Multi-robot mapping for rescue robotics

S. Carpin V. Jucikas A. Birk
International University of Bremen
Campus Ring 1 Bremen, 28725, Germany
E-mail {s.carpin,v.jucikas,a.birk}@iu-bremen.de

Abstract

We illustrate our progresses in developing multi-robot systems to be used for mapping in rescue scenarios. The problem we are currently investigating is to combine poor quality multiple maps produced by different robots into a single map to be used by human operators. In particular we motivate our approach and we illustrate the different techniques we implemented and that are at the moment being compared.
Keywords: *stochastic processes, robot mapping*

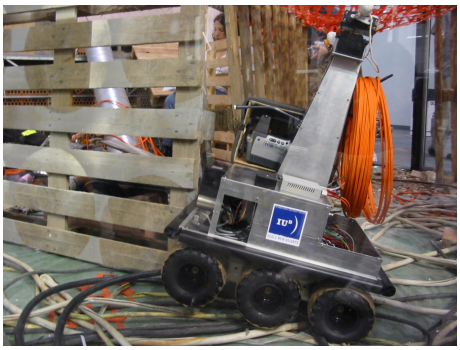
1 Introduction

One of the main tasks to be carried out by robots operating in a rescue scenario is to produce useful maps to be used by human rescue operators. Rescue environments lack a well defined structure, because of collapsed parts and debris. As a reference scenario, one can consider the rescue arenas used in the Robocup rescue league competition. Classical mapping algorithms are hard to implement and use in these environments, as issues like feature extraction and data association are hard to address. Moreover robots are supposed to move over uneven surfaces and to face significant skidding while operating. It follows that maps generated using odometric information and cheap proximity range sensors turn out to be very inaccurate. In the robotic systems we have developed this aspect is even more emphasized by our choice to implement simple mapping algorithms for their real-time execution on devices with possible low computational power [4],[7]. One of the possible ways to overcome this problem is to use multiple robots to map the same environment. The multi-robot approach has some well known advantages in itself [10],[1], most notably robustness. In the rescue framework, multi-robot systems are even more appealing because of the possibility to perform a faster exploration of the inspected area, thus increasing the chances to quickly locate victims and hazards. As the goal is to gather as much information as possible, it is evident that the maps produced by different robots will only partially overlap, as they are likely to spread around in different regions and not to stick together for the whole mission. It is then a practical issue of enormous importance to merge together such partially overlapping maps before they are used by the human operators. The problem of map merging can be seen as an optimization problem, i.e. it is necessary to find the merging parameters that produce the best matching between partially overlapping maps, accordingly to some metric. For this reason we implemented some well known optimization algorithms and we compared them with a new iterative optimization procedure we recently developed [6]. All the results we provide are not obtained via simulations, but from data collected while running our robots in the newly setup IUB rescue arena [3]. In the following we will describe the robots we developed for rescue robotics and that has been tested in the robocup rescue competition. We will then define the map merging problem and we will describe the different approaches that can be used to solve the problem. Finally, we will give numerical results and offer the conclusions.

2 The IUB rescue robots

The results presented later in section 4 are based on real-world data collected with the IUB rescue robots. The robots fall into two classes, the so-called papa goose and the mama goose design. Papa geese (figures 1.a and 1.b) are 6-wheeled robots with a footprint of $400mm \times 450mm$. Mother geese (figures 1.c and 1.d) are smaller with a footprint of $300mm \times 400mm$ and they are based on a tracked drive. In addition to their mechanical differences, they differ in their sensor payload, which is much higher for robots of the papa type. A detailed description of the robots is found in [4].
For the purpose of this paper, the merging of noisy maps of an unstructured environment, both robots type can be viewed to behave in the same way. The default localization method is odometry based on motor encoder data and the robots' kinematics. The precision of the odometry is significantly improved by

