Algorithms for Improving Speed and Accuracy of Automated Three-Dimensional Reconstruction With a Depth Camera Mounted on An Industrial Robot

Rishi Malhan
Center for Advanced Manufacturing,
University of Southern California,
Los Angeles, CA 90007
e-mail: rmalhan@usc.edu

Rex Jomy Joseph
Center for Advanced Manufacturing,
University of Southern California,
Los Angeles, CA 90007
e-mail: jomyjose@usc.edu

Prahar M. Bhatt
Center for Advanced Manufacturing,
University of Southern California,
Los Angeles, CA 90007
e-mail: praharbh@usc.edu

Brual Shah
Center for Advanced Manufacturing,
University of Southern California,
Los Angeles, CA 90007
e-mail: brualsha@usc.edu

Satyandra K. Gupta
Center for Advanced Manufacturing,
University of Southern California,
Los Angeles, CA 90007
e-mail: guptask@usc.edu

Three-dimensional reconstruction technology is used in a wide variety of applications. Automatically creating accurate pointclouds for large parts with complex geometries usually requires expensive metrology instruments. We are interested in using low-cost depth cameras mounted on commonly available industrial robots to create accurate pointclouds for large parts automatically. Manufacturing applications require fast cycle times. Therefore, we are interested in speeding up the 3D reconstruction process. We present algorithmic advances in 3D reconstruction that achieve a sub-millimeter accuracy using a low-cost depth camera. Our system can be used to determine a pointcloud model of large and complex parts. Advances in camera calibration, cycle time reduction for pointcloud capturing, and uncertainty estimation are made in this work. We continuously capture pointclouds at an optimal camera location with respect to part distance during robot motion execution. The redundancy in pointclouds achieved by the moving camera significantly reduces errors in measurements without increasing cycle time. Our system produces sub-millimeter accuracy. [DOI: 10.1115/1.4053272]

Keywords: computational metrology, manufacturing automation

1 Introduction

Three-dimensional (3D) reconstruction is a process of capturing the shape of a physical object. 3D reconstruction methods find their application in manufacturing, reverse engineering, medical imaging, gaming, entertainment, city planning, and several other industries. We are interested in making algorithmic advances in the automated 3D reconstruction of objects by generating high-fidelity models using low-cost RGB-D (Red Green Blue-Depth) sensors.

We need a fast, automated, accurate, and low-cost 3D reconstruction to handle medium-large scale complex parts in manufacturing inspection. The constructed model is compared with the geometric model of the part to determine the geometric errors. Widely used non-contact-based systems consist of metrology instruments with sophisticated laser scanners. The instruments are manually driven or automated by using accurate encoders in revolute joints. These highly specialized and expensive instruments can provide high accuracy.

Computer vision systems deploy low-cost infrared depth cameras where accuracy is not high. The depth cameras provide an accuracy range of 3–5 mm and are orders magnitude less expensive than high fidelity metrology instruments. However, the depth cameras are susceptible to conditions like imaging distance, environment lighting, high calibration errors, and surface properties like reflectivity and geometric complexity. Variations in these factors in real-world cause uncertainty in measurements. For instance, a larger distance of the camera from the object increases uncertainty in location of points in the pointcloud. However, if the distance is reduced, the camera loses focus. The uncertainties can not be characterized a priori, and we need a technology that advances the algorithmic foundations to capture the uncertainty and increase the system's accuracy.

Our goal is to realize an automated 3D reconstruction system equipped with an RGB-D (color and depth channels) sensor mounted to an industrial robotic arm and capture 3D pointcloud of the part that conforms to geometry with high accuracy. The problem has the following challenges: (i) calibration of the camera frame with respect to a reference frame, (ii) minimizing uncertainty while determining camera locations around the part, (iii) determining a robot trajectory with minimum cycle-time and satisfying constraints that reduce uncertainty in measurements, and (iv) merging the pointclouds in a time and memory-efficient way while accounting for uncertainty in pointclouds.

