Rough Mereology based CFill algorithm for robotic path planning

Lukasz Zmudzinski[0000−0002−9127−0895]

University of Warmia and Mazury in Olsztyn, Faculty of Mathematics and Computer Science, Słoneczna 54, 10-710 Olsztyn, Poland, lukasz@zmudzinski.me

Abstract. This paper focuses on describing the CFill algorithm for rough mereology based intelligent agent path planning. The algorithm updates the methodology of the Square Fill Algorithm by increasing its adaptiveness, adding dynamic neighbour building and proposing a way to deal with dynamic changes to the robot environment, by implementing tree based path planning. The author describes the changes to the original algorithm with example values and their outcomes.

Keywords: Path planning · Rough Mereology · Potential Fields · Robot Navigation · Mereogeometry · Robotics.

1 Introduction

Path planning for mobile robotics is one of the currently most researched scientific topics. One of the algorithms currently present for path planning using rough mereology techniques is the Square Fill Algorithm [3]. While generating a valid path to the goal for an autonomous agent, the algorithm is memory-efficient, making it hard to use for memory-low devices. Moreover, every appearance of an obstacle after calculating the path means recalculating the whole algorithm.

The author tries to challenge the problems and proposes a CFill algorithm, based on the previously mentioned. The changes that were made include replacing the method for creating potential field neighbours to a dynamic one, adding the *narrowing mode* described further in the paper and a new way of calculating paths, that tackles the problem of dynamic entities entering the map area.

The idea to implement tree based path planning for intelligent agents was lately used in works including the Rapidly exploring Random Tree*[8], achieving tasks for swarm robotics[1] or task allocations for multi-robot systems[9].

2 Rough mereology

One of the biggest challenges in robotics is to determine the position of the world elements such as robots, obstacles or navigation points like beacons. While using various sensor mounted on the mobile agent, an estimate value is always gathered, due to the probability of inaccurate readings. Those readings are prone to error, because of many environmental and mechanical factors for example: the type of surface, range sensors not picking material fluctuations or poor camera resolution qualities. This is where Mereology as a theory can help.

The main scheme of Mereology is the notion of part. Any notion of a part that is given, relates to the idea of *containment* or *partial containment* . This means that the robot entering a beforehand specified area can ascertain its connection to the area, by a degree. Given the information whether it is connected to the area, overlaps it or is its part, the robot is able to establish its position value more precisely. Given the notion carried out by Rough Mereology - *part to a degree*, a real number can be retrieved in the range of 0...1 (value of 1 meaning full inclusion, while 0 - no inclusion) to give a precise description of the element location, compared to the target area.

Rough mereology based reasoning as described in [6] employs the notion of a rough inclusion $\mu(x, y, r)$, which relation needs x is a part of y to a degree of at least r. As our reasoning is concerned with spatial objects, the rough inclusion involved in our reasoning is the one defined as $\mu(X, Y, r)$ if and only if $\frac{|X \cap Y|}{X} >= r$, where X, Y are n-dimensional solids and $|X|$ is the n-volume of X.

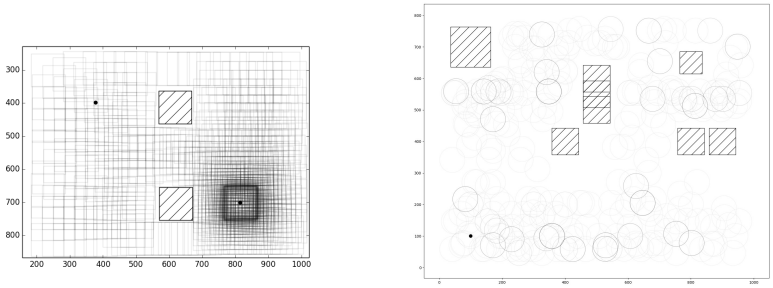
Considering a planar case of an autonomous mobile robot moving in a 3-dimensional environment, hence, spatial objects X, Y are figures assumed concept regions and $|X|$ is the area of X. The rough inclusion $\mu(X, Y, r)$ is applied in the construction of the mereological potential field. Elements of this field are square and the distance between them is defined as

