Map Merging of Rotated, Corrupted, and Different Scale Maps using Rectangular Features

Jinyoung Park, Andrew J. Sinclair
Aerospace Engineering
Auburn University
Auburn, Alabama, USA
jzp0020@auburn.edu

Ryan E. Sherrill, Emily A. Doucette, J. Willard Curtis
Air Force Research Laboratory
Eglin Air Force Base
Florida, USA

***Abstract*—Integrating data from multiple cooperative robots can be important for expanding their individual capabilities. In an environmental mapping scenario, multiple ground robots map different local areas. Algorithm complexity on merging the maps to build a global map depends on the three factors: orientation, accuracy and scale of the maps. When the three factors are all unknown, the map merging becomes a challenging problem. In this paper, a new approach on merging of two maps with the three factors are unknown. The idea is to estimate the best shared-areas by means of rectangular features. The information of dimensions and connections of maximal empty rectangles allows the algorithms to match orientations and scales, also to find overlapping points. The advantage of this approach is that a map merging is accomplished without any location estimations between the robots. This paper explains the map-merging process with an example of a simple environment, and presents a result with a practical environment.**

***Keywords*—map merging, maximal rectangles, RANSAC, multi-robot mapping, rotated maps.**

I. Introduction

Increased demand for a high degree of agent survivability and efficiency has caused an increase in task allocation of cooperative agents. Networked agents accomplish a common mission by sharing and combining their information. In an environmental mapping scenario of multiple agents, they map their areas, and share the information of their maps. The agents can be aerial or ground vehicles. This research is about map merging of multiple ground robots. The computational complexity for the map merging relies on the information from on-board sensors. When the robots know their sensory information such as compass/heading directions, real-time locations, etc., the complexity will be much reduced. Also, obstacle-detecting sensors such as laser, mono/stereo camera, sonar [3], and infrared sensors make different noise levels, and affect the map accuracy. In the case such information is unknown and only map images are given, the three differences between the maps make the map merging challenging: orientation, accuracy and scale.

Map merging can be processed with mapping simultaneously or after mapping separately. Many applications take simultaneous mapping and merging because mapping requires sensory information for location estimations or communications between the robots, and the information is also beneficial for map merging. Several estimation methods using pose or odometry information are provided in [2]. Also, simultaneous localization and mapping (SLAM) using probabilistic techniques is described in [1] and [5], and additional radio frequency sensors can be adopted for exact locations as illustrated in [4]. These single-robot mapping techniques are advanced to the multi-robot map-merging problems. Multiple efforts have researched map merging relying on communications: collaborative multi-robots build a global map by distribution exploration-tasks considering the distances between unexplored areas and the robots [6], and map merging algorithms estimate locations of the robots by recognizing landmarks using visual cameras and communications between them [7]. In the case of limited communications, the robots select communication-valid robots, and try to contact them first to share their maps [11]. However, these approaches require sufficient information to solve the map merging problem. When the merging is processed after mapping separately without such information, the three different factors between maps become important. Some researchers have done a map merging with unknown initial poses of the robots, and algorithms estimate their poses by particle filter [12]. Also, a map merging with only maps is achieved using similarity and stochastic searching algorithms. In the research, the maps have different orientations, and the approach is rotating a map to find the maximum overlapping areas in two maps [8]. Similarly, Hough spectral information can be extracted from map images, and cross correlation is computed to find the maximum similarity [9]. However, the maps used in these researches are in the same scale, and a series of rotating maps produces distortions. Another research suggests a solution to the distortion problem by topological mapping approach [10]. Since topological map illustrates an environment concisely with edges and vertices, the process becomes a graph-matching problem. Also, the degrees of edges in both maps must match exactly.

This paper suggests a novel approach to the separated map-merging problem. Since the map merging is separated from mapping process, the vehicle's location estimation is not necessary. Also, it is assumed that the three factors, orientations, accuracy and scales, are unknown. The problem statement is

This research is sponsored by Air Force Research Laboratory.

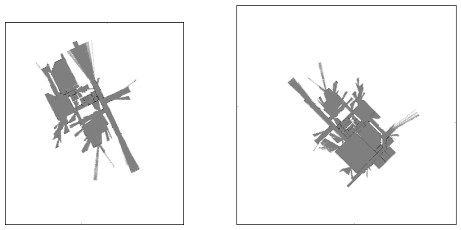

(a) map 1 (b) map 2

Fig. 1. Maps of a practical environment from two ground robots. The orientations, accuracy and scales are unknown. The result of map merging is shown in section IV.

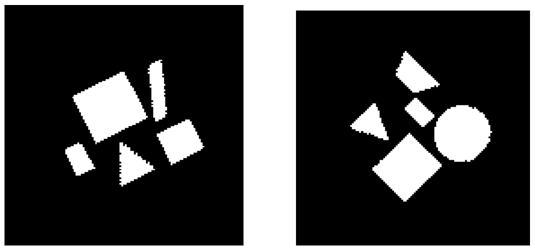

(a) map 1 (b) map 2

Fig. 2. Example of a simple environment. The maps have naturally different accuracy and scales to each other, and intentionally rotated.

given in section II. Detailed algorithms are explained with an example of a simple environment and pseudo-codes in section III. A map-merging result of a practical environment from figure 1 is provided in section IV. Finally, conclusion is provided in section V.

II. Problem Statement

Two identical robots with laser obstacle-detection sensors are used for mapping an indoor environment. Since this map merging is a separate process from mapping unlike SLAM, here it is assumed that two local maps are already acquired. The maps shown in figure 1 are the practical maps from the robots. The maps distinguish different areas with colors: empty spaces with gray, obstacles with black and unexplored areas with white. Even though the mapping robots are identical, the maps naturally have errors/corruption of several cells and different scales. The orientations of the maps are intentionally rotated and differed to each other, and the rotations make another distortions. A map merging with these conditions is challenging problem because the three factors (orientation, accuracy, and scale) are connected to each other. Further rotations to find the best shared-area produces more distortions. The more distortions/corruptions plus different scales make the problem more difficult to match the orientations, and yield more rotations.

