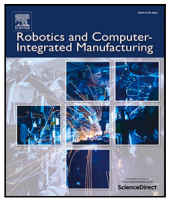
Robotics and Computer–Integrated Manufacturing 78 (2022) 102372

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim



Full length article

Planning algorithms for acquiring high fidelity pointclouds using a robot for accurate and fast 3D reconstruction

Rishi K. Malhan, Satyandra K. Gupta *

Viterbi School of Engineering, University of Southern California, Los Angeles, CA, USA

ARTICLE INFO

Keywords:
Automated Inspection
Path-constrained trajectory planning
Industrial robots

ABSTRACT

Sensors are widely used to construct a 3D model of parts by collecting data. The accuracy of the collected data depends on sensor placement with respect to the part and sensor operational range. The operational range and limitations of the sensor need to be applied as constraints while planning for robot motions that move the sensor around the part to collect data. Overly conservative constraints on sensor placement will lead to high execution times. We present an approach where we develop a robot motion planning algorithm that takes into account the camera performance constraints and produces output with low error. An RGB-D camera is used to obtain the pointcloud of the part. An offline planning method improves point density in the regions having zero or low density. Our method guarantees a high point density across the surface of the part. Results are presented on six geometries with different complexity and surface properties. We also present results on how camera parameters influence the output of our method. Our results show that algorithmic advances reported in this paper enable us to use low-cost depth cameras for producing high accuracy uniform density scans of physical objects.

1. Introduction

3D reconstruction of surfaces requires using a sensor to collect data from the part surface. Commonly used sensors have a limited operational range and field of view. This limitation requires us to move the sensor around a complex part to obtain complete information. Robots enhance the sensor capability to collect data. The use of 6 degrees of freedom (DOF) robots enables sensor position and orientations to be selected to avoid occlusions found on complex parts. In most applications, the goal is to perform the 3D reconstruction process with acceptable accuracy in the smallest amount of time.

Many commonly used sensors, such as RGB-D cameras exhibit non-uniform performance in their workspace. A depth camera might provide a large field of view and workspace. However, only a small portion of the workspace provides high accuracy. For instance, Fig. 1 shows three points observed by the camera at different locations. The location closer to the camera lens has a lower error compared to the point farther away even though they are within the operational range. Many implementations use the entire operational range of the camera to collect the pointclouds to simplify the planning process and reduce the model construction times. However, this leads to low accuracy in the model. On the other hand, an overly conservative operational range will increase accuracy, however, it will result in a significant increase in the model construction time. We need to balance data collection time and measurement accuracy during the robot motion planning phase.

We are interested in a camera placement planning approach that enables the construction of the part model with acceptable accuracy in the minimum amount of time. Camera constraints are also required to be enforced during planning. The constraints must meet the following requirements. (1) Generate camera and robot plans that produce high-quality (or low error) pointclouds. (2) The points should be measured from diverse and multiple camera locations. (3) External factors will still create regions on the measured 3D reconstruction to have missing data or low point density. We also need an online plan refinement by using live feedback during data collection to refine the robot motion and guarantee a uniform high density of points across the 3D reconstruction.

Our goal is to realize an automated 3D reconstruction cell equipped with a low-cost RGB-D sensor mounted on an industrial robot to capture a 3D pointcloud of a part that conforms to the part surface with high accuracy and guarantee uniform density. Fig. 2 shows our automated robotic 3D reconstruction cell. In our previous work [1,2], we developed a system to compute the robot motions by using the complete operating range of the camera. The system also executed only offline trajectories leading to low or zero density points. In this paper, we

* Correspondence to: University of Southern California, Los Angeles, CA, USA.
E-mail address: guptask@usc.edu (S.K. Gupta).

https://doi.org/10.1016/j.rcim.2022.102372
Received 12 January 2022; Received in revised form 18 March 2022; Accepted 4 May 2022
Available online 27 May 2022

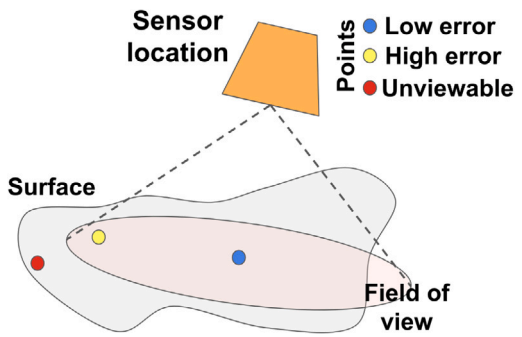


Fig. 1. Three points on a surface are viewed from different camera locations. The point closer to camera optical lens has a low error and the point farther from the lens has a high error even though both are within the field of view (FOV). Point outside the FOV is unviewable.

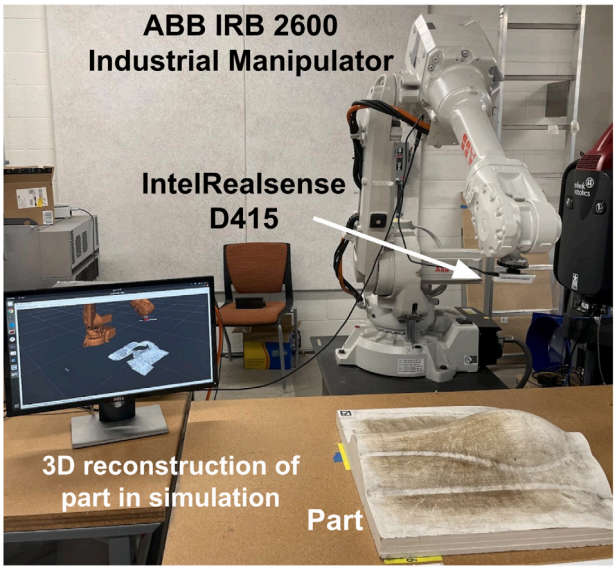


Fig. 2. Robotic inspection cell developed in this work. The video for our system is available at: https://youtu.be/6-d2MBIzApY.

extend our previous work by developing the planning constraints using a camera workspace that enables high-quality pointcloud collection. This work also guarantees uniform point density on the part with an online refinement planning module. We make the following contributions in this work: (1) Develop camera performance constraints that will accept high-quality data using the specific camera operational range. (2) Robot motion planning under the camera performance and robot kinodynamic constraints. (3) An online trajectory refinement method to identify missing low or zero density regions on the part and improve the point density.

2. Related work

Broadly, the problem of robot motion planning for 3D reconstruction involves three main components: set of camera viewpoint (location) generation, camera path generation, and pointcloud augmentation. We discuss the relevant work done on viewpoint generation in the first paragraph and the work done on camera path generation through the viewpoints in the second paragraph of this section. The third paragraph focuses on how the pointclouds captured during robot motion execution are augmented to create the output cloud.

Camera viewpoints are used to position the camera around the part. Glorieux et al. used a targeted viewpoint sampling with an optimization strategy to get the viewpoints in [3]. Bircher et al. developed an online method in a receding horizon fashion by sampling possible future viewpoints in a geometric random tree [4]. Vasquez-Gomez et al. developed a next best view algorithm to determine viewpoints for recreating an arbitrary object [5]. Raffaeli et al. developed a clustering-based approach to optimize the number of viewpoint samples [6]. Jing used randomized sampling-based and medial object-based methods to sample the viewpoints for a higher coverage [7]. González-Banos et al. have developed a random-art gallery algorithm to sample the viewpoints [8]. Viewpoint planning was also solved using reinforcement learning in [9,10]. A MATLAB toolbox-based approach for path planning for non-destructive testing was presented in [11]. Another coverage planning method for eddy current-based non-destructive testing was developed in [12]. Most of these works are focused on generating viewpoints using a complete camera operational range. We develop a method that will enforce camera performance constraints during path generation.

The sensor needs to be moved to the viewpoints by finding a path that minimizes an objective function. Generally, the objective is to reduce execution time. A survey on coverage plan generation is presented in [13]. Siyan-Dong et al. [14] presented a multi-robot collaborative coverage planning based on the traveling salesman problem (TSP). Papadopoulos et al. developed an algorithm called the random inspection tree algorithm. The algorithm generated a coverage path planning concurrently while generating the viewpoints [15]. Janoušek has implemented a method to speed up coverage queries, which is helpful in concurrent path planning and viewpoint generation [16]. The work done in [17] presented a sampling-based subroutine to improve the coverage path by making asymptotically optimal local changes on the feasible path. The authors also presented a hybrid planner to fill in the gaps to get a complete coverage path of complex 3-dimensional structures [18]. The popular approaches for solving coverage path planning are Lin–Kernighan traveling salesman heuristic [19], rapidly exploring random trees [20], probabilistic road map [21] and their modifications. Reinforcement learning has been used for mapping and exploration in coverage planning [22,23]. Most of the approaches do not incorporate all the robot kinematic and dynamic constraints while solving the problem. In this work, the robot trajectory is generated over the viewpoints under reachability, singularity, continuity, velocity, collision, and camera performance constraints.

The pointclouds captured during robot trajectory execution have to be augmented to produce the output cloud. A widely used approach is to use a grid of cubic volumes of equal size, called voxels to discretize the mapped area. The work done in [24,25] uses such a representation. An oct-tree-based representation was used in the work done in [26]. Whereas, an R tree was used in [27]. Another solution is to use a hierarchical data structure that was proposed in [26,27]. Besides the point-based methods, an alternative is to use a fully volumetric data structure to implicitly store samples of a continuous function [28–30]. In these methods, depth maps are converted into signed distance fields and cumulatively averaged into a regular voxel grid. Newcombe et al. [31] provides a truncated sign distance function implicit surface representation with a GPU accelerated voxel grid pipeline. The work done in [32] provides a hashing scheme to overcome the difficulties posed by hierarchical structures and simple voxel grids. Unlike the conventional point, mesh, and voxel-based 3D reconstruction representations, [33,34] employed a 3D geometry-based representation by learning a continuous 3D mapping. Rigid Fusion [35] and Deep Deform [36] creates non-rigid 3d models. We use a voxel-based method to generate the output cloud in this paper. Our method efficiently handles the massive number of pointclouds that continuous camera triggering obtains. Now we will discuss the terminology we use throughout the paper.

3. Terminology

A pointcloud is used to represent the 3D geometry of a part. A point in a pointcloud is represented by the coordinate vector $\langle x, y, z\rangle$ in the Cartesian space. The true part geometries we use in the test cases are represented by reference pointclouds that are captured using high-accuracy metrology instruments. The reference cloud is denoted as $\mathcal{P}$. A 3D reconstruction of the part using our method is represented by a pointcloud denoted as $\widetilde{\mathcal{P}}$.

Pointclouds are used to construct a 3D grid of cubes called as a voxel grid. The data structure makes it computationally efficient to perform operations on the cloud. The cube has an associated side length which is chosen based on the resolution we need. A voxel represents several points together as a center point of the voxel. We denoted such voxelized representations as $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$, where v_i is a 3D cubic voxel.

