Multi–object handling for robotic manufacturing

Mirko Ferrati, Simone Nardi, Alessandro Settimi, Hamal Marino, Lucia Pallottino

***Abstract*—The purpose of this work is to move a step toward the automation of industrial plants through full exploitation of autonomous robots. A planning algorithm is proposed to move different objects in desired configurations with heterogeneous robots such as manipulators, mobile robots and conveyor belts. The proposed approach allows different objects to be handled by different robots simultaneously in an efficient way and avoiding collisions with the environment and self–collisions between robots. In particular, the integrated system will be capable of planning paths for a set of objects from various starting points in the environment (e.g. shelves) to their respective final destinations. The proposed approach unifies the active (e.g., grasping by a hand) and passive (e.g., holding by a table) steps involved in moving the objects in the environment by treating them as end–effectors with constraints and capabilities. Time varying graphs will be introduced to model the problem for simultaneous handling of objects by different end–effectors. Optimal exploration of such graphs will be used to determine paths for each object with time constraints. Results will be validated through simulations.**

***Index Terms*—Robotics, Intelligent Robots, Factory Automation, Manufacturing Automation, Motion Control**

I. Introduction

The upward trend of industrial processes automation and the advent of smart factories in the context of the fourth industrial revolution rely on the complete and easy integration of autonomous robots in complex environments.

Such robots may differ in shape and capabilities, but should work together in a highly dynamic assembly chain, where the chain itself requires a continuous reconfiguration based on different typologies of processed item. Moreover, in applications where objects cannot have a standardized shape, or in complex end–of–lines setups that require a particular orientation of the items, a completely integrated and autonomous solution is missing. Two examples of such dynamic environments are the online shopping warehouses of Amazon [1] and Ocado [2] where products are moved by autonomous mobile shelves or by a complex system of conveyor belts. However, in both cases, the final manipulation of objects is still handled by humans at the end of the line. Both companies are researching a way to completely automate the process, see for example the Amazon Picking Challenge [3]. Our aim is to propose a unified framework where a set of heterogeneous manipulators and mobile robots are managed autonomously in order to pick, move, grasp and manipulate different products. Moreover, the framework should take into account objects of arbitrary shape and manipulators of different grasping capabilities. Indeed, in this case, new robot typology can be added to the system with small effort (e.g. a change of a configuration database).

All authors are with Centro di Ricerca "E. Piaggio", University of Pisa, 56122 Pisa, Italy. A. Settimi is also with the Dept. of Advanced Robotics, Istituto Italiano di Tecnologia, via Morego, 30, 16163 Genova. L. Pallottino is also with the Dept. of Information Engineering, University of Pisa, 56122 Pisa, Italy. (mirko.ferrati@gmail.com, alessandro.settimi@for.unipi.it, hamal.marino@centropiaggio.unipi.it, s.nardi@mail.dm.unipi.it, lucia.pallottino@unipi.it)

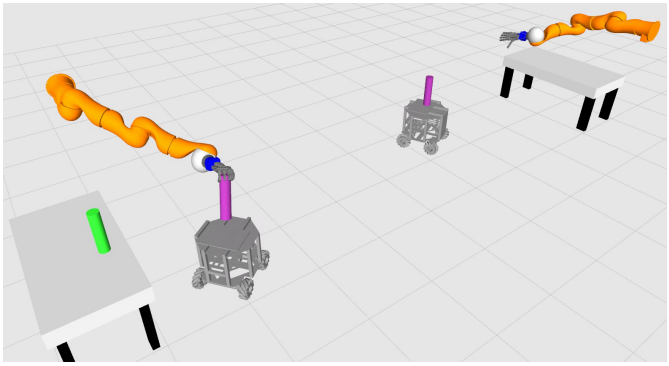

Fig. 1. An example industrial–like scenario. Multiple object have to be carried across a factory, robotic arms are used to arrange the objects on available mobile robots and on workbenches.

The integration of the proposed framework within a smart-factory could be done through a management system, integrated with the online shopping software, that schedules requests for the warehouses robots to handle the queue of customer orders. An example of this management system can be found in [4].

Recent research on smart-factories automated planners focuses on solutions to automatize specific industrial scenarios. Such works do not ensure adaptability to an already existing structured environment: on the contrary, they are usually designed ad–hoc on the final application scenario. For example, in [5] and [6] the proposed approaches rely on a substantial plant reconfiguration or design.

A complete framework for task assignment, planning and coordination of multiple mobile robots is described in [7], where the capability of integrating the system in existing environments is obtained. However, this work does not consider the integration with assembly lines or more generic manipulation of objects (e.g. placing items into shipping boxes), but rather considers only the planning of movement of goods. A mixed cooperation between mobile robots and robotic arms in manufacturing scenarios with multiple objects is shown in [8] and [9], even if limited to a single manufacturing cell. We believe that it is fundamental to realize solutions that are adaptable to different industrial scenarios. Indeed, this would allow to deploy standard robots in existing structured environments without modifying them. At the same time, we aim for an integrated mobile/manipulator system able to handle multiple objects simultaneously and a dynamic assembly chain, as the previous cited Amazon and Ocado cases.

