Automated planning in hybrid domains for in-space robot assembly tasks

Mariana Dias Cunha
mariana.d.cunha@tecnico.ulisboa.pt
Instituto Superior Técnico
December 2021

***Abstract*—Recent developments in the area of automation of in-space additive manufacture and assembly of structures has resulted in the development of in-space mobile robots that perform assembly and logistics operations inside the International Space Station (ISS). Consequently, studies and improvements in the area of robot Task-Motion Planning (TMP) need to be made in order to improve robot navigation. Inspired by this, we have created a hybrid domain using PDDL that describes mobile robots navigating through a 2D approximation sampled roadmap that are responsible for moving, loading and assembling modules.**

We have chosen a state of the art TMP approach called MPTP, however this approach does not take into account that some map areas cannot be included in the motion plan once the robot places modules in it. We have implemented an approach that allows the motion planner to be informed of the map availability according to the actions expanded by the task planner.

We defined a scenario to test and compare the use of task and motion planners separately versus a mixed TMP approach and we were able to get results that showed that using a mixed approach in domains such as the space assembly domain can be highly beneficial since it is aware of physical motion constraints while task planning.

Keywords: Hybrid domain; PDDL; Mobile robot navigation; Task planning; Motion planning

I. Introduction

There has been a collaboration between ISR-Lisboa and MIT Space Systems Laboratory in the context of autonomous robotic assembly of space structures using on-orbit additive manufacturing. By enabling the autonomous robotic assembly of space structures, near-Earth science can be improved which is the main goal of this international partnership. Since 2014 there is an on board 3D printer operating in the International Space Station (ISS) which enables the production of 3D parts and tools that can be used for repairing or improving existing hardware. This technology has the potential of being extended to fabricate bigger parts of space structures that can then be autonomously assembled in space, making way for the ISS to be able to repair and re-provision and thus being able to maintain and upgrade its own structure. By means of path planning and assembly of parts this would make way for exploration and scientific operations of space structures.

One of the ISR-Lisboa and MIT Space Systems Lab collaboration renovation proposal report for 2020 proposed goals is to understand how additive manufacturing of components for on-orbit assembly can be translated into assembly via high-level path and task planning. This work is related to this objective which concerns the autonomous assembly of parts via high-level path and task planning. The work in [1] shows a thorough study of different architectures involving in space printing and assembly of parts, one of them involves a mobile robot that is operating in an environment where a 3D printer printed three parts. In this example the robot uses proximity operations to assemble the 3D printed modules next to the printer in reverse order. We use a similar architecture where there is a mobile robot and printed modules that need to be assembled into a desired goal configuration and thus need a task and motion plan. This type of planning is hybrid planning since it involves discrete and continuous aspects and it is integrated in the field of task-motion planning. This is an area that has been evolving and different approaches have been emerging: some where task and motion planning are done in a separate way and others where this is done in a combined way, where task planning has a form of integrating motion planning. We want to contribute to the study of these different task-motion planning approaches in the context of the space assembly of 3D printed parts in order to know what would be more appropriate and produce better results in terms of solution task-motion plans. We focus on studying which type of task-motion planning approach is better suited for a scenario of a robotic autonomous assembly of modules through experimental results in a simulated environment. In this paper we show a definition of a space assembly domain into PDDL where mobile assembler robots can move and assemble parts inside an environment like the ISS and we choose a state of the art task-motion planning approach (MPTP) to run tests using the space assembly domain. We also develop an algorithm that incorporates updated information on the roadmap points' availability into MPTP. By comparing task-motion planning separate and mixed approaches we are able to provide an answer for which type of approach is better suited for domains where the tasks to carry out include assembly of structures and logistics operations inside a space station.

II. Background

A. Classical Planning and PDDL

Classical planning is the act of finding a sequence of actions that leads from an initial to a desired goal state called a **plan** which is called **optimal** if it is the one with the lowest cost. There is a popular language used to define classical planning problem domains called Planning Domain Definition

Language (PDDL) [2] which is capable of representing the four main aspects of a classical planning domain: the initial state, the actions that can be executed in each state, the effects of each of those actions and the goal state test.

In PDDL, a state is represented by conjunctions of fluents, for instance, $At(robot_1, area_1) \wedge At(robot_2, area_2)$ for describing a state in which multiple robots are in different areas, where $At(robot_1, area_1)$ is a **predicate**, which is a relation between one or more objects, and $robot_1$ and $area_1$ are instantiated **objects** of the object types $robot$ and $area$, respectively. An action is represented by an **action schema**, which specifies its name and defines the necessary **preconditions** for the action to be able to be executed, as well as the **effects** that it produces. If the preconditions of an action a are satisfied by the state s, then a is **applicable** in s. An action schema may contain variables that can be assigned values. An action can have multiple models that make it applicable in a state s. In order to define a planning problem using PDDL we need to provide a domain description where we define all necessary action schemas and predicates with their respective variables, and a problem description which has to specify the objects that will instantiate the predicates and the action schemas by replacing all the free variables, and it also has to specify an initial state and a goal condition.

Several versions of PDDL have emerged through the years, one of them being PDDL2.1 [3] which can handle temporal considerations (scheduling) and numeric considerations (resources) and introduced, for that purpose, numeric fluents for continuous change, plan-metrics and durative/continuous actions. A **durative action**, besides having preconditions and effects, also has a duration and a way to assign time to each precondition and effect by specifying that each one occurs *at start*, *at end* or *over all* the action. A **numeric fluent** is a variable that has a value throughout the plan and applies to zero or more objects from the PDDL domain. Both actions and durative actions' effects can change the value of a numeric fluent. It is declared with a name, an object name and object type, as follows: `(battery-level ?r - rover)`, where the numeric fluent allows every rover object from the domain to have a variable that represents its battery level. An extension of PDDL2.1 is PDDL+ which was created to enable the modelling of mixed discrete-continuous domains. It also provides a more flexible model of continuous change through the use of processes and events and supports modelling of exogenous events.

B. Hybrid Systems

A **hybrid system** is one that contains both continuous and discrete variables. Discrete variables are the ones associated with discontinuous transitions between different states and continuous variables represent continuous evolution within a state. In the context of this thesis, since we are dealing with mobile robots, there is the need for a hybrid plan since we need both a task and a motion plan for the robot.