(a) Papa goose



(b) Papa goose in the red arena

(c) Mama goose

(d) Mama goose in the orange arena

Figure 1: The IUB robots at the Robocup 2003 competition in Padua, Italy

fusing compass data into the orientation estimation of the robot. Nevertheless, the basic pose estimation suffers from the well known problems of accumulative errors, leading to one of the noise sources in the map-building process. For gathering obstacle data, each robot has a low-cost laser scanner. This PB9-11 laser-scanner from Hokuyo Automatic covers 162 degrees in 91 steps up to 3m depth with a 1cm resolution (figure 2). The sensor provides rather accurate data, but it suffers from occasional spurious readings adding hence to the uncertainty in the map data.

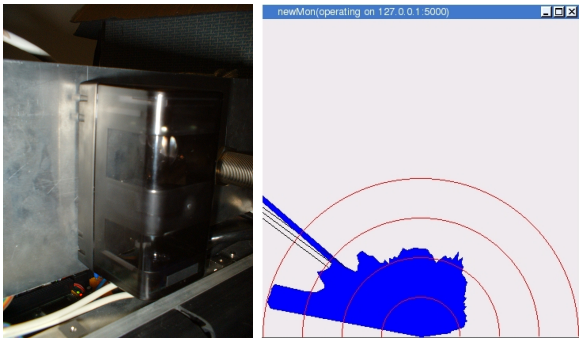


Figure 2: On the left, the Hokuyo PB9-11 sensor. On the right, a snapshot of the data provided by the PB911 sensor. It is possible to see that the robot is facing a corner in the walls (on its left, two meters ahead), as well as spurious readings (on the right)

Data acquired by onboard sensors is in transmitted in realtime to the operator station via the Framework Architecture for Selfcontrolled and Teleoperated Robots (FAST-Robots), an object-oriented network control architecture framework for robotics experiments that supports mixed teleoperation and autonomy [9]. On the operator station a map is built. Figure 3 illustrates a map produced while the robot was operating in a rescue arena setup by NIST. Maps are organized as occupancy grids. The occupancy grid accounts for three different kinds of information, namely obstacle detected, free area detected, and no information available. This information is expressed in terms of *beliefs*. The robot starts with a completely empty grid. At every time step, the input from the range scanner is acquired. By combining this with the actual pose (x,y and orientation θ) coming from the odometry measurement subsystem, it is possible to update the beliefs of the covered grid cells. Technically, every grid cell holds an integer value, initially set to 0. This means that no information is available for that grid cell. When an obstacle is detected in the grid cell, the value is incremented. When the grid cell is de-

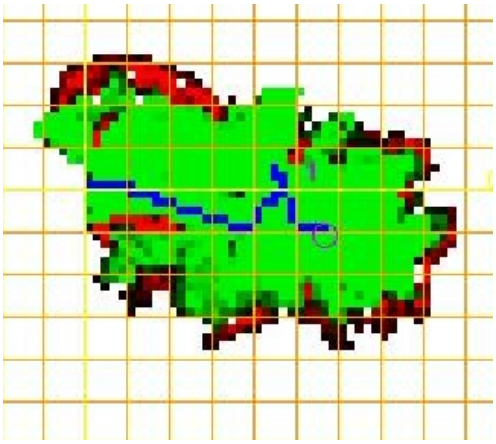


Figure 3: A map produced by the IUB rescue robot while performing in a rescue arena. Green points indicate free space, red points ocuppied space and white points indicate unknown space. The path followed by the robot is indicated by the blue dots, and the current robot position is indicated by the circle. The number *1* in the middle of the map has been put by the human operator to indicate the position of a found victim.

termined to be free, such value is decremented. Both increments and decrements are bounded. This means that such beliefs cannot arbitrarily grow or decrease when the robot is standing at a fixed position (similar to what happens with [5]).

Algorithm 1 provides the algorithmic sketch of the procedure. In order to find out if a cell is free or not (lines 5 and 11), we remind that once the position and orientation are known this is a matter of simple geometric computation. In the current setup, the current position and orientation of the robot are based on odometry and a magnetic compass, respectively.