For this purpose, we developed an integrated system capable of planning the paths of a sequence of objects from various starting points in the environment (e.g. shelves) to their respective final destinations. Multiple robotics hardware may be required to perform the objects motion or to manipulate objects such as manipulators, hands and mobile robots. For example, in Fig. 1 an industrial–like scenario is represented

where multiple objects are grasped by a robotic arm, one after another. The objects are then brought by one of the available mobile robots to the other robotic arm to be placed on another workbench. This can be associated to an intermediate phase of a manufacturing process. Our approach unifies all the active and passive steps involved in the objects motion treating them as end–effectors. The object path is planned considering each end–effector constraints and capabilities. Moreover, the returned plan will avoid collisions with the environment and self–collisions between robots. The proposed contribution can be summarized in two main results. First, we propose an extension of our previous results from [10], in which single object moving has been investigated and tested, both in real experiments and simulations scenarios such as an assembly chain with five Kuka arms. In the aforementioned simulations and experiments, different kinds of object handling have been tested, such as: pick-and-place, handoff between two robotic arms, use of a support surface to re-orient the object. Moreover different objects have been used, including both simple geometries such as a ball and a cylinder, and complex ones such as a colander and a pitcher.

The extension presented here consists in a modelization of floating base robots, such as mobile manipulators, that complies with the end–effector concept in [10]. Such modelization allows the inclusion of mobile manipulators in planning paths for the single object case. Second, we introduce a new planner capable of managing a queue of multiple different objects along the assembly chain at the same time by using a time varying graph as defined in [11]. Such graphs are used in traffic flow optimization algorithms to compute the shortest time path of multiple items that are simultaneously crossing a network: indeed, our modelization generates a network from the set of end–effectors and our planner optimizes the flow of objects using shortest time search. Finally the proposed approach is validated through a set of simulations with different numbers of end–effectors and objects.

II. Extension to System Modeling

Similarly to [10], in this work each entity that is able to act/apply a grasp/support on an object is considered an *end–effector*, see Fig. 2 for different examples of end-effectors. On the other hand, the relative configuration between the end–effector and a specific grasped object is a *grasp*. Finally, *workspaces* are sub-regions of the environment where an object can be grasped by two or more end–effectors. For example in Fig. 3a), the classical workspace is reported in red, yellow and green for the right, both right and left, and left arms respectively. With the proposed approach the workspace is represented by the three regions on the table reported in Fig. 3b). For example in $w2$ an object can be grasped by the left hand if it lays on the table (i.e., grasped by the end-effector associated to the table) or if it is grasped by the right hand.

It is possible to distinguish between *movable* end–effectors (such as serial manipulator end–effector) and *non–movable* ones (such as a surface in the environment providing stable object support). The difference between such end–effectors is that the absolute position of an object handled by a movable end–effectors can be changed without changing grasp.

It is worth noting that from a planning perspective, it is not necessary to distinguish between movable and non–movable end–effectors, because they can both interact with an object with their own set of grasps and they can exchange objects through a set of interaction transitions. For more details and examples on the concepts introduced above please refer to [10].

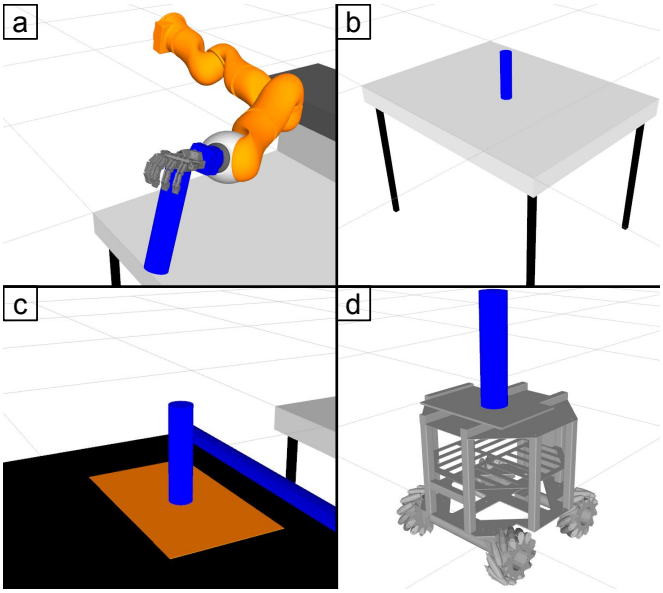

Fig. 2. In these figures different types of end–effectors and grasps are reported, the blue cylinder is the object considered for grasping. (a) Kuka LWR robotic arm, movable end–effector. (b) Table, non–movable end–effector. (c) Conveyor belt, movable end–effector. (d) Mobile robot, movable end–effector.

