Communication-based and Communication-less approaches for Robust Cooperative Planning in Construction with a Team of UAVs

Elena Umili[1], Marco Tognon[3,2], Dario Sanalitro[2], Giuseppe Oriolo[1], Antonio Franchi[4,2]

***Abstract*—In this paper, we analyze the coordination problem of groups of aerial robots for assembly applications. With the enhancement of aerial physical interaction, construction applications are becoming more and more popular. In this domain, the multi-robot solution is very interesting to reduce the execution time. However, new methods to coordinate teams of aerial robots for the construction of complex structures are required. In this work, we propose an assembly planner that considers both assembly and geometric constraints imposed by the particular desired structure and employed robots, respectively. An efficient graph representation of the task dependencies is employed. Based on this framework, we design two assembly planning algorithms that are robust to robot failures. The first is centralized and communication-based. The second is distributed and communication-less. The latter is a solution for scenarios in which the communication network is not reliable. Both methods are validated by numerical simulations based on the assembly scenario of Challenge 2 of the robotic competition MBZIRC2020.**

***Index Terms*—multi-robot systems, task planning, aerial vehicles application**

I. Introduction

In the last decades, Unmanned Aerial Vehicles (UAVs) become extremely popular in a wide range of applications. Recently, the advance of aerial physical interaction led to new applications ranging from contact-based inspection [1] to transportation [2] and assembly. In the fields of manipulation of large objects and assembly of structures, the use of a multi-robot system is becoming very popular [3]. It allows to increase the overall payload and manipulation capabilities, and from the other side, as well as to perform multiple tasks in parallel, minimizing the execution time.

In this work, we focus on the second aspect, i.e., multi-robot assembly tasks. In particular, we consider a group of UAVs that have to cooperatively assemble a structure composed of several elements, sharing the tasks, the resources, and the working environment. The robots must autonomously find the best assembly plan that optimizes the available resources and minimizes the assembly time. We aim at conceiving a task planning algorithm for the group of UAVs under assembly constraints imposed by the specific structure, and geometric constraints due to the sharing of the work-space and resources.

Assembly planning is the process of finding a valid sequence of operations to assemble a product made by different parts. The problem is a NP (Non-deterministic Polynomial time) complete problem [4]. Many algorithms have been proposed to solve assembly planning problems [5]–[8]. However, most of the proposed solutions do not embed in the problem formulation *geometric constraints* imposed by the characteristics of the specific robots. Different robots have different embodiment and feasible movements that imply specific constraints during the interaction with the environment. Additionally to geometric constraints, *assembly constraints* should be considered as well. These are imposed by the particular envisioned construction and ensure at every step of the assembly process its correctness and solidity. Nevertheless, assembly constraints alone do not grant the feasibility of the task considering the available agents.

In most of the previous works, the two types of constraints are treated separately: first, a high-level assembly plan is computed considering assembly constraints only. Subsequently, robot actions and movements are planned considering geometric constraints only [9], [10]. This approach implicitly assumes that for any given assembly plan, it is possible to find a set of feasible robot motions. Though, this is difficult to be met in practice. Some works attempted to combine assembly and geometric planning. In [11], [12] the two aspects are treated separately, but the two planners are implemented in a loop-like framework. The low-level (motion) planner can influence the decisions of the assembly planner at the higher-level.

In this paper, we aim at designing an assembly planner for a group of UAVs assembling a wall-like construction made by blocks. Assembly and geometric constraints are considered at the same planning level. In particular, we shall show that the application of these constraints results in a directed acyclic graph [13], here called "assembly graph", which encodes both assembly and geometric dependencies between tasks.

Aiming at real applications, we assume that robots may fail actions and be temporarily (or permanently) inoperative. This requires the system to be adaptive with respect to the number of available robots [14]. We design the proposed approach such that to be robust to changes in the number of operative robots. This makes the execution time highly variable, stressing the need for fine coordination and re-allocation mechanisms. If needed, the proposed online planner can explore other feasible assembly plans in real-time without a global re-planning,

[1]Sapienza University of Rome, Rome, Italy, umili@diag.uniroma1.it, oriolo@diag.uniroma1.it
[2]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France, dario.sanalitro@laas.fr, antonio.franchi@laas.fr,
[3]Autonomous Systems Lab, Department of Mechanical and Process Engineering, ETH Zurich, 8092 Zürich, Switzerland, mtognon@ethz.ch
[4]Robotics and Mechatronics lab, Faculty of Electrical Engineering, Mathematics & Computer Science, University of Twente, Enschede, The Netherlands a.franchi@utwente.nl

This work was partially funded by: a grant of the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) with the LAAS-CNRS, Toulouse, France; the ANR, Project ANR-17-CE33-0007 MuRoPhen; and the European Union's Horizon 2020 research and innovation programme under grant agreement ID: 871479 AERIAL-CORE

making the approach more flexible to unforeseen events.

Our objective is to optimize the use of available robots minimizing the construction time. With this aim, we implement an *online assembly planner* that, at each iteration, decides the task to be executed according to the current and close future system state (robots and construction).

Similarly to self-triggered control [15], the state prediction is used to check assembly constraints in advance and allocate tasks before they become feasible. Early task allocations yield more efficient cooperation, at the cost of higher uncertainty.

Another important point in multi-agent applications is the possibility to exploit inter-robots communication to enhance the agent's cognitive capabilities. Nevertheless, unstable communication networks could dramatically impact the performances of communication-based strategies for multi-robot coordination. In these conditions, distributed strategies that are robust to communication failures are advisable [16], [17].

To face these two communication scenarios, we design two algorithms. The first assumes an ideal error-free communication network. Under this assumption, tasks are allocated to the robots by a central decision entity which, relying on the communication with the robots, has complete knowledge of the system. The central planner performs assembly planning, task allocation, fault detection, and task re-allocation.

In the second scenario, we assume that robots cannot communicate. Under this assumption, the usual paradigm is *swarm intelligence* [18], i.e., single agents are able to coordinate with each other in a distributed way, relying on local information only. However, distributed paradigms have been used for construction applications in only a few cases [19]–[21]. In the proposed approach, every agent a plans and executes its tasks in an online fashion, without communicating its own plan to the others. Nevertheless, the actions of a have consequences on the construction state. The other agents can sense and estimate the construction state, exploiting it to coordinate their plan with that of a and to reallocate tasks if needed. The designed reallocation mechanism guarantees the resilience of the system. Additionally, we employ an initial task allocation optimized to equally share the tasks between the agents and to minimize the need for coordination during execution.