$$K(X, Y) = min\{max_r \mu(X, Y, r)\}, max_s \mu(Y, X, s)\}.$$

Classical methodology of potential fields works with integrable force field given by formulas of Coulomb or Newton which prescribe force at a given point as inversely proportional to the squared distance from the target. In consequence the potential is inversely proportional to the distance from the target. The basic property of the potential is that its density (force) increases in the direction toward the target. We observe this property in our construction. We start in our construction with a rough inclusion – a similarity measure formalized in the theory of rough mereology. Its basic construct is a *rough inclusion*. Rough inclusion is a ternary relation $\mu : U \times U \times [0, 1]$ and its primitive formulas of the form $\mu(x, y, r)$ read *'the object x is a part to object y to a degree of at least r'*. A rough inclusion can be used to induce a distance function as well as primitive relations of elementary geometry: betweenness and nearness.

Geometry induced by means of a rough inclusion can be used to define a generalized potential field: the force field in this construction can be interpreted as the density of squares that fill the workspace and the potential is the integral of the density. We present now the details of this construction, see also [3, 4].

The potential field generation methods called Square Fill Algorithm introduced by Osmiałowski and Polkowski [3] was presented with some alternative modifications to the algorithm and/or team based path planning in [7, 11, 2, 10, 5]. The main differences between the algorithm and CFill algorithm will be presented in the following section. You can see a graphical comparison of both field creation methods in Fig. 1.



(a) (b)

Fig. 1: The figure on the left (a) shows the potential field generated by the Square Fill Algorithm. The potential field generated by the CFill algorithm is shown in the right figure (b).

3 Creating potential fields using CFill algorithm

The CFill algorithm takes the core element of the Square Fill Algorithm - potential fields neighbour generation - and modifies it for better memory optimization and environment adaptation. This comes with two vast changes to the way the field is generated.

First of all, the potential field neighbours are now created dynamically. Instead of eight neighbours at set positions as in the Square Fill Algorithm, the algorithm specifies a *neighbours* variable, that holds the number of neighbours that should be created around the potential field (at equal intervals). This is achieved by calculating the angle α in radians as following:

$$\alpha = \frac{350}{neighbours} * \frac{\pi}{180} \tag{1}$$

where n is the selected number of neighbours to be created. The angle is multiplied each time a new neighbour is created in an iteration i, until the total angle remains smaller than 2π radians. To calculate each neighbour position (x', y') with a distance d from the potential field (x, y) the following equation is used:

$$\begin{aligned} x' &= sin(\alpha_i) * d \pm x \\ y' &= -cos(\alpha_i) * d + y \end{aligned} \tag{2}$$

Secondly, the generated potential field is sparse in open spaces and gets more dense between obstacles. Variable named *narrowing_distance* was proposed to achieve that. Whenever it is set, the algorithm checks the distance between the current potential field and all obstacles and goes into the *narrowing mode* whenever the distance is smaller than *narrowing_distance*. The *narrowing mode* has two tools that can be used for creating a more dense potential field: creating more neighbours around a potential field or decreasing the step size for smaller distances. Ideally both tools should be used to fit through narrow passages. An example of this method can be seen on Fig. 2.