In this proposed approach, first, the algorithms extract rectangular features from empty areas, and place the maps in the orthogonal orientations. Further rotations will be 90°-rotations (90°, 180°, 270°) and do not introduce further distortions. This is because occupancy grid maps represent environments by square cells. Also, the algorithms find the same orientation and scale by estimating the best shared-areas. Finally, a merged map is accomplished by overlapping the areas.

III. Map Merging

A. Rectangular Maps

The idea of the R-map is the integration of empty cells of the grid map into a maximal empty rectangle. The key issue of R-map algorithm is finding maximal empty rectangles in the form of non-overlapping rectangles by calculating the values in an image represented with binary numbers. Since grid maps express its environment with two cases, occupied and free elements, a grid map is considered as a binary matrix, 0 representing obstacles and 1 representing free spaces. The R-map algorithm finds maximal empty rectangles (MER) and their connections by manipulating the numbers. The features of those MERs are the key resources to solve the map merging problem. Further details can be found in [13]

A simple environment example is given in figure 2. A grid map of the environment is defined as an $(m \times n)$ binary matrix, G. Matrix Gx is initialized as a copy of G. Also, a matrix cx has size $((m+1) \times (n+1))$ with an additional row and column with elements equal to zero. The first $(m \times n)$ elements of cx are obtained by right-to-left row-wise summations of contiguous empty spaces in G. For example, if row-wise contiguous three elements in Gx are all 1's, the corresponding elements in cx are 1, 2 and 3 from right to left.

$$Gx(y,x) \in \{0,1\} \tag{1a}$$
$$cx(y,x) \in \{0,\cdots,n\} \tag{1b}$$

$$cx(y,x) = \begin{cases} cx(y,x+1)+1 & \text{if } Gx(y,x)=1 \quad \text{(2a)} \\ 0 & \text{if } Gx(y,x)=0 \quad \text{(2b)} \end{cases}$$

Here, x is a column number within $1 \leq x \leq n$, and y is a row number within $1 \leq y \leq m$. The survey to find maximal empty rectangles progresses column by column. This survey of each column considers the maximal rectangles that can be formed with the left edge of the rectangles in that column. The candidate rectangles under current consideration are stored in T. Each entry in T stores the width and the top row number of that rectangle. As the survey for each element proceeds down the x-th column, the value of the next element $cx(y+1,x)$ is compared to the value of the current element $cx(y,x)$. There are four possibilities of the comparison. In the first possibility, if these values are equal, then no changes are made to T. In the second possibility, if the $(y+1)$-th value is greater than the y-th value, then a new rectangle is added to T. The new rectangle has a top row equal to $(y+1)$ and a width equal to $cx(y+1,x)$. In the third possibility, if the $(y+1)$-th value of cx is less than the y-th value, the areas of the rectangles, A, in T are calculated and compared to the area of the largest previously found rectangle. The larger area is updated to the largest area, A_{max}. As the areas are calculated, the widths of the rectangles with width greater than $cx(y+1,x)$ in T are updated to $cx(y+1,x)$. Of all the resulting rectangles with width $cx(y+1,x)$ only the one with the lowest row number is retained, and the other rectangles with this width (if any) are removed from T. In the forth possibility, because the $(m+1)$-th value of cx is zero, all rectangles are removed at the end of the column. The survey then proceeds to the next column. After the survey to the last column, the largest rectangle is the maximal empty rectangle and the property of the rectangle is saved as $r_i = [y_i, x_i, w_i, h_i]$ where (y, x) coordinates of the upper-left corner, width and height of i-th largest rectangle. Algorithm 1 illustrates how to find an MER.

Algorithm 1 Maximal Empty Rectangles

```
1:  procedure FINDMER($Gx$)
2:      $[m, n]$ = size of $Gx$
3:      compute $cx$
4:      define $w = 0$
5:      define $A_{max} = 0$
6:      $k = 1$
7:      for $x = 1$ to $(n+1)$ do
8:          for $y = 1$ to $(m+1)$ do
9:              if $cx(y, x) \geq w$ then
10:                 $w = cx(y, x)$
11:                 $T(k) = [y, w]$
12:                 $k = k + 1$
13:             end if
14:             if $cx(y, x) < w$ then
15:                 $y0 = T(k, 1)$
16:                 $w0 = T(k, 2)$
17:                 $A = w0 \times (y - y0)$
18:                 if $A > A_{max}$ then
19:                     $r = [x, y0, w0, y - y0]$
20:                     $A_{max} = A$
21:                 end if
22:             end if
23:         end for
24:     end for
25: end procedure
```

For the elements of an MER from cx, the corresponding elements of Gx are updated to zeros. This updating ensures that the subsequent iteration only searches the remaining free space. The process of finding an MER and updating its elements to zeros are iterated until all of the elements in Gx have been set to zero or until a desired threshold is reached in terms of the number or size of the maximal rectangles, related to the desired R-map resolution. To find what neighbors are connected to the rectangles, the rectangles need to be labeled: the elements of r_i are updated to i. Thus the values of surrounding elements of the rectangle indicate connections of the rectangle, c_i.

$$c_{iU} = \{G(y-1, x) \ \cdots \ G(y-1, x+w)\} \tag{3a}$$
$$c_{iD} = \{G(y+h, x) \ \cdots \ G(y+h, x+w)\} \tag{3b}$$
$$c_{iL} = \{G(y, x-1) \ \cdots \ G(y+h, x-1)\} \tag{3c}$$
$$c_{iR} = \{G(y, x+w) \ \cdots \ G(y+h, x+w)\} \tag{3d}$$

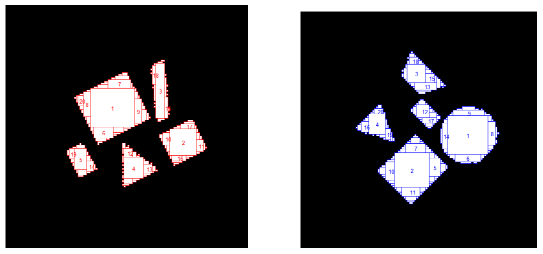


Fig. 3. Results from R-map algorithm. Map 1 and map 2 have 142 and 174 rectangles, respectively. The first twenty rectangles are labeled only for a clearer view.