The two proposed methods are validated by numerical simulations based on the assembly scenario of Challenge 2 of the robotic competition Mohamed Bin Zayed International Robotic Competition 2020 (MBZIRC2020)[1].

The rest of the paper is organized as follows. In section II we formulate the assembly and geometric constraints, describing the representation of dependencies between tasks and the current construction state. In section III we define the single atomic actions composing a task. The communication-based and the communication-less planning algorithms are described in section IV and V, respectively. In section VI simulation results are presented. Section VII concludes the paper.

[1]https://www.mbzirc.com/

II. Assembly planning

General assembly planning is the problem of finding a valid sequence of operations to assemble a product made by different parts. It has been shown that it is a complete NP problem [4]. It requires specific data structures to encode temporal constraints between different parts and to represent all possible object states during the assembly process [22].

In this paper, we focus on block-based constructions (like walls), namely objects that can be assembled adding parts to one sub-assembly. This kind of construction includes a wide variety of objects. Rather than using AND/OR graphs [23] (complete but memory inefficient data structure), we represent the assembly problem with a directed acyclic graph (DAG) of dependencies between the blocks:

$$G = (V, E), \tag{1}$$

where V is the set of vertexes and E is the set of edges. Each vertex of the graph is a block, while edges between blocks represent assembly constraints. In particular, an edge $e = (b_i, b_j)$ belongs to E if block b_i must be placed *after* block b_j, or, in other words, if b_i depends on b_j.

We call $C(t)$ the assembly state at time t, which is the set containing all the blocks already assembled in the construction at time t. Given the assembly graph G and the assembly state $C(t)$, the block $b \in V$ is considered *feasible* if the outgoing neighborhood of b, denoted as $N_G(b) = \{b_j \in V \,|\, e = (b, b_j) \in E\}$, is contained in $C(t)$. We denote with $V_{\text{feasible}}(t)$ the set of feasible blocks at time t, such that:

$$V_{\text{feasible}} = \{b \in V | N_G(b) \subseteq C(t)\}. \tag{2}$$

The problem of computing a sequence of assembly operations respecting all the dependencies in E is equivalent to find a topological ordering of the graph vertexes. This operation can be performed offline in linear time with known algorithms, e.g., Kahn's algorithm [24].

Notice that G may not contain a Hamiltonian path, i.e., a tour in G that visits each vertex exactly once. As a consequence, more than one topological ordering can be found. Furthermore, the execution times of the tasks are non-deterministic and the scheduled sequence of assembly operations may not be respected due to unexpected events. For these reasons, our approach runs online, deciding at every iteration the next task allocation among the feasible ones.

A. Example

Our approach has been evaluated on an L-shaped wall made of bricks of different sizes (see Fig. 1). This model of example is similar to the target construction of Challenge 2 in the robotic competition MBZIRC2020.

The set E contains all brick dependencies which are defined by the following assembly and geometric constraints:

1) Each brick can be placed only if it lies on the ground or entirely on other bricks previously placed on the wall;
2) Each brick can be placed in an empty row if it is the central brick of that row, or if the row is not empty and the brick is adjacent to another one already placed;

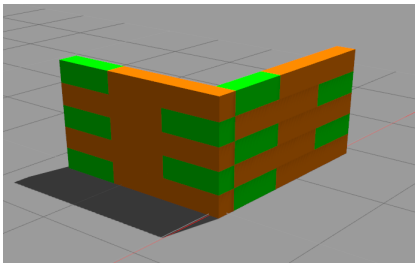

Fig. 1: Example of construction: a 3D wall made of bricks of different size.

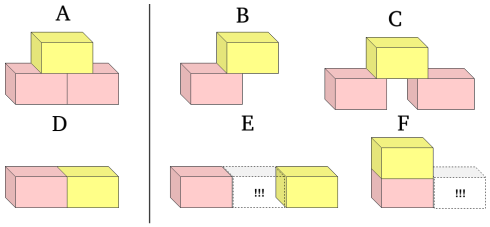

Fig. 2: Assembly and geometric constraints: on the left the accepted configurations, on the right the forbidden ones. Colored in pink the bricks already placed on the wall, in yellow the new one to be placed, in dashed line the brick that is impossible to place in the future due to control inaccuracy or robot embodiment. In particular, case B violates the first constraint only, case C violates constraints 1 and 2, and case E and F violate constraint 2 and 3, respectively, while respecting the others.

3) Each brick can be placed if the difference in height with the bricks already placed at the left and at right is shorter than the length of the UAV gripper.

Constraint 1) ensures wall stability during the construction. Notice that it doesn't depend on the robot performing the task. On the other hand, the following constraints are UAV-specific geometric constraints. Constraint 2) is set to avoid the creation of 'holes' in one line of bricks. Filling the hole requires a perfect brick positioning, which is hard to achieve using current flying robots. Constraint 3) is imposed by the robot embodiment. It is set to ensure that the UAVs are always able to reach the missing brick positions from above without colliding with the bricks already assembled. Figure 2 shows some allowed and forbidden bricks configurations. Finally, Figure 3 graphically shows G resulting from the application of the constraints to the considered wall model.

We remark that these assembly constraints are specific for the construction of the wall here considered using UAVs. In general, assembly constraints must be carefully set by problem experts. They are necessary to avoid unstable configurations of the construction, as well as deadlocks and assembly sequences with low chances of success.

III. Task definition

Each part of the construction is considered as a single task that is individually assignable to one robot using the assembly

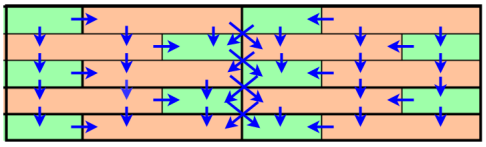

Fig. 3: Assembly graph for the wall. Each block (brick) corresponds to a vertex in V. The blue arrows represent the directed edges in E.

graph defined in the previous section. At every time t, a generic task can be:

- *Closed*: the task is completed;
- *Assigned*: a robot is executing the task;
- *Open*: the task is not completed and not yet assigned.