There are a lot of different hybrid system based planning approaches that emerged over the years. One of them is UPMurphi [4] (2009) and it performs universal planning using a model checking based algorithm for hybrid and nonlinear systems. It is also able to read problem specifications from PDDL+ files and plan for problems with time and resources constraints. Another hybrid planning approach is DReach [5] (2015) which also uses PDDL+. This approach is able to accommodate nonlinear change by encoding problems as nonlinear hybrid systems and then applying Satisfiability Modulo Theories (SMT) (we refer the reader to the work in [6] about SMT). This planner finds plan tubes instead of concrete plans because it solves a δ relaxation of the problem. It also presents heuristics for improving SMT variable selection and pruning and can prove plan non-existence. A different approach that also uses an SMT encoding of PDDL+ domains is SMTPlan+ [7] (2016) which is also able to deal with nonlinear arithmetic and it can use any SMT solver. This planner is also efficient in proving plan non-existence up to a certain bound and has proven to outperform UPMurphi and DReach in terms of time and number of instances it is able to solve using different types of domains. A more recent hybrid system based planning approach is Optimization Modulo Theories (OMT) [8] (2018) which consists of SMT solving with optimization capabilities. It is a task planner capable of finding optimal solutions and working in a multi-robot environment with concurrent actions.

Another hybrid planner is POPF-TIF which is a task planner relevant for this work. POPF [9] is a forward-chaining state-based search planner designed to solve temporal-numeric problems supporting the concept of partial order planning. It handles continuous linear numeric change and is based on grounded forward search combined with linear programming (LP). It does not sequentially build a plan, instead it builds partially ordered collections of actions. The partial ordering is achieved by delaying the commitment to ordering the decisions, timestamps and values of the numeric parameters by managing sets of constraints as actions start and end, meaning that the precise embedding of actions in time is delayed until constraints emerge. This approach benefits from the informative search control of forward planning and at the same time it has some level of flexibility due to its late commitment strategy. The partial ordering is done in a way that ensures plan consistency. Since it is a temporal planner, POPF uses PDDL2.1 which supports durative-actions.

An important aspect of POPF is that it imposes total ordering in steps that change the value of a variable v, imposing the order in which the steps are added to the plan. The planner also forces conditions that depend on active process effects to stay within those processes. In general, this approach represents a middle ground between least and total commitment. It implements a cost-optimisation approach meaning that it seeks a plan that minimizes the total plan cost.

The extension POPF-TIF was introduced in [10] and was also explored in [11] where the authors extended the concept of using an external advisor. This planner's goal is to handle problems with numeric Timed Initial Fluents (TIF). POPF-TIF implements a better heuristic evaluation and the addition of alternative search methods using a combination between

Enforced Hill Climbing (EHC) and Best First Search (BFS). It is also able to include exogenous events that assign values to fluents but not exogenous events that add or delete some effect. This is possible with the TIFs and the way it is done is by separating each domain fluent into two parts: one that is changed by the action and another that is changed by TIFs. An example of a fluent `x` would be represented in the domain as `(at end (increase (x) (external)))` where the value of `x` is increased by the value of the `(external)` variable that is a TIF and is calculated externally.

POPF-TIF is equipped with the ability to be connected to an external solver/advisor which allows the planner to perform sophisticated mathematical operations, which is something that PDDL alone is not prepared to handle. The connection between the planner and the external advisor is done through semantic attachments. This external advisor can be used to compute numeric information and more effective heuristic values. This feature makes POPF-TIF a powerful framework especially for solving problems with numeric goals.

C. Task-Motion Planning

Task planning refers to the process of finding a discrete sequence of actions that lead from a starting state to a desired goal state. Motion planning, also referred to as path planning, is the process of finding a sequence of collision-free poses, with the respective position and orientation values, that form a path to get from a starting point to a desired goal pose. When planning in robotics there is the need to have both a task plan of the discrete actions and a motion plan describing the robot's motion, but combining both is a complex problem. Task-Motion Planning (TMP) involves an interaction between the decision-making process on both the discrete and continuous domains. Different approaches have been suggested for TMP and this is an area of research that keeps getting attention. Initial TMP approaches would work by doing task planning first and then using the resulting sequence of actions to instruct the motion planner. This is done under the assumption that the robot will be able to carry out each motion with no restrictions, which is sometimes not the case, since there may be some geometric constraints. Other approaches try to mix task and motion planning, presenting a task-motion interface.

Some state of the art task-motion planning approach examples include Iteratively Deepened Task and Motion Planning (IDTMP) [12] (2016) which is a constraint based task-planning approach that uses SMT as a way of making the task planner aware of geometric constraints and have some level of motion feasibility awareness at the task planning level. It is probabilistically complete and is able to handle domains with diverse actions as well as model kinematic coupling. Another TMP approach is PDDLStream [13] (2020) that extends PDDL to incorporate sampling procedures. It also presents two new algorithms: Binding and Adaptive. These algorithms reduce PDDLStream planning to solving a series of finite PDDL problems. The Adaptive algorithm balances the time spent searching and sampling and aggressively explores many possible bindings, outperforming existing algorithms, particularly in tight-constrained and cost-sensitive problems, by greedily optimizing discovered plans. PDDLStream can be used to plan for real-world robots operating using a diverse set of actions. A more recent TMP approach is Motion-Planning-aware Task Planning approach (MPTP) [14] (2021) which uses a task planner combined with an external solver which has a motion planner incorporated. It uses POPF-TIF as the task planner, but a different one could be used. It uses PDDL2.1 to define the task level actions and it works by defining numeric fluents in the domain and taking advantage of semantic attachments to have an external solver calculate some numeric fluent variables including the motion cost and the respective path plan returning the values to the task planner.

Even though all three approaches are capable of doing task-motion planning they have limitations. PDDLStream is undecidable and its algorithms are semi-complete, meaning that it is complete only under feasible instances. IDTMP and MPTP are both probabilistically-complete. The task-motion interaction in IDTMP is done by the use of abstraction and refinement functions while MPTP uses semantic attachments as its task-motion interface. Another difference is the fact that, while PDDLStream and IDTMP approaches both involve TMP for manipulation, MPTP is focused on TMP for navigation. MPTP involves a motion planning aware task planner, taking motion costs and motion plan feasibility into account.