Here, c_{iU}, c_{iD}, c_{iL} and c_{iR} are the upper-, lower-, left- and right-connections of the r_i, respectively. The unique numbers except 0's from equation 3 are the connections of the r_i.

$$c_i \subset \{c_{iU}, c_{iD}, c_{iL}, c_{iR}\}, \quad 0 \notin c_i \tag{4}$$

Therefore, the input to the R-map algorithm is a binary matrix, G, and the output is the properties of maximal empty rectangles and their connections, r and c. Figure 3 shows the results from the R-map algorithm. The empty spaces in the maps 1 and 2 consist of 142 and 174 rectangles, respectively. The first twenty rectangles are labeled only in each map for a clearer view. Algorithm 2 shows the process of R-map, and algorithm 1 is processed at line 4.

Algorithm 2 Rectangular Maps

```
1: procedure RMAP($G$)
2:     define $Gx = G$
3:     while sum(sum($Gx$)) $\neq 0$ do
4:         $r$ = FindMER($Gx$)
5:         $Gx$ = GridMapUpdate($Gx, r$)
6:     end while
7:     $c$ = ComputeConnections($G, r$)
8: end procedure
```

B. Orthogonal Orientations

Merging maps requires solving for the offset, scaling, and orientation that relates the maps. As will be described further in section III-C, this will be done by matching rectangles in the two maps. However, the rectangles of a given area can appear quite different depending on the orientation of the maps. As an example, in figure 3, rectangle 1 in map 1 corresponds to some of the same area as rectangle 2 in map 2. But the rectangles are significantly different size due to the different orientations of the two maps. In this sense, R-maps have preferred directions: the orthogonal directions along which they search for the height and width of the rectangles. Human-constructed environments tend to have a lot of spaces made up of polygons with orthogonal angles. For this type of environment, it can be advantageous to rotate the map to line up with the rows and columns of the R-map algorithm before performing the R-mapping. This has two advantages: first, empty spaces can be more efficiently filled with fewer

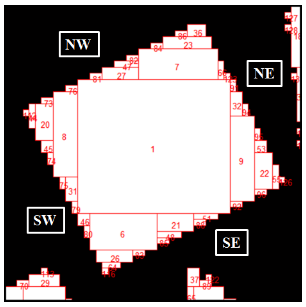


Fig. 4. Zoomed in image of the top-left of figure 3a for the rectangular empty space. A series of the largest neighbor along each edge-direction divide which rectangles are on which side of the environment.

direction	1st	2nd	3rd	4th	5th
upper	7	23	36	·	·
lower	6	26	64	116	·
left	8	20	44	112	·
right	9	22	55	126	·

TABLE I. The largest neighbor of rectangle 1 along each edge-direction.

rectangles and second, solving for the relative orientation between the maps has been reduced to four possibilities which are rotations of multiples of 90 degrees. Aligning the environment with the map grids proceeds with two steps. First, finding points on the edges of the empty spaces by sorting the neighboring rectangles. Second, finding angles of those edges by random sample consensus (RANSAC) [14].

The points on each edge are sorted out by recognizing the largest neighbor along each edge. A series of the largest neighbors along each edge-direction divide which rectangles are on which side of the environment. Figure 4 is the top-left portion of the environment of figure 3a. Rectangle 1, the largest one in the environment, has neighbors around it. Table I shows the series of the largest neighbors to each edge-direction. Searching a series of the largest neighbors stops when there are no further connections on any direction. For example, the largest rectangles to upper-direction are 7, 23 and 36. Those rectangles divide their neighbors: left neighbors of them are on NW-edge, and right neighbors of them are on NE-edge, and those neighbors are denoted as upper-left and left-upper connections of rectangle 1. Since the series of the largest neighbor to upper-direction ends at third connection, the series to the other directions also take up to third connection.

$$c_{1UL} = \{7, 23, 27, 36, 47, 82, 84, 86\} \tag{5a}$$
$$c_{1LU} = \{8, 20, 44, 73, 76\} \tag{5b}$$
$$c_{1UR} = \{7, 23, 36, 66\} \tag{5c}$$
$$c_{1RU} = \{9, 22, 32, 53, 55, 94\} \tag{5d}$$
$$c_{1DL} = \{6, 26, 46, 64, 80\} \tag{5e}$$
$$c_{1LD} = \{8, 20, 31, 44, 45, 75\} \tag{5f}$$
$$c_{1DR} = \{6, 21, 26, 48, 64, 83, 85\} \tag{5g}$$
$$c_{1RD} = \{9, 22, 55, 92, 96\} \tag{5h}$$

Equation 5 shows the lists of edge-touching rectangles on the eight directions from rectangle 1. As it can be seen in figure 4, rectangles on two directions cover one edge of the environment: c_{1UL} and c_{1LU} for NW-edge, c_{1UR} and c_{1RU} for NE-edge, c_{1DL} and c_{1LD} for SW-edge, and c_{1DR} and c_{1RD} for SE-edge. Also, the largest rectangle in the environment, rectangle 1, can be the edge-touching rectangle. Thus in general, the list of rectangles on each edge of an empty space having the largest rectangle i can be written as shown in equation 6.