Notice that $C(t)$ results in the set of closed tasks. Additionally, we denote as $A(t)$ and $O(t)$ the set of assigned and open tasks, respectively. It results that $V = C(t) \dot{\cup} A(t) \dot{\cup} O(t)$.

We call *pick-and-place* the complete macro-action which can be decomposed in a sequence of 4 atomic actions:

1) *fly-to* the position in the map where the brick is stored,
2) *pick* the brick using the gripper,
3) *fly-to* a safe point close to the wall,
4) *place* the brick on the wall.

We assume that a low-level motion planner and position/force controller are provided to allow the robot to execute each action. In this work, we do not focus on these low-level aspects. Nevertheless, the atomic actions 2) and 4) are considered non-deterministic, i.e., we suppose that they may fail with a constant probability p. The two atomic actions must be then repeated until they succeed; eventually, they will succeed with probability 1. Furthermore, we consider that a robot may stop working at any time (e.g., because of malfunctioning sensors or of a crash). The robot will not complete the task which must be re-assigned to another agent.

IV. Communication-based task allocation

Assuming an error-free communication network, the task allocation can be performed using a classical centralized approach. A single decision entity, called *the planner*, is responsible for the task allocation and re-allocation. The planner can run in one of the robots or on a remote computer.

We remark that, at every time instant t, the planner should know the assembly graph, G, and the current assembly state, $C(t)$, $A(t)$ and $O(t)$. Each UAV communicates with the planner in order to request and receive a new task defined by b_{next}. After completing b_{next}, it notifies the planner of the success or failure of the tasks. In the following, we detail the algorithm from both the planner and the robot points of view.

Notice that if the planner cannot directly sense the assembly state, the latter can be inferred collecting the robots' notifications (we assume that false notifications are not possible).

A. Planner Side

Algorithm 1 shows the pseudo-code of the task allocation method. This process is triggered by the request message of

Algorithm 1: online task allocation

Input: new task request from robot r_i
Output: $message$
t_r: current time;
$C(t_r)$: current construction state;
V: goal construction state;
if $C(t_r) = V$ **then**
  message.type $\leftarrow$ go-home;
else
  $S(t_r)$ = feasibleBricks($C(t_r) - A(t_r)$) ;
  if $S(t_r) = \emptyset$ **then**
    $R(t_r, r_i)$ = collectRiskyTasks(r_i);
    if $R(t_r, r_i) = \emptyset$ **then**
      message.type $\leftarrow$ wait;
    else
      $b_{\text{next}} \leftarrow$ bestHeuristicValueBrick($R(t_r, r_i)$);
      message.type $\leftarrow$ task;
      message.task $\leftarrow b_{\text{next}}$;
      message.risky = true;
  else
    $b_{\text{next}} \leftarrow$ bestHeuristicValueBrick($S(t_r)$);
    message.type $\leftarrow$ task;
    message.task $\leftarrow b_{\text{next}}$;
    message.risky = false;
recordAssignmentTime(b_{next});
sendMessage(message)

a new task by a robot r_i at time t_r. At each request, the algorithm performs the following activities:

1) *task classification*: open tasks are classified to compute a set of next allocation candidates: $Candidates \subseteq O(t_r)$;
2) *decision making*: if $Candidates$ is not empty a task $b_{\text{next}} \in Candidates$ is selected;
3) *communication*: the planner allocates b_{next} to r_i sending all the information about the task to r_i. In case b_{next} is not available, the planner sends a special message to r_i.

This sequence of activities is maintained also in the communication-less algorithm.

In the following, the criteria used in the task classification, as well as the policies used in the decision-making activities are explained in detail.

Task classification: The planner partitions the set of open tasks $O(t_r)$ into three categories:

$$O(t_r) = S(t_r) \,\dot\cup\, R(t_r, r_i) \,\dot\cup\, U(t_r). \tag{3}$$

- $S(t_r)$ is the set of candidates for *safe allocation*, i.e., open tasks that are feasible according to the assembly state at time t_r, and in particular to $C(t_r)$:

$$S(t_r) = \{b \in O(t_r) | N_G(b) \subseteq C(t_r)\}. \tag{4}$$

  Allocating a task in $S(t_r)$ guarantees that the brick will be placed respecting the assembly constraints. All the task dependencies are satisfied even before the robot starts the task execution.
  However, the chances of finding a safe allocation decrease with the number n of robots composing the team. This motivates the search also for "risky" allocations.
- $R(t_r, r_i)$ is the set of candidates for a *risky allocation*, i.e., the bricks whose dependencies are not closed at the current time t_r, and that are likely to be closed soon according to the planner estimate based on the available information. In particular, $R(t_r, r_i)$ contains all the open tasks whose dependencies are closed or are assigned to other robots that will complete their task before r_i will attempt to execute the place action. Notice that to compute $R(t_r, r_i)$, the planner must be able to build beliefs about the completion time $ct(r, b)$ of any task b assigned to robot r. These beliefs can be also used to estimate if a robot failed and the corresponding task must be reassigned.
- $U(t_r)$ is the set of *unfeasible* tasks. It contains all the bricks not classified in other categories. They will not be considered for an allocation.

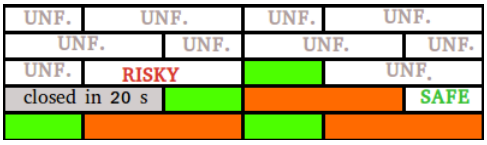

Fig. 4: The current wall is composed of the colored bricks, that are the closed tasks. The grey brick is assigned but not yet closed and it is expected to be closed at time 20 seconds. The white bricks are open and not assigned tasks. In this configuration the planner finds two candidates allocation: one safe and one risky task (the time expected to close this task is higher than 20 seconds).

The next brick to allocate, b_{next}, is chosen among safe or risky candidates:

$$Candidates = S(t_r) \,\dot\cup\, R(t_r, r_i). \tag{5}$$

An example of task classification for the wall model in a particular assembly state is shown in Fig. 4.