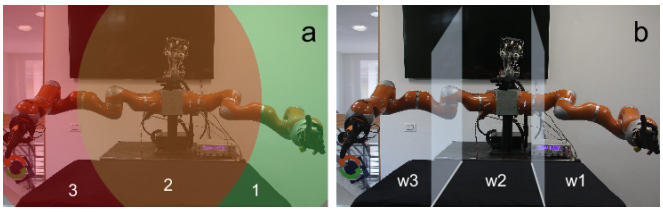

Fig. 3. (a) Considering a dual–arm setup the reachable regions of the left and right arm are respectively reported in green and red, while the one of the table is represented by the table itself. (b) Using the intersection between simple approximations of these regions (cuboids), we defined the workspaces for this scenario.

A. Mobile robots as movable end–effectors

In this work the concept of end–effector is extended to mobile robots that can be seen as *movable end–effectors* with a floating base. These robots can be considered equipped with either a grasping device or a simple flat surface, that provides multiple ways to grasp an object, in the sense defined above. Mobile robots are hence compliant with the end–effector definition, and their grasps are the ways they can carry around an object. The exchanging workspaces between mobile robots and other end–effectors are obtained by intersection between the collision–free configuration space of the mobile robots and the workspace of other end–effectors (see, for example, Fig. 4). Note that, depending on the grasp capabilities, end–effectors do not necessarily share a workspace. For example, a mobile robot with a surface does not share the table workspace while it does with a fixed base manipulator. Similarly, if the mobile robot is equipped with a manipulator it can exchange an object with the table, i.e, the end–effectors share a common workspace.

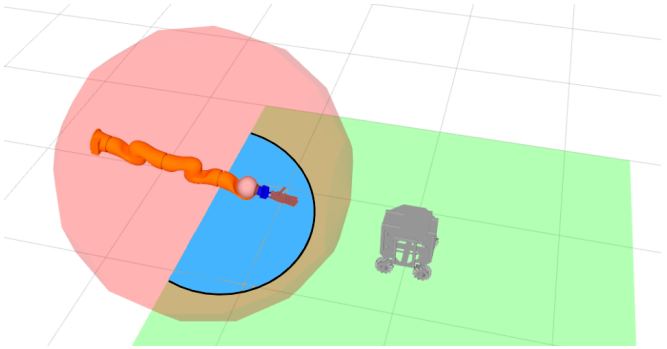

Fig. 4. The exchanging workspace (reported in blue) between a robotic arm and a mobile robot is the intersection of the single ones (reported in red for the robotic arm, and in green for the mobile robot).

III. The Time–Varying Graph

The planning of a single object to be deployed through a sequences of end–effectors has been modeled as a graph exploration problem in [10]. In order to extend the approach to the case of multiple objects a Time–Varying Graph [12], [13], [11] must be considered. Indeed, in case of multiple objects to be deployed, the planning based on static graphs can lead to the undesired behaviour of an end–effector grasping more than one object simultaneously. To avoid a trivial and non efficient solution, of handling objects once at a time in the whole environment, the graph must change in time. With the proposed algorithm we will be able to plan the motion of objects, ordered in a queue, with different end–effectors grasping different objects simultaneously.

A node of the graph is an object state $S_{e,g,w}$, where the object is (i) grasped by the end–effector e, with (ii) a particular grasp g and (iii) in a specific workspace w. The transition of an object from one state to another leads to an object position changing, this is due by a passage between different end–effectors or by a displacement between different workspaces. For the sake of simplicity, given nodes $i = S_{e_i,g_i,w_i}$ and $j = S_{e_j,g_j,w_j}$ connected by an arc $a_{ij} = (i, j)$ it holds that $g_i = g_j$ if and only if $w_i \neq w_j$. In other words, an object may be passed between end–effectors without changing workspace and if there is a workspace change the object is grasped by the same end–effector.

In this framework, a single object plan P is a sequence of nodes and arcs, coupled with the time T_i at which each transition a_{ij} starts. For each object o to be the deployed, a different graph G_o is generated. Graphs are not necessarily identical, but may contain states associated to the same end–effector.

In Fig. 5 the graph related to the example reported in Fig. 3 is represented.

In order to plan shortest path for multiple objects, we need to change the arcs' cost according to the end–effector occupation time (time to accomplish a grasp, to move the object and to leave it). Each arc a_{ij} of each graph, has a cost c_{ij} which represents an estimate of the time required to move an object from the source node i to the target node j. Such definition builds a time dependent network or a time varying graph [13]. In Fig. 6 a possible trend of the arc a_{ij} cost with respect to time is shown. When an end–effector e is occupied at a certain time, all the arcs involved with that end–effector

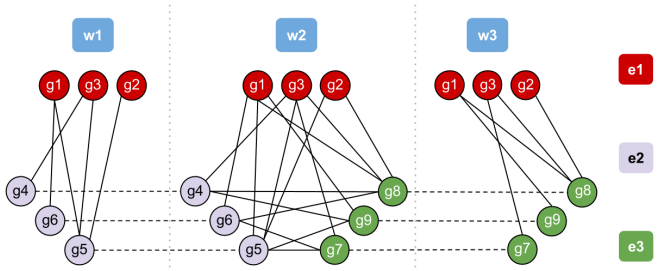

Fig. 5. Graph corresponding to the example in Fig. 3: each node represents a state of the object being grasped by the end effector e_i ($i = 1, \ldots, 3$) with the grasp g_k ($k = 1, \ldots, 9$), in workspace w_m ($m = 1, \ldots, 3$). Solid lines represent grasp transitions within the same workspace; dashed lines represent object displacements between workspaces with the same grasp.