The robot, camera, and part used in the setup are rigid objects in the Cartesian space. Locations of these objects are represented by attaching a frame O. The frame O defines the position of the frame origin and orientation of three-unit direction vectors along the X, Y, and Z axes also known as the pose of the frame. Our work uses the frames attached to the camera (O_c), part (O_p), robot end-effector (O_e), and robot base (O_b). O_b is also used as the global reference frame. A homogeneous transformation matrix T establishes the relationship between a pair of frames [37]. For instance, points represented in coordinate system O_c can be transformed to O_b using ${}^{b}T_{c}$.

A pose in the Cartesian space with which the camera frame O_c aligns is also called as a camera viewpoint I. A set of viewpoints can be defined around the part to generate a state space. A sequence of viewpoints within the state space defines a camera path $P = \{I_1, I_2, \ldots, I_k\}$.

A serial link manipulator comprises of a sequence of rigid links connected by revolute joints. A configuration vector $\vec{q}$ describes the positions of all the joints. For a K degrees of freedom (DOF) robot, $\vec{q} \in \mathbb{R}^K$. In this work, we use a 6-DOF ABB IRB 2600 industrial manipulator that makes K=6. All such robot configurations $\vec{q}$ belong to a space called the configuration space ($\mathcal{C}$-space).

We need to represent robot paths needed for moving the camera in terms of robot joints. We will refer to these paths as configuration space paths. A configuration space path τ in $\mathcal{C}$-space is defined as $\tau = \{\vec{q}(u) | \vec{q}(u) \in \mathcal{Q}_f\}, u \in [0, 1]$, where u is a trajectory parameter and $\mathcal{Q}_f$ is the set of feasible robot configurations in $\mathcal{C}$-space. The robot joint configuration and pose of the flange are related by forward kinematics or FK. Joint velocity ($\dot{\vec{q}}$) in and the Cartesian velocity of the flange in Cartesian space $\dot{x}$ are related by the robot Jacobian J as $\dot{x} = J \times \dot{\vec{q}}$, where $J \in \mathbb{R}^{6\times N}, \dot{x} \in \mathbb{R}^6$ and $\dot{\vec{q}} \in \mathbb{R}^K$. The robot description and functionalities are stored under robot model $\mathcal{R}$.

4. Problem formulation

4.1. Background

An initial pointcloud estimation $\mathcal{P}_0$ of the part geometry is used for robot motion planning and construction of a voxel grid data structure $\mathcal{V}$. $\mathcal{P}_0$ may contain points with high errors and is only an estimate of the geometry. $\mathcal{P}_0$ is obtained by sampling points on the CAD model (if available) of the part and introducing noise that results from part manufacturing and localization errors. $\mathcal{P}_0$ can also be obtained by taking the robot on a few user-defined viewpoints over the working surface and augmenting all the pointclouds. Section 6 describes the generation of $\mathcal{P}_0$.

$\mathcal{P}_0$ is converted to a voxel representation $\mathcal{V}$ and the set of points in a voxel v_i are denoted as $P(v_i)$. The collected points during the reconstruction process are represented as $\mathcal{P}_c$ and mapped to $\mathcal{V}$. The true point will be within a voxel in $\mathcal{V}$ since manufacturing and localization errors are taken into account while constructing $\mathcal{P}_0$. Each surface voxel in $\mathcal{V}$ is used to construct a corresponding $\mathcal{N}$-neighborhood having at least N diverse points. The details on computation of $\mathcal{N}$ are discussed in Section 7.3 and binary diversity function diverse(p_i, p_j), $i \neq j$ is defined in Section 5. The points in the $\mathcal{N}$-neighborhood are then reduced to a single point. Reducing all such $\mathcal{N}$-neighborhoods produces the output pointcloud $\tilde{\mathcal{P}}$.

A point p in $\mathcal{P}_c$ is filtered before it is mapped to $\mathcal{V}$. A point admittance function $\eta(p, \mathcal{M})$ makes a binary decision (True or False) on whether an observed point p should be admitted into $\mathcal{P}_c$ or rejected thereby controlling the quality of the data. Details on the admittance function and its characterization using a camera performance model ($\mathcal{M}$) are discussed in Section 5. The Section also introduces the parameters that the function takes. We will now discuss the problem statement for our work.

4.2. Problem statement

The method takes as an input an initial pointcloud $\mathcal{P}_0$, robot model $\mathcal{R}$, the camera performance model $\mathcal{M}$, $\eta(p, \mathcal{M})$, and N. Our goal is to find an output pointcloud $\tilde{\mathcal{P}}$ of the part using all the captured $\mathcal{P}_c$. We formulate the problem in three steps given by Eqs. (1), (2), and (3).

$$\begin{aligned}
&\text{Add point } p \text{ to } P(v_i) \text{ if} && (1)\\
&(i)\ p \in v_i\\
&(ii)\ \forall v_i \in \mathcal{V}, \exists \mathcal{N}_i\\
&(iii)\ \forall i, j \ \text{diverse}(p_i, p_j) = True,\ i \neq j\\
&(iv)\ \forall p \in \mathcal{N}_i,\ \eta(p, \mathcal{M}) = True\\
&\text{Terminate point insertion if} && (2)\\
&(i)\ \text{cardinality of } \mathcal{N}_i \geq N\\
&\text{Finally, construct } \tilde{\mathcal{P}} \text{ from } \mathcal{P}_c \text{ by averaging points in } \mathcal{N}_i && (3)\\
&\text{where, } P_c = \bigcup_i P(v_i)
\end{aligned}$$

$\mathcal{R}$ is constructed based on the robot used. $\mathcal{M}$ is decided by the camera used in the system. The selection of N is a user-specific choice. The selection is based on the accuracy required from the final pointcloud. The accuracy is related to N by representing the uncertainty of a point captured by the camera as a Gaussian distribution. σ is the variance for the distribution of error for a point. The new variance for N points will be given by $\sigma/\sqrt{N}$. Increasing N improves accuracy but will lead to higher execution times with diminishing gain in accuracy. Section 8 will further present the influence of N on the accuracy.

5. Camera performance model

An RGB-D camera captures an image using a 2D grid of pixels where each pixel will also have an associated depth value (referred to as the Z-depth in this work). The depth camera uses the intrinsic parameters to convert the 2D image into a 3D pointcloud that is represented in camera frame O_c, also the camera depth optical frame for Intel RealSense D415 camera used in our work. Camera intrinsic properties are a geometric transformation incorporating the optical center and focal length. The pointcloud is then converted to a global reference frame O_b using the transformation ${}^{b}T_{c}$, also called the camera extrinsic property. The conversions can be referred in Fig. 3(right).

All the points obtained in the global reference frame do not have uniform accuracy. Larger distances of the points on the surface from O_c lead to high errors. The points are out of focus when they get too close. Refer to Fig. 3(left) that illustrates the field of view of the camera. There is an acceptable range which we set within which the points can be successfully captured by the camera and the error is low. It is important to keep the observable points within the acceptable range from the camera frame while capturing the pointclouds.

The camera performance model in our method captures three parameters, α, β, and γ to characterize the camera performance. Fig. 4

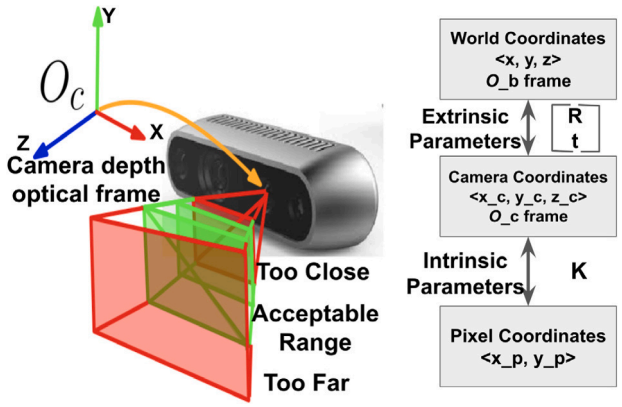


Fig. 3. (Left) The field of view for the D415 depth camera is illustrated as a frustum. The Center region of the frustum is the acceptable distance of the surface from the camera frame. (Right) A block diagram showing the relationship between the conversion of coordinates between world, camera, and pixel frames. R and t are the rotation and translation elements that make up the homogeneous transform for extrinsic parameters. K is the matrix representing intrinsic parameters.

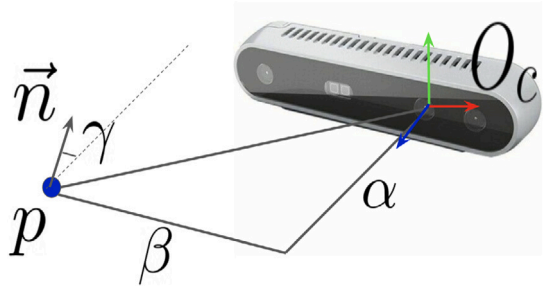


Fig. 4. The parameters α, β, and γ for an observed point p in the world with respect to the camera frame O_c is illustrated. $\vec{n}$ is the normal vector that corresponds to p.

illustrates a point p viewed by the camera. α is the Z-depth or normal distance of the point from O_c. β is the radial distance of p from O_c that defines how far out the point is from the camera frame within the field of view. The point p also has an associated normal vector $\vec{n}$ of the underlying surface. The angle γ is made by this normal and the direction Z-axis of the camera frame O_c.

α, β, and γ govern whether the observed point will be labeled as acceptable or unacceptable data. Let the error of the point captured p with respect to the true point p^* be computed as the second-order norm $\|p^* - p\|_2$. The observed points from the camera have a distribution associated with the error. We represent the variance of the distribution by σ. As the point moves away from or gets too close to the camera frame, the σ increases that are given by the σ vs. α profile. Points are not recorded if the value of α decreases below a threshold since the observable world goes out of view. The σ vs. β profile emphasizes the increase in σ as the point moves radially outward from the camera centerline extended along the depth direction. Finally, the parameter γ captures whether the camera is normal to a point on the surface or it is oriented at a very high angle that increases the σ.

Determining the camera performance model involves finding the admittance function $\eta(\cdot)$. Many non-linear complex relationships can exist for a sensor and a more sophisticated function will lead to better output. In this work, we empirically determine the lower and upper on acceptable α, β, and γ that make a linear admittance function η. The empirical relationship is found using the following steps.

1. *Planar surface reference:* A high fidelity pointcloud of a plane surface is obtained using the Hexagon Romer arm with an integrated laser scanner used in our work as a reference metrology instrument.
2. *Depth camera scan:* The planar surface is then scanned by the depth camera from various camera viewpoints. Values of α, β, and γ are computed for each observed point p_i on the planar surface.
3. *Error computation:* The error for p_i is computed as deviation from the true point p_i^t using second order norm $\|p_i^t - p_i\|_2$. Variance of the error is obtained by using error values obtained from multiple measurements taken for the point p_i.
4. *Determining bounds:* The lower and upper limits on α, β, and γ are selected by using user-defined threshold on the computed variance value.

Fig. 5 shows a 3D plot with values of α, β, and γ plotted on the axes with units meters, meters, and radians respectively. A colormap is overlaid on the cuboid representing the values of a variance σ. σ is computed for all the deviation errors for all the observed points captured in small elements within the plot. We can observe that for a α value of less than 0.25, we do not obtain any valid point since the surface goes out of the camera field of view. Fig. 5 also shows three other 2D plots where the average error for each element is plotted. For each plot, two of the three parameters are kept constant and the third parameter is allowed to vary. The values of the constant parameters are shown in the legend. We can notice that the aforementioned increasing trend of error with respect to increasing values of the parameters holds for all the cases. We will now define all the constraints that will be applied for camera path planning and robot trajectory planning based on the aforementioned discussion.