Algorithm 2 shows how to infer $R(t_r, r_i)$, i.e., the set of risky task candidates. Let us define $N(t_r)$ as the set of open bricks depending on closed tasks and at least one assigned task at time t_r, then

$$N(t_r) = \{b \in O(t_r) | N_G(b) \subseteq (C(t_r) \cup A(t_r))\} - S(t_r). \tag{6}$$

The bricks in $N(t_r)$ can populate $R(t_r, r_i)$. For each brick b in $N(t_r)$, the time $t_{plc}(r_i, b)$ at which robot r_i will execute the place action is predicted. The assembly state estimated at time $t_{plc}(r_i, b)$ is $C(t_{plc}(r_i, b))$. If the brick b is feasible according to the estimated assembly state, b is a possible risky allocation.

Decision making: it is the process of selecting a brick b_{next} among the candidates sets $S(t_r)$ or $R(t_r, r_i)$ computed in the previous step. In the proposed communication-based algorithm, safe allocations are always preferred to risky ones.

Algorithm 2: collect risky tasks

Input: new task request from robot r_i and $S(t_r) = \emptyset$,
Output: $R(t_r, r_i)$
t_r: current time;
$O(t_r)$: set of open tasks at time t_r;
$A(t_r)$: set of tasks assigned at time t_r;
$S(t_r)$: set of safe allocations at time t_r;
$R(t_r, r_i) = \emptyset$;
$N(t_r)$ = feasibleBricks$(C(t_r) \cup A(t_r)) - S(t_r)$;
for *brick* b_n *in* $N(t_r)$ **do**
  $t_{plc}(r_i, b_n)$ = predictPlaceTime(r_i, b_n);
  $C(t_{plc}(r_i, b_n))$ = $C(t)$;
  for *brick* b *in* $A(t_r)$ **do**
    if $b.tplace < t_{plc}(r_i, b_n)$ **then**
      $C(t_{plc}(r_i, b_n)) = C(t_{plc}(r_i, b_n)) \cup b$;
  feasible = checkBrickFeasibility$(C(t_{plc}(r_i, b_n)), b_n)$;
  if *feasible* **then**
    $R(t_r, r_i) \leftarrow R(t_r, r_i) \cup b_n$;
return $R(t_r, r_i)$;

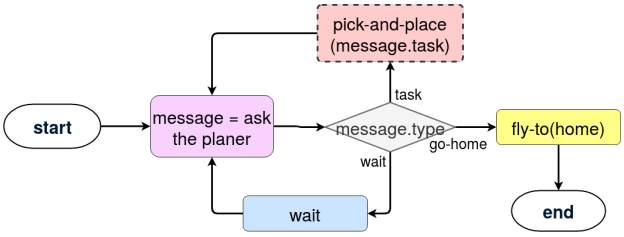

Fig. 5: Robot life cycle in the communication-based scenario. Operations in solid line boxes are atomic actions, while the ones in dashed line boxes are complex macro-action.

The planner tries to allocate a safe task. Only if $S(t)$ is empty (no possible safe allocations), $R(t)$ is computed and one task in it is chosen. This kind of **cautious policy** can represent a non-optimal choice in the general case. We shall show in the next section this is true for the communication-less case.

Finally, the algorithm chooses one brick in the selected class using a heuristic defined by the user. The heuristic choice will prefer some particular assembly plans over other possible ones, influencing the final construction time. In Sec. VI some heuristics for a specific case are compared and discussed. In particular, the tested heuristics were designed to always have non-empty candidates set. In this way, the planner is always able to assign a task to each request.

B. Robot Side

Next, we describe the algorithms from the robot side. Each robot cannot decide its task, it only executes what the planner commands. Figure 5 illustrates their simple behavior. The generic robot sends a message to request a new task and warn the planner in case of changes of the construction state. The planner replies using one of the following messages:

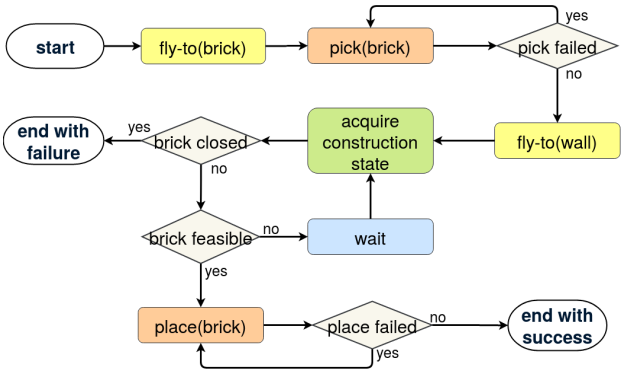

Fig. 6: Pick-and-place: task execution control flow

- **Task message**: it contains the description of the new task the robot must execute;
- **Wait message**: the robot must wait because the planner did not find a task allocation candidate. The robot waits for a fixed amount of time and repeats the request;
- **Go-home message**: the construction is complete achieving the goal, and the robot can terminate the mission.

Robots r_i continuously demand tasks until the go-home message is received. This mechanism ensures reliability even in scenarios where no more than one robot can accomplish the assignment due to possible robot failures. Figure 6 shows the pick-and-place control flow.

After the start signal, the robot flies over the bricks stock (*fly-to* action) and try to pick it (*pick* action). Action *fly-to* is always considered executed with success. On the other hand, the *pick* and *place* actions are repeated in a loop until their fulfillment. Once the pick is executed and the robot is over the wall ready for the place, it has to first check the task status (if closed or not) and feasibility (if assembly dependencies are satisfied). In the best-case scenario, the robot will place the brick. However, it could happen that its given task has been already closed by another robot or that the task was a risky one and all the dependencies are not yet satisfied. The first case happens when a pick-and-place operation takes more time than expected. Because of the wrong estimation of the pick-and-place time, the planner re-assigns the task considering that the corresponding robot failed. In the second case, when a robot receives a risky task, it might be not feasible at the moment of the place. The robot must wait until all the dependencies are fulfilled and the task becomes feasible.

Notice that the over and underestimation of the execution time causes the lack of candidates and inopportune re-allocations, respectively. This implies the waste of resources and delays in total execution time. It is then advisable to embed an online prediction mechanism for the task execution time.

