Task-Consistent Path Planning for Mobile 3D Printing

Julius Sustarevas[1], Dimitrios Kanoulas[1], Simon Julier[1]

***Abstract*— In this paper, we explore the problem of task-consistent path planning for printing-in-motion via Mobile Manipulators (MM). MM offer a potentially unlimited planar workspace and flexibility for print operations. However, most existing methods have only mobility to relocate an arm which then prints while stationary. In this paper we present a new fully autonomous path planning approach for mobile material deposition. We use a modified version of Rapidly-exploring Random Tree Star (RRT*) algorithm, which is informed by a constrained Inverse Reachability Map (IRM) to ensure task consistency. Collision avoidance and end-effector reachability are respected in our approach. Our method also detects when a print path cannot be completed in a single execution. In this case it will decompose the path into several segments and reposition the base accordingly.**

I. Introduction

Construction is one of the most important human activities, yet it remains a largely labour-intensive process. It is dangerous [1], [2] and can be very wasteful [3]. As a result, there is a rising interest in the use of robotic Construction Automation Systems (CAS) to aid low-skilled activities, reduce accidents and reinvigorate stagnating productivity [4] in the sector. It could also help in satisfying demands for new housing, boosted by the rapid growth of urbanisation [5]. It could complement the shrinking workforce of our ageing populations [6], [7].

3D printing (or material deposition) has proven to be a cost effective [8] construction method, and several types of printing CAS have been developed [9]–[11]. However, to date only stationary gantry or crane-type systems [12] have achieved commercial success [13], [14].

In this paper, we focus on the use of Mobile Manipulators (MM) for 3D-Printing in construction. An MM consists of an agile, articulated arm that is mounted on a mobile base which can traverse the construction site (see Fig. 1). MMs have several advantages over gantry systems. They have a potentially unlimited workspace. They are easy to deploy. The agility of the high Degree of Freedom (DoF) base-arm system means that they can reach into difficult or confined spaces.

Two paradigms for MMs have been proposed: *printing-while-stationary*, and *printing-in-motion*. In a printing-while-stationary CAS [15]–[18], the robot moves itself to a number of different locations. At each location it holds its base stationary and printing is carried out. Printing-in-motion generalises this concept and allows the robot to print material while the base is in motion. Although printing-while-stationary is technically easier to achieve, it has a number of limitations. To print something larger than the robot, such systems require many relocations, which might lower print quality, introduce unnecessary discretisation of the print and give rise to task-sequencing problems [19], [20].

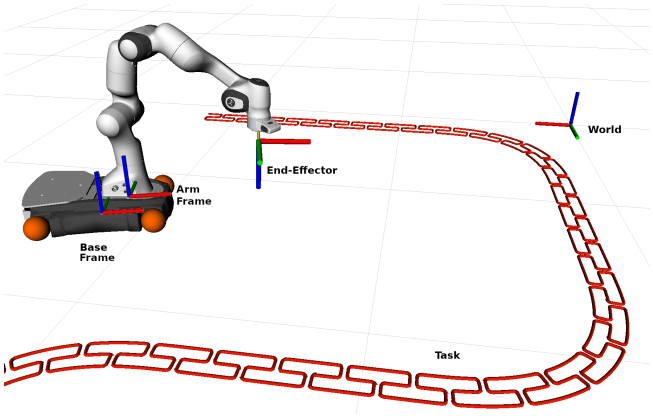


Fig. 1: Problem illustration: a tether-less Mobile Manipulator, equipped with an extruder is depositing material along a pre-described task path.

There have been only a handful of works that have explored printing-in-motion. The first is the Digital Construction Platform (DCP) [16], which consists of a high-DoF manipulator mounted on a mobile hydraulic crane. It printed a 10 m wide circular foam formwork while stationary, but only deposited a single line on the ground while moving. The second was our earlier work, Mobile Agile Printer (MAP) [21] which aimed to validate the printing-in-motion approach. Given a manually specified base path, the system showed that the end-effector error when path-tracing in-motion is only marginally greater than when the base is stationary. Another work comes from Nanyang Technological University (NTU) [22] and it is the first system to deposit concrete from a moving platform. Like MAP, it used a manually-specified base path to keep the printing task within the arm workspace. Subtracting this path from the task allowed arm motion to be planned in the local arm frame. Although these systems demonstrate the feasibility of printing-in-motion, they required manual specification of the base path and thus fall short of being fully autonomous. Our last work Youwasps [23] demonstrated autonomous clay deposition where only the print path had to be specified. However, the crude assumptions made by the path planning method only further highlights the need to explicitly explore path planning for mobile 3D printing.

In this paper, we develop an algorithm for autonomous task-consistent path planning for mobile 3D printing. Given a print path, a kinematic robot description, and constraints on manipulator orientation for material deposition, the al-

[1]Department of Computer Science, University College London, Gower Street, WC1E 6BT, UK. julius.sustarevas@cs.ucl.ac.uk

gorithm computes a base path and keeps the task inside the arm workspace. Our approach extends Rapidly-exploring Random Trees (RRTs), specifically RRT*, by using modified Inverse Reachability Maps (IRMs). The IRMs are used to bias and constrain an RRT* search to ensure task-consistent paths. The underlying graph is used to detect if the print is achievable in a single execution, or if relocation is necessary.

The structure of this paper is as follows. The next section describes and introduces the task-consistent planning problem statement. Related work is described in Sec. III. The proposed algorithm is described in Sec. IV and it is evaluated in a set of experiments reported in Sec. V. Finally, conclusions and future directions are discussed in Sec. VI.

II. Problem Statement

Consider the problem illustrated in Fig. 1. A tether-less Mobile Manipulator, equipped with a single circular-nozzle extruder, has been tasked with printing a 3D structure. The print task is defined by layered print paths. The extruder must follow these paths while depositing material. The deposition rate is assumed to be slow and constant. The extruder must be orthogonal to the printing surface, and the material must form a continuous structure. In this paper, all print surfaces are parallel to the ground plane, and so the head must be oriented to point down.