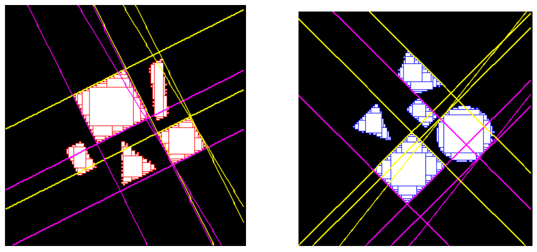
$$NW_i = \{i, c_{iUL}, c_{iLU}\} \tag{6a}$$
$$NE_i = \{i, c_{iUR}, c_{iRU}\} \tag{6b}$$
$$SW_i = \{i, c_{iDL}, c_{iLD}\} \tag{6c}$$
$$SE_i = \{i, c_{iDR}, c_{iRD}\} \tag{6d}$$

Since the rectangles in an R-map are in the maximum sizes, larger rectangles are located closer to the center of an empty space. Thus, equation 6 should be applied to the larger rectangles, $i = 1, \cdots, N$, to extract rectangles on the edges of empty spaces, where N is a user-select variable and expected between 5 and 10 because a map environment mostly has a rectangular empty space within the first five rectangles also, smaller rectangles than r_{10} may not the largest rectangle in their empty spaces and $N > 10$ can be redundant computational expenses.

Next, finding angles of the edges to align the edges with the R-map axes. Two rectangles from each edge can make a line segment. Note that the coordinates of an edge-touching vertex of a rectangle depend on the four edges in equation 6 and are easily obtained using the properties of the rectangles: (x, y) for NW-edge, $(y, x + w)$ for NE-edge, $(y + h, x)$ for SW-edge and $(y+h, x+w)$ for SE-edge. Combinations of two rectangles from each given set of equation 6 make all possible line segments. The number of two-combinations is denoted in equation 7.

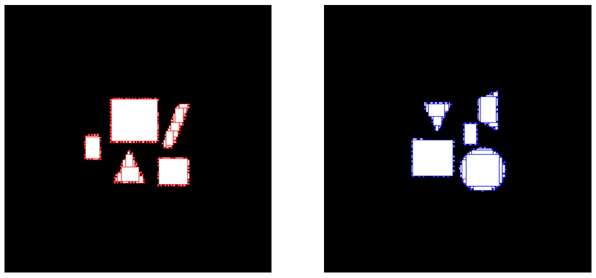
$$_{n_{edge}}\mathrm{C}_2 = \binom{n_{edge}}{2} = \frac{n_{edge}!}{2!(n_{edge} - 2)!} \tag{7}$$

Here, n_{edge} is the number of elements in a corresponding edge in equation 6. For example, according to equations 5a, 5b and 6a, NW_1 has 14 elements ($n_{NW} = 14$), and equation 7 tells that there are 91 line segments on NW-edge. With those lines, to count the number of line segments with similar slopes, the idea of RANSAC is applied here. In this case, both sample data set and parameters are the same as line segments, and it is a counting line segments of the data set laying in a region made by a parameter allowing a tolerance, $\pm\tau$, in y-axis. The number of data set within the region is noted as *inlier*. This process is applied to the four edges. For the each line segment, if there is a line segment with the same slope on the facing-edge (NW-SE and NE-SW), the two parallel lines are selected as a pair of valid. The example in figure 5 illustrates the valid lines: one yellow and one pink lines are a pair of parallel lines. Finally, the slope of a line segment with the most many *inlier* becomes the rotation angle, Λ, to place a map in the orthogonal orientation. The true initial rotation angles of figure 5a and 5b are $\Lambda_0^{(1)} = 27°$ and $\Lambda_0^{(2)} = 225°$ counterclockwise, and the algorithm returns calculated rotation



(a) map 1 (b) map 2

Fig. 5. Pairs of parallel lines: one yellow and one pink lines are a pair. The initial rotation angles are $\Lambda_0^{(1)} = 27°$ and $\Lambda_0^{(2)} = 225°$ counterclockwise, and estimated angles are $\Lambda^{(1)} = -26.5651°$ and $\Lambda^{(2)} = -45°$ for map 1 and map 2, respectively.



(a) map 1 (b) map 2

Fig. 6. Results from orthogonal-orientations algorithm. Orthogonal oriented maps are acquired through rotating the initial maps by estimated angles: $\Lambda_0^{(1)} + \Lambda^{(1)} = 0.4349°$ and $\Lambda_0^{(2)} + \Lambda^{(2)} = 180°$ for the map 1 and map 2, respectively.

angles of $\Lambda^{(1)} = -26.5651°$ and $\Lambda^{(2)} = -45.0°$, respectively. Therefore, orthogonal oriented maps are acquired by rotating the initial maps by the estimated angles as shown in figure 6: $\Lambda_0^{(1)} + \Lambda^{(1)} = 0.4349°$ and $\Lambda_0^{(2)} + \Lambda^{(2)} = 180°$ for the maps 1 and 2. This angle estimation has small errors due to corruptions and distortions in maps. Algorithm 3 illustrates how to estimate rotation angles.

Algorithm 3 Orthogonal Orientations

```
1: procedure ANGLE($r, c$)
2:     define $\tau$
3:     define $inlier = 0$
4:     for $i = 1$ to $N$ do
5:         extract $edge = [NW_i; NE_i; SW_i; SE_i]$
6:         for $j = 1$ to $4$ do
7:             $lines =$ combinations of $edge(j,:)$
8:             for $p = 1$ to $n_{lines}$ do
9:                 $[(X_1, Y_1), (X_2, Y_2)] = lines(p)$
10:                $slope = (Y_2 - Y_1)/(X_2 - X_1)$
11:                $region = [(X_1, Y_1 + \tau), (X_2, Y_2 + \tau);$
                             $(X_1, Y_1 - \tau), (X_2, Y_2 - \tau)]$
12:                for $q = 1$ to $n_{lines}$ do
13:                    $[(x_1, y_1), (x_2, y_2)] = lines(k)$
14:                    if $[(x_1, y_1), (x_2, y_2)] \subset region$ then
15:                        $inlier = inlier + 1$
16:                    end if
17:                end for
18:                $lines(p) = [lines(p), slope, inlier]$
19:            end for
20:        end for
21:    end for
22:    $valid_{NW-SE} =$ lines with same slope on NW-SE
23:    $valid_{NE-SW} =$ lines with same slope on NE-SW
24:    $\Lambda = slope$ of a line with most many $inlier$
           from $[valid_{NW-SE}, valid_{NE-SW}]$
25: end procedure
```