The IUB rescue robots obtained the 4th place during the 2003 Robocup Rescue competition held in Padova (Italy) and the 2nd place during the 2004 American Open. The robots are currently being improved with map merging capabilities. The map merging is again done on the operator station, so that a human rescue worker gets an overview of the overall terrain including location marks for victims and hazards.

3 Map merging

We start with the formal definition of a map.

Definition 1 *Let N and M be two positive real numbers. An $N \times M$ map is a function*

$$m : [0, N] \times [0, M] \to \mathbb{R}.$$

Algorithm 1 Mapping procedure

```
1:  Initialization: fill the grid map with 0s
2:  loop
3:    Get data from scanner: vector v of MAX_READINGS distances
4:    Get x,y and θ from odometry
5:    for n ← 1 to MAX_READINGS do
6:      if v[n] < RELIABLE_DISTANCE then
7:        Let G[i][j] be the corresponding occupied
          grid cell (computed from x,y,θ and v[n])
8:        if G[i][j] < MAX then
9:          Increase G[i][j]
10:       for all intermediate free cells G[i][j] do
11:         if G[i][j] > MIN then
12:           Decrease G[i][j]
13:     else
14:       Discard v[n]
```

We furthermore denote with $I_{N\times M}$ the set of $N \times M$ maps. Finally, for each map, a point from its domain is declared to be the reference point. *The reference point of map m will be indicated as $R(m)$.*

The function m is a model of the beliefs encoded in the map. For example, one could assume that a positive value of $m(x, y)$ is the belief that the point (x, y) in the map is free, while a negative value indicates the opposite. Moreover, the absolute value indicates the degree of belief. The important point is that we assume that if $m(x, y) = 0$ no information is available. From now on, for sake of simplicity, we will assume $N = M$, but the whole approach holds also for $N \neq M$.

Definition 2 *Let x,y and θ be three real numbers and $m_1 \in I_{N\times N}$. We define the $\{x, y, \theta\}$-*transformation *to be the functional which transforms the map m_1 into the map m_2 obtained by the translation of $R(m_1)$ to the point (x, y) followed by a rotation of θ degrees. We will indicate it as $T_{x,y,\theta}$, and we will write $m_2 = T_{x,y,\theta}(m_1)$ to indicate that m_2 is obtained from m_1 after the application of the given $\{x, y, \theta\}$-transformation.*

Definition 3 *A* dissimilarity function *ψ over $I_{N\times N}$ is a function*

$$\psi : I_{N,N} \times I_{N,N} \to \mathbb{R}^+ \cup \{0\}$$

such that

- $\forall m_1 \in I_{N,N}\ \psi(m_1, m_1) = 0$
- *given two maps m_1 and m_2 and a transformation $T_{x,y,\theta}$, then $\psi(m_1, T_{x,y,\theta}(m_2))$ is continuous with respect to x, y and θ.*

The dissimilarity function measures how much two maps differ. In an ideal world, where robots are able to build two perfectly overlapping maps, their dissimilarity will be 0. When the maps cannot be superimposed the ψ function will return positive values.
Having set the scene, the map matching problem can be defined as follows.

> Given $m_1 \in I_{N,N}$, $m_2 \in I_{N,N}$ and a dissimilarity function ψ over $I_{N\times N}$, determine the $\{x, y, \theta\}$-transformation $T_{(x,y,\theta)}$ which minimizes
> $$\psi(m_1, T_{(x,y,\theta)}(m_2)).$$

The devised problem is clearly an *optimization* problem over $\mathbb{R}^3$. To solve optimization problems like this one, different algorithms have been proposed [11] in literature, like *multipoint hill climbing* or *simulated annealing*. In multipoint hill climbing, a population of random transformations is created, and each one is update by following a gradient descent. The approach