1. *Acceptable Range Constraint:* The parameters of a camera view point I from the observed point p located in the world need to be constrained with the limits defined over α, β, and γ.

$$\begin{aligned} \alpha_{lb} \leq \alpha \leq \alpha_{ub} \\ \beta_{lb} \leq \beta \leq \beta_{ub} \\ \gamma_{lb} \leq \gamma \leq \gamma_{ub} \end{aligned} \tag{4}$$

where lb and ub are the lower and upper bounds on the parameters. We use only the range constraint given by Eq. (4) for planning purposes to make sure that the camera viewpoints are generated within the camera field of view. Other constraints can also be incorporated for a more sophisticated plan.

2. *Performance Constraint:* The lower and upper bounds on the three camera parameters are given by the point admittance function $\eta(p, \mathcal{M})$. $\eta(p, \mathcal{M})$ makes a binary decision of whether to accept or reject the point. The performance constraint is then defined as:

$$\eta(p, \mathcal{M}) = True \tag{5}$$

The η function used in our work applies the following thresholds on the three parameters to filter the points from original pointcloud. $\alpha : [0.26, 0.4], \beta : [0, 0.2], \gamma : [0, 1.0]$.

3. *Diversity:* Each point has an associated camera frame from which it was captured. The diversity value is then calculated using Eq. (6). The new point will be rejected if the value is less than a threshold δ.

$$\|\langle \alpha_i, \beta_i, \gamma_i \rangle - \langle \alpha, \beta, \gamma \rangle\|_2 \leq \delta, \forall i \in [1, k] \tag{6}$$

where, k is the number of points currently present in the $\mathcal{N}$-neighborhood and $\langle \alpha, \beta, \gamma \rangle$ is the vector for camera parameters of the new incoming point.

6. System overview

Fig. 6 gives the overview of our system along with the individual modules and the libraries we have used. The framework is built on top of Robot Operating System (ROS) architecture and uses libraries such as MoveIt, open3d, scipy, Intel RealSense for ROS, and RViz. Input to the system is as follows:

1. A pointcloud estimation $\mathcal{P}_0$.

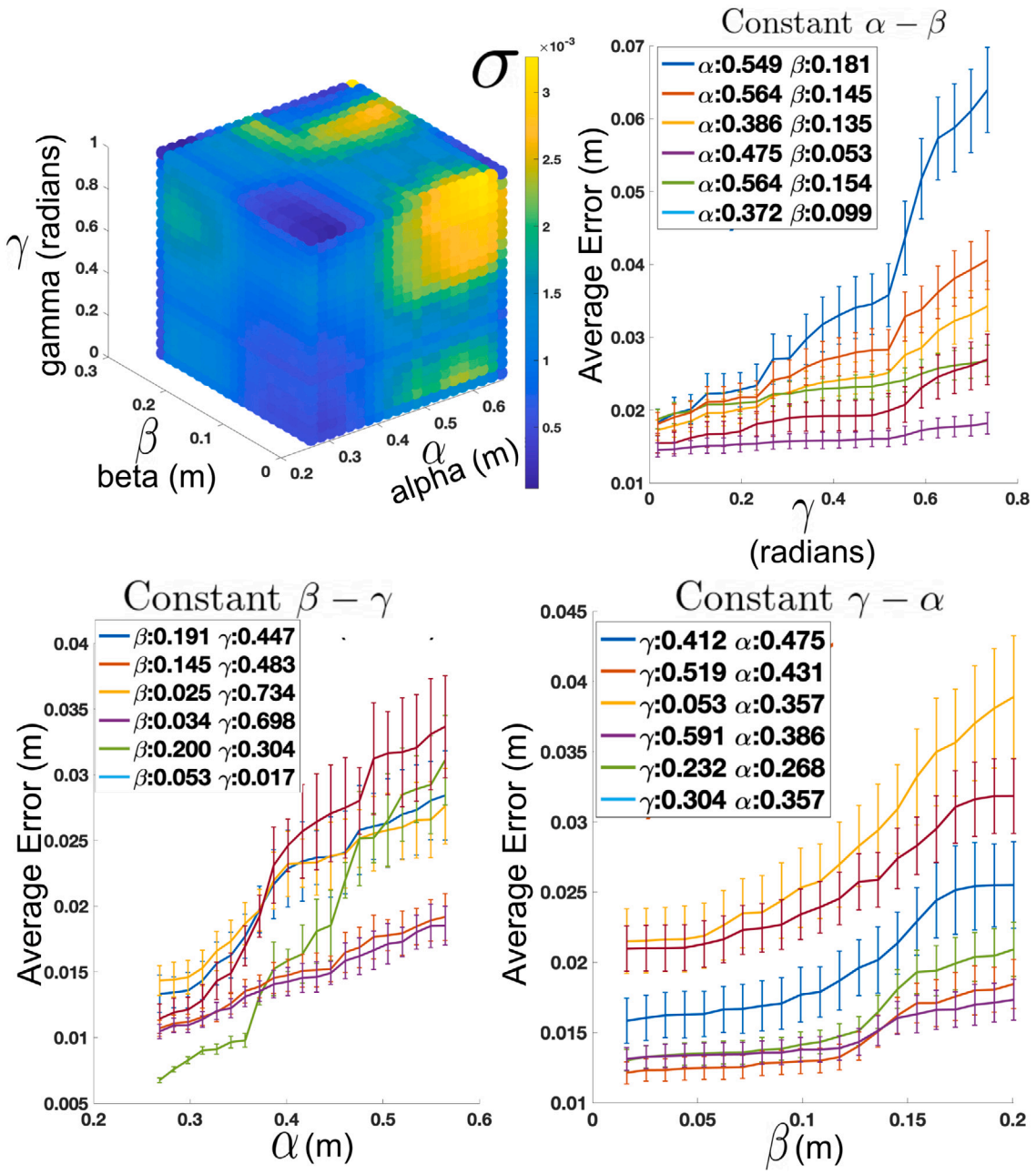


Fig. 5. (Top, Left) 3D colormap cuboid showing the variance for every small element $\Delta\alpha, \Delta\beta$, and $\Delta\gamma$ within the range of values observed in the dataset produced from the planar surface. (Top, Right) Profile for deviation error with respect to γ for a few sampled constant α and β. (Bottom, Left) Profile for deviation error with respect to α for a few sampled constant β and γ. (Bottom, Right) Profile for deviation error with respect to β for a few sampled constant γ and α. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2. The user estimated transformation ${}^{e}T_{c}$ and ${}^{b}T_{p}$ that will be used for camera localization and placement of voxel grid $\mathcal{V}$ with respect to the robot base.

Our system performs the 3d reconstruction using the following different components.

1. *Camera localization:* The transformation ${}^{e}T_{c}$ can be estimated using the CAD model of the camera mount but is not accurate to produce reliable measurements due to manufacturing and assembly errors. Our system uses a multi-stage optimization strategy to localize the camera with sub-millimeter accuracy and the process is described in the appendix. Another variant for an accurate camera localization method can also be found in the work done in [38].

2. *Generation of $\mathcal{P}_0$:* $\mathcal{P}_0$ is generated by sampling points over the surface of the CAD (if available). However, $\mathcal{P}_0$ is not the true representation of the real part due to manufacturing and localization errors. Additional points are therefore introduced in $\mathcal{P}_0$ by sampling the measurement uncertainty for each point to include all possible regions where the true point can exist. For instance, in our implementation, we used a maximum position and orientation uncertainty of 40 mm and 30 ° respectively for each point. Each point on the CAD model was then be perturbed with many positions and orientation errors sampled within the limits −40 to 40 mm and −30° to 30° respectively. The resultant $\mathcal{P}_0$ is a cloud of points around the CAD surface that will entail the true point.

When the CAD model is not available, the robot will have to take the depth camera to a few predefined viewpoints over the part and augment all the points that correspond to the part without filtering them using admittance function η.

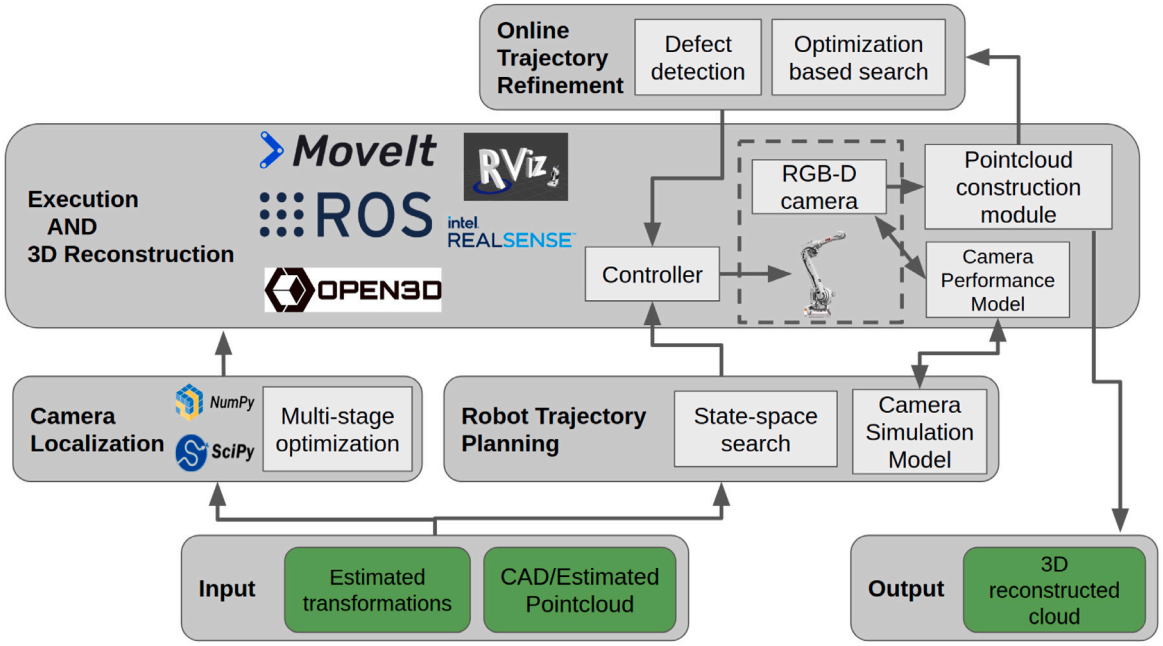


Fig. 6. Block diagram representing the system architecture used by our method during reconstruction.

3. *Camera Path Generation:* Viewpoints are generated around the part using the CAD model. A state-space search is then performed over feasible viewpoints using a simulated camera model under the camera performance model constraints to find P. Section 7.1 describes this step.

4. *Robot Trajectory Generation:* The robot trajectory is generated in the C-space from the camera path P. Robot trajectory generation algorithm enforces robot-specific constraints like reachability, collision avoidance, path consistency, continuity, and velocity on top of the camera performance constraints to find an executable robot trajectory. Section 7.2 describes this step. Pointclouds are captured and augmented using the 3D reconstruction step of our method.