A. Specification of the 3D-Printing Task

Let W be the world-fixed frame and E be the end-effector frame. The printing task is a parameterised path ($\in \mathbb{R}^3$) in the world frame that E must follow. To capture the orientation constraint with respect to the printing surface, the task is assumed to be provided as a trajectory of homogeneous transformations ${}^{W}T(s) \in SE(3)$. The orientation constraint is defined as the z-axis of E and specified as the third column of the matrix $T(s)$. In this work, we assume that the task is layered vertically and E is always constrained to point downwards.Therefore,

$$T(s) := \begin{pmatrix} 1 & 0 & 0 & T_x(s) \\ 0 & 1 & 0 & T_y(s) \\ 0 & 0 & -1 & T_z(s) \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (1)$$

The scalar parameter $s \in [0, s_f]$ denotes the progress along the task path, e.g. the length of task path printed. The path is completed when $s = s_f$.

B. Path Planning for Mobile Material Deposition

Given a task $T(s)$, the path planner seeks to produce a continuous path $b(s) = (b_x, b_y, b_\theta) \in SE(2)$ that the base frame will follow. We formalise this as follows.

The frame fixed to the robot base is B. The robot arm is rigidly fixed to the robot base. The arm root link is the frame A. The homogeneous transformation expressing pose of A in frame B is ${}^{B}A$.

At a point s along the task, the base pose is ${}^{W}B(s)$. To ensure task consistency, the end-effector pose in the arm frame must be

$${}^{A}E(s) = \left({}^{W}B(s)\,{}^{B}A\right)^{-1}\,{}^{W}T(s). \quad (2)$$

The base pose determines the task pose in the arm frame. Therefore, this task-consistency constraint imposes the constraint that $b(s)$ must be chosen such that the end-effector trajectory ${}^{A}E(s)$ is feasible. This means that, $\forall s$, the inverse kinematics (IK) for the pose ${}^{A}E(s)$ are defined and can form a continuous solution in the arm's joint space. Note, that because the nozzle is circular, there is a symmetry about the Z-axis of the end-effector. Thus these are 5 Dof IK calls and the end-effector yaw is left unconstrained.

In addition to the task-consistency described in (2), there are several important considerations that the path planner must take into account:

a) Dynamic and Evolving Obstacles: As the material is deposited, the robot creates and reshapes obstacles in its environment. The robot must not collide with these dynamic obstacles otherwise the print will be ruined. One way to address this problem is to treat the geometry of the entire print task as a static obstacle. However, this very crude assumption leads to conservative trajectories and often renders the solution infeasible. Therefore, the planner must explicitly plan for dynamic and evolving obstacles.

b) Discontinuous Printing: Although the end-effector must trace out a continuous path, it might not be possible to complete the task in a single execution [23]. As the robot creates dynamic obstacles during the print process, the robot might not be able to reach the next point in the task without having to interrupt printing and reposition itself. The planning algorithm must offer a principled way to detect when a break in the print process when it is necessary and find a solution that incorporates it.

c) Maintaining Arm Responsiveness: A Mobile Manipulator is composed of two different systems with very different fidelity characteristics. A robotic arm has a limited workspace, but is precisely encoded and has a repeatability of the order of 0.1 mm. Therefore, when the arm is tasked to move to a pose ${}^{A}E(s)$, the arm will achieve this with little error. By contrast, the robot base can move freely over a 2D surface, but its external pose can only be estimated via tracking systems such as SLAM or GPS. The errors with such systems can be at least several centimetres. Also, as the mobile base moves throughout the construction site, it can experience disturbances including those due to uneven terrain or breakaway friction. Although implementing disturbance rejection is out of scope of this paper, it introduces two requirements for the planner.Firstly, the algorithm should minimise the base path $b(s)$ to avoid unnecessary motion. Secondly, $b(s)$ should be chosen to keep the task in a highly manipulable or reachable region of the arm. For example, paths which require the arm to be fully extended would not allow some corrective motions.

To address these needs, we developed a novel planning algorithm that uses RRT* as a framework. RRT* was chosen as it is an optimal algorithm [24] and allows to minimise the base motion. To address the requirement of arm responsiveness requires detailed discussion on robot workspace and manipulability. These are reviewed next.

III. Previous Work on Task-Consistent Path Planning

A. Planning and Manipulability

One of the earliest works in this area was by Nagatani et al. [25] who developed a mobile manipulator-based robot to write on a wall. This required the solution of a task-consistent planning problem. Two key insights were used to address this. First, they treated the problems of planning the base and the arm separately to simplify the planning task. Second, they argued that the arm planner should plan trajectories in manipulable regions of the configuration space. They used Yoshikawa's definition of the manipulability ellipsoid, which is a representation of a set of possible velocities the end-effector can take [26]. If the Jacobian of a specific joint configuration is J, $m = \sqrt{det(JJ^\intercal)}$ can be interpreted as a distance from singularity configurations [27]. As a result, the robot arm will be able to reject disturbances quickly. Using this intuition, they extended an RRT algorithm to validate graph edges by checking manipulability as well as collisions.

Oriolo et al. [28] showed that injecting a bias into the RRT sampling step makes it possible to add additional control to a solution. Therefore, by providing suitable judicial extensions to RRTs, it is possible to ensure task consistency, manipulability and other constraints in the path.

Although manipulability provides a measure of robustness to a solution, Zacharias et al. [27] identified a number of difficulties based on the fact that the Jacobian is used. The manipulability measure is only computed locally, is unable to handle constraints, and mixes spatial and angular units making the computed solution difficult to interpret. To overcome these limitations, the Reachability Map (RM) was developed.

B. Reachability and Inverse Reachability

A Reachability Map (RM) provides a measure, for each point in the arm's workspace, of how hard it is for the arm to place an end-effector at that point. Following Zacharias' methodology, the RM is built as follows. The arm workspace is discretised into a set of voxels. Each voxel is assigned a reachability index.

