Switching formation of robotic swarms in occluded environments using sampling-based algorithms

Sreejeet Maity
Indian Institute of Science
Bangalore, India
sreejeetm@iisc.ac.in

Nilay Srivastava
Indian Institute of Science
Bangalore, India
nilays@iisc.ac.in

Mohit Kumar Gupta
Indian Institute of Science
Bangalore,India
mohitg@iisc.ac.in

***Abstract*—This paper introduces a sampling-based algorithm to reconfigure the classical multi-robot motion planning problem to realize a wide range of significant formations and internal switching patterns. A swarm of geometrically identical robots does not call for a unique target for each agent. As a result, any possible permutations of the target configurations are mathematically valid, reducing complexity significantly. With the help of the above-mentioned class of algorithms, we explored and successfully implemented specific formations of our interest. The contributions can prove handy in many real-life industrial problems such as object transportation in occluded environments in the presence/absence of static obstacles ensuring zero agent-agent and obstacle-agent collision.**

***Index Terms*—swarm robotics, motion-planning, sampling, switching formation**

I. Introduction

A multi-robot system is a collection of robotic units that cooperate to solve problems of various interests. One of the crucial aspects of swarms is their ability to configure them in multiple configurations. This feature enables the agents to implement a particular class of cooperative tasks. Apart from that, the central theme of motion planning is to find collision-free paths for each swarm agent through an occluded environment in the presence of obstacles. Some significant applied problems related to swarm robotics are more concerned with creating formation. One such example is the object-carrying problem in unknown, occluded environments. While carrying an object of any typical geometry, balance is maintained by placing agents in particular positions, and thus formation making becomes necessary.

A group of agents in a swarm should go through thorough motion planning to achieve the desired formation. We also need to ensure that the motion of the individual agents should be collision-free. On graphs, the multi-robot motion planning problem is reconfigured as pebble motion [2]. For simplicity, if we consider identical robots [3] gives us an insight into the unlabelled version of pebble motion on graphs[2]. [1] provided a sampling-based algorithm UPUMP that helps us to achieve the same. We wish to use the UPUMP algorithm to generate formations of various interests. However, [1] showed that a swarm might be composed of groups of non-identical agents.

In this paper, we start with geometrically identical disc robots. Since all the robots are similar, we reduce it to an unlabelled problem as described in [2]. We implement various classes of formations by tuning the number of robots. The major contribution of this paper revolves around algorithms that enable us to successfully switch between any start configuration to any pre-specified target configuration.

The paper is organized as follows. Section II discusses the mathematical preliminaries. Section III takes into account the algorithms required to achieve our desired goal. Section IV describes the experimental setup. Section V deals with the results of our experiments and the limitations of our algorithms. Section VI discusses further work and application of our algorithm.

II. Mathematical Preliminaries

A. Primordial terminologies

We start with a robot r which is supposed to work in a workspace W. In this paper, we consider the geometry of the workspace to be rectangular. Let C = $\{c_1, c_2, ..., c_m \mid c_i \epsilon F\}$ be a set of m single robot configurations be called configuration. Let configuration C occupied by the robots be called placements. Now, we denote by $F(r)$ the free space of a robot r: the set of all collision-free single-robot configurations. Given s, $t \epsilon F(r)$, there exists a path for the robot r from s to t is a continuous function $\pi : [0, 1] \rightarrow F(r)$, such that $\pi(0) = s$, $\pi(1) = t$. To extend our argument of free space, we can assume with mathematical certainty that the geometrically identical robots share the same free space. Mathematically, given two identical robots r and r_1, their free space are respectively equal to $F(r)$ = $F(r_1)$.

Let R = $\{r_1$,r_2,...,$r_m\}$ be a set of m geometrically identical robots in the workspace W. And we define $r(c) \subset$ C, for c ϵ F, to represent the section of the workspace which is covered by a robot r ϵ R placed in the single-robot configuration c. Note that two robots from R collide, when placed in $c, c' \epsilon$ F, if $r(c) \cap r(c') \neq \phi$.