D. MPTP

MPTP is the planning approach that will serve as baseline for this work. This planning approach is meant for navigating in large knowledge-intensive domains, it is probabilistically complete and returns a solution plan that is optimal at the task level. The MPTP framework is also prepared to deal with belief space planning, i.e. motion planning under motion and sensing uncertainty in partially-observable state-spaces, although this is not explored within the scope of this master's thesis. MPTP presents a task-motion interface layer that allows the motion planner to inform the task planner of the motion feasibility as well as the associated costs.

In this approach, there is a set of discrete actions that can be expanded by the task planner. Every time the task planner expands an action that requires robot motion, an external solver is called/triggered. When this happens, the discrete symbolic parameters are converted to the corresponding continuous geometric instantiations. These geometric instantiations are pre-sampled upon knowing the map of the environment (a roadmap-based sampling is used, specifically PRM). For each call of the external solver different motion plans are obtained for these instantiations and the best one is chosen according to a specific metric. Finally, the cost associated with the chosen motion plan is returned to the task planner and it corresponds to the cost of the associated action. Because the motion cost is returned as every action is expanded, the resulting plan will be optimal at the task level. The cost used in this approach is the sum of the trajectory length and the cost associated with motion and sensing uncertainty. However, MPTP is capable of supporting any cost function and since uncertainty is not

within the scope of this thesis, the respective motion and sensing uncertainty cost will not be considered.

Since this approach will be used to accomplish this thesis' goal, we provide a more practical and in depth explanation of how it works. MPTP uses POPF-TIF [10] as the task planner because it is a temporal task planner that can handle numeric time initial fluents. The work in [10] and [11] introduces the concept of semantic attachments, allowing the POPF-TIF planner to have an external solver call and MPTP takes advantage of this concept to incorporate the motion planner.

According to [11], a **semantic attachment** evaluates numeric fluents using externally specified functions. For this interface to work, all the domain's numeric variables are categorised either as indirect variables (V^{ind}), direct variables (V^{dir}) or free variables (V^{free}). The planner, in this case POPF-TIF, determines the value of the V^{dir} variables and when these values change they affect the V^{ind} variables. The V^{ind} variables are calculated by the external function/advisor based on the information provided by the planner. The V^{free} variables are the remaining variables evaluated by the planner but they do not trigger any external computation. Figure 1 shows an overall view of the described structure.

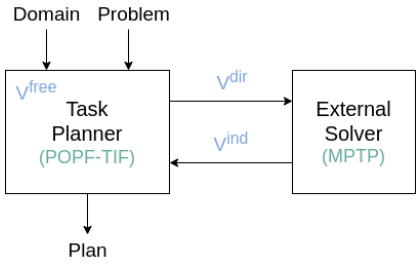


Fig. 1. Overview of the MPTP structure (V^{dir} - direct variables; V^{ind} - indirect variables; and V^{free} - free variables that stay within the task planner)

In other words, every time POPF-TIF expands an action it passes the V^{dir} variables to the external function and in turn the advisor returns the V^{ind} variables to the planner, so a semantic attachment can be seen as a function that is dependent on the V^{dir} variables and computes the V^{ind} variables. More precisely, when POPF-TIF first updates a state, the V^{dir} and V^{free} variables are computed. If any of the V^{dir} values changed, then the external function is called to compute the V^{ind} variables and it receives all the V^{dir} values as input.

III. Methodology

A. The Space Assembly Domain

We consider a mobile robot navigating through a corridor inside the International Space Station (ISS) that holds, moves and assembles modules and we refer to it as the space assembly domain. To simplify the problem, we consider a 2D approximation of the ISS Columbus laboratory (which is the European lab inside the ISS) as a corridor of 3×6 meters. This means that the environment's map is known a priori. As can be seen in figure 2 we design the corridor as a grid of 1×1 meter squares so that each of these squares can be used as a location to the task planner when defining the domain.

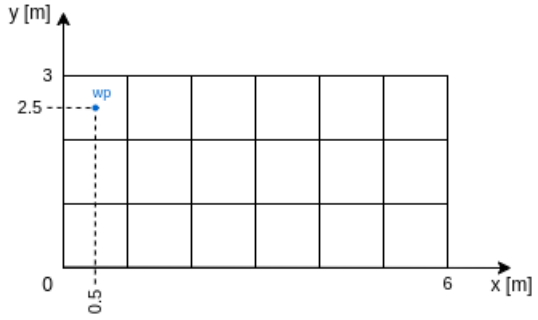


Fig. 2. Representation of a 3×6 meter 2D ISS corridor approximation in an xy coordinate axis (wp stands for waypoint and represents an example of a sampled roadmap point and its respective coordinates)

We use a roadmap-based sampling method where a map of the environment is defined by waypoints (which are the sampled points that constitute the map). In order to guarantee that each location contains at least one waypoint, the center of all grid squares is manually added to the roadmap and labeled as shown in figure 3 where it is also shown how the locations are referred to in the problem file: each row with a letter in alphabetical order and each column with a number $i \in \mathbb{R}_0$.

Fig. 3. The 2D ISS corridor approximation with a waypoint for each location and the used nomenclature (the rows are identified by letters and the columns by numbers, so the location containing $wp0$, for example, is location $a0$)

In this domain we define 3 types of objects: (1) **location**, an area of the ISS corridor corresponding to a square from the grid approximation; (2) **module**, these are assumed to be equally sized cubes (squares in our 2D approximation) corresponding to the parts that the robot has to assemble; we also assume that a module occupies all the area covered by the location where it is placed; (3) **robot**, the mobile robot that can grab modules and navigate through the ISS.