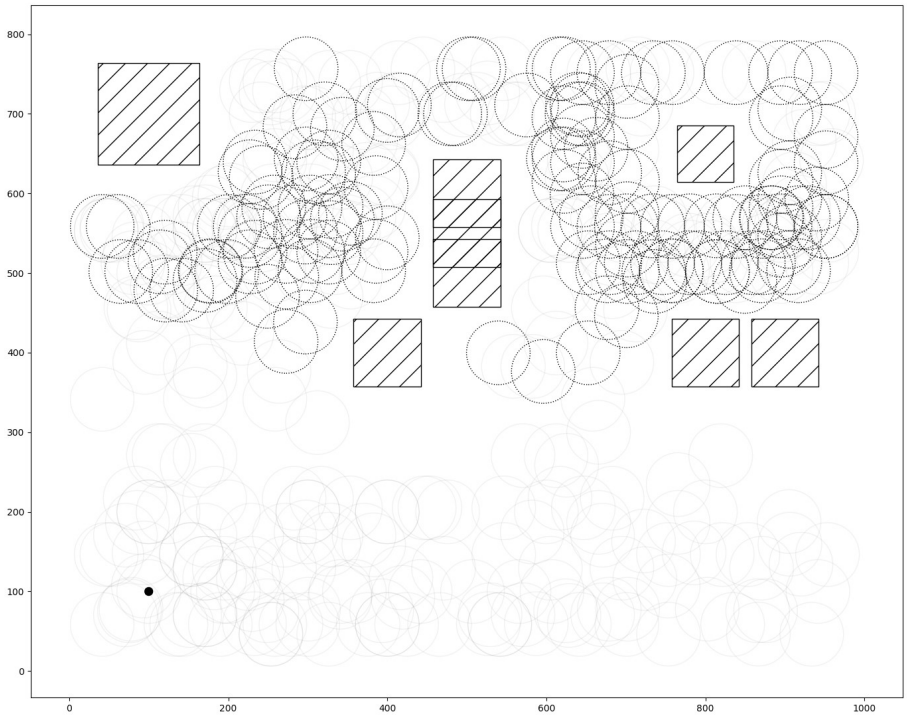


Fig. 2: Additional potential fields (represented by dotted circles) that were created by running the algorithm with the *narrow_distance* parameter set to 60 units.

3.1 CFill Algorithm

The CFill algorithm is composed of the following steps (pseudocode can be seen in Algorithm listing 1):

1. Set the following variables:

- *neighbours* - contains how many neighbours should be created,
- *step* - the incrementation step distance between potential fields,
- *radius* - the radius of the created potential fields,
- *goal* - position of the goal for the algorithm,
- *potential_fields* - list to store created potential fields,
- *narrow_distance* - distance to activate the narrowing mode (optional),
- *narrow_step* - increment step distance for narrowing mode (optional),
- *narrow_neighbours* - neighbours in narrowing mode (optional).

2. Create an empty queue Q,
3. Set the *is_clockwise* variable to **True**,
4. Add the first potential field at the *goal* location with a *distance* of 0,
5. Enumerate through all potential fields in Q,
6. Check the conditions, if any is **True**, remove the potential field from Q and go to next potential field:
   - Check, if the potential field is out of bounds of the map,
   - Check, if a potential field at that location is already created,
   - Check, if the potential field isn't overlapping an obstacle.
7. If the narrowing mode is set and the potential field is in the distance of *narrow_distance* to obstacle:
   - Create neighbours with number *narrow_neighbours* and *narrow_step* for distance incrementor from the potential field as described above,
   - Add the neighbours to queue Q,
8. If the narrowing mod is not set or the potential field isn't in the distance of *narrow_distance* to obstacle:
   - Create neighbours of the potential field with the number *neighbours* and *step* for distance incrementor as described above,
   - Add the neighbours to queue Q
9. Change the *is_clockwise* value to opposite,
10. Remove the potential field from queue Q,
11. Add the potential field to the *potential_fields* list.