B. Unlabelled multi robot motion planning

In the unlabelled multi robot motion planning, we start with an unlabelled problem U = (R, S, T), where R ($|R| = m$) be the set of m geometrically identical robots and S = $\{s_1$,s_2,...,$s_m\}$, T = $\{t_1$,t_2,...,$t_m\}$ be the start and target placements, respectively ensuring $|S|$ = $|T|$ = m. The goal of the problem is to find the path π_u = $\{\pi_1$,...,$\pi_m\}$ where, π_i

is the path for the i^{th} robot such that $\pi(0)$ = $s_i \epsilon S$ and $\pi(1)$ = $t \epsilon T$.

C. Pebble motion problem

Before we introduce the focal algorithm in the next section, we take a moment to discuss a tweaked version of a pebble motion problem. In the pebble motion problem, we place m pebbles in a pebble graph $G(V,E)$ with $|V| = n$, taking $n > m$. We can move pebbles along the edges of the same graph, which allows us to achieve a series of configurations. However, we add one additional constraint such that only one pebble can move at a time.

In a broader mathematical sense, let us consider two placements P_1 = $\{p_{11}$,...,$p_{1m}\}$ and P_2 = $\{p_{21}$,...,$p_{2m}\}$ in a graph $G(V,E)$. These two placements are valid in a pebble motion problem if (p_{1i},p_{2j}) ϵ E and $p_{1k} = p_{1l} \; \forall \; k \neq i$ and $l \neq j$.

We can check whether a pebble motion problem has a solution or not by introducing a simple mathematical test which we call as signature [1] throughout this paper. Let V' be a pebble placement of a pebble problem $P(G,S,T,m)$ and let $\{G_1,...,G_h\}$ be the set of maximal connected subgraphs of G, where G_i = (V_i, E_i).The signature of V' is defined as $\{|V_i \cap V'|\}_{i=1}^{h}$. Now, we claim that the equivalency of two placements V', V'' on a pebble graph G is given by signature(V', G) = signature(V'', G).

Continuing from the mathematical premise described in the last section, we can say For every pebble problem $P(G,S,T,m)$ such, there exists a pebble path from S to T if and only if signature(S,G) = signature(T,G) or $S \equiv T$.

III. Switching Formation using UPUMP

The major contribution of our paper is to use the agents in the swarm to realize various formations of our interest. This section is reserved for anatomizing the construction and algorithms that help us achieve the same. In the next section, we have explicitly shown the effectiveness of our algorithm by featuring a subset of our results.

A. Formation Generation

We start by constructing our desired formation set as follows F_g = {A,|,M,N,C,λ,ϕ}, where each $x \; \epsilon \; F_g$ is a configuration essential to form the particular formation with $|x|$ = m (number of robots). Each x is formed within a bounding box of an area scaled by a fraction of the configuration space. After generation, this formation is randomly rotated and shifted somewhere in the configuration space.

B. Application of UPUMP

This section will introduce the algorithms we designed to solve the unlabelled motion planning problem using UPUMP. We use disc robots to switch from one formation to another, equivalent to finding a path in an unlabelled motion planning problem. Since this is an unlabelled motion planning problem, we can use the UPUMP algorithm to find a path for a switching formation problem.

Algorithm 1 FORMATION_GENERATION

```
1: $F_g \leftarrow \{A, |, M, N, C, \lambda, \phi\}$
2: $Shape \leftarrow USER\_SELECT(F_g)$
3: $F \leftarrow OBTAIN\_CONFIGURATION(Shape)$
4: $S \leftarrow RANDOM\_CONFIGURATION()$
5: $R \leftarrow RANDOM\_SAMPLE(\pi, -\pi)$
6: $F \leftarrow ROTATE(F, R)$
7: $F \leftarrow SHIFT(F, S)$
8: return $F$
```

a) Generating Pebble Graphs: We start by generating a pumped configuration, which is a configuration PC such that $|PC|$ = $n \geq m$. We then use the EDGE PLANNER mechanism to form the edges between the vertices generated by the pumped configurations. Let there be two vertices v,$v' \; \epsilon$ PC then there exists an edge (v, v') which can also be denoted by $\pi_{v,v'}(\theta)$. We need to ensure $r(\pi_{v,v'}(\theta)) \cap r(u) = \phi$ where $u \; \epsilon$ PC and $u \neq v$,v'.