The reachability index for the jth voxel is calculated by fitting a sphere inside the voxel and sampling a set of N random end-effector poses on its surface. The z-axis of these poses is aligned with the tangential normal vector of the sphere at each point. For each pose, the Inverse Kinematics (IK) is computed. Let R_j store the total number of IK solutions that exist. The Reachability Index (RI) for the jth cell is then given by:

$$RI_j = 100\frac{R_j}{N}. \tag{3}$$

The interpretation is as follow [27]. When $RI_j = 0$, no IK solutions exist and the voxel is not reachable. When $RI_j = 100$ then many IK solutions exist and the voxel is easy to reach. Intermediate values provide a measure of the degree of difficultly of placing an end-effector in that voxel.

Subsequently, Zacharias et al. [29] showed how the RM could be used to optimise the placement of a static base for an object manipulation task. They computed the RM for a robot. They cross-correlated it with the set of end-effector tasks the robot had to undertake. This resulted in the reachable task position in the robot local frame. Then inverting this position for a given task in world frame could suggest a base pose such that the task is in a highly reachable region of the arm. This work demonstrated that, when determining the robot base pose, the RM can be used.

To avoid expensive correlation operations and generalise robot base pose evaluation, Vahrenkamp et al. developed the Inverse Reachability Map (IRM) [30]. While the RM computes the reachability of a point in the arm frame A, the IRM aims to evaluate base poses in the task-frame (e.g. the required end-effector pose to pick up an object). This way IRM allows derivation of viable base poses given a task pose. The construction of the IRM is described in the next section.

IV. RRT* for Task-Consistent Path Planning

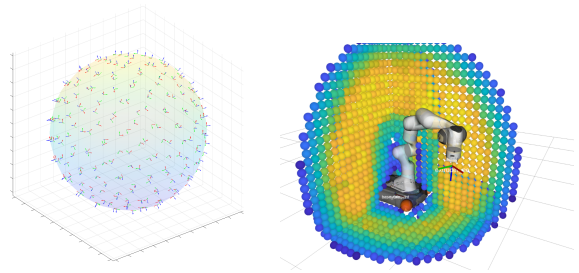
In this section, we describe our approach for task-consistent path planning. This consists of two phases. The first is the off-line construction of the Inverse Reachability Map (IRM), creating a cached data structure. The second is to carry out online planning using the cached IRMs and a modified RRT* planning algorithm.

A. Inverse Reachability Map Construction

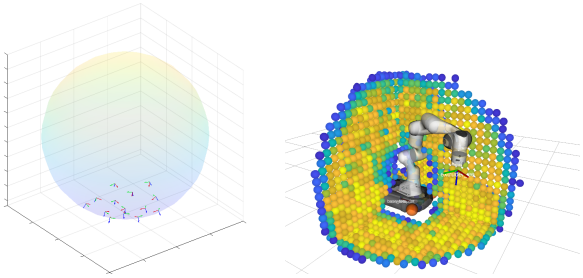
We construct a task-specific Inverse Reachability Map (IRM), which captures the reachability quality of a particular base placement ${}^W b(s)$ for a printing task ${}^W T(s)$. We start by constructing a Reachability Map (RM) in the arm frame A. We construct the RM by extending Zacharias et al.'s sampling-based method [27] to account for the print head orientation constraint. This constraint is specified as $T_{j=3}$, the third column of the matrix $T(s)$ in (1) and in general could be a function of the task process variable s. Since in this paper we only consider the case when the end-effector needs to be oriented straight down, $T_{j=3} = (0,0,-1,0)^\intercal$.

An orientation-constrained RM can then be constructed as follows. The arm workspace is discretised into cubic voxels (we use side length 5 cm) and N (we use $N = 200$) test poses are sampled from each as described in Sec. III. We then discard poses that are not close to the orientation constraint. We use the condition $(\text{acos}(|P_{3,3}|) < 0.5) \wedge (P_{3,3} < 0)$, where P is the transformation matrix of a test pose. The number of existent IKs R_j then defines the reachability intex of cell j as in (3). This way the RM is indicative of point in robot arm space where the constrained task will be reachable. Fig. 2 shows the reachability maps computed using the original (unconstrained) and modified (constrained) algorithms.

Enforcing the orientation constraint on sample poses reduces the number poses considered, and in turn, increases sensitivity of RI to non-existent IKs. It also reduces the spatial distribution of the poses inside the voxel, see Fig. 2. We addressed this by applying an equally weighted $3 \times 3 \times 3$ smoothing kernel to the voxel grid to produce the



(a) Original reachability map.



(b) Constrained reachability map.

Fig. 2: Comparison of the original and constrained reachability maps. RI values, normalised over their individual maps, shown in blue (low) to yellow (high). Left: the sphere in one voxel showing a set of sampled poses. Right: The resulting Reachability Map.

smoothed Reachability Map in Fig.3. The smoothing does not take away from the metric as it effectively expands sampling volume and distribution. Implementation-wise, the RM is a three-dimensional array describing the function $(x,y,z) \rightarrow RI(x,y,z)$, where x,y,z are the centre coordinates of a voxel. We used the smoothed RM to compute the

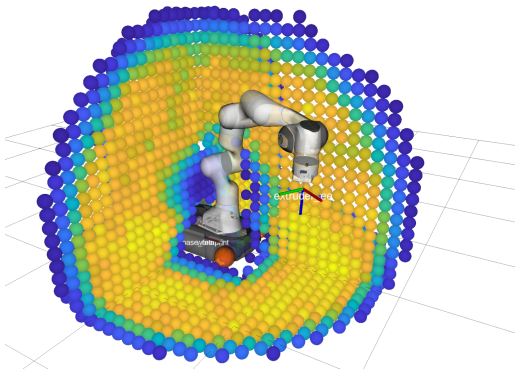


Fig. 3: Smoothed orientation-constrained reachability map.