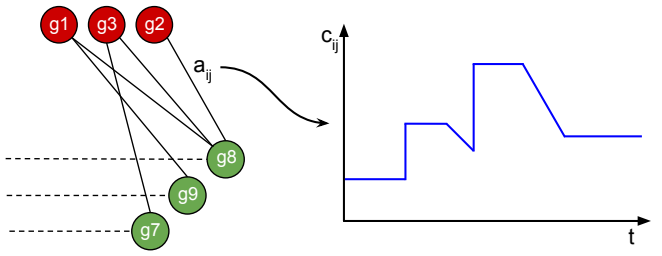

Fig. 6. Arc a_{ij} cost with respect to time. The difference from the static shortest–path problem is that the arc cost is generalized from a constant to a time–variable function. For an example of an arc cost in a real experiment see Fig. 9.

can be penalized with a higher cost which represents the need to wait for all the other objects to be deployed by end–effector e. Since the time required for an end–effector to perform the task is known based on the system status, all the arcs weights are also known when planning for the next object in the list. By using a discrete search with temporal constraints on the weighted graph, we automatically select the best sequence of end–effectors to deploy the object through its desired position.

The time $c_{i,j}$ required to move an object from node $i = S_{e_i,*,*}$ to node $j = S_{e_j,*,*}$ depends on the starting time T_i of the transition. If the end–effector e_i is occupied during the time interval $[t_a, t_b]$ such that $T_i \in [t_a, t_b]$, the cost c_{ij} of arc a_{ij} is increased by a penalty value $p(t)$ that depends on t_a and t_b. Same considerations hold for end–effector e_j.

The penalty cost is chosen as the following:

$$p(t) = \begin{cases} t_b - t, & \text{if } t_a - W_{ij} < t < t_b \\ 0, & \text{otherwise} \end{cases}, \forall\, t \in \mathbb{R}_+. \qquad (1)$$

where W_{ij} is the expected time needed to perform the transition a_{ij} neglecting all the other objects, which can be computed with an heuristic measure defined by the application scenario.

Such penalty cost focuses on minimizing the time required, but different functions could be considered in the future in order to consider grasp reliability and/or the amount of hand–offs.

Without loss of generality we assume to have a FIFO ordered list of objects to be deployed. In this case, we can compute a single object plan by considering only the occupation of end–effectors by previous objects in the queue. Note that the FIFO ordering assumption does not restrict the

proposed approach validity. Indeed, the theorem presented in [14] allows to turn any non–ordered problem into a FIFO one (in polynomial time) if waiting on nodes is allowed, as in our case.

In the following sections we will first describe the planning algorithm for a single object and then we extend it to the multiple objects case.

IV. Single object algorithm

The planning algorithm for a single object moved by multiple end–effectors proposed in [10] is now briefly reported for reader convenience. With respect to the original algorithm, modifications are proposed to adapt it to handle time varying graphs. Given the set of plans obtained for prior objects in the list and given initial and desired object position, the outcome of the algorithm is a plan for the current object that is compatible with all the other plans.

An important concept described in details in [10] is the conversion between a path in the object graph and a set of commands in Cartesian space that are given to each end–effector. As an example, an arc that represents a change of workspace is translated into two Cartesian end-effector poses, respectively in the source and target workspace.

We define with:

- C_{Init}, and C_{Final} the initial and final object configurations in the Cartesian space,
- T_{Init} the initial time of the object plan,
- G the time varying graph,
- P_C the Cartesian plan,
- $S_{\text{Init}}= S_{e_i,g_i,w_i}$ and $S_{\text{Final}}= S_{e_j,g_j,w_j}$ the initial and final object configurations in the graph,
- P_{HL} the timed plan on the graph,
- Arc_{Id} the identity of the arc that makes the Cartesian planning fail.

The algorithm consists of several functions that are briefly described next:

- *GetInformationFromDB*: all the information the planner needs are loaded from a database, these information include the end–effectors, the grasp sets for each of them and the workspaces;
- *GenerateTimeVaryingGraph*: the retrieved information, along with previous objects plans, is used to create the time varying graph through the time–based arc costs penalization;
- *CartesianToGraph*: this function translates a Cartesian positions C into a state S of the graph;
- *ShortestTimePath*: time-dependent A* is used to find the shortest path (in terms of time execution) in the graph from S_{Init} to S_{Final};
- *GraphToCartesian*: kinematic utilities are used to convert each arc in the graph plan to low–level commands such as move or grasp/ungrasp;
- *GetFailedConversion*: in case the GraphToCartesian function returns a failure due to a collision with the environment the arc that leads to a collision is returned;
- *Backtracking*: a backtracking procedure is performed whenever a collision–free path for a robot cannot be found. In this case the graph is accordingly updated by removing the arc returned by the GetFailedConversion function. The shortest path is thus computed on the updated graph.