Algorithm 2 multi-point hill climbing

Require: $numSteps \geq 0$
Require: $numBestSamples > 0$
Require: *samples* array contains different samples for position/rotation and is sorted according to their dissimilarity
Ensure: $samples[1]$ is the best sample found

```
1:  step ← 0
2:  while step < numSteps do
3:    clear newSamples
4:    for all s in samples do
5:      for all Δs in possible position/rotation changes do
6:        Generate a new sample ns ← s + Δs
7:        c_ns ← ψ(m_1, T_ns(m_2))
8:        if c_ns < c_s then
9:          Add ns to newSamples
10:       else
11:         Discard sample ns
12:   step ← step + 1
13:   samples ← newSamples
14:   sort samples according to their dissimilarity
15:   samples ← samples[1..numBestSamples]
```

is illustrated in algorithm 2. Using a set of transformations rather than a single one, the algorithm decreases the chances of getting trapped into local minima. Simulated annealing is a well known alternative or complementary technique. The advantage is that in the early stage of its execution simulated annealing allows to accept samples which do not result in an improvement in terms of the cost function. Algorithm 3 shows our implementation of the simulated annealing approach for map fusion. In addition, we

Algorithm 3 simulated annealing

Require: $numSteps \geq 0$
Require: $numBestSamples > 0$
Require: *samples* array contains different samples for position/rotation and is sorted according to their fitness
Require: $fitnessRatio \geq 1$
Ensure: $samples[1]$ is the best sample found

```
1:  step ← 0
2:  bestFitness ← c_samples[1]
3:  lastFitness ← +∞
4:  while step < numSteps AND lastFitness/bestFitness > fitnessRatio do
5:    clear newSamples
6:    for all s in samples do
7:      for all Δs in possible position/rotation changes do
8:        Generate a new sample ns ← s + Δs/step
9:        c_ns ← ψ(m_1, T_ns(m_2))
10:       if c_ns < c_s then
11:         Add ns to newSamples
12:       else
13:         Discard sample ns
14:   step ← step + 1
15:   samples ← newSamples
16:   sort samples according to their fitness
17:   samples ← samples[1..numBestSamples]
18:   lastFitness ← bestFitness
19:   bestFitness ← c_samples[1]
```

recently introduced a further algorithm [6] which performs a randomized search based on random walks on the space of possible transformations, i.e. translations and rotations. The algorithm performs a random walk over the space of possible transformations. The interesting part is that new transformations are generated using a Gaussian distribution whose parameters are updated at each iteration. This idea comes from a recent motion planning algorithm we recent developed [8]. It is possible to prove that the proposed algorithm is probabilistic complete, i.e. if enough time is allotted it will find the optimal transformation which obtains the best overlapping between the maps being considered.

4 Numerical Results

In our implementation a map is a grid of 200 by 200 elements, whose elements can assume integer values between -255 and 255. This is actually the output

Algorithm 4 Random walk

Require: $numSteps \geq 0$

$k \leftarrow 0, \quad t_k \leftarrow t_{start}, \quad \Sigma_0 \leftarrow \Sigma_{init}, \quad \mu_0 \leftarrow \mu_{init}$
$c_0 \leftarrow \psi(m_1, T_{t_{start}}(m_2))$
while $k < numSteps$ **do**
  Generate a new sample $s \leftarrow x_k + v_k$
  $c_s \leftarrow \psi(m_1, T_s(m_2))$
  if $c_s < c_k$ OR $RS(t_k, s) = s$ **then**
    $k \leftarrow k + 1, \quad t_k \leftarrow s, \quad c_k = c_s$
    $\Sigma_k \leftarrow \text{Update}(t_k, t_{k-1}, t_{k-2}, \ldots, t_{k-M})$
    $\mu_k \leftarrow \text{Update}(x_k, t_{k-1}, t_{k-2}, \ldots, t_{k-M})$
  else
    discard the sample s