Inverse Reachability Map as follows. The domain of possible base poses is a $[-r,r]^2 \times [-\pi,\pi]$, where r is arm reach. The domain of task heights is $[0,r]$. We discretise these domains into a set of voxels using equally spaced points. We use 5 cm increments for Cartesian dimensions and 5° for rotations (determined experimentally). The voxel centre coordinates for the ith voxel are $(x_i,y_i,\theta_i,z_i) = (b_i,z_i)$. The first three represent a possible base pose b_i. The last is the task height. An Inverse Reachability Index (IRI) is assigned to each voxel:

$$IRI(b_i,z_i) := RI(R(\theta_i)^{-1}(x_i,y_i)^\intercal,z_i), \tag{4}$$

where $R(\theta_i)$ is a planar rotation matrix and so RI is appropriately indexed by an x,y,z tuple. It is not shown in (4), but indexing into the RM must be done in the arm-frame, so the indexing is offset by ${}^BA$. Implementation-wise, the resulting IRM is a four-dimensional array that stores IRI values associated with discretised base poses and task heights i.e. $IRM := (b_i,z_i) \rightarrow IRI(b_i,z_i)$.

The computation time for the RM and the IRM strongly depends on the voxel resolution. To simplify our implementation, we used serialised service request calls using *MoveIt!* [31] and *Trac-IK* [32]. As such, the computation took about 11 h. A more optimised implementation for computing non-task-specific RM and IRM is available [33].

In the next section we show how the IRM is used to validate a base pose. In order to only consider base poses of high IRI value, we threshold or *prune* the IRM first. We do this by ranking all the IRM voxels by their IRI value. Then, the IRI value of the bottom portion, say 20%, of the voxels is overwritten to be zero. Portion size is determined experimentally and is elaborated on in the evaluation. This way, a voxel having a non-zero IRI value implies it is in the top 80% of available poses. If this condition is used to validate a base pose, the algorithm will guarantee a minimal IRI value throughout the path. Fig. 4 illustrates and IRM. Such pruning, effectively, discards the low value (blue) regions shown.

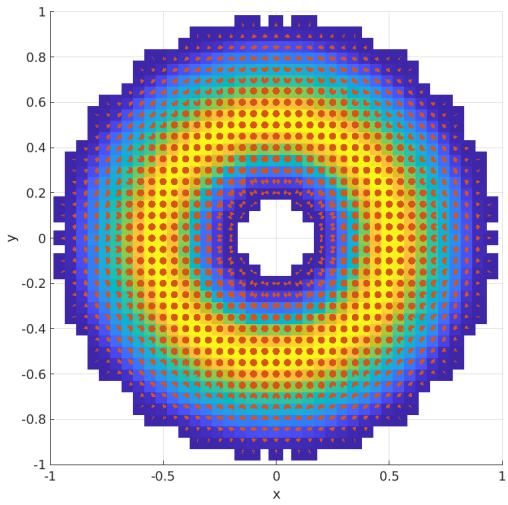


Fig. 4: Visualisation of Inverse Reachability Map. Voxels shown are *projections* of $SE(2)$ onto $\mathbb{R}^2$, task height is kept constant $z_i = 0$. The colour brightness indicates value of IRI summed over rotation dimension θ of $SE(2)$. Non-empty voxels along θ dimension are shown as red lines drawn from centers of projected voxels.

B. Task-Consistent RRT*

In this section we use the RRT* algorithm as a framework to solve the task-consistent path planning problem. RRT* grows a graph using a parent-child relationship between randomly sampled points in a search space and maintains the accumulative path costs. For detailed description of RRT* please see [24]. We adopt RRT* by firstly defining a search space for the base poses $b(s)$. Then, extend it by redefining

how it samples and validates graph vertices and edges.

Eq. (2) shows that $T(s)$ constrains $b(s)$ to path-dependent reachable regions in $SE(2)$. To capture this variation, the search space is extended with the process variable $SE(2) \times [0, s_f]$. A graph vertex in this space is denoted as $q := (b, s) = (x, y, \theta, s)$. Moving backwards along s is not allowed and so, the following metric on the search space is defined:

$$d(q_1, q_2) = \begin{cases} \|q_1 \ominus q_2\|_2, \text{if } s_2 > s_1 \\ \infty, \text{if } s_2 \leq s_1. \end{cases} \quad (5)$$

where $\ominus$ means that angle difference is used for θ. We also define two functions using the cached IRM. These are used for sampling and validation during path planning. Firstly, recall that the IRM assigns IRI values to voxels whose centres are $(b_i, z_i) = (x_i, y_i, \theta_i, z_i)$, We can approximately *index* into IRM by rounding values to nearest voxel centers:

$$IRI(b, z) = IRI(\arg\min_{b_i, z_i}(\|(b_i, z_i) \ominus (b, z)\|) \quad (6)$$

We define a function *inIRM* that, given a graph vertex $q = (q_x, q_y, q_\theta, q_s)$, indexes into the IRM and evaluates if the task point $T(q_s)$ is reachable from (q_x, q_y, q_θ):

$$\begin{aligned} &inIRM(q) = \\ &IRI((q_x - T(q_s)_x, q_y - T(q_s)_y, q_\theta, T(q_s)_z)) > 0 \end{aligned} \quad (7)$$

The IRM is already thresholded as described in Sec. IV-A and so, IRI being positive implies the value was above the threshold, and q is considered suitable.

Furthermore, the IRM can be used to draw samples of valid base poses for a given $s = s^*$. First, fix z^* to be the task height $T(s^*)_z$ rounded to nearest voxels center z_i. We can then use the IRI values as a probability density function to sample from the IRM. We define $pdfIRM$ and $sampleIRM$ as such:

$$z^* = \arg\min_{z_i}(\|T(s)_z - z_i\|) \quad (8)$$

$$pdfIRM(b_i) = \frac{IRI(b_i, z^*)}{\sum_{b_i} IRI(b_i, z^*)} \quad (9)$$

$$sampleIRM(T(s^*)) = (T(s^*)_x + b_x, T(s^*)_y + b_y, b_\theta), \quad (10)$$

where $b \sim pdfIRM$. Therefore, for a given s^* we can draw samples of base poses, from the IRM placed at the task point $T(s^*)$.