C. Shared Triangles

When the edges of empty spaces are not aligned with the map grids, the R-map algorithm returns smaller rectangles to fill out edge-near areas. However, when the spaces are in the orthogonal orientations so their edges are aligned with the map grids, the spaces can be fitted with fewer and larger rectangles (the spaces may have micro-rectangles around larger rectangles due to distortional uneven edges). As shown in figure 6, once the empty spaces are in the orthogonal orientations, rectangles can be kept in the same dimensions while rotating if the rotation angles are 90°-angles (this is an advantage of using the maximal size of rectangles). Since both maps in figure 6 are in the orthogonal orientations but the orientation-matching is unknown, an effort is required to select one from the four of 90°-rotation angles to match their orientations. In this step, the shared-triangle algorithm provides the best three shared-rectangles from each map by means of rectangular and triangular features, where the rectangular features are from the R-map algorithm and the triangular features are from triangular formations of three selected rectangles. A set of two shared-triangles (a triangle by the best three rectangles from each map) helps to find the same orientation and scale.

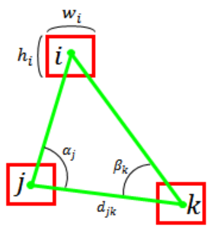
The maps from the previous steps can have corruptions from obstacle-sensing errors and distortions by non-90°-rotations (0°∼360°). Therefore, a tolerance, δ, is allowed in finding shared-triangles. Due to the tolerance, every rectangle in map 1 should have a list of corresponding rectangles in map 2. Recall that r_i has the property of $[x_i, y_i, w_i, h_i]$.

$$r_i^{(1)} \triangleq \left\{ \overline{r_i}^{(2)} : (w_i^{(1)} \pm \delta) \cap (h_i^{(1)} \pm \delta) \right\} \tag{8}$$

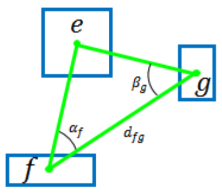
In the form of a triangle by three rectangles, equation 8 implies that $r_i^{(1)}$ is a vertex of a triangle in map 1 and it has multiple corresponding vertices in map 2, depicted as $\overline{r_i}^{(2)}$. Hence, three of equation 8 make a triangle in map 1, and its multiple corresponding triangles in map 2. The list of all possible triangles, T, is obtained by combinations of three rectangles in map 1. Equation 9 is counting three-combinations.

$${}_{n_r^{(1)}}\mathrm{C}_3 = \binom{n_r^{(1)}}{3} = \frac{n_r^{(1)}!}{3!(n_r^{(1)} - 3)!} \tag{9}$$

Here, $n_r^{(1)}$ is the number of rectangles in map 1. According to equation 9, figure 6a has 169 rectangles and there are 790,244 possible three-combinations out of them. Notice that



(a) map 1



(b) map 2

Fig. 7. A set of shared-triangles. (a) A combination of $\Delta^{(1)} = \{r_i, r_j, r_k\}$ in map 1 makes a triangle. Each rectangle from $\Delta^{(1)}$ have corresponding rectangles in map 2 within a tolerance. (b) One of possible triangles by $\Delta^{(1)}$'s corresponding rectangles, $\Delta^{(2)} = \{r_e, r_f, r_g\}$. The feature vector, FV, contains the triangular and rectangular features.

the computational burden can be reduced by eliminating smaller rectangles because the micro-rectangles near the edges of empty spaces have non-remarkable features. When the rectangles with area smaller than 10 are eliminated from the list of $r^{(1)}$, $n_r^{(1)}$ becomes only 14, and the number of combinations will be much reduced to 364. Depending on the value of δ, the number of corresponding rectangles in map 2 can vary. A triangle in map 1 and one of its corresponding triangle in map 2 are a set of shared-triangles. The rectangular and triangular features are extracted from the set and compared to find the most similar triangles. Figure 7 illustrates a set of triangles by $\{r_i, r_j, r_k\}^{(1)}$ and $\{r_e, r_f, r_g\}^{(2)}$. To find rectangles in not only the dimensions but also their formations, the area of each rectangle and the two of vertex angles, α and β, are adopted as the features. Also, the areas are normalized for comparisons by the distance between the second and third rectangles, d. Each triangle has a feature vector, FV.

$$FV^{(1)} = \begin{bmatrix} (w_i \times h_i)/d_{jk} \\ (w_j \times h_j)/d_{jk} \\ (w_k \times h_k)/d_{jk} \\ \alpha_j \\ \beta_k \end{bmatrix}, FV^{(2)} = \begin{bmatrix} (w_e \times h_e)/d_{fg} \\ (w_f \times h_f)/d_{fg} \\ (w_g \times h_g)/d_{fg} \\ \alpha_f \\ \beta_g \end{bmatrix} \tag{10}$$

A cost function, J, with the feature vectors is defined.

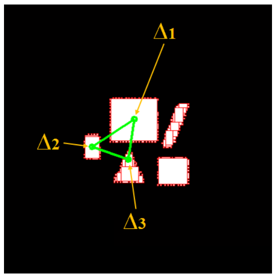
$$J = \left|FV^{(1)} - FV^{(2)}\right|^T \cdot W \cdot \left|FV^{(1)} - FV^{(2)}\right| \tag{11}$$

Here, W is a weight matrix to give weights to desired features. In this example, an identity matrix is applied. The minimum cost refers a set of the best shared-triangles.