The goal is to have the robot assemble the modules into a desired configuration. A robot may not be carrying anything, which is encoded by the predicate `(empty ?r)`, where `r` is the robot, or it may be holding a module, which is encoded by the predicate `(loaded ?r ?m)`, where `m` is the module that is being carried by the robot `r`. In either case, the robot can move from one location to another using the high-level action `move_robot`. To describe the current location of a robot we use the predicate `(at ?r ?l)`, where `l` is the

location where the robot r is. If a location has a module placed on it, it is encoded by the predicate (on ?m ?l), where m is the module placed on location l. If a location does not have a module on it, then this corresponds to the predicate (clear ?l), which encodes that the location l is clear. Another predicate defined in this domain is (adjacent ?l0 ?l1), where l0 and l1 are two distinct locations that are adjacent to each other. This predicate has a crucial role in this domain, since we consider that a robot cannot leave a module floating inside the ISS, one of the preconditions for a robot to unload a module is that it has to be assembled to another module, which can be translated as: only unload the module if there is another module on an adjacent location. The action that encodes a robot unloading/assembling a module is assemble. For simplicity purposes we assume that all modules can be assembled to each other with no restrictions. For a robot to pick up a module it has to be within its reach, which means that the robot has to be at a location adjacent to a different location with a module in it. The robot also has to be empty for it to be able to pick up a module since a robot can only grab one at a time. The action load is the one responsible for encoding a robot grabbing a module.

In this domain there are also the following numeric fluents:

- (act-cost): direct variable that models the cost associated with the actions; it will be used as the metric to minimize by the task planner;
- (extern): the motion cost, an indirect variable that is returned by the external solver. In the action that involves robot movement, after being computed by the external solver, this variable's value will be added to the act-cost value as an effect of this action;
- (triggered ?from ?to): direct variable used for the action that involves robot motion (move_robot) whose value is 1 at the beginning when the action is expanded and 0 once the action duration is complete. Every time this variable changes it triggers the motion planner in the external solver and gives two locations as arguments - the from and to positions where the robot is and to where it wants to move, respectively;
- (occupied ?loc): direct variable used for the assemble action that takes as argument the location where a module is being placed in order to let the external solver know that this location is now occupied. This is the core of our approach, we program the external solver to remove the waypoints that are within this occupied location for the motion planner to know that they are no longer available.
- (unoccupied ?loc): direct variable used for the load action. This is the opposite of the occupied variable, since its goal is to inform the external solver that a module was removed from a location which means that this location is now unoccupied and the external solver can register that the respective waypoints became available for the motion planner to use.

B. Expansion of the MPTP Approach

We choose the MPTP approach as the most appropriate to use in the context of this dissertation because, while other TMP approaches are more focused on TMP for manipulation, MPTP is focused on solving TMP for navigation which is what the space assembly domain requires since it revolves around a mobile robot carrying and assembling modules.

The communication between the task planner (POPF-TIF) and the external solver is done through the use of V^{dir} and V^{ind} variables. In our approach there are four V^{dir}: act-cost, triggered, occupied and unoccupied; and one V^{ind}: extern which is calculated by the external solver. The external solver defined by the MPTP approach mainly consists of a motion planner and it works by using (triggered ?from ?to) and (increase (act-cost) (extern)). The numeric fluent triggered is what allows the motion planner to receive information about the start and goal locations of the robot motion that the action involves and the indirect variable extern is the motion cost computed and returned to the task planner by the external solver/motion planner. The act-cost is the direct variable that the task planner uses to accumulate the cost of the expanded actions. Consequently, (increase (act-cost) (extern)) is used as an effect in the move_robot action, since it involves robot motion, to increase the act-cost variable by the motion cost value returned by the motion planner, while the remaining actions that do not involve robot motion use something like (increase (act-cost) n) where n is a value defined by the task planner which corresponds to the action's cost.

In order to accomplish this works' goal, we change the external solver to incorporate our approach as well as the motion planner defined by MPTP. Our goal is to see if the task plan changes when the high-level actions result in motion constraints in cases like the space assembly domain. In this domain a robot can move modules from one location to another. By doing so, the sampled robot poses that are contained within the new module's location are now unavailable and this may imply that the robot has to find a different path to go around the module, which may result in a larger path with a higher motion cost. A case like this would mean that the task plan would have to change according to the motion planner information in order to find the optimal plan.

We extend the external solver to keep track of a list of currently available waypoints. This includes all the map points that belong to locations that do not have a module placed in it. Every time an assemble action is expanded by the task planner, the (occupied ?l) numeric fluent informs the external solver of the location l that is now occupied and the list of available waypoints is updated by removing all the points contained within the given location. A similar thing happens whenever the task planner expands a load action: the (unoccupied ?l) fluent informs the external solver of the location that has just been unoccupied and the list is updated by adding the waypoints within the respective location.

The way that the motion planner works in MPTP is by expanding the waypoints one by one between the initial and goal locations. By doing this, it calculates and stores the cost of each different available path between the two locations returning the smallest cost at the end. The cost we use is the sum of the distance between all the waypoints that constitute the path. Our implementation gives the motion planner access to an updated list of available waypoints at all times so whenever the motion planner is expanding the possible paths between two locations it checks if the next waypoint is available and in case it is not available, the motion planner skips it and does not take into account for the path.

The external solver can only return numeric variable values to POPF-TIF. This means that in case there are modules blocking all possible paths between two locations it is not possible for the external solver to inform the task solver that there is no feasible motion plan for that particular action. To get around this situation we make the external solver return a very high cost whenever there is no path available in order for the task planner not to include that action in the final plan. Since the goal is to find a plan that minimizes the `act-cost` variable which corresponds to the sum of all action costs from the task planner output plan, the motion cost has an impact on the chosen plan actions.

IV. Implementation

POPF-TIF (implemented in C++) works with the following inputs: the PDDL domain file, the problem file and a user defined external solver/advisor (which is a dynamically loaded shared library). When an external solver is provided, POPF-TIF calls it using a function which has the state as argument as well as a boolean variable that states if it wants to use a heuristic or not. The MPTP approach defines the motion planner in a function called $callExternalSolver$ that receives the two given arguments, one of them being the current state. This includes not the predicates defined on the $problem.pddl$ file, but a map from the names of the numeric fluents defined on the $domain.pddl$ file to their respective current values.