In order to use the RRT* framework to solve the problem at hand, we redefine RANDCONFIG and ISVALID routines. These, as well as the RRT* framework, are presented here:

- **Line 1:** The input consists of the task $T(s)$ and a set of n starting base poses $\{q_{start}\}$. The $\{q_{start}\}$ are sampled using $sampleIRM(T(0))$.
- **Line 2:** All q_{start} are set as root nodes.
- **Line 4:** In each iteration, random configurations q_{rand} are sampled using RANDCONFIG (described further on).
- **Line 5:** The graph vertex closest to q_{rand} is found via a nearest-neighbour search using the measure in (5).
- **Line 6:** The EXTEND function uses the ISVALID routine to validate the edge between $q_{nearest}$ and q_{rand} at intervals ε_{inc} and up to a distance ε_{reach}. We use $\varepsilon_{inc} = 0.01$ and $\varepsilon_{reach} = 0.1$. If $q_{nearest}$ is reached, *isReached* is set to true. EXTEND could be used to impose motion constraints. For simplicity, in this paper we use a holonomic base and do not model kino-dynamic constraints.
- **Lines 7–9:** Routines NEAR and CHOOSEPARENT find the set $\{q_{near}\}$ of graph vertices in a region of radius ε_{near} around q_{new}. We use $\varepsilon_{near} = 0.3$ The closest vertex q_{min} is made the parent of q_{new}. The parent-child assignment is also used to calculate the accumulative cost $cost(q_{new}) = d(q_{min}, q_{new}) + cost(q_{min})$.
- **Lines 10:** The REWIRE routine changes the parent of any vertices in $\{q_{near}\}$ to q_{new} if this leads to a lower cost.
- **Lines 12:** The algorithm terminates if the q_{new} added to the tree is a goal vertex.
- **isValid:** In the standard RRT*, this only checks for collisions with the environment. We extend it to detect collisions with $T([0, q_s])$. That is, the print task printed up to q_s. Also, q is only considered valid if $inIRM(q)$ is true. This assures that the edges, which, are validated at increments ε_{inc} all satisfy the task-consistency constraint.
- **RandConfig:** It is common for state sampling to include a bias towards the goal. Here, a goal state is returned with probability β_{goal} (we use $\beta_{goal} = 0.01$). However, the 3D printing processes necessitates that the whole task is printed before reaching the goal. Therefore, we also bias the sampling towards the *task frontier*. The main graph keeps track of the configuration furthest along the printing process q_{max}. We can then use $s_{max} = q_{max,s}$ to bias the random configuration sampling and draw $q_{rand,s}$ values from a normal distribution centred at s_{max} with standard deviation s_{var}. Thus, more poses are drawn around the *task frontier*. In this paper we use $s_{var} = 0.1 s_f$. Note, it is not shown above, but RANDCONFIG samples new configurations until one is validated using ISVALID

Task-Consistent RRT*

```
1: procedure RRT* FRAMEWORK({q_start}, T(s))
2:     graph.insert(∅, {q_start})
3:     while True do
4:         (q_rand, isGoal) ← RandConfig(s_max)
5:         q_nearest ← Nearest(graph, q_rand)
6:         (q_new, isReached) ← Extend(q_nearest, q_rand)
7:         {q_near} ← Near(graph, q_new)
8:         q_min ← ChoseParent(q_new, {q_near}))
9:         graph.insert(q_min, q_new)
10:        Rewire({q_near}, q_new)
11:        if isGoal and isReached then
12:            Break
```

Redefined Routines

```
procedure ISVALID(q)
    valid ← ¬isInCollisionWithEnvironment(q)
    valid ← valid ∧ ¬isInCollisionWithTask(q, T([0, q_s]))
    valid ← valid ∧ inIRM(q)
    return valid

procedure RANDCONFIG(s_max)
    if rand() < β_goal then
        (s_rand, isGoal) ← (s_f, True)
    else
        (s_rand, isGoal) ← (max(0, min(s_f, s ~ N(s_max, s_var))), False)
    (x, y, θ) ← sampleIRM(T(s_rand))
    return q_rand = (x, y, θ, s_rand)
```

These modifications have three effects. First, collisions with the task are considered dynamically. Second, the graph is constructed only using configurations from reachable regions inside the IRM. Lastly, edges are densely validated to fall inside sufficiently high value IRM voxels.

C. Task Discontinuity Detection

Lastly, using a sampling method to grow a tree along a print path allows us to detect when the printing task cannot be accomplished in a single continuous trajectory. Analytically, this arises when, for some point in process s_I, all pairwise edges between a search graph vertex with base pose b_I (that can reach $T(s_I)$) and base poses $b_{I+\varepsilon_{inc}}$ (that can reach $T(s_I+\varepsilon_{inc})$) cannot be validated, e.g. are in a collision with obstacles or previously deposited material. Such an event will, by construction, cause our proposed *RRT* search to stall. The algorithm will grow the graph until $s_{max} = s_I$, at which point, random configurations q_{rand} will all fail to grow the tree beyond s_{max}. Therefore, the proposed algorithm counts how many iterations of the RRT* loop went by with s_{max} not changing value. If this number breaches a threshold (in this paper we use 300), an interruption is detected. Then, a set of new starting poses $q_{start,new} = sampleIRM(T(s_I+\varepsilon_{inc}))$ and a new graph path planning problem is started for the remaining task $T(s \in [I; s_f])$. The *interruption* relationship is captured by assigning q_{max} as a separately kept *interruption parent* to all configurations $q_{start,new}$.

V. Evaluation

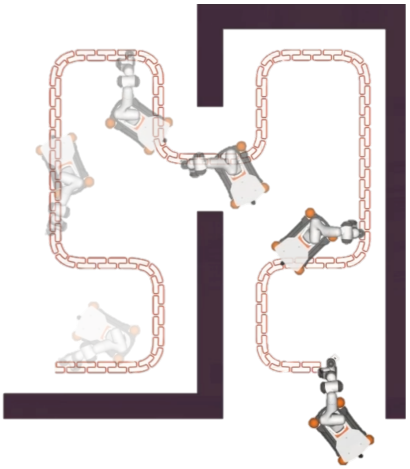
We evaluate the performance of the algorithm using a model of 7DoF Franka Panda robot arm mounted on Kuka Youbot robot base. Two print tasks are used. The first task explores how $b(s)$ is found, and how it avoids obstacles and already deposited material. We also discuss finding the joint space solution for the corresponding $E(s)$. The second task explores interruption detection when the algorithm is given an unfeasible task.