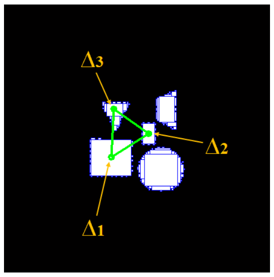
$$\Delta^{(1)} = \{r_i, r_j, r_k\}^{(1)}_{\min(J)} \tag{12a}$$

$$\Delta^{(2)} = \{r_e, r_f, r_g\}^{(2)}_{\min(J)} \tag{12b}$$

Here, $\Delta^{(1)}$ and $\Delta^{(2)}$ denote the best shared-triangles in map 1 and map 2 in the order of size. Figure 8 shows $\Delta^{(1)}$ and $\Delta^{(2)}$ with the green triangles. The first and second shared-rectangles are selected correctly ($\Delta_1^{(1)} = \Delta_1^{(2)}$ and $\Delta_2^{(1)} = \Delta_2^{(2)}$) but the last rectangle in the triangular empty space in each map is not the same ($\Delta_3^{(1)} \neq \Delta_3^{(2)}$). It occurs because of the



(a) map 1



(b) map 2

Fig. 8. Results of shared-triangles algorithm. A set of the best shared-triangles has min(J). Δ_1's and Δ_2's in both maps are correct shared-rectangles, but Δ_3's are incorrect due to different scales.

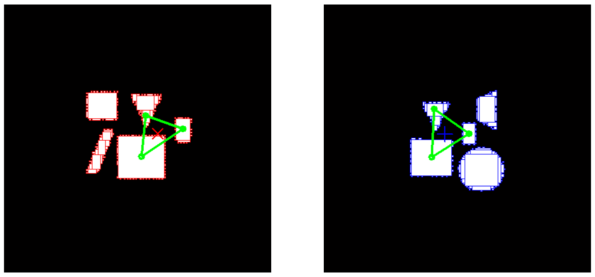
difference in scales. Even though maps are in different scales, the algorithm usually selects the correct Δ_1's and Δ_2's in both maps, but not for the last rectangles, Δ_3's. This is because Δ_1's and Δ_2's are larger ones even among all rectangles across map environments, and most of them are separated to each other with some distance (even two neighboring larger rectangles have some distance between their center points due to their dimensions). However, Δ_3's are smaller ones, thus similar rectangles can exist across the map because of the non-rectangular environments or corruptions. Therefore, $\Delta^{(1)}$ and $\Delta^{(2)}$ can have Δ_3's of similar dimensions but at slightly different locations, not at big different locations because FV contains features for the triangular-shape matching. Algorithm 4 contributes how to find the best set of shared-triangles.

Algorithm 4 Shared Triangles

```
1: procedure TRIANGLE($r^{(1)}, r^{(2)}$)
2:     define $\delta$
3:     $k = 1$
4:     for $i = 1$ to $n_r^{(1)}$ do
5:         $R_i^{(2)} = [r^{(2)} : (w_i^{(1)} \pm \delta) \cap (h_i^{(1)} \pm \delta)]$
6:     end for
7:     $T^{(1)} =$ combinations of $r^{(1)}$
8:     for $p = 1$ to $n_T^{(1)}$ do
9:         define $FV^{(1)}(p)$
10:        $T^{(2)} =$ combinations of $R^{(2)}$
11:        for $q = 1$ to $n_T^{(2)}$ do
12:            define $FV^{(2)}(q)$
13:            $J = \left|FV^{(1)} - FV^{(2)}\right|^T \cdot W \cdot \left|FV^{(1)} - FV^{(2)}\right|$
14:            $shareT(k) = [T^{(1)}, T^{(2)}, J]$
15:        end for
16:    end for
17:    $[\Delta^{(1)}, \Delta^{(2)}, J] = shareT_{\min(J)}$
18: end procedure
```

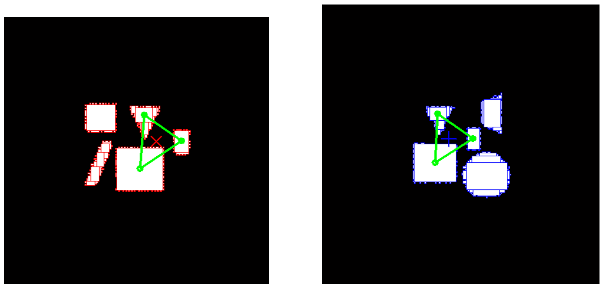
D. Rescaling and Merging

The R-map algorithm produces the resources, r and c, from map images to estimate rotation angles and the best shared-triangles. As maps are updated by each estimating process, the resources need to be re-produced for the next estimations. Therefore, the map-merging algorithm is a series



(a) map 1 (b) map 2

Fig. 9. Orientation-matched maps. Bisector-vectors of triangles in figure 8 give final rotation angle, $\Lambda_f = 180°$. Orientations are matched through rotating map 1 by Λ_f. Δ_3's are still incorrect shared-rectangles.



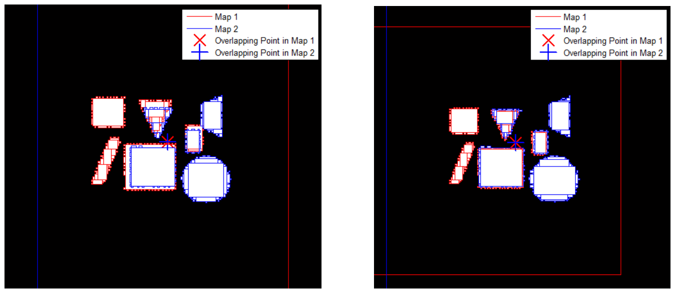
(a) map 1 (b) map 2

Fig. 10. Scale-matched maps, $ratio = 1$. Figure 9 has $ratio = 1.1429$, and scales are matched through rescaling map 2 by the $ratio$. Δ_3's are correct shared-rectangles. The center points of the triangles are the overlapping points.