In addition to the $domain.pddl$ and $problem.pddl$ files, MPTP also needs three more files in order to work and to know the environment's map: (1) $edge.txt$ - contains a list of all the edges in the form of pairs of connected waypoints; if we use the map on figure 3 an example of an edge would be $(wp0, wp1)$ which are two adjacent locations and therefore they are connected; (2) $waypoint.txt$ - contains a list of all the map's points, each with a respective set of coordinates x, y and θ for the orientation; once again, if we use the map on figure 3, an example of a waypoint contained in this file would be $wp0[0.5, 2.5, 0]$; (3) $region\_poses$ - lists the correspondence between each location and the respective set of waypoints that is contained within it; using the map on figure 3, what would appear in this file for location $a0$, for example, would be: $a0$ $wp0$, since $wp0$ is the only waypoint inside location $a0$. The problem file can change according to whatever example we want to test out. Since there can be an infinite number of problem files and consequently edge, waypoint and region_poses files (because these three change according to the map of the problem) we automated the process of creating these four files by developing a python script to quickly be able to define new problems.

Besides defining the space assembly domain and problem, our approach consists in improving MPTP in order for it to know and update which waypoints are available while the task planner is expanding the actions to find a solution plan. We start by initializing a variable called $available\_wp\_list$ which is a list of all the available waypoints. This list is initialized with all the waypoints in the provided map and then we read from the $problem.pddl$ file in order to know which are the initial module locations that we initially remove from the $available\_wp\_list$. Then we take advantage of the fact that the external solver can receive information from the task planner with the use of numeric fluents and define for that purpose the variables `(occupied ?l)` and `(unoccupied ?l)` where `l` is the occupied or unoccupied location, respectively. Every time the task planner expands an `assemble` or `load` action, one of the $at\ start$ effects of each of these actions is to change the value of the variables `occupied` and `unoccupied`, respectively, to 1.

Inside the external solver function there is a loop that goes through all the numeric fluents received as the state to check which ones have a value bigger than zero (every numeric fluent is initialized as zero so the ones which are zero did not suffer any changes). Since an action was expanded, at least one direct variable changed, in this case either `(occupied ?l)` or `(unoccupied ?l)` changed to 1. In the MPTP approach there is a variable called $region\_mapping$ which is a map from all the locations to their respective list of waypoints. In case there was an `assemble` action expanded, this triggers the code to loop through the list of waypoints from $region\_mapping$ associated with location `l` to remove all of them from the $available\_wp\_list$. On the other hand, if the task planner expanded a `load` action, this invokes the code to loop through the waypoints stored in the $region\_mapping$ variable associated to location `l` and add all of them back to the available waypoints list.

Whenever the `(triggered ?from ?to)` numeric fluent changes to 1 after a `move_robot` action has been expanded, the external solver will execute the motion planner code which expands the waypoints in order to find all possible paths between the $from$ and to location variables sent by the task planner and returning an external cost (which corresponds to the `extern` variable) of the path with the smallest cost. There is a loop inside the motion planner that starts with the $from$ location, gets all of its child nodes, i.e. all the roadmap's poses that are directly connected to it according to the list of edges from the $edge.txt$ file, and iterates through all of them, expanding all the possible waypoints until reaching the goal location. In order to incorporate our approach into the motion planner we add a step that verifies if a child node is present in the list of available waypoints and only expand it in case it is. If it is not on the list it means that that waypoint has a module placed on it and the robot cannot use that point as a part of its

path. Since the list is constantly updated while the planner is running, the motion plan can be done with updated information on the available paths. Algorithm 1 shows a pseudocode of the motion planner behaviour with our approach incorporated, using the euclidean distance as the motion cost.

```
Data: Roadmap (sampled poses and edges), start:
      starting location, goal: goal location
Result: extern
cost ← 0
wp_list ← list of currently available roadmap poses
to_expand ← start
expanded ← empty
while to_expand is not empty do
    current_node ← first element of to_expand
    /* All waypoints that form edges
       with current node                    */
    child_nodes ← Find children of current_node
    expanded ← add current_node
    foreach child not in expanded do
        if child exists in wp_list then
            to_expand ← child
            /* Calculate euclidean
               distance between current_node
               and child                    */
            d ← sqrt((x_curr − x_child)^2 + (y_curr − y_child)^2)
            cost_curr_node ← cost_curr_node + d
            if child is goal then
                Store this path's cost
                Break out of for loop
            end
        end
    end
    to_expand ← remove current_node
end
extern ← Choose smallest path cost
Print motion path into output file
```

Algorithm 1: Motion planner with waypoint removal

There are situations that may require some additional steps due to the way the planner and the external solver work. One situation is when there is no available path between the start and the goal locations, in other words when the path is blocked by modules. In this case we would want to inform the task planner that there is no available path between those two locations. However, POPF-TIF is only prepared for the external solver to return the indirect variables (V^{ind}) with numerical values in order to update the V^{dir} variables internally. For this reason, in our approach, whenever the motion planner is unable to find a path because it is blocked and there are no available roadmap points between the two locations, it returns a very high motion cost, for example of 10000, so that the task planner will not choose that specific `move_robot` action to be included in the output plan.

Another situation to take into account is that whenever the goal waypoint is not available, instead of returning a high cost we just temporarily add it and expand it. This is because the task planner has the ability to know that the action cannot be done since it is able to use the predicate `(clear ?l)` to know that the goal location is actually not clear and has a module on it. Because the task planner will exclude this action, the external solver does not need to.

V. Results and Discussion

A. Motion Plan: Using Task Planner vs Using Motion Planner

To test the difference between using only the task planner (and getting a motion path by changing the way the domain is defined) and using the task planner along with a motion planner with the domain definition described in section III-A we made a simple change in the domain file: we added the precondition `(at start (adjacent ?from ?to))` to the `move_robot` action, which forces the robot to only move between adjacent locations. This way the solution plan outputted by the task planner will have a description of the locations where the robot has to walk through in order to get from one location to another. When the external solver is being used, the `move_robot` action's cost is the cost returned by the motion planner. Since we are only using the task planner in this test we changed this cost to a fixed value of 1.

The problem that was defined for this test is shown in figure 4 where the blue squares represent the initial positions of the three modules (locations $c0$, $c1$ and $b7$, respectively) and the red circle represents the mobile robot (with initial position $a0$). The goal is to move $m2$ to its goal position which is location $b6$. This problem requires the robot to move close to module 2 and then move once again to take the module to its goal position. For this test we ran the task planner with two different domains: the one described in section III-A and the same one but with both changes mentioned above.