Algorithm 1 CFill Algorithm

```
1: procedure CFILL
2:     set neighbours, step, radius, goal, direction
3:     set n_distance, n_step, n_neighbours
4:     potential_fields ← new empty array
5:     Q ← new empty array
6:     potential_field ← goal_x, goal_y, current_distance
7:     Q.append(potential_field)
8: For field in Q:
9:     if Q is empty then end
10:    if field.out_bounds or field.created or field.collision then
11:        Q.remove(field)
12:        continue
13:    if narrowing_mode is True then
14:        new_neighbours ← define_neighbours(direction, n_neighbours, n_step)
15:    else
16:        new_neighbours ← define_neighbours(direction, neighbours, step)
17:    Q.append(new_neighbours)
18:    direction ← not direction
19:    Q.remove(field)
```

4 Path planning using tree graphs

Making the potential field sparse in CFill algorithm compared to the Square Fill Algorithm required implementing a new way of calculating paths. The previously used method of picking neighbours by selecting new path points by finding the nearest mereological distance between connected potential fields can't be used in CFill algorithm at most times. The proposed algorithm for path planning is devised of two steps: creating possible paths tree and selecting a path, based on navigation to the tree root.

4.1 Creating possible paths tree

Below are the steps taken to build a tree graph on the generated mereological potential fields. An example of a proposed tree can be seen in Fig. 3.

1. Append the goal position to the $working\_fields$ list,
2. Set the $tree\_radius$ parameter responsible for determining how far should the algorith look for neighbouring potential fields,
3. Enumerate through the created potential field list until all potential fields have their *parent* parameter set,
4. Check the following condition for enumerated field:
   - Potential field distance is smaller or equal to $tree\_radius$,
   - Potential field is not in the $working\_fields$ list,
   - Potential field does not have the *parent* parameter set.

5. If the conditions are met:
   – Assign the *parent* parameter of current potential field to the closest mereological field from the $working\_fields$ list,
   – Add the potential field to the $working\_fields$ list.

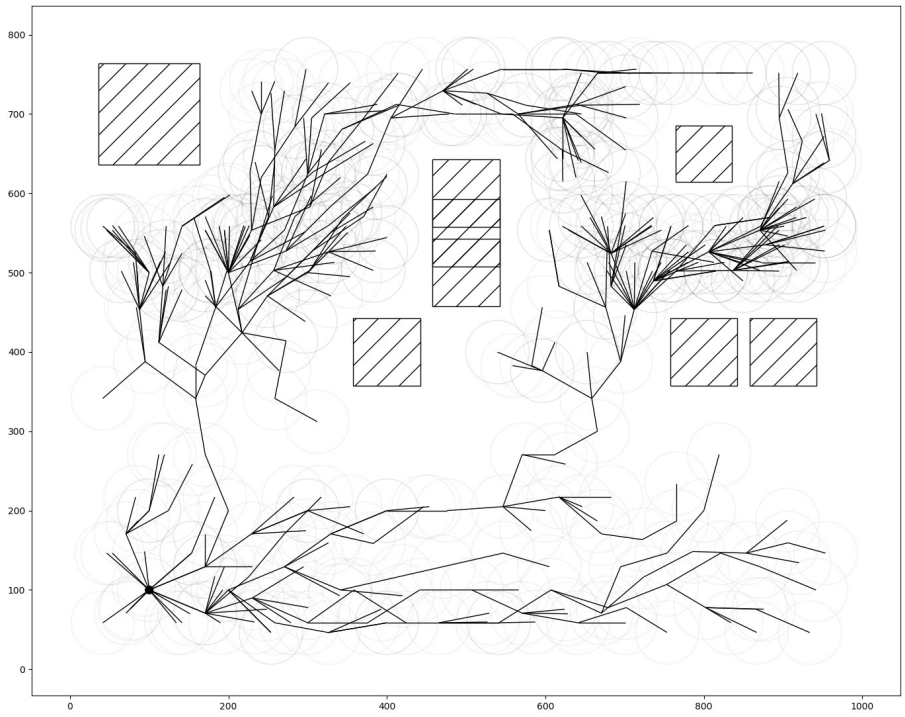


Fig. 3: Path tree graph generated from the modified mereology potential field, with the root at the goal position by visiting all created fields.