Algorithm 2 PUMPED_CONFIGURATION(n)

```
1: $V \leftarrow \phi$
2: while $|V| \neq n$ do
3:     $v \leftarrow RANDOM\_SAMPLE()$
4:     $valid \leftarrow TRUE$
5:     for all $v' \; \epsilon \; V, v' \neq v$ do
6:         if $r(v) \cap r(v') \neq \phi$ then
7:             $valid \leftarrow false$
8:         end if
9:     end for
10:    if $valid$ then
11:        $V \leftarrow V \cup \{v\}$
12:    end if
13: end while
14: return $V$
```

Algorithm 3 EDGE_PLANNER(V,v,v')

```
1: $\pi_{v,v'} \leftarrow (v, v')$
2: $V_\pi \leftarrow DISCRETIZE(E)$
3: for all $u \; \epsilon \; V, u \neq v, v'$ do
4:     for all $w \; \epsilon \; V_\pi$ do
5:         if $r(u) \cap r(w) \neq \phi$ then
6:             $\pi_{v,v'} \leftarrow \phi$
7:             return $\pi_{v,v'}$
8:         end if
9:     end for
10: end for
11: return $\pi_{v,v'}$
```

b) Connecting Pebble Graphs: This section briefly discusses the two algorithms, namely CONNECT and CON_GEN.

The first algorithm(CONNECT) generates all possible edges between two pebble graphs. The second algorithm(CON_GEN) removes the interfering edges that can cause a collision between the robots. As given in [1] the problem of finding paths between pumped configurations transforms into the problem of finding an independent set in a graph. We generate the set of pairs $D = \{(v, v') \mid v \in V,\ v' \in V',\ \pi_{v,v'} \neq \phi\}$. We say that two pairs $((v, v'), (u, u')) \in D$ if there exists $\theta \in [0, 1]$ such that robot $r \in$ R placed in $\pi_{v,v'}(\theta)$ collides with another robot r' $\in$ R placed in $\pi_{u,u'}(\theta)$. All the valid edges are then added to the road-map graph H. It is to be noted that every two pairs $((v, v'), (u, u')) \in D$ interfere, such that $v = u$ and $v' = u'$ interfere by definition.

Now, we construct the interference graph I, whose vertices are elements of D. By their interference, we connect vertices of I by an edge if they interfere. Going by the definition, every independent set of size m of the vertices of I represents a collection of m non-colliding single-robot paths. It is to be noted that finding independent sets as the problem in its default is NP-Hard [4]. We have to develop a heuristic by examining graph vertices one by one and the order estimated by the random permutation of the vertices; a new vertex is added to the set only if it is not connected to any other vertices in the set.

Algorithm 4 CONNECT(G(V,E), G'(V',E'), H(V$_H$, E_H))

```
1: $Path\_Edges \leftarrow (v, v')$
2: for all $v \in V, v' \in V'$ do
3:     $Path\_Edges \leftarrow (v, v')$
4: end for
5: $\{(C_1, C_1'), ..., (C_q, C_q')\} \leftarrow CON\_GEN(Path\_Edges)$
6: for i ← 1 to q do
7:     $V_H \leftarrow \{C_i, C_i'\}$
8:     $E_H \leftarrow (C_i, C_i')$
9: end for
10: return $H$
```

c) Query: We use CONNECT algorithm defined in the previous section to check the connection between the start and target graphs through the sampled pebble graphs.