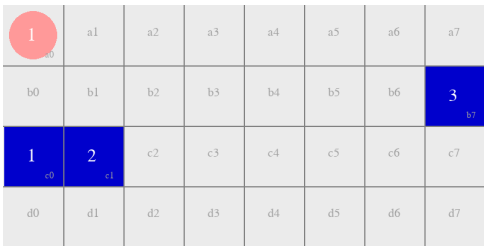


Fig. 4. Visual representation of the initial state of a problem in a 4×8 map with three modules and one mobile robot where the blue squares represent the module positions and the red circle represents the initial robot position

The output plan for the original domain in figure 5(a) is composed by four actions and the total plan cost is 4. There is a move action for the robot to go from region $a0$ to $b1$, when in $b1$ the robot can execute a load action and load module $m2$ which is within the robot's reach. Then another `move_robot` action is executed from $b1$ to $a6$ where the robot can then unload module $m2$ at its goal location $b6$ assembling it to module $m3$ which is in the adjacent location $b7$. By running the task planner alone with this domain we cannot have an idea of the path that the robot has to take

in each `move_robot` action. However, by running the task planner using the modified domain, as figure 5(b) shows, the output plan discriminates all the locations that the robot goes through and thus having a motion plan. As the plan indicates, to get from $a0$ to $c2$ the robot goes through locations $a1$, $a2$ and $b2$ and in the second `move_robot` action the robot starts at its current position $c2$ and it goes through locations $c3$, $c4$, $b4$ to finally reach $c5$. The total plan cost is 10.

```
; States evaluated: 12
; Cost: 4.000
; Time 0.13
0.000: (move_robot r1 a0 b1)  [20.000]
20.001: (load r1 m2 b1 c1)  [5.000]
20.002: (move_robot r1 b1 a6)  [20.000]
40.003: (assemble r1 m2 m3 a6 b6 b7)  [10.000]
```

(a)

```
; States evaluated: 148
; Cost: 10.000
; Time 0.35
0.000: (move_robot r1 a0 a1)  [20.000]
20.001: (move_robot r1 a1 a2)  [20.000]
40.002: (move_robot r1 a2 b2)  [20.000]
60.003: (move_robot r1 b2 c2)  [20.000]
80.004: (load r1 m2 c2 c1)  [5.000]
80.005: (move_robot r1 c2 c3)  [20.000]
100.006: (move_robot r1 c3 c4)  [20.000]
120.007: (move_robot r1 c4 b4)  [20.000]
140.008: (move_robot r1 b4 b5)  [20.000]
160.009: (assemble r1 m2 m3 b5 b6 b7)  [10.000]
```

(b)

Fig. 5. POPF-TIF output after running the problem defined in figure 4 with: (a) - the domain from section III-A; and (b) - the modified domain that forces robot movement between adjacent locations

Comparing both results we can see that using the domain definition to force this kind of motion plan implies a higher cost. In problems that require a larger amount of move actions the cost would increase exponentially, proving that making the task planner do a motion plan is not an efficient approach compared to using a domain that implies less actions and combining it with an external motion planner.

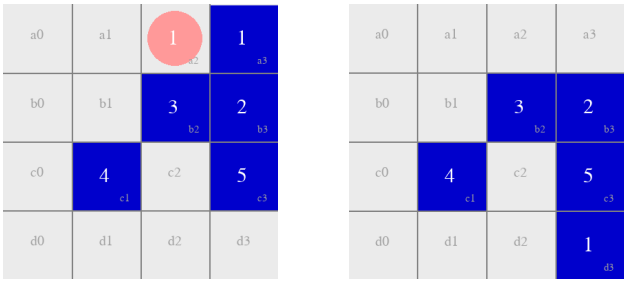
B. Motion Plan With Waypoint Removal

In order to see the difference between the motion plan produced by the original MPTP approach and the one produced by MPTP with waypoints update we define the problem in figure 6 where a robot needs to go around some modules in order to get to its goal location. Figure 6(a) shows the initial configuration of the five modules and the initial robot position which is $a2$. The goal configuration is shown in figure 6(b) where the only module that changed position was $m1$ which started in location $a3$ and whose goal location is $d3$.

After running POPF-TIF with both approaches, both output plans were the same following sequence of actions:

1) `(load r1 m1 a2 a3)`
2) `(move_robot r1 a2 d2)`
3) `(assemble r1 m1 m5 d2 d3 c3)`

Both plans include one `load` action where the robot loads module $m1$, followed by one `move_robot` action where the robot goes from $a2$ to $d2$ and then an `assemble` action



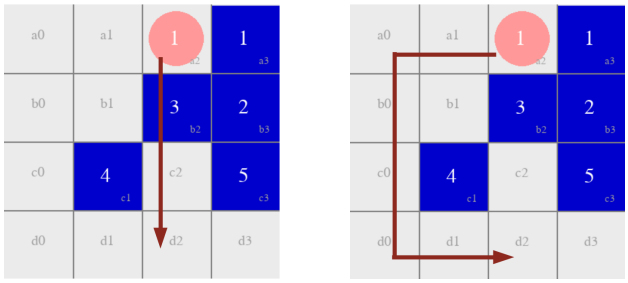
(a) Initial state **(b)** Goal state

Fig. 6. Visual representation of a problem in a 4×4 map with five modules and one mobile robot where the blue squares represent the module positions and the red circle represents the initial robot position

where the robot assembles module $m1$ to $m5$. The difference between the two plans is in the motion path and consequently in the total plan cost. We consider a cost of 1 for both the `load` and `assemble` actions. The motion planner cost is the calculated euclidean distance between all the waypoints included in the robot's path from one location to another. In this case we are using a map that has one waypoint for every location which corresponds to its center coordinates. For this reason, the motion cost between every two adjacent locations is going to be 1. The labels of the roadmap points follow the same ordering logic of figure 3.