4.2 Picking a path

After the tree is generated, a path for the autonomous agent can be chosen. This means following the steps:

1. If the robot is at the goal position - end algorithm,
2. Add the robot position to *path_points* list,
3. Find the mereological potential fields closest to the agent start position and set it to
4. Do the following, until the full path is constructed:
   – Add the *current* potential field position to *path_points* list,
   – Get the *parent* parameter of the potential field and assign it to *current*.

4.3 Path smoothing

After the path points are acquired the author uses a smoothing method modified for rough mereology path planning algorithms presented in [10]. An example of the smoothed path based on the path generated from the path tree can be found in Fig. 4. The smoothing method has the following steps:

1. Smoothing weights α and β are selected,
2. The following steps are iterated over n times until they produce a result:
   - Data weight α is applied and moves the position of the point x_k depending on the position of the previous x_{k-1} and next x_{k+1} point on the given path:
   $$x_k = x_k + \alpha(x_{k-1} + x_{k+1} - 2x_k) \tag{3}$$
   - Counter balacing the updated position $x+k$ with a chosen smooth weight β, so that a straight line isn't created.
   $$y_k = y_k + \beta(x_k - y_k) \tag{4}$$
   - The value of the path point is updated, if the distance from the point to any obstacle is greater than the computed collision distance. In other cases, the value remains the same.

4.4 Dynamic changing environment

Due to the nature of the proposed path planning method, contraty to the Square Fill Algorithm, it is possible to use the generated path system for dynamic environments, without recalculating the potential field and/or the proposed path. When a new obstacle is added after the initial calculations, the potential fields in the vacinity get blacked out by setting their *active* parameter to $False$. The effect of this is that all branches of the path tree starting or going through that point are turned off. From this point, finding a new path is similar to initial finding of the path as described above - an active point closest to the current robot position is searched for and a new path is selected.

5 Conclusion

The CFill algorithm is the successor to the Square Fill Algorithm proposed by Osmialowski and Polkowski in [4]. The algorithm focuses on delivering an updated method of creating the mereological potential field, by doing so dynamically, allowing parametrization, instead of static content. This allows the creation of a set number of neighbours based on the *neighbours* parameter passed as an input along with the *step* like in the predecessor.

This manipulation allowed the implementation of the *narrowing mode*, which gives the possibility of creating more dense potential fields around obstacles,