V. Communication-less Task Allocation

Here we present a communication-less algorithm for task allocation and re-allocation. The robots operate in the same world domain described so far, but without communications among them nor with a remote decision entity. Therefore, it is

not possible to allocate tasks one by one during the execution as in the previous case. Conversely, tasks have to be assigned to the robots offline, before the execution starts. Nevertheless, we do not force robots following the precomputed plan. We rather designed an on-line re-allocation strategy that, during the construction process, can change the order of tasks and execute new ones not originally included in their plan to face unforeseen events. This *stigmergy* mechanism is suitable for treating possible failures since each agent is uniquely responsible for a subset of tasks. At the same time, the proposed strategy can handle unexpected events. We highlight that without the presence of a communication network, the use of some onboard sensors is necessary to estimate the construction state and react accordingly.

A. Initial task allocation

Defining n the number of agents and considering $0 < i \leq n$, we partition the set of tasks V into n subsets, A_i, containing the tasks initially assigned to robot i such that:

$$V = \dot{\bigcup}_{i=1}^{n} A_i. \tag{7}$$

Tasks must be assigned in order to minimize possible dependencies between tasks belonging to different robots. This reduces possible waiting periods necessary for a task to become feasible. Our offline task allocation approach aims at minimizing and maximizing the *inter-* and *intra-dependencies*, respectively. The firsts are dependencies connecting tasks assigned to different agents, while the seconds are dependencies between tasks assigned to the same agent.

Let us consider u and v vertices of a graph G. They are *connected* if there exists a path of any length connecting u to v. Let us define with $conn_G(u)$ the set of vertices connected with u through a directed path in G, and with $conn_G(A)$, where $A \subseteq V$, the set of vertices connected in G with at least one vertex in A. In other words:

$$conn_G(u) = \{v \in V \mid \exists \text{ a path from u to v}\} \tag{8}$$

$$conn_G(A) = \bigcup_{u \in A} conn_G(u). \tag{9}$$

Let us define a *cluster* as a set of bricks densely connected with each other and poorly connected with the rest of the bricks. Each robot will be assigned a cluster A_i. Then, the aim is to minimize the following function:

$$\sum_{i=1}^{n} |conn_G(A_i) \cap (V - A_i)|, \tag{10}$$

where $|\star|$ is the cardinality of any generic set $\star$. To evaluate a task allocation, we need to consider the number of:

- *direct inter-dependencies*: bricks assigned to different robots connected by a *path* with length 1 (called *arc*);
- *indirect inter-dependencies*: bricks assigned to different robots connected by a *path* with a length greater than 1.

The reason is to account for the *transitivity* of assembly constraints. In fact, there are *temporal* constraints whereby if brick A has to be placed after brick B and brick B after brick C, it follows that brick A has to be assembled after brick C.

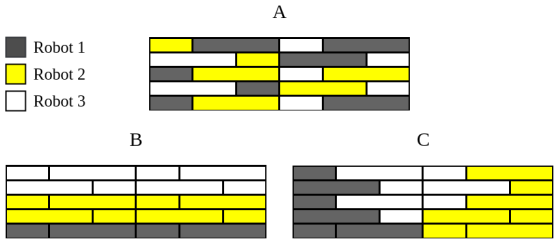

Fig. 7: three examples of task allocation for $n = 3$ robots. The brick colour indicates the robot the brick is assigned to. Case A is an example of sparse allocation, it is not a good allocation because it has a big number of direct inter-dependencies. Case B has a small number of direct inter-dependencies but a lot of indirect intra-dependencies. Case C has few direct and indirect inter-dependencies, and it's therefore the best allocation among the three.

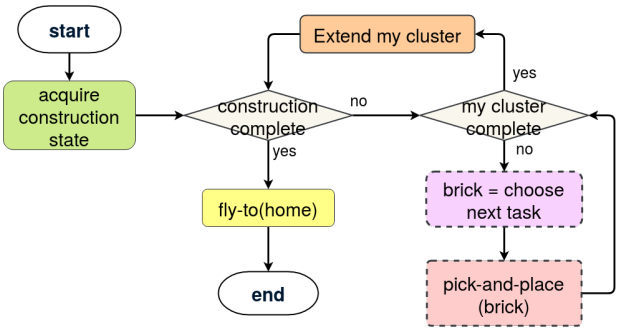

Fig. 8: Robot algorithm in the communication-less scenario.

Figure 7 shows different allocations for a wall with $n = 3$.

- *Case A* is an example of *sparse allocation* where there are many *direct* inter-dependencies.
- *Cases B* (like Case C) shows sets composed by adjacent bricks, therefore direct inter-dependencies are minimized. However, Case B shows many indirect inter-dependencies. Indeed, $conn_G(A2) \supseteq A_1$ and $conn_G(A_3) \supseteq A_2 \cup A_1$. In this example, robots' operations can run in parallel only in a minimal part. In fact, robot 1 can start immediately, while robot 2 is obliged to wait that robot 1 has almost completed its tasks before starting, and robot 3 has to wait until robot 2 has almost completed its tasks before starting.
- *Case C* has a small number of direct inter-dependencies and not many indirect inter-dependencies representing the best allocation shown until now. However, even in case C, inter-dependencies cannot be completely avoided because the graph of the considered wall is connected (there are not disconnected parts).

B. Robot life cycle

Hereafter, we present the algorithm concerning each aerial vehicle (see Fig. 8). At the beginning of each new task, the

Algorithm 3: choose next task

Output: b_{next} – next task to execute
$S_i(t)$ = feasibleBricks($C(t)$) $\cap A_i$;
$\hat{C}_i(t) \leftarrow C(t)$;
taskList = topologicalOrder(V);
for *b in taskList* **do**
  if *b in* $O(t) \cap A_j$, *j≠i* **then**
    if *checkBrickFeasibility*($\hat{C}_i(t)$, *b*) **then**
      $\hat{C}_i(t) \leftarrow \hat{C}_i(t) \cup b$;
$R_i(t)$ = feasibleBricks($\hat{C}_i(t)$) $- S_i(t)$;
bestSafeBrick = bestHeuristicValueBrick($S_i(t)$);
bestRiskyBrick = bestHeuristicValueBrick($R_i(t)$);
b_{next} = decisionMaking(bestSafeBrick, bestRiskyBrick);
if $b_{next} \in S_i(t)$ **then**
  return b_{next};
tempt $\leftarrow 0$;
feasible $\leftarrow$ false;
while *not feasible and tempt*$<$ *threshold* **do**
  $C(t)$ = observeConstructionState();
  feasible = checkBrickFeasibility($C(t)$,b_{next});
  if *feasible* **then**
    return b_{next};
  else
    tempt $\leftarrow$ *tempt* $+ 1$;
if *bestSafeBrick not null* **then**
  return bestSafeBrick;
return shortestPathReallocation($C(t)$, b_{next})

robot detects the construction state by means of its sensors. This operation is necessary to decide future actions. If the construction is complete the robot can terminate its mission. Otherwise, it has to first close the tasks composing its cluster and then, if the construction is still incomplete, it has to extend its cluster boundaries taking care of the other tasks not yet closed by other robots.