5. *3D Reconstruction:* Pointclouds are continuously captured instead of discretely capturing them at the viewpoints. An efficient method to handle the pointclouds in a parallel thread is described in Section 7.3 that merges the pointclouds by accepting points that reduce the overall inaccuracy of the output cloud $\tilde{\mathcal{P}}$. A data structure of voxel grid $\mathcal{V}$ is used to process the incoming pointclouds filtered using the camera performance constraints.

The points in each $\mathcal{N}$ neighborhood of the voxels in $\mathcal{V}$ are then averaged to find the points in $\tilde{\mathcal{P}}$.

6. *Online Trajectory Refinement:* Section 7.4 describes this module which continuously examines regions with zero or low point density, also called anomalous regions on the part. The regions do not satisfy pointcloud density requirement. The module generates a refinement plan over such regions to capture more points.

Once the trajectory execution is complete the system returns the output $\tilde{\mathcal{P}}$. We will now discuss the 3D reconstruction algorithm and how the aforementioned components fall in place.

7. 3D reconstruction algorithm

7.1. Camera path generation

Feasible state space of viewpoints is constructed around the part and then a greedy planning method is used to find the camera path.

7.1.1. State space generation

Camera path needs to be designed such that the optical frame follows the part surface geometry. The optical frame also needs to be within the acceptable distance from the surface. Hence, we can use the surface voxels in $\mathcal{V}$ and shift them outward by the acceptable distance to construct the state space. The method first downsamples $\mathcal{V}$ from the resolution of 0.5 mm to 40 mm to generate a new voxel representation $\mathcal{V}_c$. The coarser resolution controls the viewpoints in state-space since a large number of viewpoints are unnecessary and increase computation cost. The coarse resolution is empirically selected based on the number of viewpoints required to cover the part surfaces, camera field of view, and part geometries used in our test cases. The Center of each voxel in $\mathcal{V}_c$ represents the position of viewpoints in the state-space (S) and also has a direction normal associated with it.

The viewpoint positions are then shifted outward along their direction normals at the acceptable distance sampled within the bounds α_{lb} and α_{ub}. We choose a range constraint of 35 cm for our state space. The camera needs to view the surface as close to the surface normal for capturing low error points. Orientation of each state is therefore assigned by using the viewpoint normal. Orientation vectors are uniformly sampled within a 3D cone about the normal direction of a voxel and each sample is added to the state space. The 3D cone allows some flexibility about the direction normal so motion plans can be easily generated without over constraining the camera to align with the normal. Fig. 7 illustrates the state space that is generated for a part. Line-1 of algorithm 1 specifies the state-space generation step.

7.1.2. Path generation

A state-space search algorithm then finds the camera path that passes through a minimum set of viewpoints generated in the state space. The algorithm takes additional inputs as the robot model $\mathcal{R}$, a simulated camera model C_s, voxel grid $\mathcal{V}$, and camera performance model $\mathcal{M}$ for path generation. The camera path generation is conducted in the following steps.

1. *KDTree Generation:* A KDTree data-structure is initialized (line-2 of algorithm 1) over all the viewpoints in S. The start state of the camera path P is the nearest neighbor to the current camera pose or a user-specified pose. The start state is appended to the path.

2. *Selection of child nodes:* A fixed K number of viewpoints from S are queried from the KDTree for the last state in P as the parent node. The

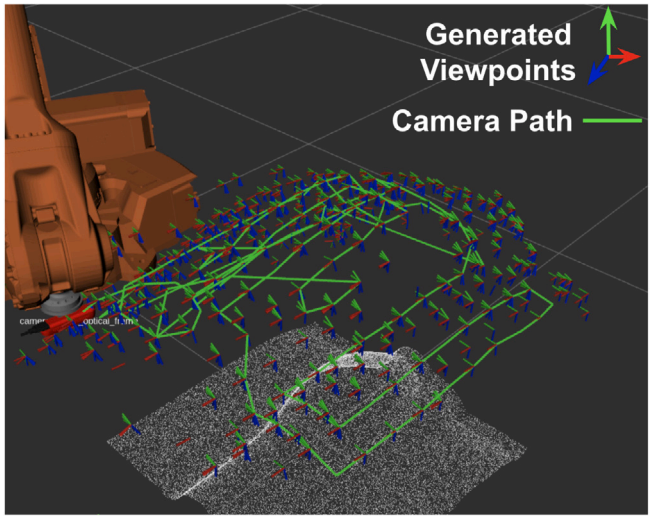


Fig. 7. The state space of feasible camera viewpoints are shown around a part. A camera path passing through a set of viewpoints is executed by the robot to capture the pointclouds across the surface.

Algorithm 1 CameraPathGeneration($\mathcal{R}$, $\mathcal{V}$, $\mathcal{M}$, C_s)

```
1: state_space ← generate_state_space(V)
2: kdtree ← generate_KDTree(state_space)
3: path.append(kdtree.query(current_camera_pose(R), K = 1))
4: while !stopping_condition do
5:    children ← kdtree.query(current_camera_pose(R), K = 20)
6:    parent ← path[−1]  rewards ← [ ]
7:    for child in children do
8:       rewards.append(V.simulate_update(parent, child, M, C_s))
9:    end for
10:   if all(rewards == 0) then
11:      expand(K)
12:      if K > state_space.size() then
13:         return path #Incomplete solution
14:      end if
15:      repeat steps 8-10
16:   else
17:      path.append(children[rewards.argmax()])
18:      V.update(parent, children[rewards.argmax()], M, C_s)
19:   end if
20:   if V.coverage_reached() then
21:      stopping_condition ← True
22:   end if
23: end while
24: return path
```

viewpoints are added to a list of child nodes for the next step. Line-5 of algorithm 1 specifies this step.

3. *Camera segment generation:* A Cartesian segment from parent node to each child node is then generated under the range constraint described by Eq. (4). Position of the camera frame while transitioning from the parent node to a child node along the Cartesian segment is maintained within the acceptable range. All the Cartesian segments formed between the parent and each child node are then evaluated using a camera simulation.

4. *Voxel data-structure update:* A camera model C_s is used to simulate how pointclouds will be captured in the real world while moving the camera along each Cartesian segment computed in the previous step. The simulated pointcloud is filtered using the point admittance function η. The pointcloud is then admitted into a simulated copy of the voxelized data structure $\mathcal{V}$.

5. *Reward computation:* A point from the filtered pointcloud is considered to be useful towards the reward computation for a Cartesian segment if it helps increase the number of points in a neighborhood $\mathcal{N}_i$ for $v_i \in \mathcal{V}$ and gets it closer to N. Given the number of new points that are considered useful from the simulated pointcloud and the total number of points in $\mathcal{V}$, the reward is computed using the following equation.

$$r = (\text{new\_points}/\text{total\_points})/\text{segment\_length} \tag{7}$$

The reward is higher for a parent-to-child segment if a large number of high quality and diverse points are added to $\mathcal{V}$ to meet our objective with a shorter travel distance in the Cartesian space.

6. *Selection of highest reward child:* The reward is computed for all segments generated from the parent to each child in the list of child nodes and the child which gives the maximum reward is chosen as the next viewpoint along the path along with the corresponding Cartesian segment. Lines 6–19 of algorithm 1 encompass steps 3–6 of the path generation process.

We will now discuss what the stopping conditions for the iterations of adding child nodes to the path are and when the algorithm returns an incomplete solution.

7.1.3. Stopping conditions

1. *Incomplete solution:* The neighborhood size we use in our approach for selecting child nodes is $K = 20$ which we obtained by manually tuning the method for low computation speed and path lengths. There are situations where no reward can be obtained while transitioning from a parent to possible child nodes (lines 10–15 of algorithm 1). The neighborhood K is then increased and more number of child nodes are evaluated (line-11 of algorithm 1). The neighborhood keeps expanding until the entire state space is exhausted. The algorithm will then return an incomplete solution. Such a situation will arise if state space is inadequately sampled and regions of the part are completely unvisited. We generate a state space that is adequate to find a complete solution.

2. *Complete solution:* A complete coverage is obtained if all the $\mathcal{N}$ neighborhoods for every voxel in $\mathcal{V}$ have contained N valid points. The algorithm then returns the path found so far.

7.2. Robot trajectory generation

A robot configuration path $\tau = \{q_1, q_2, \ldots, q_m\}$ needs to be found in the configuration space (C) of the robot corresponding to the camera path $P = \{I_1, I_2, \ldots, I_m\}$. The configuration space path is then converted to a trajectory $\tau(t)$, where t is the time. The computation efficiency and success rate of finding a trajectory reduces by enforcing all the constraints at once. Several evaluations of the constraint function are required to compute the gradient and the optimization algorithms are susceptible to local minima. We, therefore, use a hierarchical approach to this trajectory generation problem to improve the computational efficiency of generating a solution. The stages in robot trajectory generation are as follows.

7.2.1. Reachability check

Checking the reachability of the robot first before checking for meeting other constraints can save computation costs since constraints will never be met if the robot is unreachable. An inverse kinematics (IK) filter is applied during camera path generation we discussed earlier in Section 7.1. The robot IK filter provides a computationally fast reachability check before camera motion is simulated from parent to child. Given that the robot configuration at parent pose is q_{i-1}, the validity of configuration for a candidate child pose denoted by w is checked by using the logic in Eq. (8).

$$\begin{aligned} &q_i \text{ is valid if } \underline{q} \le q_i \le \overline{q} \\ &q_i \leftarrow \arg\min_q PE(q_{i-1}, w, \mathcal{R}) \\ &PE = \|\langle x_p, y_p, z_p\rangle - \langle x_w, y_w, z_w\rangle\|_2 + \\ &\sum_{i=\{bx,by,bz\}} (1 - p_i * w'_i) \\ &\text{where } p = FK(q_i) \end{aligned} \tag{8}$$

where $FK(\cdot)$ is a forward kinematics function for the robot, $\langle x_p, y_p, z_p \rangle$ and $\langle x_w, y_w, z_w \rangle$ are the 3D position only vectors for from poses p and w respectively, and bx, by, bz are the unit X, Y, and Z direction vectors of the frames corresponding to the poses p and w. The pose error PE essentially minimizes the position and orientation difference between the robot's current frame p and the target frame w. The robot is reachable at the target w if a solution is found to the aforementioned minimization problem and is within the lower and upper joint limits $\underline{q}$ and $\overline{q}$ respectively. The camera viewpoint is discarded if the robot is unreachable and the child node is not considered for planning of camera path.

7.2.2. Sequential C-space path generation

A sequential IK solver then generates a sequence of robot configurations by using an iterative optimization process given by the Eq. (9). The solver finds robot C-space path τ by introducing the path consistency constraint defined by Eq. (10) and collision constraint given by Eq. (11) to generate a continuous path.

$$\tau = \{q_i | q_i \leftarrow \arg\min_{q} PE(q_{i-1}, I_i, \mathcal{R}), \forall i \in [1, m]\} \tag{9}$$

$$\|q_i - q_{i-1}\|_\infty \leq \Delta\theta \tag{10}$$

$$in\_collision(q_i) = false \tag{11}$$

The path consistency constraint enforces that the two consecutive robot configurations q_{i-1} and q_i do not have an absolute maximum joint angle change of more than $\Delta\theta$ radians given by the infinity norm. This ensures that the path is executed continuously without dramatic robot joint angle changes. The collision check between all the bodies in the environment is done using built-in collision checking the functionality of MoveIt [39].