Algorithm 1: Single Object Plan

Data: Initial and final object position C_{Init}, C_{Final}, starting time T_{Init}, Object o, Queue of other objects plans $\mathcal{P}$
Result: Plan for object o
(*states*,*transitions*) = GetInformationFromDB(o);
G= GenerateTimeVaryingGraph(*states*,*transitions*, $\mathcal{P}$);
$G_{\text{HL}}= G$;
S_{Init}= CartesianToGraph(C_{Init});
S_{Final}= CartesianToGraph(C_{Final});
repeat
  P_{HL}= ShortestTimePath(G_{HL},S_{Init},T_{Init},S_{Final});
  if P_{HL} *is empty* **then**
    Return p not valid;
  end
  P_C= GraphToCartesian(P_{HL});
  if P_C *is empty* **then**
    Arc_{Id}= GetFailedConversion();
    G_{HL}= Backtracking(G_{HL},Arc_{Id});
  end
until P_C *is not empty*;

Such hypothesis is supported by the fact that in the multi–object scenario (see next Section, for details) the planning is organized in a FirstInFirstOut approach.

The single object planning problem can be solved by applying a time–varying graph (TVG) search algorithm, such as [12], [14], [15] or time varying A*, as described in [16]. The TVG search algorithm implemented here is time varying A*, because, thanks to simple arc cost functions, we do not experienced a particular performance hit. For future developments, the use of a more performing algorithms such as [17] will be evaluated. The A* solver, in our algorithm, is reported as ShortestTimePath(graph, start state, start time, end state) function in Algorithm 1.

Note that the introduction of mobile robots does not affect the graph exploration algorithm at all, while the conversion between graph plan and Cartesian positions, performed by the GraphToCartesian function, require a different inverse kinematic algorithm.

V. Multi-object serial planning

As reported in previous Section, occupied end–effectors are used to generate time dependent arc costs. Consequently, the planner chooses objects paths considering parallel paths in the graph, searching for the shortest time one.

Since we are assuming an ordered queue of objects, we use a FIFO approach when planning for multiple objects. Thus, for each object o_i in the queue, we can safely ignore the rest of the queue $o_j, j > i$, and consider only the timed paths of objects o_j with $j < i$.

The multi–object planner is composed of two main parts: a loop that generates a timed plan for each object in the queue through Algorithm 1, and a parallel motion planning where the Cartesian commands are used to generate joint space collision free trajectories for each end–effector as shown in next Subsection. Once the first part has generated the set of Cartesian plans $\mathcal{P}$, those plans are reordered by the starting time of each transition and merged into a sequence of Cartesian commands χ.

Algorithm 2: Global high level planning and execution

Data: Queue of objects $o \in \mathbb{O}$, with their initial and final positions C_{Init_o},C_{Final_o} and starting time T_{Init_o}
initialization;
$\mathcal{P}$=empty;
while $\mathbb{O}$ *is not empty* **do**
  o = select next object from $\mathbb{O}$;
  p=plan single object(C_{Init_o},T_{Init_o},C_{Final_o},o,$\mathcal{P}$);
  if *p is valid* **then**
    add p to $\mathcal{P}$;
  else
    Even in infinite time, the object cannot be moved;
    Skip object;
  end
end
χ = reorder and merge plans($\mathcal{P}$);
foreach *Cartesian command in* χ **do**
  plan low-level parallel motions and execute;
  if *Failure in motion planning* **then**
    /*Local replan*/
    Plan motions with different grasps but same end–effectors;
    if *Failure in motion planning* **then**
      /* Global replan*/
      restart algorithm with C_{Init_o} = current object positions;
    end
    change motion plan and continue execution;
  end
end

After the path reordering joint space trajectories are generated taking into account that low level failures can occur when single end–effector motion planning fails due to collision along the planned trajectory. Usually a simple local re–plan that involves the same end–effectors is enough to solve the problem, but if the new plan involves changes in the timings or in the end–effectors involved in subsequent object handling, a global re-plan is required. In case of such failures affecting the global plan, Algorithm 2 can be re-initialized with the current objects positions as the initial states. Indeed, with the proposed approach the initial Cartesian positions of the objects are not constrained to be in any particular workspace.

Note that once the multi-object planning algorithm has found a global solution, the execution does not utilize the information about the starting time of each command. Indeed a low level control can be performed by the end–effector e once previous commands in the list have already been started and in case e is ready to be used.

A. Motion planning for different end–effectors

The hierarchical structure of our approach leads to a decoupling low level path planning algorithm for the execution of *move, grasp, ungrasp* commands. The motion planning algorithm for each end–effector is implemented with RRT* [18] and RRT-Connect [19] with collision checking and kinematic constraints. In particular, collision checking for mobile robots is implemented using a 2D occupancy grid of the whole setup, while for arm manipulators is implemented a 3D collision check between robot links and the possible sub-set of objects in the environment that may collide with the arm. Self collisions are enabled only for the arm manipulators. Finally, for simpler end–effectors such as tables and conveyor belts analytical formulas are used.