A. Obstacle and Deposited Material Avoidance

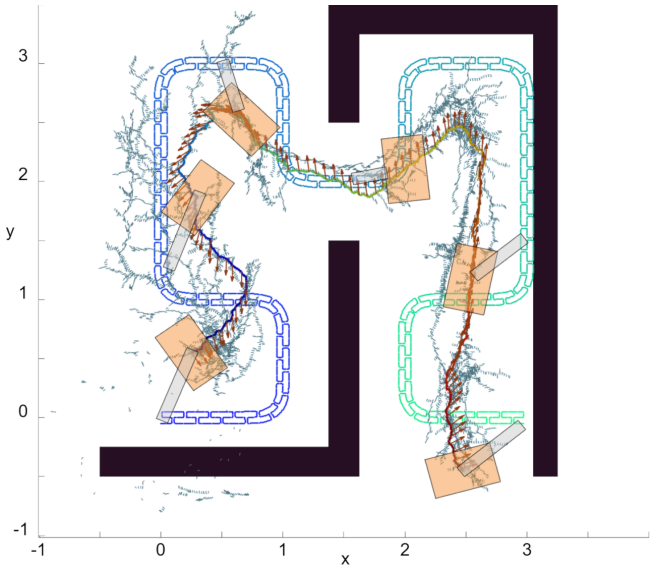
To clearly illustrate how the deposited material is avoided, we focus on a complex single layer task. This allows us to capture the shrinking free space that $b(s)$ must traverse.

Fig. 5(a) shows the task. The robot starts in the bottom left corner and prints a 47.4 m long, single layer curve. It uses the filling pattern which is often used in concrete 3D printing. The task travels close to the static obstacles (black), especially in the middle. Therefore, the simple heuristic described in Sec. II, of assuming that the whole task is a static obstacle prior to execution, would render this task infeasible. Therefore this task is an ideal test case for our planning approach.

Fig. 5(b) shows the space the RRT* explores to find a solution $b(s)$. Note that the jagged nature of the base trajectory arises because, in this initial implementation, we do not model kino-dynamic constraints. Implementing these constraints is a topic for future work. The top left corner of Fig. 5(b) shows that the tree branches do not propagate in presence of obstacles and the evolving task. The highlighted solution $b(s)$ found allows the arm to trace the task trajectory while the base is in motion. In particular, when the robot passed through the narrow gap in the wall, it has to drive with the arm oriented so it can print behind it. Rectangles showing base and arm footprint at a number of points along the print



(a) Illustration of robot executing the task.



(b) A path search illustration. Static Obstacles (Black), Print Task (Blue to Green), $b(s)$ found (Blue to Red), RRT* tree exploration, visible in blue, shown only in 2D.

Fig. 5: The first task illustrating task-consistent base path planning using extended RRT*. Video available at: `youtu.be/guyZEsBgakE`

illustrate the scale and how the base path is associated with the task.

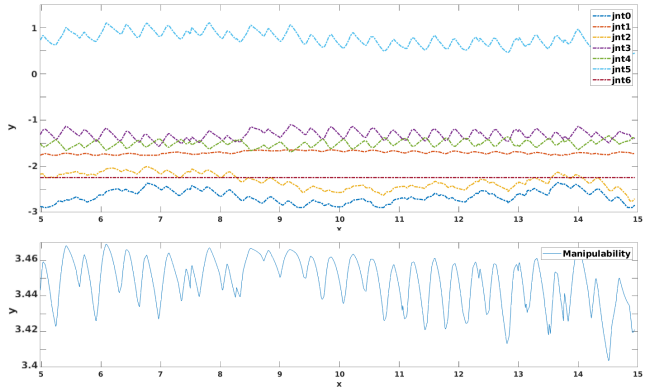
Path planning for this task was carried out 100 times using the IRMs using the method in Sec. IV-A. Several pruning values were chosen. Aggregate results for planning time, success rate, $b(s)$ length and end-effector trajectory error are presented in Table I. Here, we used `ikunc` solver of Robotics Toolbox for Matlab [34] to find the corresponding end-effector poses $E(s)$. The plan was considered successful if the combined translational and rotational *ikunc* error was less than 0.01. Data illustrates how discarding larger portions of the IRM increases the strictness of the constraint it imposes. In turn, this leads to increased planning time, but also lower error and higher success rate.

Fig. 6 shows a segment of a successful base path, task and $E(s)$, which corresponds to the print task in the local arm frame. Throughout this path, the reachability index of

Pruning (%)	Time (std.) (s)	Success (%)	$b(s)$ length (std.) (m)	Error (std.) (10^{-4}m)
None	15.9 (6.1)	71	12.2 (0.9)	2.5 (4.2)
10	16.4 (4.1)	74	12.1 (0.9)	1.9 (3.2)
20	15.9 (2.9)	76	12.2 (0.8)	2.3 (3.2)
50	23.5 (12.5)	98	12.0 (0.9)	0.5 (0.7)
70	44.5 (33.4)	100	12.5 (1)	0.2 (0.4)

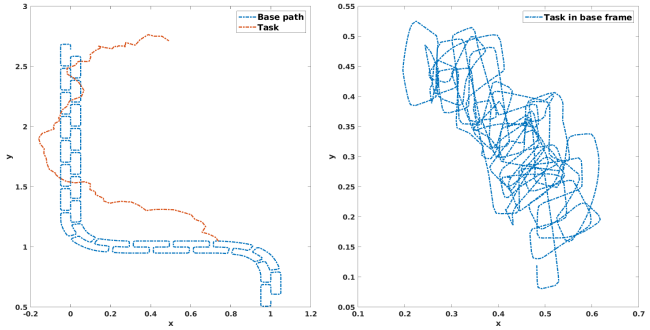
TABLE I: Planning time and success rate dependency on IRM pruning over 100 sample runs.