7.2.3. C-space trajectory generation

C-space path generated in the previous step needs to be time parameterized for the robot controller to execute it. Additional constraints like velocity and singularity are introduced in this step and the sequential problem given by Eq. (9) is resolved to find a trajectory $\tau(t)$.

1. *Velocity:* Velocity constraint for ith configuration q_i along the path is given by the robot Jacobian at previous configuration q_{i-1}. The robot Jacobian defines the relationship between robot joint velocity $\dot{q}$ and the Cartesian velocity of the tool attached to the robot. The tool frame in our case is the camera frame $\dot{I}$ and the velocity is represented as $\dot{I} = J(q) * \dot{q}$. For a small-time change Δt, the relation is expressed as $\Delta I = J(q) * \Delta q$. The velocity constraint is then given by the following equation.

$$\underline{\dot{q}} \leq \text{inv}(J(q_{i-1})) * (I_i - I_{i-1}) \leq \overline{\dot{q}} \tag{12}$$

where $\underline{\dot{q}}$ and $\overline{\dot{q}}$ are the robot lower and upper velocity limits, inv() is a linear algebraic inverse function, and I_i and I_{i-1} are the ith and $(i-1)th$ camera viewpoint in the sequential path P.

2. *Singularity:* Robot singularity constraint is also given by the robot Jacobian for a given configuration q_i as:

$$\det(J(q_i) * J(q_i)^T) \geq \omega \tag{13}$$

where det() is the determinant of the queried matrix J^T is the transpose of the Jacobian matrix. ω is a constant. Higher values of ω ensure good robot manipulability at a camera viewpoint which makes it feasible to generate online trajectory refinement plans to improve point density.

7.2.4. Point-to-point trajectory generation

The trajectory found using the sequential solver is used to capture pointclouds. Other trajectories are needed for moving the robot from one point to the other in free space without tracing Cartesian paths. We use a sampling-based planner from MoveIt [39] for such point-to-point planning. More sophisticated approaches have been studied and can be used where the air-time or the time between transitioning from one point to the other is minimized for further improving the overall execution time [40].

7.3. Pointcloud construction

The pointclouds are captured continuously in a parallel thread as the robot executes the trajectory $\tau(t)$. We developed an efficient method to handle the massive number of pointclouds. The method can deal with a finer resolution to cover more intricate details of the part geometries and also saves a larger number of points in the cloud before generating the output improving our previous approach in [1,2]. The pointcloud processing method is used both for the simulation environment used by the camera path generation and for the real-time execution. The difference is that the simulation environment used a camera model C_s whereas the pointclouds in real time are captured using the real RGB-D sensor. We will now discuss the elements of the pointcloud construction method.

7.3.1. Pointcloud filtering

A pointcloud captured from the camera is first transformed to a robot base frame using the corresponding ${}^bT_c$. All the outliers are removed and the constraint-based on point admittance function η given by Eq. (5) is applied to eliminate high error points. The resulting pointcloud is then mapped to $\mathcal{V}$.

7.3.2. Point mapping

Each point in the incoming cloud is checked if it belongs to $\mathcal{V}$ and the points that do not belong to any voxel are removed. More points are admitted during the scan in the same way. The voxels are processed to the output pointcloud by using an *update rule* in our method.

7.3.3. Update rule

The surface voxels in $\mathcal{V}$ are identified using the intersection of voxels with the rays projected from viewpoints in the camera path P. The first voxel to intersect the ray is a candidate surface voxel. If the candidate surface voxel has neighbors greater than a threshold number then it is labeled as a valid surface voxel in $\mathcal{V}$. Invalid candidate voxel is disregarded and the next intersecting voxel is considered for evaluation. The method allows us to disregard but not eliminate the voxels that may be filled out with noisy data and consider the voxels that have a high density of neighbors.

Every surface voxel v_i^s in the original grid $\mathcal{V}$ also has a direction normal associated with it. A cylindrical geometry is generated for each v_i^s with the principal axis (longer dimension) along the surface normal. The center of each cylinder coincides with the center of v_i^s. The principal axis aims at capturing points that would be spread along the normal due to the Z-depth uncertainty of the camera. The radius of the cylinder aims at capturing points spread due to the lateral uncertainty of the camera. We choose a radius of 1.5 mm in our work and a principal axis of 20 mm.

All the points that fall within this cylinder (may belong to other voxels) are considered as the $\mathcal{N}$-neighborhood of voxel v_i^s. These are the points that we use to determine whether N diverse points have been attained for the voxel v_i^s. The $\mathcal{N}$-neighborhood points are then checked for outliers and any point with a deviation of more than $2 * \sigma$ is rejected. The remaining points are averaged and become the new location of voxel v_i^s. The voxel v_i^s is considered as low-density voxel if the points within the $\mathcal{N}$-neighborhood are less than N.

The update-rule is applied to $\mathcal{V}$ after a threshold total number of points within $\mathcal{V}$ is reached. The update rule also helps with saving memory as points are averaged and reduced to a single point. Successive iterations of the update-rule gradually allow the surface voxels to converge to the true surface of the part.

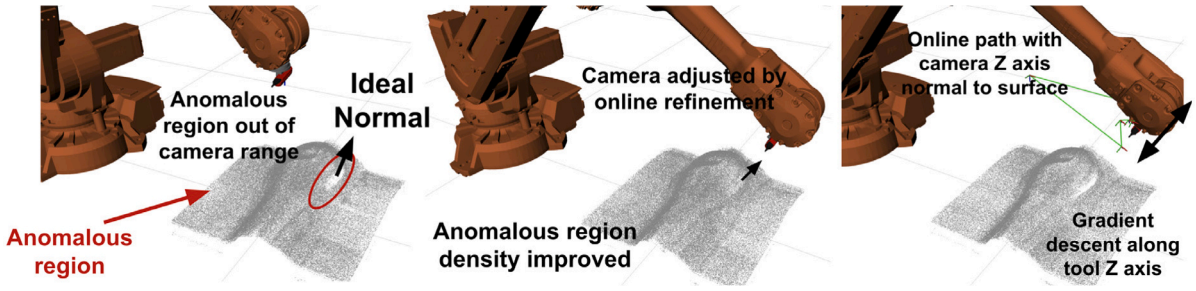


Fig. 8. The camera was not oriented normal to the ideal surface normal of the double curvature region leading to zero or low point density, also called anomalous regions (left). The point density of anomalous regions can be improved by aligning the camera with the ideal normal (center). An example online camera path generated using the refinement module to increase point density is shown (right).

7.4. Online trajectory refinement

The observations we accept during the scanning process correspond to the conservative bounds expressed by the camera performance constraint. The camera path is generated such that all points captured by the simulated camera are within the bounds and the part is fully covered. However, complete coverage may not be possible in reality due to simplifications in the camera model or external environmental factors. There may exist regions with zero or low point density as the $\mathcal{N}$-neighborhood for surface voxels in $\mathcal{V}$ did not reach the desired minimum number of points N.

It is necessary to pay particular attention to such anomalous regions and orient the camera with respect to surface normals of the anomalous region to improve the density of the points. Fig. 8(left) illustrates an example where pointcloud obtained by the robot during the execution of camera path had anomalous regions with low or zero pointcloud density. The region was situated where the part geometry had double curvature and high error points were being captured by the camera which was rejected by the camera performance constraint.

Fig. 8(center) shows how aligning the camera with the normal vector of an anomalous region can help to increase the point density since the robot slows down and pays particular attention to such regions. The online trajectory refinement module in our system is responsible for such online corrections in the robot trajectory. We will now discuss the sequence of steps that go into the online trajectory refinement module.

7.4.1. Identification of unobserved clusters

The grid $\mathcal{V}$ is used to identify the unobserved regions on the part during the scanning process. We first identify the voxels in $\mathcal{V}$ that have low or zero density of points. A custom clustering algorithm is then used to cluster the regions such that the minimum number of voxels in the cluster is greater than a threshold. This allows us to ignore several smaller regions on the part as noise since the voxel size used is 0.5 mm and many clusters are identified and may not necessarily be an unobserved region. The unobserved regions are used to identify anomalous regions.

7.4.2. Identification of anomalous regions

The simulated camera is used to determine what regions will be covered by the remainder of the camera path that has not been executed. A cluster will be an anomalous region if it does not fall under the simulated camera coverage. The anomalous regions will not be visited by the robot in the offline plan and are holes in the part model. The robot is then commanded to stop the execution of the trajectory and first execute the online trajectory to improve the anomalous region point density before resuming the offline trajectory.

7.4.3. Online trajectory generation

The generation of an online trajectory to fill the anomalous regions is done in two steps.

1. *Viewpoint generation:* The anomalous region is discretized into smaller regions using a 3D surface grid. Each element on the grid has a direction normal vector associated with it. Camera frames that are oriented with the normal vector are generated such that they obey the range constraint (Eq. (4)) and the robot reachability constraint (Eq. (8)). The set of all such camera frames for every grid element across the grid are the viewpoints for the online planner.
2. *Optimization based execution:* Robot is then moved from the location where it was stopped to one of the generated viewpoints. The motion allows the camera to be oriented normal to the small grid element of the anomalous region improving the density of points for that region. A gradient descent-based motion primitive is also used based on a predefined set of actions if the density does not improve. For instance, the camera pose is changed along the Z-depth axis in an increasing direction until new points are obtained. If no information is added, the motion is reversed to the decreasing direction. Similar motions are executed along the camera tilting axis as well.

The stopping condition used during online execution is based on a time-out. The online execution is terminated if the point density cannot be restored in the concerned region.

8. Results

8.1. Test cases

Six parts A, B, C, D, E, and F as shown in Fig. 9 are used as test cases in this paper. The part geometries have single and double curvatures, convex and concave contours, corners and cavities, and different reflectivity properties due to wood, aluminum, and plastic materials used to manufacture them. The variation makes it a challenging case study to benchmark the performance of our method. The length, width, and height of parts A, B, C, D, E, and F are 380 x 280 x 80 mm, 500 x 450 x 200 mm, 300 x 600 x 300 mm, 570 x 830 x 95 mm, 480 x 470 x 480 mm, and 400 x 1150 x 90 mm respectively. The dimensions cover medium–large industrial parts and the robot is reachable over all the parts and can successfully move the camera to cover their surfaces.

8.2. Algorithm performance

8.2.1. Reconstruction performance

Baseline approaches are used to provide insights into the importance of using the four main components described in this paper: range constraint during planning, camera performance model constraint, minimum number of observations (N) and diversity constraint, and online plan refinement methods.