The proposed algorithm for selecting b_{next} is illustrated in Algorithm 3. After the estimation of the future construction state, the proposed approach performs a task classification which is followed by a decision making phase and an emergent task reallocation.

1) Future construction state estimation: The first step consists in estimating the construction state in the near future. In this scenario, each robot r_i decides its task relying on the perceived state $C(t)$, without any knowledge regarding other robots, e.g., their state nor which task they are executing. On the contrary, robot r_i can sense $C(t)$ and make a hypothesis on the future construction state, called $\hat{C}_i(t)$. In particular, r_i will add to the hypothetical construction all the bricks pre-assigned to other agents that can be assembled to the current construction. If we define the set of the open bricks $O(t)$ as the bricks not yet placed in the construction, we have that:

$$O(t) = V - C(t). \tag{11}$$

$\hat{C}_i(t)$ contains both closed bricks and all the open bricks assigned to the other robots that do not have dependencies with open bricks assigned to r_i:

$$\hat{C}_i(t) = C(t) \cup \left(A_{j\neq i} \cap \left(V - conn_G(O_i(t))\right)\right), \tag{12}$$

where $A_{j\neq i} = \bigcup_{j=1,j\neq i}^{n} A_j$, and $O_i(t) = A_i \cap O(t)$. Figure 9 shows an example of how each robot computes $\hat{C}_i(t)$.

2) Task classification: At this stage, r_i tags all the open tasks, in its subset A_i, in *safe*, *risky* and *unfeasible*, according to the current construction state $C(t)$ and the future construction state $\hat{C}_i(t)$. The classification follows the same principle described for the communication-based planner algorithm:

$$A_i \cap O(t) = S_i(t) \,\dot\cup\, R_i(t) \,\dot\cup\, U_i(t), \tag{13}$$

where

- $S_i(t)$ contains *safe* tasks depending on closed tasks only
$$S_i(t) = \{b \in (A_i \cap O(t)) | N_G(b) \subseteq C(t)\}. \tag{14}$$
- $R_i(t)$ is the set of *risky* tasks, namely the ones depending on at least one brick contained in $\hat{C}_i(t)$ and not in $C(t)$
$$R_i(t) = \{b \in (A_i \cap O(t) - S_i(t)) | N_G(b) \subseteq \hat{C}_i(t)\}. \tag{15}$$
- $U_i(t)$ is the set of *unfeasible* tasks, containing the rest of open tasks of r_i
$$U_i(t) = A_i \cap O(t) - S_i(t) - R_i(t). \tag{16}$$

In Fig. 9 an example of future wall estimation and task classification for a certain wall state and a particular initial task allocation is shown.

3) Decision making: Once bricks are classified, the robot has to choose one target brick in the set of risky or safe candidates.

$$candidates_i(t) = S_i(t) \,\dot\cup\, R_i(t) \tag{17}$$

If in the communication-based approach the best policy was to choose safe options, in this case, the cautious policy is not the most effective. Indeed, the cautious policy pushes the agents to place bricks within the boundaries of their sub-construction as long as they can and to place the bricks on the borders only when no other choices are possible. In general, this makes tasks that depend on other robots to be postponed, resulting in a non-optimal cooperative behavior. In case G cannot be divided into n disconnected parts, bricks located on the border of a sub-construction A_i are connected with at least another subset A_j. This means that r_i might wait for a brick assigned to robot r_j or vice versa. Placing bricks at the border can potentially be the riskiest task. But, at the same time, they are the most necessary to let other agents continue their tasks.

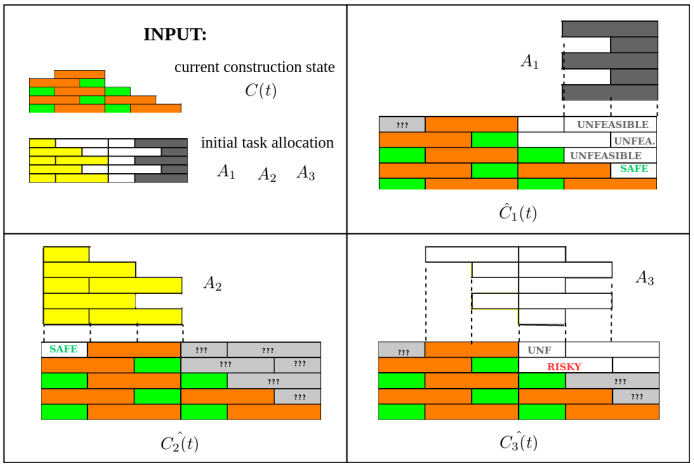

Fig. 9: Wall estimation and task classification in the communication-less scenario. The current wall state $C(t)$ is composed by the coloured green and orange bricks only, the wall estimated by the robots are $\hat{C}_1(t)$, $\hat{C}_2(t)$ and $\hat{C}_3(t)$. They contain the closed tasks (green and orange bricks) plus the bricks that others can place in the best hypothesis (grey bricks). According to the estimation, each robot classifies its open tasks in safe, risky or unfeasible.

4) Emergent task reallocation: After the decision making process, if the task selected is safe the robot can start the execution, otherwise, if it is risky, the robot needs to wait until the task becomes feasible or to take actions to make the task feasible. In particular, when robot r_i selects a risky option b_r, it remains in an observation state for a while. During this time it monitors the construction state and, as soon as b_r becomes feasible, it starts the execution.