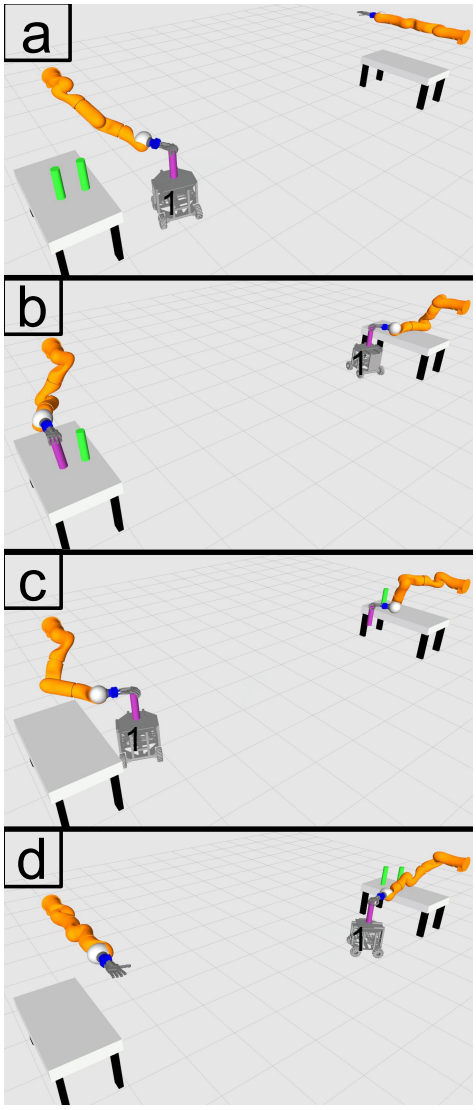

Fig. 7. Multiple objects are carried across the factory using two robotic arms and a mobile robot. In (a) a robotic arm takes the first object, which is given to the other robotic arm by the mobile robot in (b). In (c) the third object is put on the mobile robot while the second one is put in the final destination. In (d) the third object is carried by the mobile robot to the robotic arm on the right.

VI. Validation

To validate our approach, different kinematic simulations have been performed. Real experiments to validate grasping capabilities and the low level motion planner have been carried out for a dual–arm setup, using objects of different shapes such as kitchen tools, in our previous work [10]. Using the same framework, also a supply chain scenario formed by 5 Kuka LWR arms and a conveyor belt has been simulated for moving a single object. Videos referred to these examples, and

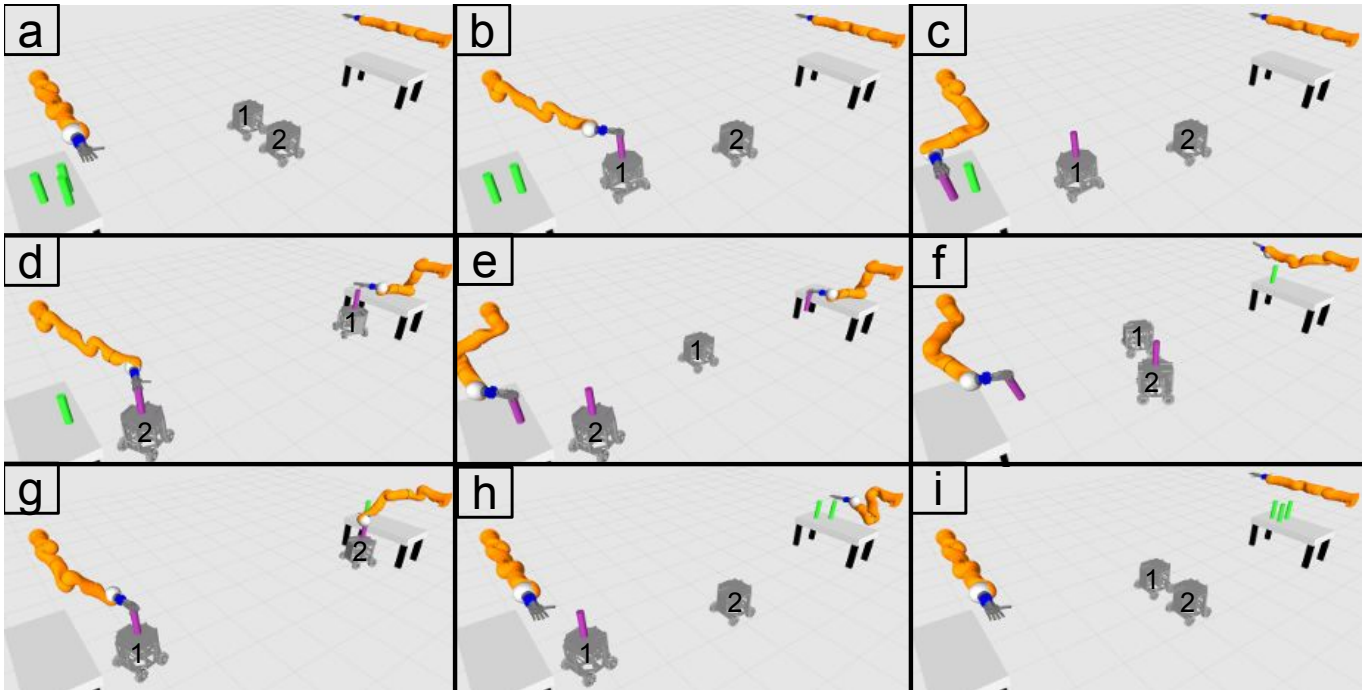

Fig. 8. A second mobile robot is added, availability of the two robots is automatically handled by the time varying graph leading to a faster task accomplishment. The three objects are brought from a side of the scenario to the other by the mobile robots. Each time the robotic arm on the left put the cylinders on the currently available mobile robot. The various snapshots are temporally contiguous (from (a) to (i)).