of the mapping system we described in [7]. According to such implementation, positive values indicate free space, while negative values indicate obstacles. As anticipated, the absolute value indicates the belief, while a 0 value indicates lack of knowledge. The function ψ used for driving the search over the space S is defined upon a map distance function borrowed from picture distance computation [2]. Given the maps m_1 and m_2, the function is defined as follows

$$\psi(m_1, m_2) = \sum_{c \in \mathcal{C}} d(m_1, m_2, c) + d(m_2, m_1, c)$$

$$d(m_1, m_2, c) = \frac{\sum_{m_1[p_1]=c} \min\{md(p_1, p_2) | m_2[p_2] = c\}}{\#_c(a)}$$

where

- $\mathcal{C}$ denotes a set of values assumed by m_1 or m_2,
- $m_1[p]$ denotes the value c of map m_1 at position $p = (x, y)$,
- $md(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$ is the Manhattan-distance between p_1 and p_2,
- $\#_c(m_1) = \#\{p_1 | m_1[p_1] = c\}$ is the number of cells in m_1 with value c.

Before computing D, we preprocess the maps m_1 and m_2 setting all positive values to 255 and all negative values to -255. In our case then $\mathcal{C} = \{-255, 255\}$, i.e., locations mapped as unknown are neglected. A less obvious part of the linear time implementation of the picture distance function is the computation of the numerator in the $d(m_1, m_2, c)$-equation. It is based on a so called distance-map *d-map*$_c$ for a value c. The distance-map is an array of the Manhattan-distances to the nearest point with value c in map m_2 for all positions $p_1 = (x_1, y_1)$:

$$\text{d-map}_c[x_1][y_1] = \min\{md(p_1, p_2) | m_2[p_2] = c\}$$

The distance-map *d-map*$_c$ for a value c is used as lookup-table for the computation of the sum over all cells in m_1 with value c. Algorithm 5 gives the pseudocode for the three steps carried out to built it.

Algorithm 5 The algorithm for computing *d-map*$_c$

for $y \leftarrow 0$ to $n - 1$ **do**
  for $x \leftarrow 0$ to $n - 1$ **do**
    if $M(x, y) = c$ **then**
      d-map$_c[x][y] \leftarrow 0$
    else
      d-map$_c[x][y] \leftarrow \infty$
for $y \leftarrow 0$ to $n - 1$ **do**
  for $x \leftarrow 0$ to $n - 1$ **do**
    $h \leftarrow \min($*d-map*$_c[x - 1][y] + 1,$ *d-map*$_c[x][y - 1] + 1)$
    d-map$_c[x][y] = \min($*d-map*$_c[x][y], h)$
for $y \leftarrow n - 1$ downto 0 **do**
  for $x \leftarrow n - 1$ downto 0 **do**
    $h \leftarrow \min($*d-map*$_c[x+1][y]+1, d-map_c[x][y+1] + 1)$
    d-map$_c[x][y] = \min($*d-map*$_c[x][y], h)$

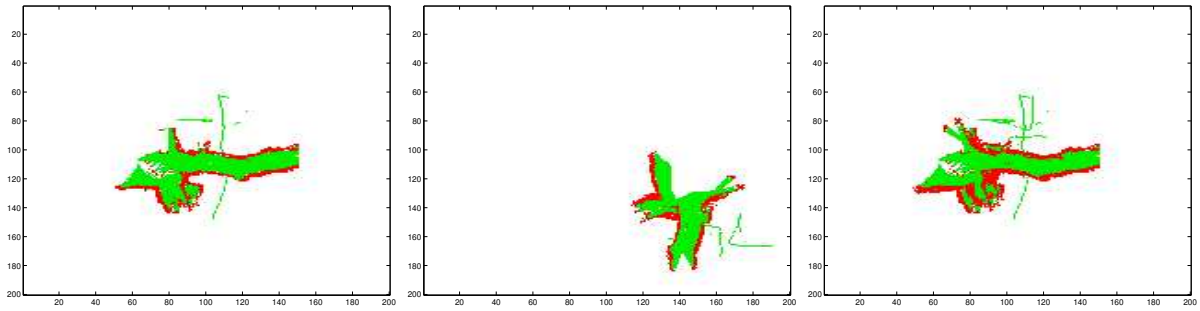
It can be appreciated that to build the lookup map it is necessary just to scan the target map for three times. In this case it is possible to avoid the quadratic matching of each grid cell in m_1 against each grid cell in m_2. We have implemented the three algorithms illustrated in the former section and we compared their performance while merging different maps in terms of speed of computation and accuracy. In addition we implemented the naive brute force algorithms that tries all the possible transformation to determine the optimal one. The results are illustrated in table 4 where we compare the different algorithms in terms of quality of matching and of execution speed, while an example of matching is illustrated in figure 4.