In case b_r depends on a task assigned to a robot that failed, r_i will indefinitely stay in the observation state. For this reason, a **threshold-based reallocation** mechanism is implemented. If b_r is still unfeasible after a certain maximum time, r_i decides to allocate to itself one of the bricks needed to make b_r feasible. Indirectly, it assumes that the robot originally responsible for these bricks failed. We remark that the reallocated brick, b_{reall}, is a feasible brick among the direct or indirect dependencies of b_r, accordingly to $C(t)$:

$$b_{reall} \in \{b \in conn_G(b_r) \cap O(t) | N_G(b) \subseteq C(t)\}. \quad (18)$$

In particular, b_{reall} is chosen among the brick connected with b_r by the shortest path.

VI. Simulation Results

The two task allocation algorithms were tested in simulation, using a framework based on the Robotic Operative System (ROS) and the Gazebo robotic simulator. The simulated environment resembles the scenario of Challenge 2 of the MBZIRC competition (see Fig. 10). The target construction used for testing is the wall depicted in Fig. 1. The probability of atomic actions to fail was set to 0.25. We evaluate the task

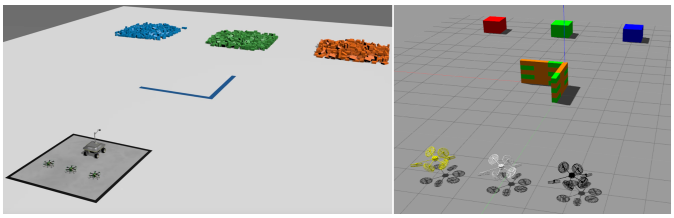

Fig. 10: On the left the MBZIRC –challenge 2– environment. On the right the simulated environment used to test our task allocation algorithms

allocation methods according to the *global execution time* and the *average active ratio*, defined by ar as:

$$ar = \frac{1}{n} \sum_{i=0}^{n} \frac{a_i}{w_i}, \quad (19)$$

where a_i and w_i are the time robot i spent in an *active* and *waiting* states, respectively. For the decentralized communication-less algorithm we recorded also the total number of *dropped bricks*, caused by a small re-allocation threshold, as well as the total number of *emergent reallocations*.

A. Communication-based task allocation results

The centralized communication-based task allocation was first tested with different heuristics for selecting the target task among several candidates without exploiting the possible risky allocations. The best heuristic was then used with also risky allocations to achieve the final result. The heuristics used for the wall model are hereafter explained:

- h0) *Random choice*: first we tested the algorithm without using any heuristics, the decision-making module simply chooses one random task among the class of candidates ($S(t)$ or $R(t)$). The results are used as a baseline;
- h1) *Distance from the base middle point*: here the candidate bricks are evaluated according to their position in the wall. In particular, the selected brick is the closest to the middle point of the wall. Given the assembly constraints explained in section II, constructing the wall starting from the center increases the number of feasible tasks in the next steps;
- h2) *Distance from the central vertical line*: the selected brick is the closest to the vertical line located in the center of the wall;
- h3) *Position in the wall and brick length*: this heuristic combines the position in the wall and the brick dimension. Longer bricks closer to the central vertical line are selected first. If the bricks are considerably different in dimensions, like in our test example, placing first the longest ones increases the number of feasible options in the next steps.

As it is clear from Tab. I, h3) is the best heuristics and the performance further improves with the use of the risky allocation strategy. Nevertheless, we remark that heuristics

Heuristic	Risky allocations	Execution time	Active ratio
h0	No	267 s	60%
h1	No	263 s	62%
h2	No	262 s	62%
h3	No	249 s	66%
h3	**Yes**	**169 s**	**88%**

TABLE I: Results of the centralized communication-base task allocation method.

are strongly linked to this particular scenario. Other heuristics could be investigated in different scenarios.

B. Communication-less task allocation results

The decentralized communication-less task allocation algorithm was tested in the same scenario, using the offline task allocation (Case C) shown in Fig. 7. For the decision making of Algorithm 3, a random policy with uniform probability has been employed. In Fig. 11 the results obtained varying the re-allocation threshold between 1 and 10 seconds are shown. For each value, the statistical results over 100 simulations are represented through a box and whisker plot. The red line indicates the median value, while the box extends from the lower to upper quartile values of the data. The whiskers show the range of the non-outliers data and flier are outliers data.

One can notice that the execution time grows with bigger threshold values, while the active ratio decreases otherwise. On the other hand, the total number of reallocation is higher with low threshold values. Notice that the number of dropped bricks is minimum with a re-allocation threshold between 4 and 5. These values provide the best performance, namely the maximum active ratio, the minimum execution time, and the minimum number of re-allocations.

VII. Conclusion and Future Work

In conclusion, this paper proposes two task allocation algorithms for the construction of structures made by blocks, using a group of cooperative UAVs. Both are based on directed acyclic graph, here simply called assembly graph, to encode the temporal constraints between the tasks. This is also used to plan and execute feasible sequences of assembly operations.

The first method is a centralized algorithm based on *ideal communication* conditions. In this scenario, tasks are allocated to the robots in real-time through messages exchanged with the planner. The latter implements the decisional part of the algorithm, while the robots actuate performs the tasks. To improve the performances, the planner exploits an estimation of the future construction state to allocate tasks that are not yet feasible but that will be in the near future. The planner is also responsible for detecting robot failures and for the reallocation of tasks. This algorithm achieves the best results in terms of global execution time and average active ratio. However, it requires robots-planner communication which results in a single point of failure for the method.