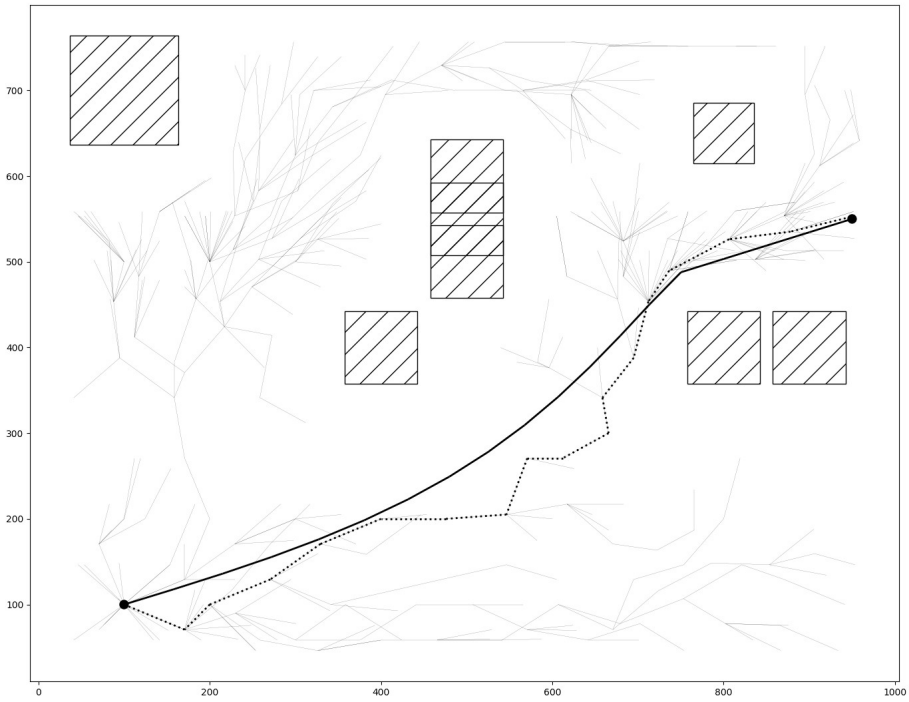


Fig. 4: The figure shows the initial path selected from the path tree graph (shown by the dotted line) as well as the smoothed out path (represented by the solid line) with parameters: $n = 100$, $\alpha = 0.1$ and $\beta = 0.1$.

while making them sparse in open terrain - greately reducing the memory needed to calculate the mereologic potential field.

Moreover, due to how the algorithm created potential fields, the old way of implementing path finding was deprecated. A new solution was presented using tree graphs that were based on the earlier created potential fields. The greatest advantage of this, is that the algorithm is now valid in a changing environment, since adding new obstacles after the path tree was created, does not induce the need of recalculating the whole potential field or generated paths.

Future work will include further dive into the subject of using CFill in changing environments and bigger comparison of the algorithm with other rough mereology based ideas. More work can be done in the field of the path tree creation as only one method was tested on how to pick branch roots for the mereological potential field.

References

1. Divband, S.M., Zahadat, P., Ghofrani, J., Hamann, H.: Adaptive Path Formation in Self-Assembling Robot Swarms by Tree-Like Vascular Morphogenesis, Distributed Autonomous Robotic Systems, Springer International Publishing, pp. 299–311, 2019
2. Gnys P.: Mereogeometry Based Approach for Behavioral Robotics. In: Polkowski L. et al. (eds) Rough Sets. IJCRS 2017. Lecture Notes in Computer Science, vol 10314. Springer, Cham, 2017
3. Osmialowski, P.: On path planning for mobile robots: Introducing the mereological potential field method in the framework of mereological spatial Reasoning. Journal of Automation, Mobile Robotics and Intelligent Systems (JAMRIS) 3(2), pp. 24-33, 2009
4. Osmialowski P., Polkowski L.: Spatial Reasoning Based on Rough Mereology: A Notion of a Robot Formation and Path Planning Problem for Formations of Mobile Autonomous Robots, In: Transactions on Rough Sets XII, pp.143-169 (2010)
5. Szmigielski A., Polkowski L.: Computing from words via rough mereology in mobile robot navigation, In: Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on
6. Polkowski L.: Rough mereology: A new paradigm for approximate reasoning, In: International Journal of Approximate Reasoning, Volume 15, Issue 4, pp. 333-365, November 1996
7. Polkowski, L., Zmudzinski, L., Artiemjew, P.: Robot navigation and path planning by means of rough mereology, In: Proceedings of IEEE International Conference on Robotic Computing, 2018.
8. Véras, L. G., Medeiros, F. L., Guimarães, L. N. (2019). Rapidly exploring Random Tree* with a sampling method based on Sukharev grids and convex vertices of safety hulls of obstacles. International Journal of Advanced Robotic Systems.
9. Zhou, X., Wang, H., Ding, B., Hu, T., Shang, S.: Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm, Expert Systems with Applications, v. 116, pp. 10-20, 2019
10. Zmudzinski, L., Artiemjew, P.: Path planning based on potential fields from rough mereology, In: Proceedings of International Joint Conference on Rough Sets, IJCRS'17, Olsztyn, Poland, Lecture Notes in Computer Science (LNCS), vol. 10303, pp. 158-168. Springer, Heidelberg (2017)
11. Zmudzinski, L., Polkowski, L., Artiemjew, P.: Controlling robot formations by means of spatial reasoning based on rough mereology, In: Advances in Robotics Research, vol. 2(3), pp. 219-236, Techno-Press (2018)