5 Conclusions

We have reported on our experience implementing and developing different algorithms for fusing partially overlapping maps produced by multiple robots exploring different parts of the disaster area. Different techniques have been implemented and compared over data gathered running real robots in the IUB rescue arena. The random walk algorithm we have recently introduced shows promise and appears to be an interesting approach to investigate.

Map	Brute force	Multipoint	Simulated Annealing	Random walk
Case 1	15493.3/3.06955	6.07066/4.14678	9.3172/3.2926	1.00072/4.4132
Case 2	43462.5/3.48718	8.97039/4.42068	12.9822/3.64833	1.3959/4.3684
Case 3	38994.7/6.52768	7.77098/6.92824	12.0962/6.68582	1.24792/7.1016
Case 4	16798.7/3.43908	7.2469/3.81709	9.73415/3.67773	1.07923/4.0440

Table 1: Comparative results. In each column the first number is the time spent (in seconds) and the second number is the obtained dissimilarity value.



(a) (b) (c)

Figure 4: Different maps created by the robots and their combination obtained by our matching technique.

References

[1] T. Arai, E. Pagello, and L.E. Parker. Guest editorial advances in multirobot systems. *IEEE Transactions on Robotics and Automation*, 18(5):655–661, 2002.

[2] A. Birk. Learning geometric concepts with an evolutionary algorithm. In *Proc. of The Fifth Annual Conference on Evolutionary Programming*. The MIT Press, Cambridge, 1996.

[3] A. Birk. The iub rescue arena, a testbed for rescue robots research. In *Proceedings of SSRR 2004*, 2004.

[4] A. Birk, S. Carpin, and H. Kenn. The iub 2003 rescue robot team. In *Robocup 2003*. Springer, 2003.

[5] S. Biswas, B. Limketkai, S. Sanner, and S. Thrun. Towards object mapping in dynamic environments with mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelliegent Robots and Systems*, pages 1014–1019, 2002.

[6] S. Carpin and A. Birk. Stochastic map merging in rescue environments. In *Robocup 2004*. Springer, 2004. Accepted for pubblication.

[7] S. Carpin, H. Kenn, and A. Birk. Autonomous mapping in the real robot rescue league. In *Robocup 2003*. Springer, 2003.

[8] S. Carpin and G. Pillonetto. Motion planning using adaptive random walks. *IEEE Transactions on Robotics and Automation*, To appear.

[9] H. Kenn, S. Carpin, M. Pfingsthorn, B. Liebald, I. Hepes, C. Ciocov, and A. Birk. Fast-robots: A rapid-prototyping framework for intelligent mobile robots. In *Proceedings of the 3rd International Conference on Artificial Intelligence and Applications*, pages 76–81. ACTA Press, 2003.

[10] L.E. Parker. Current state of the art in distributed autonomous mobile robots. In L.E. Parker, G. Bekey, and J.Barhen, editors, *Distributed Autonomous Robotic Systems 4*, pages 3–12. Springer, 2000.

[11] S. Russel and P. Norwig. *Artificial Intelligence - A modern approach*. Prentice Hall International, 1995.