The second is a distributed algorithm which uses *none communication* between agents. A preliminary *ideal plan* is

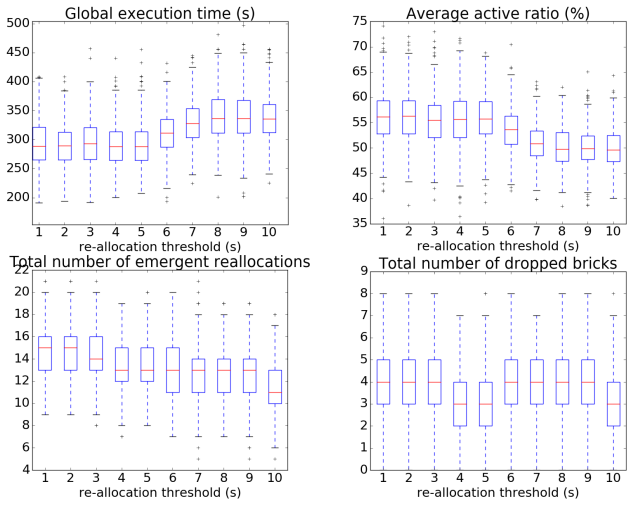

Fig. 11: Box and whisker plots showing results obtained with the decentralized communication-less task allocation, using different fault threshold values. Each box shows the statistical distribution of the global execution time, average active ration, emergent reallocations, and dropped bricks, respectively, over 100 simulations.

computed offline to minimize the direct and indirect dependencies between tasks allocated to different agents. However, if inter-dependencies cannot be avoided, the robots must coordinate each other exploiting only the observation of the environment. Each robot autonomously decides the order of tasks execution and when to re-allocate tasks in order to cover possible fails of another robots. This approach is more robust to communication-failures, but at the expense of lower performance. Nevertheless, results can be enhanced tuning the system parameters, e.g., the reallocation threshold.

A possible future work is to suitably blend the communication-based and less approaches in order to have a strategy that exploits communication as much as possible, but, at the same time, is robust to the failure of the communication network. Other improvements of the heuristics could be obtained by the use of learning-based methods, such as genetic algorithms or reinforcement learning.

References

[1] M. Tognon, H. A. Tello Chávez, E. Gasparin, Q. Sablé, D. Bicego, A. Mallet, M. Lany, G. Santi, B. Revaz, J. Cortés, and A. Franchi, "A truly redundant aerial manipulator system with application to push-and-slide inspection in industrial plants," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1846–1851, 2019.

[2] M. Bernard, K. Kondak, I. Maza, and A. Ollero, "Autonomous transportation and deployment with aerial robots for search and rescue missions," *Journal of Field Robotics*, vol. 28, no. 6, pp. 914–931, 2011.

[3] D. Sanalitro, H. J. Savino, M. Tognon, J. Cortés, and A. Franchi, "Full-pose manipulation control of a cable-suspended load with multiple UAVs under uncertainties," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2185–2191, 2020.

[4] L. Kavraki, J. Latombe, and R. H. Wilson, "On the complexity of assembly partitioning," *Information Processing Letters*, vol. 48, no. 5, pp. 229–235, 1993.

[5] R. H. Wilson and J. Latombe, "Geometric reasoning about mechanical assembly," *Artificial Intelligence*, vol. 71, no. 2, pp. 371–396, 1994.

[6] D. Halperin, J. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, no. 3-4, pp. 577–601, 2000.
[7] J. Cortés, L. Jaillet, and T. Siméon, "Disassembly path planning for complex articulated objects," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 475–481, 2008.
[8] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus, "Ikeabot: An autonomous multi-robot coordinated furniture assembly system," in *2013 IEEE Int. Conf. on Robotics and Automation*, Karlsruhe, Germany, Oct. 2013, pp. 855–862.
[9] M. Dogar, A. Spielberg, S. Baker, and D. Rus, "Multi-robot grasp planning for sequential assembly operations," *Autonomous Robots*, vol. 43, no. 3, pp. 649–664, 2019.
[10] H. Mosemann and F. M. Wahl, "Automatic decomposition of planned assembly sequences into skill primitives," *IEEE transactions on Robotics and Automation*, vol. 17, no. 5, pp. 709–718, 2001.
[11] R. Lallement, J. Cortés, M. Gharbi, A. Boeuf, R. Alami, C. J. Fernandez-Agüera, and I. Maza, "Combining assembly planning and geometric task planning," in *Aerial Robotic Manipulation*. Springer, 2019, pp. 299–316.
[12] J. Munoz-Morera, F. Alarcon, I. Maza, and A. Ollero, "Combining a hierarchical task network planner with a constraint satisfaction solver for assembly operations involving routing problems in a multi-robot context," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, 2018.
[13] K. Thulasiraman and M. N. S. Swamy, *Graphs: theory and algorithms*. John Wiley & Sons, 2011.
[14] L. E. Parker, "Alliance: An architecture for fault tolerant multirobot cooperation," *IEEE transactions on robotics and automation*, vol. 14, no. 2, pp. 220–240, 1998.
[15] C. Nowzari, "Self-triggered optimal servicing in dynamic environments with acyclic structure," *IEEE Transactions on Automatic Control*, vol. 58, pp. 1236–1249, 05 2013.
[16] L. E. Parker, "Distributed intelligence: Overview of the field and its application in multi-robot systems." in *AAAI Fall Symposium: Regarding the Intelligence in Distributed Intelligent Systems*, 2007, pp. 1–6.
[17] M. Tognon, C. Gabellieri, L. Pallottino, and A. Franchi, "Aerial co-manipulation with cables: The role of internal force for equilibria, stability, and passivity," *IEEE Robotics and Automation Letters, Special Issue on Aerial Manipulation*, vol. 3, no. 3, pp. 2577 – 2583, 2018.
[18] J. Kennedy, "Swarm intelligence," in *Handbook of nature-inspired and innovative computing*. Springer, 2006, pp. 187–219.
[19] S. Yun, M. Schwager, and D. Rus, "Coordinating construction of truss structures using distributed equal-mass partitioning," in *Robotics Research*. Springer, 2011, pp. 607–623.
[20] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014.
[21] ——, "Distributed multi-robot algorithms for the termes 3d collective construction system," in *Proceedings of Robotics: Science and Systems*. Institute of Electrical and Electronics Engineers, 2011.
[22] P. Jiménez, "Survey on assembly sequencing: a combinatorial and geometrical perspective," *Journal of Intelligent Manufacturing*, vol. 24, no. 2, pp. 235–250, 2013.
[23] R. A. Knepper, D. Ahuja, G. Lalonde, and D. Rus, "Distributed assembly with and/or graphs," in *Workshop on AI Robotics at the Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014.
[24] A. B. Kahn, "Topological sorting of large networks," *Communications of the ACM*, vol. 5, no. 11, pp. 558–562, 1962.