$E(s)$ was on average 91.8 which is in the top 10 percentile of the forward Reachability Map of the arm. It can be seen that as the base travels, the local task is situated in a small, close region that shifts as the task shifts in the arm frame. The joint-space solution reflects the pattern of the task as well as the small stutter in the base path solution. Lastly manipulability, $\sqrt{\det JJ^\intercal}$, computed from $j(s)$, is shown to tightly oscillate around a constant non-zero value, confirming that the task remains in a manipulable region. Here, we only show that arm reachability was maintained. Evaluating how this would effect disturbance rejection is left for future work.



(a) Top: Joint-space solution for this section of $E(s)$. Bottom: arm manipulability evaluated at the joint values

Fig. 6: Analysis of $E(s)$ in task and joint-space.



(a) Left: Segment of $T(s)$ and $b(s)$. Right: Corresponding $E(s)$

B. Task Discontinuity Detection

To illustrate task discontinuity, we consider the task shown in Fig. 7. A letter P is to be printed. The robot starts in the middle of the letter, proceeding along the arc and to the base. Then the next layer is elevated above the first one by 3 cm and starts at the base of the letter proceeding towards the top. The algorithm receives both layer trajectory as a single task definition. An obstacle is deliberately situated in the environment to make it impossible for the task to be executed in a single print.

The figure shows how the solution is constrained by the environment and the evolving obstacles created by the print. The base must navigate in front of the arm and above the obstacle. Then as the robot continues to print the second layer, it can successfully reach the middle of the letter. However, the task itself prevents the base from reaching the left side of the second layer. At this point, the search stalls and an interruption is detected. The base is then allowed to relocate to the other side and continue printing.

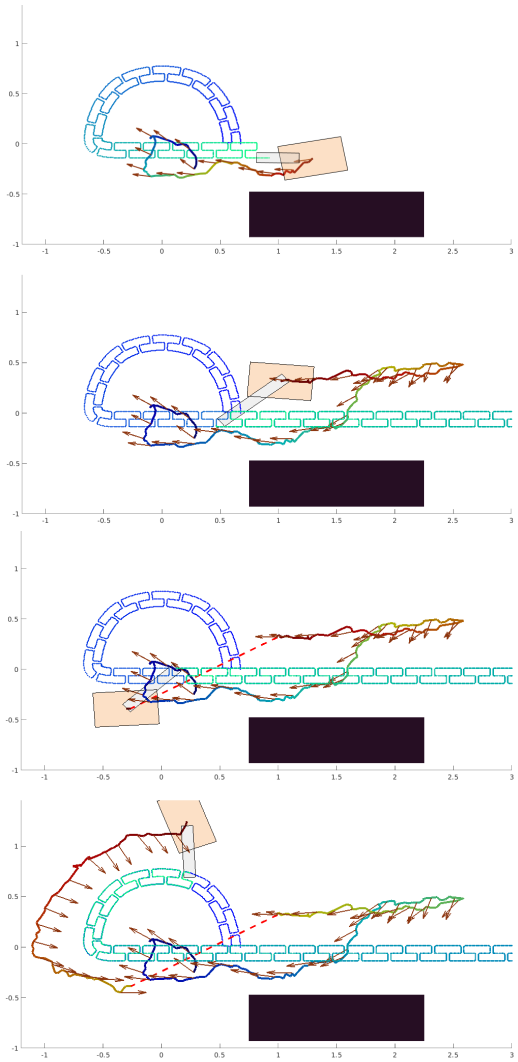


Fig. 7: Robot base path solution when task interruption is necessary. The series of pictures (top to down) show 25%, 58%, 60% and 90% task completion. Static Obstacle (Black), Task being printed (Blue to Green as oldest to newest). Base poses linked through relocation (Red dashed), RRT* tree is omitted for presentation clarity.

This 11.47 m long task takes significantly longer to plan for, average planning time of 10 attempts being at 116 s. This is because the interruption detection relies on the RRT* search stalling. If the stall threshold is decreased, an interruption can be detected more quickly. However, this is less reliable and a false positive interruption can

be detected as a small amount of stalling might happen simply by navigating a narrow path. On the other hand, throughout experimentation, the algorithm never produced a false negative. Since RRT edges must be validated and in small increments, task infeasibility always means the tree fails to progress and stalls.

VI. Conclusions and Future Work

This paper tackles the Task-Consistent Path Planning problem in mobile 3D-Printing. Our experiments show that the proposed solution, based on RRT* and Inverse Reachability Maps, is able to find base paths that facilitate material deposition along prescribed task trajectories. However, a number of limitations will be addressed in future work. The method proposed makes it likely but does not guarantee the existence of a continuous joint-space trajectory of the manipulator. This could be addressed by integrating this work with Cartesian path planner like Descartes [35] or other advanced path planners [36], [37]. Furthermore, there are examples in the literature that show that IRM computation could be made faster [33] or only light-weight features of IRM used instead [38]. Lastly, perhaps the most interesting implication of this work is the potential to use it in task decomposition. Task discontinuities, as described in this paper, could make for a natural way to segment long trajectories and allocate them to multiple robots.

References

[1] International, Labour, and Organization, "Facts on Safety at Work Report," pp. 1–2, 2005.

[2] HSE, "Construction statistics in Great Britain," 2019.

[3] Faniran and Caban, "Minimizing waste on construction project sites," *Eng. Constr. Archit. Manag.*, vol. 5, no. 2, pp. 182–188, 1998.

[4] R. Agarwal, S. Chandrasekaran, and M. Sridhar, "Imagining Construction's Digital Future," McKinsey, Tech. Rep. Exhibit 1, 2018. [Online]. Available: https://www.mckinsey.com/industries/capital-projects-and-infrastructure/our-insights/imagining-constructions-digital-future

[5] HM Government, "Construction 2025. Industrial Strategy: Government and industry in partnership," *UK Gov.*, no. July, p. 78, 2013.

[6] United Nations, "Ageing." [Online]. Available: http://www.un.org/en/sections/issues-depth/ageing/

[7] T. Bock, "The future of construction automation: Technological disruption and the upcoming ubiquity of robotics," *Autom. Constr.*, vol. 59, pp. 113–121, nov 2015.