Baseline-1 does not use any of the aforementioned components and the camera viewpoints are generated around the part without the range constraint. A camera path is then generated using the same planner as

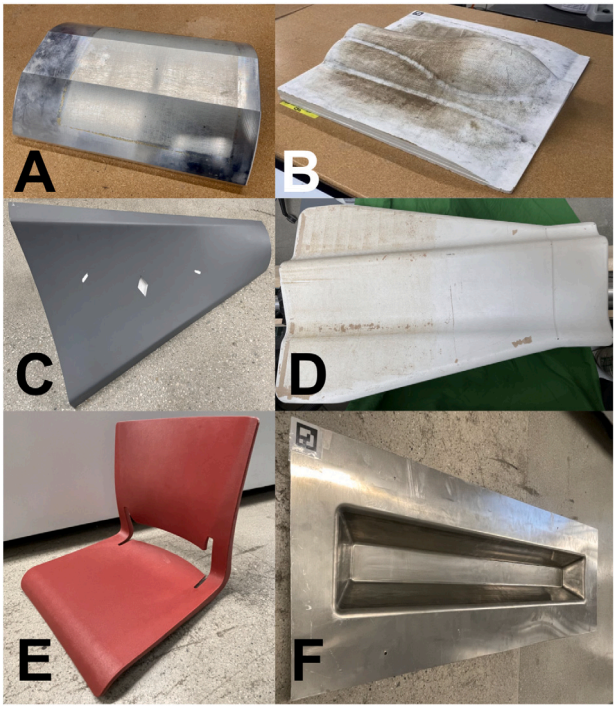


Fig. 9. Parts A, B, C, D, E, and F that are used in our work. The parts are fabricated with complex geometries and different surface properties to provide insights into the performance of our method.

we discussed in Fig. 7. The simulated camera model for the planner does not truncate points based on the point admittance function η and all the points are used to fill the $\mathcal{N}$-neighborhood. The method aims at collecting raw camera data during the scan and then post-processing it to create the output pointcloud. The collected points will be a mix of low and high error points.

The baseline-2 method generates the state space using the range constraint. All the viewpoints are within the Z-depth distance specified by lower and upper limits of α from the part. Camera performance and diversity constraint are not applied and therefore accepting all the camera points without truncating them using η. The method collects raw pointcloud data during the scan, similar to baseline-1. Although, in this case, the camera distance from the surface is maintained within the acceptable range. The collected data therefore should have lower error compared to baseline-1. However, this would still have a mix of high and low error points.

The third set of pointclouds is generated using all the constraints but not using the online refinement to fill the anomalous regions. The approach imposes the range, camera performance, and diversity constraints while generating the camera path and capturing the pointclouds continuously during execution. We label this approach as *offline only*. The output pointcloud will only have low error points since *eta* function eliminates high error points. The overall quality of the data will also be better than baseline-1 and 2 since diverse measurements are taken for each point on the surface. The offline-only approach also provides insights into how zero or low-density regions can leave holes in the output while only using offline planning.

Lastly, the fourth set of pointclouds uses both offline and online refinement. The approach imposes all the constraints and also obtains points for anomalous regions online to guarantee the pointcloud density is consistent throughout the scan. We label this method as *offline + online*.

Fig. 10 shows a comparison between all the aforementioned methods using a point deviation colormap for all the parts. The output pointcloud $\tilde{\mathcal{P}}$ is overlaid on the reference pointcloud $\mathcal{P}$ using iterative closest point (ICP) algorithm and point to point correspondence is found. Each point in $\tilde{\mathcal{P}}$ is then assigned a color based on the deviation error from the reference cloud. The colormap demonstrates the overall accuracy of the surface scan.

Table 1
Average and maximum values of point deviation error in millimeters computed using the points across the entire surface for all the test parts used in our work. The baseline methods and our approach (offline + online planning and execution).

	Baseline-1		Baseline-2		Our Approach	
Part	Avg	Max	Avg	Max	Avg	Max
A	7.2	13.6	4.8	11.6	1.4	2.3
B	5.2	12.9	3.3	8.6	1.5	3.3
C	6.5	15.6	4.4	12.5	1.9	2.9
D	4.8	12.7	3.8	10.4	1.7	2.8
E	5.6	13.2	4.1	13.6	1.4	3.8
F	6.4	18.3	5.7	12.8	2.1	4.3

We can observe that the errors are higher for baselines 1 and 2 since we do not use range, camera performance, and diversity constraints during planning and execution. A good post-processing technique will not be able to generate high-quality output construction since the raw data collected contains high error points. Baseline-1 and 2 both show the behavior since η function is not used during planning and diversity in data collection is not enforced either. Deviation errors using baseline-2 are lower than deviation errors using baseline-1 since range constraint is enforced in baseline-2 providing slightly better quality data. Significant improvements are further obtained in the offline approach as we enforce all the constraints. The offline-only approach, however, leaves anomalous regions in the pointcloud since anomalous regions are not identified and rescanned online. Offline and online components together are required to generate a complete scan of the surface filling the anomalous regions and also marginally improving accuracy due to the addition of more high-quality points.

We also summarize the values from the point deviation map in the Table 1 for all the parts. The average and maximum error across all the points for a part are computed and presented in millimeters. We can observe similar improvements in the error values using our approach (offline + online) as compared to baseline-1 and baseline-2.

High average and maximum errors are generated for parts manufactured using reflective material. Parts A and E are examples of such materials. Errors also increased in the regions having double curvature or cavities. Double curvature offers challenges to the planner to align the camera very close to the surface normals since a good estimation of the surface normals is not available during the scan. The poor surface normal estimation is due to localization and manufacturing errors that create uncertainty in the location of the part. A good quality construction is obtained at single curvature and mildly curved convex regions. We expect the observations to extend to a wide range of similar geometries as our test cases cover commonly used industrial parts.

8.2.2. Planning performance

An alternate way to guarantee a high density of points across the part is to first scan the part using an offline plan, then identify the anomalous regions, and lastly, generate another offline plan to scan the anomalous region. The process is repeated iteratively until the density is improved across the entire part. The insights into the effectiveness of using an online approach over the iterative offline approach are provided by comparing against the iterative offline + replanning method as a baseline. Three iterations of replanning paths over anomalous regions were sufficient to cover the entire part and produce the same quality of output pointcloud $\mathcal{P}$. Part B was used for obtaining the Table 2 that

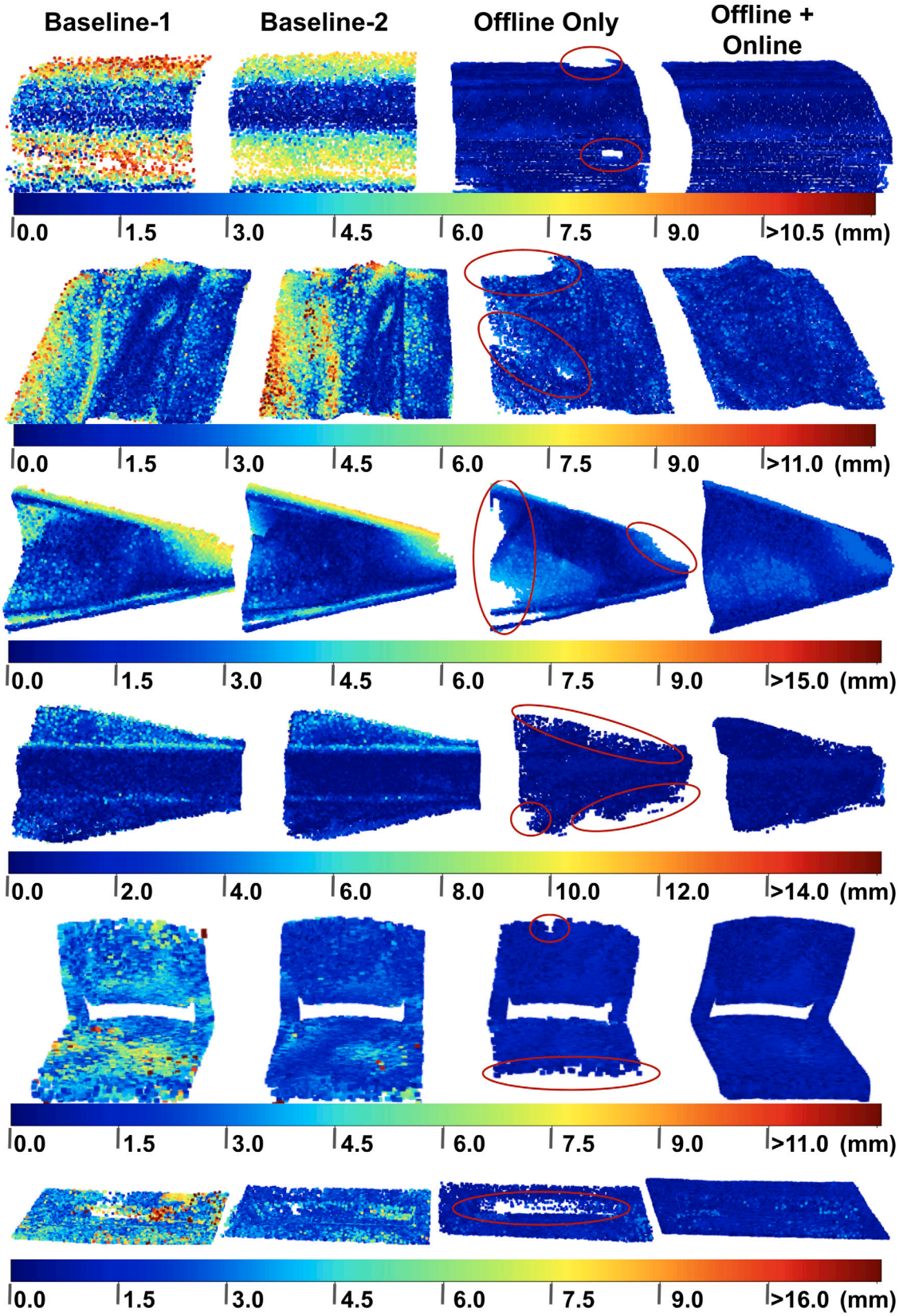


Fig. 10. Point deviation error plotted as a colormap across the surface of all the test parts used in our work. The deviation errors are computed in mm. Baseline approaches 1 and 2, offline only execution, and offline + online execution approaches are used to present the colormaps. The circled contours on the pointclouds generated using offline only method indicate zero or low density point regions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Path lengths in meters for offline+replan approach used for comparing against the offline+online approach are presented. Offline only path length are presented as a reference for obtaining pointclouds without filling the anomalous regions.

Part	Offline + Replan	Offline	Offline + Online
A	4.1	2.7	3.1
B	12.2	6.8	8.5
C	9.3	4.7	6.7
D	15.3	10.6	12.1
E	10.2	6.5	7.8
F	17.9	9.5	13.6

provides the lengths of the paths (used as a surrogate for execution time) for offline + replan offline execution, and the offline + online execution.

We can observe that the path length for offline + replan is higher than offline + online due to the execution of multiple passes required to acquire the desired pointcloud. The path length of offline + online is lower than offline + replan, but higher than offline only. Although, the offline-only approach is not acceptable since regions with zero or low density of points are present.