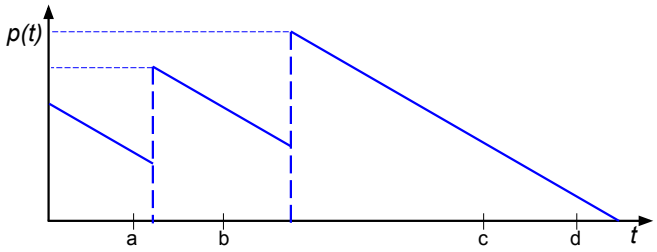

Fig. 9. Arc penalty cost associated to the workspace change of the mobile robot in the example of Fig. 8.

the others reported in this section, can be found online[1].

In the following the different scenarios consisting of two Kuka LWR arms, one or two mobile robots and objects of different shapes are considered. In figures we will refer to the different robots as: KL (robotic arm on the left), KR (robotic arm on the right), M1 (mobile robot 1) and M2 (mobile robot 2). In the first scenario, we show a simple setup where three similar objects need to be moved from a table to another across the room. The only possible path is trough M1, which moves from a table to the other three times. This shows the seamless integration of mobile robots in our framework. The scenario is reported in Fig. 7 where only some snapshots of the performed test are shown. In Fig. 7a KL takes the first object, which is given to KR by M1 in Fig. 7b. In Fig. 7c KL puts the third object on M1 while KR is placing the second one on its workbench. Finally in Fig. 7d the third object is carried to KR.

[1]https://www.youtube.com/channel/UCb6ECYixj3Tobh9hpSYJ7Ug

The penalty cost associated to the arc representing the change of workspace of the mobile robot in Fig. 8 is shown in Fig. 9.

In order to show the effects of our time varying planner, we add a second mobile robot to the previous experiment. Thus, the three cylinders can be moved in parallel between tables, and the total time of the task is obviously reduced. Note that the planner respects the constraint imposed by KR, as the timings of actions never force KR to work on multiple cylinder at the same time. Some screenshots are shown in Fig. 8. First, KL puts the object on the M1 and then retrieves the second object (Fig. 8a, 8b, 8c). While M1 brings the robot to KR, M2 approaches KL to take the second object (Fig. 8d). The third object is thus given to M1 (which is again available) by KL and M2 brings the second object to KR (Fig. 8e–i)). The sequence of Cartesian commands χ for this scenario, reported for clarity, consists of the followings:

1) *move* KL towards *object1*
2) *grasp object1* using KL
3) *move* M1 and KL towards a common location
4) *grasp object1* using M1, *ungrasp* from KL
5) *move* KL towards *object2*
6) *grasp object2* using KL
7) *move* M2 and KL towards a common location *AND move* M1 and KR towards a common location
8) *grasp object2* using M2, *ungrasp* from KL *AND grasp object1* using KR, *ungrasp* from M1
9) *move* KR towards *object1* final location
10) *ungrasp* from KR (*object1* reaches its final location) *AND move* KL towards *object3*
11) *grasp object3* using KL
12) *move* M2 and KR towards a common location *AND move* M1 and KL towards a common location
13) *grasp object3* using M1, *ungrasp* from KL *AND grasp object2* using KR, *ungrasp* from M2
14) *move* KR towards *object2* final location
15) *ungrasp* from KR (*object2* reaches its final location)
16) *move* M1 and KR towards a common location

17) *grasp object3* using KR, *ungrasp* from M1
18) *move* KR towards *object3* final location
19) *ungrasp* from KR (*object3* reaches its final location)

Finally, in Fig. 10, the same robot setup as in Fig. 8 is used, the goal is now to assembly different objects for which reorientation is requested to achieve desired final arrangement. The re–orientation is automatically computed by the single object graph planner since the final configuration of the objects is associated to a different node of the graph.