Algorithm 5 Query(H,S,T)

```
1: for all $G$ in $H$ do
2:     $H \leftarrow CONNECT(S, G, H)$
3:     $H \leftarrow CONNECT(G, T, H)$
4: end for
5: if S,T not connected in H then
6:     return FAILURE
7: end if
8: return $RETRIEVE\_PATH(H, S, T)$
```

d) Retrieve Path: We deploy three major algorithms in this section.

The first one is CONNECTIVITY_MATRIX algorithm which returns a matrix M such that $M_{ij} \in \{0, 1\}$. When $M_{ij} = 1$ it indicates that there is a connection between i^{th} graph and j^{th} graph else there is no connection between them. The first index in the M represents the start graph and the last index represents the target graph.

Algorithm 6 RETRIEVE_PATH(H,S,T)

```
1: $\Pi \leftarrow \phi$
2: $M \leftarrow CONNECTIVITY\_MATRIX(H, S, T)$
3: $way \leftarrow WAY\_FINDER(M)$
4: $(C_0, ..., C_l) \leftarrow FIND\_PATH(way)$
5: for i ← 1 to l do
6:     if $C_{i-1} \equiv C_i$ then
7:         $G \leftarrow pebble\ graph\ of\ C_i$
8:         $\pi \leftarrow PEBBLE\_SOLVER(G, C_{i-1}, C_i)$
9:         $\Pi.append(\pi)$
10:    else
11:        $\Pi.append(\pi_{C_{i-1}, C_i})$
12:    end if
13: end for
14: return $\Pi$
```

The second algorithm that we have devised is the WAY FINDER. This algorithm returns an array containing the indices of the connected graph from start to target.

The third algorithm, PATH RETRIEVAL, uses the way obtained from the WAY FINDER to find the edges in the road-map graph H for the respective graphs given by the WAY FINDER.

IV. Experimental Setup

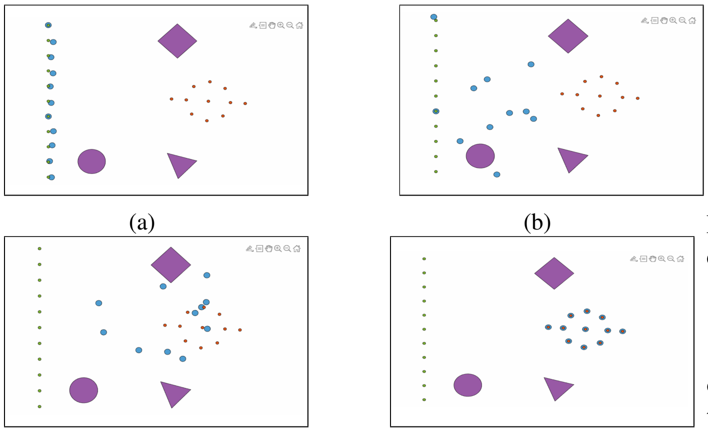
We take m translating disc robots and generate three obstacles in the configuration space. We expect the robots to go from a start configuration to a target configuration by generating a path to avoid the obstacles and inter-agent collision. The disc radius is 1 unit, and the safety margin is 0.1 unit. We anchored a fixed number of robots for each formation [Table I]. The algorithm has two parameters that affect its performance. The first parameter is g which is the number of sampled pebble graphs. The second parameter is q which is the number of connections generated between two sampled pebble graphs. We take different values of g and q to comment on the performance of our algorithm.

TABLE I: Number of robots required for a particular formation

Formation	Number of robots (m)
A	10
M	11
C	7
N	10
ϕ	11
λ	7
$\mid$	7,8,10,11

V. Results and Discussions

The planned motion for the case of robots starting with start configuration - straight, targeting to target configuration - phi is illustrated in Figure 1.



(a) (b)

(c) (d)

Fig. 1: Case scenario: start configuration - straight line, target configuration - phi with g = 2 and q = 500

We run the algorithm for many different case scenarios to check its validity. Figure 2-5 illustrates different case scenarios. It is to be noted that for more number of robots with same q we had to conduct more number of trial runs to find a path. Therefore, as the number of robots increases the probability of finding a path for a fixed q decreases.