of estimations of the three unknown factors reproducing R-maps. The R-map algorithm firstly takes the maps from ground robots. Using the list of rectangles and their connections, the orthogonal-orientation algorithm estimates a rotation angle, Λ, to place initially rotated maps in the orthogonal orientations. The orthogonal oriented maps are defined as map_r, and corresponding new rectangles, r_r, are required for the next estimations. In the finding shared-triangle step, the algorithm provides the best shared-triangles, $\Delta^{(1)}$ and $\Delta^{(2)}$. Since Δ's are the common triangles in both maps, matching them yields the same orientation. The final rotation angle, Λ_f, is obtained by matching bisector-vectors passing through the midpoint between Δ_2 and Δ_3 from Δ_1. Since the maps are already in the orthogonal orientations, the final rotation angle should be one of the 90°-angles. Rotating map 1 by Λ_f makes map 1 and map 2 in the same orientation as shown in figure 9. In this example, $\Lambda_f = 180°$. Their orientations are matched but Δ_3's in both maps are still incorrect shared-rectangles because of the different scales. For the scale matching, as discussed in section III-C, Δ_1's and Δ_2's are mostly the correct shared-rectangles, thus matching a dimension of Δ_1's yields the same scale. A scale ratio is defined as shown in equation 13.

$$ratio = w^{(1)}_{\Delta 1} / w^{(2)}_{\Delta 1} \tag{13}$$

Here, $w^{(1)}_{\Delta 1}$ and $w^{(2)}_{\Delta 1}$ are the widths of $\Delta^{(1)}_1$ and $\Delta^{(2)}_1$, respectively. Note that the heights of them can be taken instead of the widths. By simply rescaling map 2 by $ratio$, both maps will


Map 1
Map 2
Overlapping Point in Map 1
Overlapping Point in Map 2
Map 1
Map 2
Overlapping Point in Map 1
Overlapping Point in Map 2

(a) without rescaling. (b) with rescaling.

Fig. 11. Merged maps. (a) Merging maps of figure 9 makes a worse fit. (b) Merging maps of figure 10 makes a better fit.

finally become in the same scale. From the example of figure 9, $ratio$ is 1.1429. This implies that the map 1 is 1.1429 times larger than the map 2 and the map 2 is rescaled by 1.1429. Since the map 2 has been changed, entire process from R-mapping should be repeated with the rescaled maps. Thus each iteration estimates orientation and scale to match the maps. The iteration is processed until the maps are in the same scale. Finally, figure 10 shows the maps in the same orientation and scale. After the orientation and scale matchings, $ratio = 1$, the shared-triangles algorithm returns the correct set of $\Delta^{(1)}$ and $\Delta^{(2)}$. The $ratio = 1$ means that the orientations and scales are matched, and the common areas are found. Therefore, a merged map is accomplished by overlapping the maps. The overlapping points are each center of $\Delta^{(1)}$ and $\Delta^{(2)}$. Figure 11a illustrates a merging map without rescaling, and the maps 1 and 2 have a worse fit. However, in figure 11b, the merged map has a better fit after rescaling. Algorithm 5 refers the map merging process.

Algorithm 5 Map Merging

```
1: procedure MERGE($map^{(1)}, map^{(2)}$)
2:     $[r^{(1)}, c^{(1)}] = RMAP(map^{(1)})$
3:     $[r^{(2)}, c^{(2)}] = RMAP(map^{(2)})$
4:     $\Lambda^{(1)} = ANGLE(r^{(1)}, c^{(1)})$
5:     $\Lambda^{(2)} = ANGLE(r^{(2)}, c^{(2)})$
6:     $map^{(1)}_r$ = rotate $map^{(1)}$ by $\Lambda^{(1)}$
7:     $map^{(2)}_r$ = rotate $map^{(2)}$ by $\Lambda^{(2)}$
8:     define $ratio = 0$
9:     while $ratio \neq 1$ do
10:        $r^{(1)}_r = RMAP(map^{(1)}_r)$
11:        $r^{(2)}_r = RMAP(map^{(2)}_r)$
12:        $[\Delta^{(1)}, \Delta^{(2)}] = TRIANGLE(r^{(1)}_r, r^{(2)}_r)$
13:        $\Lambda_f$ = compare bisector-vectors of $\Delta^{(1)}$ and $\Delta^{(2)}$
14:        $map^{(1)}_r$ = rotate $map^{(1)}_r$ by $\Lambda_f$
15:        define $ratio = w^{(1)}_{\Delta 1} / w^{(2)}_{\Delta 1}$
16:        rescale $map^{(2)}_r$ by $ratio$
17:    end while
18:    $MAP$ = overlap centers of $\Delta^{(1)}$ and $\Delta^{(2)}$
19: end procedure
```