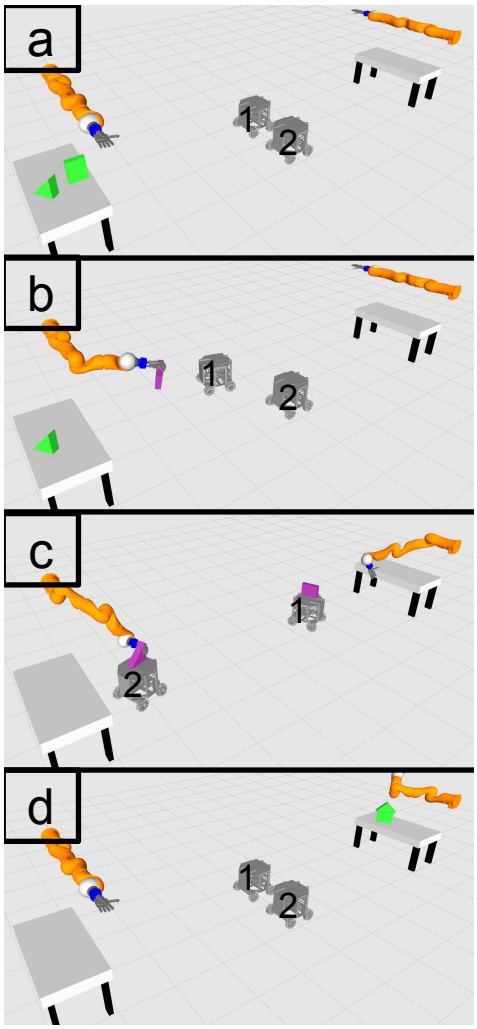

Fig. 10. Using different paths to move different objects: in this scenario, two pieces need to be mounted, and both are reoriented during their path from initial to final configuration in order to obtain the desired assembly. The two object are brought in the correct order to be assembled, from (a) to (d).

VII. Conclusions

In this work we proposed a novel approach to handle objects moving in industrial scenarios with heterogeneous robots. In particular the approach is thought to be used in the domain of smart factories where complete automation is the key for competitive solutions. This work is an extension to our previous work [10] to handle multiple objects at the same time and include mobile robots in the environment, and it is based on a time varying graph on which a shortest path is performed to find the optimal solution. All the code of this framework is available online[2].

Acknowledgments

This work is supported by the grant no. 645599 "SoMa" -Soft-bodied intelligence for Manipulation- within the H2020-ICT-2014-1 program.

References

[1] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9, 2008.
[2] K Boyer, T Hult, and M Frohlich. Ocado: An alternative way to bridge the last mile in grocery home delivery. *Case No. 602-057*, 1, 2002.
[3] Amazon picking challenge website, http://amazonpickingchallenge.org, 2015.
[4] Hao Luo, Ji Fang, and George Q Huang. Real-time scheduling for hybrid flowshop in ubiquitous manufacturing environment. *Computers & Industrial Engineering*, 84:12–23, 2015.
[5] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. Incremental task-level reasoning in a competitive factory automation scenario. In *AAAI Spring Symposium: Designing Intelligent Robots*, 2013.
[6] Shiyong Wang, Jiafu Wan, Di Li, and Chunhua Zhang. Implementing smart factory of industrie 4.0: an outlook. *International Journal of Distributed Sensor Networks*, 2016, 2016.
[7] Basilio Bona, Luca Carlone, Marina Indri, and Stefano Rosa. Supervision and monitoring of logistic spaces by a cooperative robot team: methodologies, problems, and solutions. *Intelligent Service Robotics*, 7(4):185–202, 2014.
[8] Simon Bogh, Casper Schou, Thomas Rühr, Yevgen Kogan, Andreas Dömel, Manuel Brucker, Christof Eberst, Riccardo Tornese, Christoph Sprunk, Gian Diego Tipaldi, et al. Integration and assessment of multiple mobile manipulators in a real-world industrial production facility. In *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, pages 1–8. VDE, 2014.
[9] Alwin Hoffmann, Andreas Angerer, Andreas Schierl, Michael Vistein, and Wolfgang Reif. Service-oriented robotics manufacturing by reasoning about the scene graph of a robotics cell. In *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, pages 1–8. VDE, 2014.
[10] Hamal Marino, Mirko Ferrati, Alessandro Settimi, Carlos Rosales, and Marco Gabiccini. On the problem of moving objects with autonomous robots: a unifying high-level planning approach. *IEEE Robotics and Automation Letters*, 1:469–476, 2016.
[11] Kenneth L Cooke and Eric Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of mathematical analysis and applications*, 14(3):493–498, 1966.
[12] Stuart E Dreyfus. An appraisal of some shortest-path algorithms. *Operations research*, 17(3):395–412, 1969.
[13] Stefano Pallottino and Maria Grazia Scutella. Shortest path algorithms in transportation models: classical and innovative aspects. In *Equilibrium and advanced transportation modelling*, pages 245–281. Springer, 1998.
[14] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37:607–625, 1990.
[15] Evangelos Kanoulas, Yang Du, Tian Xia, and Donghui Zhang. Finding fastest paths on a road network with speed patterns. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 10–10. IEEE, 2006.
[16] Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes. Bidirectional A∗ search for time-dependent fast paths. In *Experimental Algorithms*, pages 334–346. Springer, 2008.
[17] Mostafa K Ardakani and Madjid Tavana. A decremental approach with the a algorithm for speeding-up the optimization process in dynamic shortest path problems. *Measurement*, 60:299–307, 2015.
[18] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt*. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, page 1478. IEEE, 2011.
[19] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[2]http://dualmanipulation.bitbucket.org