The motion paths outputted by the *output_motion.txt* file for both approaches is shown in figure 7. For the first approach the cost was 5. This corresponds to a cost of 1 from the `load` action plus 1 from the `assemble` action and a remaining cost of 3 for the `move_robot` action. The output motion path that could be seen by the *output_motion.txt* file was the following sequence of waypoints: $wp2 \rightarrow wp6 \rightarrow wp10 \rightarrow wp14$. This motion path is highlighted in figure 7(a). For the second approach with waypoints removal the total plan cost was 9. This corresponds to the sum of a cost of 1 from the `load` action with another cost of 1 from the `assemble` action with a remaining cost of 7 for the `move_robot` action. This move action has a higher cost than the previous approach since the waypoints from all the locations that contain a module were removed. This includes waypoints $wp6$, $wp7$, $wp9$ and $wp11$ which the robot is not able to use for its path. For this reason, the outputted motion plan was a sequence of roadmap points that go around the modules as follows: $wp2 \rightarrow wp1 \rightarrow wp0 \rightarrow wp4 \rightarrow wp8 \rightarrow wp12 \rightarrow wp13 \rightarrow wp14$. A visual representation of this motion path can be seen in figure 7(b).

Comparing both results we conclude that using the roadmap points update approach is beneficial in situations like the one described in this section where there are obstacles that are unknown to the task planner due to the way the domain is defined (which is a domain that implies less actions and whose advantages were explained in section V-A). Even though results from figure 7(b) imply a higher cost than the results from figure 7(a), the motion plan is better and more accurate since the one in figure 7(a) would fail at the robot's execution phase due to physical motion constraints that were not taken



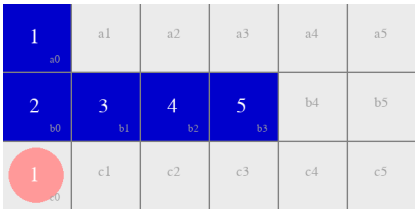
(a) Original MPTP **(b)** With waypoint removal

Fig. 7. Motion plan, represented by the red arrows, outputted by MPTP and by MPTP with the waypoint removal approach for the problem in figure 6

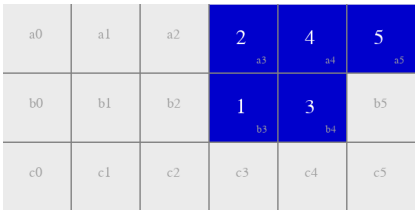
into account during the planning phase.

C. Columbus Lab Scenario

This scenario was created to show the difference between two outputted task plans when doing task-motion planning separately compared to a mixed approach as is the case with MPTP but with our roadmap points update improvement and it consists of the domain described in section III-A with a map that is similar to a 2D approximation of the ISS Columbus lab. The map used in this test can be seen in figure 2 and uses the location's names and the exact roadmap points pictured in figure 3, which means that each location contains a roadmap point that corresponds to its center coordinates. The problem that was defined for this test is depicted in figure 8 where there is a mobile robot and five modules that are to be all moved to a different configuration.



(a) Initial state



(b) Goal state

Fig. 8. Visual representation of the Columbus lab corridor problem in a 3×6 map with five modules and one mobile robot where the blue squares represent the module positions and the red circle represents the initial robot position

After running this problem using only the task planner POPF-TIF we get the output solution plan detailed in table I. This is a valid plan and all `move_robot` actions have at least one possible path with no modules blocking it, which means that if we were to do motion planning after this task plan it would be possible to find a valid motion plan with no modules blocking the way. We proceeded to manually build the motion plan with the smallest cost possible by following the given task plan and taking into account all the module positions upon the execution of each `load` and `assemble` action. We then calculated the euclidean distances of all the paths that make up the motion plan in order to know the total plan cost of solving this problem using task and motion planning separately. The cost of each action is shown in table I in front of the respective actions as well as the total plan cost at the bottom of the table which was 55 for this approach.

TABLE I
TASK PLAN FOR PROBLEM IN FIGURE 8 USING THE TASK PLANNER (POPF-TIF) AND THE COST ASSOCIATED WITH EACH ACTION

Actions	Cost
`0.000: (load r1 m2 c0 b0) [5.000]`	1
`0.001: (move_robot r1 c0 a4) [20.000]`	6
`20.002: (move_robot r1 a4 a2) [20.000]`	2
`40.003: (assemble r1 m2 m5 a2 a3 b3) [10.000]`	1
`50.004: (load r1 m4 a2 b2) [5.000]`	1
`50.005: (move_robot r1 a2 b4) [20.000]`	5
`70.006: (assemble r1 m4 m2 b4 a4 a3) [10.000]`	1
`70.007: (move_robot r1 b4 b5) [20.000]`	1
`90.008: (move_robot r1 b5 a1) [20.000]`	7
`110.009: (load r1 m1 a1 a0) [5.000]`	1
`110.010: (move_robot r1 a1 b2) [20.000]`	2
`130.011: (assemble r1 m1 m2 b2 a2 a3) [10.000]`	1
`140.012: (load r1 m3 b2 b1) [5.000]`	1
`140.013: (move_robot r1 b2 b5) [20.000]`	5
`160.014: (assemble r1 m3 m4 b5 b4 a4) [10.000]`	1
`160.015: (move_robot r1 b5 b2) [20.000]`	5
`180.016: (load r1 m5 b2 b3) [5.000]`	1
`180.017: (move_robot r1 b2 b5) [20.000]`	5
`200.018: (assemble r1 m5 m4 b5 a5 a4) [10.000]`	1
`200.019: (move_robot r1 b5 b2) [20.000]`	5
`220.020: (load r1 m1 b2 a2) [5.000]`	1
`225.021: (assemble r1 m1 m2 b2 b3 a3) [10.000]`	1
TOTAL:	55

Then we ran this problem using MPTP along with the waypoint removal approach and we obtained the solution task plan detailed in table II which also shows the cost associated with each action, being the `move_robot` action cost the one calculated by the external solver. This test's motion plan was outputted in the *output_motion.txt* and the total solution plan cost for this approach was 41.