The savings in correcting the density online compared to iteratively executing replanned trajectories is higher for parts B and C. Parts B and C were found to have multiple anomalous regions with poor point density. We can expect the behavior whenever multiple anomalous regions are found since the robot has to finish execution and then travel back to all anomalous regions from the home location as compared to executing shorter paths in online execution. Parts A, E, and F required longer trajectories even for the online planner and had one anomalous region which led to similar execution times for replanning and online methods. We can expect fewer improvements in path lengths when fewer anomalous regions are present.

8.3. Performance analysis

8.3.1. Selection of point admittance function η

Our *eta* function uses bounds on α, β, and γ as $[0.26, 0.4], [0, 0.2]$, $[0, 1.0]$ respectively. A more sophisticated η function can also be used with our method which will admit points more effectively based on the sensor capabilities. The selection of η is important since it trades off the accuracy of the measurement with respect to the time taken for process execution. A very conservative η function will reject several points and execution times will be very high as compared to the gains in accuracy and vice versa. We will present an experiment conducted using part B that shows the trade-off.

Fig. 11 shows the relationship between choice of η function and accuracy of the measurement using deviation error values for part B. Average deviation over all the points of the measured cloud $\tilde{\mathcal{P}}$ and true cloud $\mathcal{P}$ are used along the Y-axis of the plot and camera path length along X-axis for five different cases of η function. The bounds over α, β, and γ for the five η functions $\eta_1, \eta_2, \eta_3, \eta_4$, and η_5 are $\alpha_1 : [0.0, 0.9], \beta_1 : [0, 0.8]$, and $\gamma_1 : [0, 3.14]$, $\alpha_2 : [0.1, 0.7], \beta_2 : [0, 0.6]$, and $\gamma_2 : [0, 2.5]$, $\alpha_3 : [0.2, 0.6], \beta_3 : [0, 0.5]$, and $\gamma_3 : [0, 2.0]$, $\alpha_4 : [0.3, 0.5], \beta_4 : [0, 0.4]$, and $\gamma_4 : [0, 1.5]$, $\alpha_5 : [0.3, 0.4], \beta_5 : [0, 0.2]$, and $\gamma_5 : [0, 1.0]$ respectively. We can observe that as the bounds over the parameters for which a point will be accepted get conservative, the average error across the output reduces. However, the reduction in error is diminishing with more conservative η function at the cost of high gains in path length.

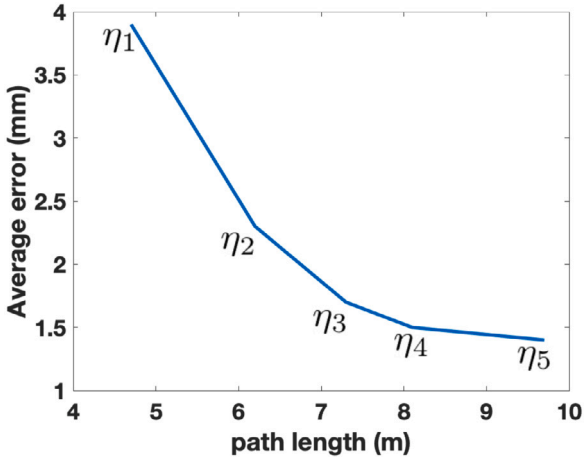


Fig. 11. The relationship between average error and path length found by using five different bounds in the η function. The functions η_1 to η_5 successively constrain the acceptable range of the α, β, and γ showing an increase in path length. However, the corresponding reduction in average error across the output pointcloud diminishes eventually.

Therefore, an optimal η function should be selected that provides the desired accuracy with least path length.

Any η function that provides more conservative bounds will admit fewer points in the data. The camera error is high for points that are at the boundary of the operational range and reduces as they get closer to the center of the operational range. The reduction in error is not linear. Reduction in error is higher in the beginning and gradually decays. Hence, making the η function more conservative is unnecessary. We recommend using more sophisticated learning-based models to find the non-linear relationship and provide reasonable bounds on the camera parameters. We also expect similar behaviors when other sensors are used with our approach.

8.3.2. Effects of N on performance

We use four different values of N as $N_1 = 20, N_2 = 60, N_3 = 100, N_4 = 225$ that is 0.1, 0.4, 0.7, and 1.5 times the number of points used to generate the results presented earlier for part B. Fig. 12 shows how the average error across all the points in output cloud $\tilde{\mathcal{P}}$ for part B varies with each of the N selected. The error reduces as the number of points increases in the data. However, a further increase in N past a value will result in increased computational efficiency and higher execution time with little gains in error reduction. The figure also provides insights into issues with the selection of a lower value of N causing insufficient points and loss of accuracy.

The variance of point error reduces by factor of $\sqrt{N}$ according to the relationship $\sigma_{new} = \sigma_{old}/\sqrt{N}$. σ_{new} will decrease faster as N increases at the beginning leading to lower values of average error in the plot. A similar relationship will be obtained irrespective of the part used. However, the reduction in error value saturates with higher values of N as the camera will not produce zero error results and will always have some error value for a point. Larger values of N will simply converge to this small value of error. However, the execution time will significantly increase with larger N so a value of 30–200 points within a $\mathcal{N}$-neighborhood is recommended.

8.3.3. Consistency in performance

Part B was also used to perform multiple consecutive 3D reconstructions to evaluate the consistency in the scans. Five different measurements were taken and the average and maximum error in millimeters across all the points are presented in the Table 3. Five measurements are also shown next to one another as a point deviation colormap in Fig. 13. The scale of the error used for the colormap is different from the

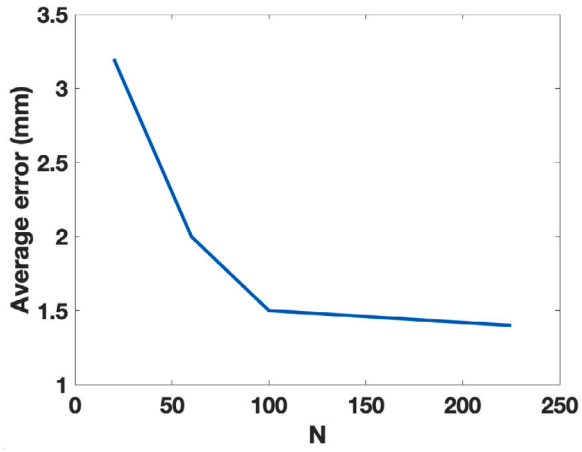


Fig. 12. Diminishing reduction in average error across the entire output pointcloud for part B is observed with increasing the minimum number of observations N for a $\mathcal{N}$-neighborhood. Additionally, a lower value of N will also lead to a high average error.

Table 3
Five measurements of average and maximum error across all points for part B are presented.

Scan Number	Average Error (mm)	Maximum Error (mm)
1	1.61	3.5
2	1.65	3.5
3	1.48	3.2
4	1.70	3.9
5	1.56	3.4

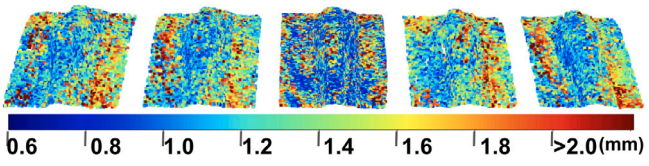


Fig. 13. Point deviation colormap for five consecutive scans of part B. Left to right correspond to scan numbers 1–5 in Table 3. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

scale previously used to compare the baseline approaches. Therefore, we notice high error red points on the deviation map. However, the red points are within the maximum error reported for part B.

9. Conclusions and future works

We have demonstrated that by characterizing the depth sensor properties using an admittance function only high-quality points were accepted during planning and execution. Camera performance model constraints were developed to also introduce diversity and a minimum number of admitted points in local neighborhoods to produce a reduced error output. Additional kinodynamic robot constraints were developed for generating robot trajectories that can execute paths to capture the high-quality pointclouds. An online planning algorithm was presented to guarantee a uniform point density throughout the part by covering the anomalous regions. The contributions from this work can also be applied to a wide range of sensors used in the industry.

We will develop a more sophisticated point admittance function using a machine learning model to find how execution times can improve with similar or higher accuracy. We will need to develop a better method for camera calibration and DH parameter estimation for the robot which was not done in this work to further optimize the accuracy of the output cloud. The camera path planning algorithm presented in this work takes the robot to the vicinity of visited regions in the future since a minimum number of observations need to be met. Execution times can be improved further by rewiring such paths and reducing the path length.

CRediT authorship contribution statement

Rishi K. Malhan: Conceptualization, Methodology, Software, Writing – original draft, Formal analysis, Visualization. **Satyandra K. Gupta:** Conceptualization, Investigation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported in part by National Science Foundation Grant #1925084. Opinions expressed are those of the authors and do not necessarily reflect the opinions of the sponsors.

Appendix. Camera localization process

The transform ${}^{b}T_{c}$ is needed to obtain points from the camera frame in the robot base or global frame. We can write the transform ${}^{b}T_{c}$ as ${}^{b}T_{e} *^{e} T_{c}$. ${}^{b}T_{e}$ is the robot end-effector transform with respect to the robot base which is determined by the forward kinematics. ${}^{e}T_{c}$ is initially estimated by the user based on the CAD model of the mount used to fix the camera to the end-effector. The estimate can also be obtained by measuring the location of the camera frame with a ruler or caliper. The estimated transformation is inaccurate and is only used by our algorithm as an input to a multi-stage optimization process that finds the improved estimate of this transform, ${}^{e}T_{c}^{*}$. The following steps describe the multi-stage optimization process.

We move the robot to different camera poses around a rectangular glass plate and capture the surface pointclouds. The camera poses follow the acceptable range constraint from the camera performance model and points within the field of view are accepted. Outlier points are removed using depth-based filters and box filters and downsampled using a 5 mm voxel size.

All the captured pointclouds must coincide after transforming to the base frame since they belong to the same planar surface. However, due to the error in estimated ${}^{e}T_{c}$, the pointclouds do not coincide. Fig. 14 illustrates how the pointclouds may look like in relation to a best-fit plane computed for all the points using the least square fit.

Every point in the clouds deviates from the best-fit plane with a value measured by the shortest distance from the plane. The average of the distances for all the points is taken as an error for the multi-stage optimization process. The algorithm aims at finding an optimal ${}^{e}T_{c}$ that minimizes the average deviation error. We will now discuss the multi-stage optimization algorithm.