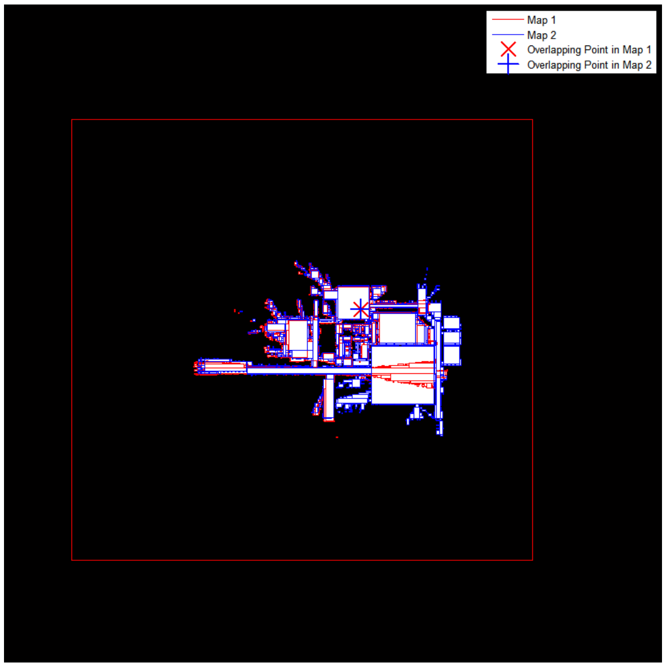


Fig. 12. Merged map of practical environment from figure 1

IV. Result

The practical maps of figure 1 is acquired from two identical ground robots. The maps naturally have corruptions and different scales. The orientations of the maps 1 and 2 are intentionally rotated by $\Lambda_0^{(1)} = 27°$ and $\Lambda_0^{(2)} = 225°$, respectively. The orthogonal-orientations algorithm estimates the rotation angles $\Lambda^{(1)} = 63.4349°$ and $\Lambda^{(2)} = 45°$ for each, thus both maps are in the orthogonal orientations through rotating the maps by $\Lambda_0^{(1)}+\Lambda^{(1)} = 90.4349°$ and $\Lambda_0^{(2)}+\Lambda^{(2)} = 270°$, respectively. With the orthogonal oriented maps, the shared-triangles algorithm returns $\Delta^{(1)} = \{r_3, r_{10}, r_{11}\}$ and $\Delta^{(2)} = \{r_2, r_6, r_{14}\}$ with $\min(J) = 68.0174$. From the best shared-triangles, the comparison of the bisector-vectors gives $\Lambda_f = 180°$, hence both maps are in the same orientation through rotating map 1 by 180°. With the same oriented maps, the shared-triangle algorithm returns $\Delta^{(1)} = \{r_3, r_{10}, r_{20}\}$ and $\Delta^{(2)} = \{r_2, r_6, r_{20}\}$ with $\min(J) = 69.4954$. The comparison of their bisector-vectors gives finally $\Lambda_f = 0°$, and the scale ratio of the two maps is $ratio = 1.0313$. Thus, map 2 is rescaled by 1.0313, and since the map 2 has been changed, entire process is repeated with the rescaled maps. After the series of the algorithms, R-map, orthogonal-orientations, and shared-triangles, the scale ratio becomes $ratio = 1.1515$ in this second iteration, and the map 2 is rescaled again by 1.1515. The entire process is repeated again with the newly estimated maps until the scale ratio meets $ratio = 1$. Finally in the third iteration, the algorithms return $\Delta^{(1)} = \{r_1, r_3, r_{13}\}$, $\Delta^{(2)} = \{r_2, r_3, r_{19}\}$, and $ratio = 1$. Figure 12 shows the final merged map with a good fit of the map 1 and map 2.

In the form of occupancy grid maps, the map 1 and map 2 in figure 1 have resolutions (361×317) and (393×393), respectively. The simulation was processed in MATLAB on a normal laptop PC and the computation time is about 80 seconds through the three iterations. Simpler environments in lower resolutions will take shorter computation time for map merging.

V. Conclusion

A new approach to the separate map merging problem has been suggested. The key technique in this approach is extracting the properties and connections of maximal empty rectangles from maps that allows to match orientations and scales, and find overlapping points. Because the three factors and sensory information are unknown, the final merged map is an estimated map by minimizing differences of the factors. As mentioned, sensory information is required for a better accuracy in a merged map.

To expect a good fit of two maps in a merged map, initial maps require two conditions. First, the environment in each map should have at least one pair of parallel edges and longer makes a better estimation. Thus, empty spaces such as rectangles or trapezoids are essential to place the maps in the orthogonal orientations by aligning the parallel edges with the map grids. Second, the initial maps should have at least three of separate rectangular spaces or a non-rectangular space to find three rectangles as shared-rectangles. Hence, the R-map algorithm produces rectangles and the shared-triangle algorithm finds three of common rectangles. Fortunately, most environments in practical cases meet the two conditions.

This approach can be modified by adopting techniques from other map-merging researches for faster computations and better accuracy.

References

[1] S. Thrun, "A probabilistic on-line mapping algorithm for teams of mobile robots," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 335-363, May 2001.

[2] S. Thrun, "Robotic mapping: a survey," Technical Report CNU-CS-02-111, Carnegie Mellon University, 2002.

[3] J. Tardós, J. Neira, P. Newman, and J. Leonard, "Robust mapping and localization in indoor environments using sonar data," *The International Journal of Robotics Research*, vol. 21, no. 4, pp. 311-330, April 2002.

[4] D. Hahnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose, "Mapping and localization with RFID technology," *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 1, pp. 1015-1020, 26 April-1 May 2004.

[5] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229-241, Jun 2001.

[6] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, pp. 476-481, 24-28 April 2000.

[7] N. Özkucur, and H. Akin, "Cooperative multi-robot map merging using fast-SLAM," *RoboCup 2009: Robot Soccer World Cup XIII*, vol. 5949, pp. 449-460, Springer Berlin Heidelberg, 2009.

[8] A. Birk, and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384-1397, July 2006.

[9] S. Carpin, "Fast and accurate map merging for multi-robot systems," *Autonomous Robots*, vol. 25, no. 3, pp. 305-316. October 2008.

[10] W. Huang, and K. Beevers, "Topological map merging," *The International Journal of Robotics Research*, vol. 24, no. 8, pp. 601-613, August 2005.

[11] K. Konolige, D. Fox, B. Limketkai, J. Ko, and B. Stewart, ”Map merging for distributed robot navigation,” *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 1, pp. 212-217, October 2003.

[12] H. Lee, S. Lee, M. Choi, and B. Lee, ”Probabilistic map merging for multi-robot RBPF-SLAM with unknown initial poses,” *Robotica*, vol. 30, no. 2, pp. 205-220, March 2012.

[13] J. Park, and A. Sinclair, ”Mapping for path planning using maximal empty rectangles,” *AIAA Guidance, Navigation, and Control (GNC) Conference*, pp. 5096, 19-22 August 2013.

[14] M. Fischler, and R. Bolles, ”Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381-395, June 1981.