[8] S. Lim, R. Buswell, T. Le, S. Austin, A. Gibb, and T. Thorpe, "Developments in construction-scale additive manufacturing processes," *Autom. Constr.*, vol. 21, no. 1, pp. 262–268, jan 2012.

[9] R. N. P. Van Woensel, T. Van Oirschot, M. J. H. Burgmans, M. Mohammadi, Ph D, and K. Hermans, "Printing Architecture: An Overview of Existing and Promising Additive Manufacturing Methods and Their Application in the Building Industry," *Int. J. Constr. Environ.*, vol. 9, no. 1, pp. 57–81, 2018.

[10] Y. W. D. Tay, B. Panda, S. C. Paul, N. A. Noor Mohamed, M. J. Tan, and K. F. Leong, "3D Printing Trends in Building and Construction Industry: a Review," *Virtual Phys. Prototyp.*, vol. 12, no. 3, pp. 261–276, Jul 2017.

[11] F. Bos, R. Wolfs, Z. Ahmed, and T. Salet, "Additive manufacturing of concrete in construction: potentials and challenges of 3D concrete printing," *Virtual Phys. Prototyp.*, vol. 11, no. 3, pp. 209–225, 2016.

[12] B. Khoshnevis and J. Zhang, "Extraterrestrial construction using contour crafting," in *23rd Annual International Solid Freeform Fabrication Symposium - An Additive Manufacturing Conference, SFF 2012*, 2012, pp. 250–259.

[13] Apis Cor, "We print buildings." [Online]. Available: http://apis-cor.com/

[14] D-shape, "D-Shape Portfolio." [Online]. Available: https://d-shape.com/portfolio-item/

[15] CyBe, "3D Concrete Printers," 2018. [Online]. Available: https://cybe.eu/technology/3d-printers

[16] S. J. Keating, J. C. Leland, L. Cai, and N. Oxman, "Toward site-specific and self-sufficient robotic fabrication on architectural scales," *Sci. Robot.*, vol. 2, no. 5, p. eaam8986, apr 2017.

[17] J. Li, P.-L. Aubin-Fournier, and K. Skonieczny, "SLAAM: Simultaneous Localization and Additive Manufacturing," *IEEE Trans. Robot.*, pp. 1–16, 2020.

[18] X. Zhang, M. Li, J. H. Lim, Y. Weng, Y. W. D. Tay, H. Pham, and Q.-C. Pham, "Large-scale 3D printing by a team of mobile robots," *Autom. Constr.*, vol. 95, pp. 98–106, nov 2018.

[19] F. Suarez-Ruiz, T. S. Lembono, and Q. C. Pham, "RoboTSP - A Fast Solution to the Robotic Task Sequencing Problem," in *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., sep 2018, pp. 1611–1616.

[20] N. Adrian and Q.-C. Pham, "MoboTSP: Solving the Task Sequencing Problem for Mobile Manipulators," 2020.

[21] J. Sustarevas, D. Butters, *et al.*, "MAP - A Mobile Agile Printer Robot for on-site Construction," in *2018 IEEE/RSJ Int. Conf. Intell. Robot. Syst.* IEEE, oct 2018, pp. 2441–2448.

[22] M. E. Tiryaki, X. Zhang, and Q.-C. Pham, "Printing-while-moving: a new paradigm for large-scale robotic 3D Printing," in *IEEE Int. Conf. Intell. Robot. Syst.*, sep 2019.

[23] J. Sustarevas, B. K. X. Tan, *et al.*, "YouWasps : Multi-Robot Construction using Mobile and Autonomous Additive Manufacturing Systems for Large-scale 3D Printing," in *2019 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2019.

[24] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," *Proc. - IEEE Int. Conf. Robot. Autom.*, no. June 2011, pp. 1478–1483, 2011.

[25] K. Nagatani, T. Hirayama, A. Gofuku, and Y. Tanaka, "Motion planning for mobile manipulator with keeping manipulability," no. October, pp. 1663–1668, 2003.

[26] T. Yoshikawa, *Foundations of Robotics*. The MIT Press, 2003.

[27] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: Representing robot capabilities," in *IEEE Int. Conf. Intell. Robot. Syst.*, 2007, pp. 3229–3236.

[28] G. Oriolo and C. Mongillo, "Motion planning for mobile manipulators along given end-effector paths," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2005, no. April, pp. 2154–2160, 2005.

[29] F. Zacharias, W. Sepp, C. Borst, and G. Hirzinger, "Using a model of the reachable workspace to position mobile manipulators for 3-d trajectories," in *9th IEEE-RAS Int. Conf. Humanoid Robot. HUMANOIDS09*, 2009, pp. 55–61.

[30] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Robot placement based on reachability inversion," in *2013 IEEE Int. Conf. Robot. Autom.*, no. 2. IEEE, may 2013, pp. 1970–1975.

[31] I. A. Sucan and S. Chitta, "MoveIt Motion Planning Framework," *ROS*, 2018. [Online]. Available: http://moveit.ros.org/about/

[32] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *Proc. IEEE RAS Humanoids Conf.*, Seoul, Korea, nov 2015.

[33] A. Makhal and A. K. Goins, "Reuleaux: Robot Base Placement by Reachability Analysis," in *2018 Second IEEE Int. Conf. Robot. Comput.*, vol. 2018-Janua. IEEE, jan 2018, pp. 137–142.

[34] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*, 2nd ed. Springer, 2017, iSBN 978-3-319-54413-7.

[35] "Descartes: Cartesian Path Planner." [Online]. Available: http://wiki.ros.org/descartes

[36] V. Suryamurthy, V. S. Raghavan, *et al.*, "Terrain Segmentation and Roughness Estimation using RGB Data: Path Planning Application on the CENTAURO Robot," in *IEEE-RAS 19th Int. Conf. on Humanoid Robots (Humanoids)*, 2019, pp. 1–8.

[37] V. S. Raghavan, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Agile Legged-Wheeled Reconfigurable Navigation Planner Applied on the CENTAURO Robot," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1424–1430.

[38] T. Welschehold, C. Dornhege, and F. Paus, "Coupling Mobile Base and End-Effector Motion in Task Space," pp. 7158–7163, 2018.