Comparing both results we can observe that the actions as well as the sequence of actions differs from the first approach to the other. This is because the second approach is aware of the motion constraints as the robot loads and assembles modules into different positions of the map and with the incorporation of the motion planner along with that information, the task planner is able to choose the actions in a more informed way, so it will switch the order in which the robot moves modules in order to avoid producing motion constraints that will imply a higher cost when doing the motion plan, which is what happened in the case of the first approach. From this comparison we can conclude that an informed mixed task-motion planning approach allows the motion planner to be aware of the physical constraints imposed by the task planner's expanded actions while planning, which will help produce

TABLE II
TASK PLAN OBTAINED BY RUNNING THE PROBLEM FROM FIGURE 8 USING A MIXED TASK-MOTION PLANNING APPROACH (MPTP) WITH WAYPOINT REMOVAL AND THE COST ASSOCIATED WITH EACH ACTION

Actions	Cost
0.000: (load r1 m2 c0 b0) [5.000]	1
0.001: (move_robot r1 c0 a4) [20.000]	6
20.002: (move_robot r1 a4 a2) [20.000]	2
40.003: (assemble r1 m2 m5 a2 a3 b3) [10.000]	1
50.004: (load r1 m4 a2 b2) [5.000]	1
50.005: (move_robot r1 a2 b4) [20.000]	5
70.006: (assemble r1 m4 m2 b4 a4 a3) [10.000]	1
70.007: (move_robot r1 b4 b5) [20.000]	1
90.008: (move_robot r1 b5 b2) [20.000]	5
110.009: (load r1 m5 b2 b3) [5.000]	1
110.010: (move_robot r1 b2 b5) [20.000]	3
130.011: (assemble r1 m5 m4 b5 a5 a4) [10.000]	1
130.012: (move_robot r1 b5 b2) [20.000]	3
150.013: (load r1 m3 b2 b1) [5.000]	1
150.014: (move_robot r1 b2 b3) [20.000]	1
170.015: (assemble r1 m3 m4 b3 b4 a4) [10.000]	1
170.016: (move_robot r1 b3 b2) [20.000]	1
190.017: (move_robot r1 b2 a1) [20.000]	2
210.018: (load r1 m1 a1 a0) [5.000]	1
210.019: (move_robot r1 a1 b2) [20.000]	2
230.020: (assemble r1 m1 m2 b2 b3 a3) [10.000]	1
TOTAL:	41

a solution plan with a smaller cost and consequently with higher quality. We can also observe that the task plan, i.e. the discrete actions that are executed as well as their order, may change when using this mixed informed approach in order to accommodate a feasible and smaller cost motion plan.

VI. CONCLUSIONS

Planning in domains like the space assembly one where a mobile robot navigates through the environment to move and assemble parts can be challenging since it involves task-motion planning. We have seen that it is possible to define the domain in a way that we can obtain some sort of motion plan by only using the task planner if we force the robot to only move between adjacent locations. We can even divide the map into smaller areas that we would define as locations in order to have a more accurate motion plan. However, by performing the tests described in section V-A we can conclude that this is not a very efficient way of performing task and motion planning combined since this implies a very high cost which only gets higher for problems that involve more robot movement actions. It is also a very time consuming approach which might make it difficult to find a solution plan within a reasonable amount of time.

We introduced an improvement into the MPTP approach that enabled the external solver to inform the motion planner of the availability of each roadmap point updating this information according to the actions expanded by the task planner while it is working to find a solution. After running some tests we were able to show that using this informed task-motion planning approach was highly beneficial in the space assembly domain in comparison to running the task planner and then the motion planner separately. This is due to the fact that the mixed TMP approach is able to produce a more realistic task plan since it takes some motion constraints into account. By doing task and motion planning separately the task planner can in some situations produce a plan that may be unfeasible due to motion constraints or it can also be a task plan that implies a higher motion cost than other possible plans.

We defined a problem as the Columbus Lab scenario where we made a 2D approximation of the ISS European lab and defined a problem with a mobile robot and some modules to assemble into a goal configuration. With this example we were able to test both TMP approaches and conclude that the mixed informed TMP approach produced better results as some physical constraints imposed by the task planner's expanded actions were taken into account during planning, which led to a sequence of high-level task actions that produced a lower cost motion plan and an overall lower total cost for the task-motion plan. This shows that using a mixed TMP approach in scenarios that involve robot navigation is very beneficial in terms of task-motion plan quality.

REFERENCES

[1] C. Jewison, D. Sternberg, B. McCarthy, D. W. Miller, and A. Saenz-Otero, "Definition and testing of an architectural tradespace for on-orbit assemblers and servicers," 2014.

[2] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson *et al.*, "Pddl— the planning domain definition language," Technical Report, Tech. Rep., 1998.

[3] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[4] G. Della Penna, D. Magazzeni, F. Mercorio, and B. Intrigila, "Upmurphi: A tool for universal planning on pddl+ problems," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 19, no. 1, 2009.

[5] D. Bryce, S. Gao, D. Musliner, and R. Goldman, "Smt-based nonlinear pddl+ planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[6] E. Ábrahám and G. Kremer, "Smt solving for arithmetic theories: Theory and tool support," in *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2017, pp. 1–8.

[7] M. Cashmore, M. Fox, D. Long, and D. Magazzeni, "A compilation of the full pddl+ language into smt," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 26, no. 1, 2016.

[8] F. Leofante, E. Ábrahám, T. Niemueller, G. Lakemeyer, and A. Tacchella, "Integrated synthesis and execution of optimal plans for multi-robot systems in logistics," *Information Systems Frontiers*, vol. 21, no. 1, pp. 87–107, 2019.

[9] A. Coles, A. Coles, M. Fox, and D. Long, "Forward-chaining partial-order planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 20, no. 1, 2010.

[10] C. Piacentini, M. Fox, and D. Long, "Planning with numeric timed initial fluents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[11] S. Bernardini, M. Fox, D. Long, and C. Piacentini, "Boosting search guidance in problems with semantic attachments," in *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

[12] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach." in *Robotics: Science and systems*, vol. 12. Ann Arbor, MI, USA, 2016, p. 00052.

[13] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.

[14] A. Thomas, F. Mastrogiovanni, and M. Baglietto, "Mptp: Motion-planning-aware task planning for navigation in belief space," *Robotics and Autonomous Systems*, vol. 141, p. 103786, 2021.