1. *Decision Variable:* Initial transformation ${}^{e}T_{c}$ estimated by the user is converted to a 6 dimensional vector $\vec{x} = \langle x, y, z, r, p, y \rangle$ where $\langle x, y, z \rangle$ are the positional coordinates of the camera frame O_c origin and $\langle r, p, y \rangle$ describe the Euler angles for unit X, Y, and Z vectors of O_c. The vector $\vec{x}$ is used as the decision variable.
2. *Stage 1:* A discrete optimization-based method is used as the first stage. The method uniformly samples around the vector $\vec{x}$ in discrete steps. The objective function used is the average over deviation distances of the points in the cloud from the best-fit plane.. The objective function is evaluated at these samples and a fixed number of good

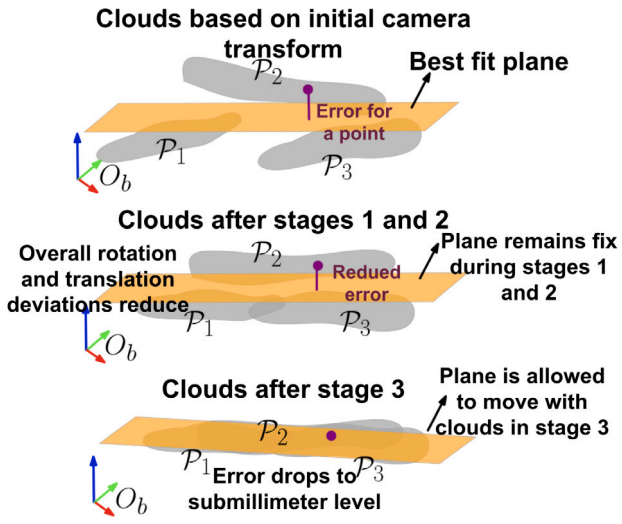


Fig. 14. $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ illustrate pointclouds collected by using user provided estimate ${}^{b}T_c'$. The clouds do not coincide in the beginning (top). Optimization stages 1 and 2 keep the plane fixed and reduce the deviations (center). Optimization stage 3 finds optimal ${}^{b}T_c^*$ and by allowing the plane to move (bottom).

samples having lower values of the objective function are selected as initial guesses for the next stage of optimization.

3. *Stage 2:* The second stage uses a gradient descent algorithm over the selected good initial guesses from stage-1. The objective function is the same as the stage-1. Gradient descent is conducted using every initial guess and the corresponding optimal decision variables are found. The optimal decision variable corresponding to the lowest objective function value is chosen as the initial guess to stage-3 of the algorithm.

Fig. 14 shows the output from Stages 1 and 2 of the algorithm. The clouds still do not coincide with the plane P as we kept it fixed during the stages. Stage-3 will now allow the plane P to move and the plane parameters will be a part of the optimization decision variable.

4. *Stage 3:* This stage extends the vector $\vec{x}$ by including the plane parameters and finds an optimal value $\vec{x}^*$ that minimizes the objective function. The objective function is same as the stage-1. Fig. 14 shows output from the stage-3 of optimization. The best-fit plane now closely coincides with the pointclouds.

The output $\vec{x}^*$ is then is converted to homogeneous transform ${}^{e}T_c^*$. ${}^{e}T_c^*$ will allow all the pointclouds to be determined with respect to O_b such that the error between the clouds and the true plane is minimum. We use this transformation for getting the pointclouds during the 3D reconstruction process. For simplicity, we will denote ${}^{e}T_c^*$ as ${}^{e}T_c$. We will now discuss the 3d reconstruction algorithm that uses the optimal camera transformation and the determined camera performance model to generate robot trajectories and capture the pointclouds.

References

[1] Rishi Malhan, Rex Jomy Joseph, Prahar Bhatt, Brual Shah, Satyandra K. Gupta, Fast, accurate, and automated 3D reconstruction using a depth camera mounted on an industrial robot, in: ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2021, 2021.

[2] Rishi Malhan, Rex Jomy Joseph, Prahar M. Bhatt, Brual Shah, Satyandra K. Gupta, Algorithms for improving speed and accuracy of automated three-dimensional reconstruction with a depth camera mounted on an industrial robot, Journal of Computing and Information Science in Engineering (ISSN: 1530-9827) 22 (3) (2022) 031012, http://dx.doi.org/10.1115/1.4053272.

[3] Emile Glorieux, Pasquale Franciosa, Dariusz Ceglarek, Coverage path planning with targetted viewpoint sampling for robotic free-form surface inspection, Robot. Comput.-Integr. Manuf. 61 (2020) 101843.

[4] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, Roland Siegwart, Receding horizon path planning for 3D exploration and surface inspection, Auton. Robots 42 (2) (2018) 291–306.

[5] J. Irving Vasquez-Gomez, L. Enrique Sucar, Rafael Murrieta-Cid, Efrain Lopez-Damian, Volumetric next-best-view planning for 3D object reconstruction with positioning error, Int. J. Adv. Robot. Syst. 11 (10) (2014) 159.

[6] Roberto Raffaeli, Maura Mengoni, Michele Germani, Ferruccio Mandorli, Off-line view planning for the inspection of mechanical parts, Int. J. Interact. Des. Manuf. (IJIDeM) 7 (1) (2013) 1–12.

[7] Wei Jing, Coverage Planning for Robotic Vision Applications in Complex 3D Environment (Ph.D. thesis), Carnegie Mellon University, 2017.

[8] Héctor González-Banos, A randomized art-gallery algorithm for sensor placement, in: Proceedings of the Seventeenth Annual Symposium on Computational Geometry, 2001, pp. 232–240.

[9] Mustafa Devrim Kaba, Mustafa Gokhan Uzunbas, Ser Nam Lim, A reinforcement learning approach to the view planning problem, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6933–6941.

[10] Christian Landgraf, Bernd Meese, Michael Pabst, Georg Martius, Marco F. Huber, A reinforcement learning approach to view planning for automated inspection tasks, Sensors 21 (6) (2021) 2030.

[11] Carmelo Mineo, Stephen Gareth Pierce, Pascual Ian Nicholson, Ian Cooper, Robotic path planning for non-destructive testing–a custom matlab toolbox approach, Robot. Comput.-Integr. Manuf. 37 (2016) 1–12.

[12] P. Olivieri, L. Birglen, X. Maldague, I. Mantegh, Coverage path planning for eddy current inspection on complex aeronautical parts, Robot. Comput.-Integr. Manuf. 30 (3) (2014) 305–314.

[13] Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, Jorge Dias, Guowei Cai, A survey on inspecting structures using robotic systems, Int. J. Adv. Robot. Syst. 13 (6) (2016) 1729881416663664.

[14] Siyan Dong, Kai Xu, Qiang Zhou, Andrea Tagliasacchi, Shiqing Xin, Matthias Nießner, Baoquan Chen, Multi-robot collaborative dense scene reconstruction, ACM Trans. Graph. 38 (4) (2019).

[15] Georgios Papadopoulos, Hanna Kurniawati, Nicholas M. Patrikalakis, Asymptotically optimal inspection planning using systems with differential constraints, in: International Conference on Robotics and Automation, IEEE, 2013, pp. 4126–4133.

[16] Petr Janoušek, Jan Faigl, Speeding up coverage queries in 3D multi-goal path planning, in: International Conference on Robotics and Automation, IEEE, 2013, pp. 5082–5087.

[17] Brendan Englot, Franz Hover, Sampling-based coverage path planning for inspection of complex structures, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 22, 2012, p. 392.

[18] Brendan Englot, Franz S. Hover, Sampling-based sweep planning to exploit local planarity in the inspection of complex 3D structures, in: International Conference on Intelligent Robots and Systems, IEEE/RSJ, 2012, pp. 4456–4463.

[19] Keld Helsgaun, An effective implementation of the lin–kernighan traveling salesman heuristic, European J. Oper. Res. 126 (1) (2000) 106–130.

[20] Steven M. LaValle, James J. Kuffner, B.R. Donald, et al., Rapidly-exploring random trees: Progress and prospects, in: Algorithmic and Computational Robotics: New Directions, Vol. 5, 2001, pp. 293–308.

[21] Lydia E. Kavraki, Petr Svestka, J.-C. Latombe, Mark H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, Trans. Robot. Autom. 12 (4) (1996) 566–580.

[22] Shi Bai, Fanfei Chen, Brendan Englot, Toward autonomous mapping and exploration for mobile robots through deep supervised learning, in: International Conference on Intelligent Robots and Systems (IROS), IEEE, 2017, pp. 2379–2384.

[23] Fanfei Chen, Shi Bai, Tixiao Shan, Brendan Englot, Self-learning exploration and mapping for mobile robots via deep reinforcement learning, in: AIAA Scitech Forum, 2019, p. 0396.

[24] R. Jain, Building an environment model using depth information, Computer 22 (06) (1996) 85–88, http://dx.doi.org/10.1109/2.507636.

[25] H. Moravec, Robot spatial perceptionby stereoscopic vision and 3d evidence grids, Perception (1996).

[26] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, Wolfram Burgard, OctoMap: An efficient probabilistic 3D mapping framework based on octrees, 34 (2012) 189–206.

[27] Sheraz Khan, Athanasios Dometios, Chris Verginis, Costas Tzafestas, Dirk Wollherr, Martin Buss, RMAP: a rectangular cuboid approximation framework for 3D environment mapping, Auton. Robots 37 (3) (2014) 261–277, http://dx.doi.org/10.1007/s10514-014-9387-y.

[28] Adrian Hilton, Andrew J. Stoddart, John Illingworth, Terry Windeatt, Reliable surface reconstruction from multiple range images, in: European Conference on Computer Vision, Springer, 1996, pp. 117–126.

[29] Brian Curless, Marc Levoy, A volumetric method for building complex models from range images, in: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, ACM, 1996, pp. 303–312.

[30] Mark D. Wheeler, Yoichi Sato, Katsushi Ikeuchi, Consensus surfaces for modeling 3D objects from multiple range images, in: Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271), IEEE, 1998, pp. 917–924.

[31] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Andrew W. Fitzgibbon, Kinectfusion: Real-time dense surface mapping and tracking, in: 10th International Symposium on Mixed and Augmented Reality, IEEE, 2011, pp. 127–136, http://dx.doi.org/10.1109/ISMAR.2011.6092378.

[32] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, Marc Stamminger, Real-time 3D reconstruction at scale using voxel hashing, ACM Trans. Graph. (ToG) 32 (6) (2013) 1–11.

[33] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, Andreas Geiger, Occupancy networks: Learning 3d reconstruction in function space, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4460–4470.

[34] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al., Local implicit grid representations for 3d scenes, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 6001–6010.

[35] Yu-Shiang Wong, Changjian Li, Matthias Nießner, Niloy J. Mitra, RigidFusion: RGB-D scene reconstruction with rigidly-moving objects, Comput. Graph. Forum 40 (2) (2021).

[36] Aljaž Božič, Michael Zollhöfer, Christian Theobalt, Matthias Nießner, DeepDeform: Learning non-rigid RGB-D reconstruction with semi-supervised data, in: Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 2020, pp. 7002–7012.

[37] Mark W. Spong, Seth Hutchinson, Mathukumalli Vidyasagar, et al., Robot modeling and control, 2006.

[38] Mingyang Li, Zhijiang Du, Xiaoxing Ma, Wei Dong, Yongzhuo Gao, A robot hand-eye calibration method of line laser sensor based on 3D reconstruction, Robot. Comput.-Integr. Manuf. 71 (2021) 102136.

[39] David Coleman, Ioan Sucan, Sachin Chitta, Nikolaus Correll, Reducing the barrier to entry of complex robotic software: a moveit! Case study, 2014, arXiv preprint arXiv:1404.3785.

[40] Tiago Rodrigues Weller, Daniel Rodrigues Weller, Luiz Carlos de Abreu Rodrigues, Neri Volpato, A framework for tool-path airtime optimization in material extrusion additive manufacturing, Robot. Comput.-Integr. Manuf. 67 (2021) 101999.