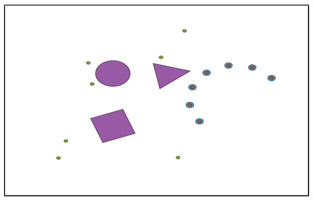


Fig. 2: Case scenario: start configuration-random, target configuration-C with g=3 and q=150

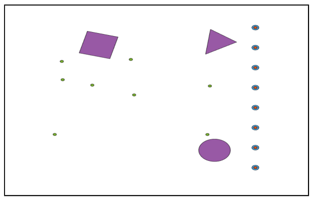


Fig. 3: Case scenario: start configuration-random, target configuration-straight with g=3 and q=250

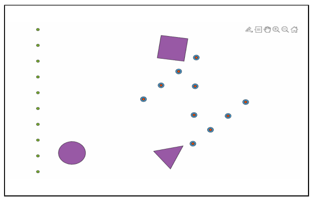


Fig. 4: Case scenario: start configuration-straight, target configuration-N with g=3 and q=800

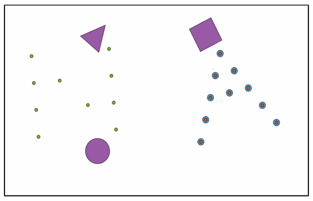


Fig. 5: Case scenario: start configuration-N, target configuration-A with g=2 and q=500

We observe that the time taken by the algorithm depends on the number of robots m, the shape of the formation, and the hyper-parameters g and q. Table II shows the time taken to find a path for the different number of robots but with the same g and q. On the other hand, Table III shows the time taken to find a path for different values of q and m.

TABLE II: Results for selected scenarios at fixed g = 2 and q = 500

Number of robots (m)	Start Formation	Target Formation	Time(in sec)
8	Random	Random	7.665
10	Random	A	18.711
8	Random	|	8.515
11	|	ϕ	29.700
10	N	A	10.416
7	λ	C	7.743
10	|	|	9.056
10	|	A	12.174
7	Random	λ	6.624
11	Random	M	21.473
7	|	C	12.475
10	Random	N	19.996

TABLE III: Results for selected scenarios at g = 3 and different q

Number of robots (m)	Start Formation	Target Formation	g	q	Time(in sec)
10	|	N	3	800	52.996
7	C	λ	3	150	4.789
7	|	λ	3	250	5.013
11	|	M	3	800	60.338
8	Random	|	3	250	8.155
7	Random	C	3	150	2.949

From Table II, we observe that with the increase in the number of robots m, the time to reach the target configuration increases for the same g and q. Similarly, for the same value of m from Table III, we note that with the decrease in the value of q, the possibility of finding a path decreases, but if found, the time taken to reach the target decreases.

The success of the algorithm highly depends on the parameters m, g, and q, which are being tuned manually, so finding a correct combination of them may require some hit and trials. Another limitation of this algorithm is that it only can work with static obstacles.

VI. FURTHER WORK

One of our goals is to develop a method for automatically selecting the algorithm parameters, as currently, we are tuning it manually. Also, we would like to explore the performance of our algorithm on other complex shape robots that can both rotate and translate, as here, we only considered translating disc robots. Developing an online version of our algorithm for static and dynamic obstacles would be another challenging problem. One place where our algorithm could fit in is the problem of object transportation. It requires the robot to be placed at a particular location under the object so that the object can be carried around with utmost stability.

REFERENCES

[1] Solovey K, Halperin D. k-color multi-robot motion planning. The International Journal of Robotics Research. 2014;33(1):82-97.

[2] Kornhauser D, Miller G and Spirakis P (1984) Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: Foundations of Computer Science(FOCS). Los Alamitos, CA: IEEE Computer Society Press, pp. 241–250.

[3] Calinescu G, Dumitrescu A and Pach J (2008) Reconfigurations in graphs and grids .SIAM Journal on Discrete Mathematics22(1): 124–138.

[4] Papadimitriou CH (1994) Computational Complexity. Reading, MA: Addison-Wesley.