This paper is based on the work presented by the authors at the 2021 ASME Computers and Information in Engineering Conference. We present a novel algorithm that finds a robot trajectory using a discrete parameter optimization process to minimize the cycle time and maintain a camera distance from the surface, which reduces uncertainty. A pointcloud merging algorithm is presented, which continuously captures clouds while the robot is executing motion. It maintains a memory-efficient data structure to update the output cloud and estimate uncertainty in the measurements. The algorithm enables us to use inexpensive RGB-D sensors like Intel RealSense D415 used in our approach and obtain sub-millimeter measurement accuracy. Our contributions in this paper are as follows: (i) a camera localization method that performs significantly better than existing approaches, (ii) a method to find the robot trajectory that minimizes the cycle time and uncertainty in measurements, and (iii) a method for merging the pointclouds which guarantees a real-time estimation of uncertainty.

Contributed by the Computers and Information Division of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received August 16, 2021; final manuscript received December 9, 2021; published online February 7, 2022. Special Editor: Mahesh Mani.

We provide a theoretical basis to show how our method performs to a sub-millimeter accuracy goal that we desire. This paper is based on the work presented by the authors at the 2021 ASME Computers and Information in Engineering Conference [1].

2 Related Work

2.1 Imaging Configuration Sampling. The robot moves the measurement sensor around the part in order to image it. Camera viewpoints or configurations in workspace have to be generated which can be used to align the camera. Glorieux et al. have used a non-random targeted viewpoint sampling with viewpoint sampling optimization to get the image configurations. They have shown that their approach reduces the inspection cycle time [2].

Bircher et al. proposed a method which plans online in a receding horizon fashion by sampling possible future image configurations in a geometric random tree [3].

Vasquez-Gomez et al. have proposed a next best view algorithm to determine image configurations for recreating an arbitrary object [4]. Raffaeli et al. presents a clustering based approach to optimize the number of viewpoint samples for inspection [5]. Jing used randomized sampling-based and medial object-based methods to sample the image configuration for a higher coverage [6]. González-Banos et al. have developed a random-art gallery algorithm to sample the image configurations [7]. Most of these works are focused on generating imaging configurations that achieve a specific objective. Our work focuses on improving accuracy by introducing a constraint on imaging configuration that minimizes uncertainty. Any method can use the constraint to generate imaging configurations. View point planning problem was posed as a reinforcement learning problem in Refs. [8,9].

2.2 Coverage Path Planning. A coverage plan is generated for robot to take the camera around the part and capture pointclouds at discrete waypoints [10]. The plan generated is optimized for cycle time and other constraints. Dong et al. [11] showed a multi robot collaborative travelling salesman problem (TSP)-based coverage planning. Papadopoulos et al. developed an algorithm called random inspection tree algorithm to perform coverage path planning concurrently while generating the viewpoints [12]. Viewpoints are locations around the part used to align the camera frame. Janoušek has implemented an method to speed up coverage queries, which is helpful in concurrent path planning and viewpoint generation [13]. Englot and Hover introduced a sampling-based subroutine to improve the coverage path by making asymptotically optimal local changes on the feasible path [14]. They have also presented a hybrid planner which uses a random planning procedure to fill in the gaps to get a complete coverage path of complex three-dimensional structures [15]. The popular approaches for solving coverage path planning are Lin-Kernighan traveling salesman heuristic [16], rapidly exploring random trees [17], probabilistic road map [18] and their modifications. The coverage plan in this work is generated by using a discrete parameter optimization method to also incorporate camera distance and velocity constraints. The existing methods do not account for this constraint as the pointclouds are not captured continuously, and thus, camera distance in between viewpoints can be violated. Reinforcement learning has been used for mapping and exploration in coverage planning [19,20].

2.3 Pointcloud Fusion. A popular approach to 3D reconstruction is by using a grid of cubic volumes of equal size, called voxels to discretize the mapped area. The work done in Refs. [21,22] uses voxel-based representation. However, the memory complexity is high in order to effectively map structures at a higher resolution. An alternative solution is to use a hierarchical data structure that was proposed in Refs. [23,24]. An oct-tree-based representation was used in Ref. [23] to offer maximum flexibility with regard to the mapped area and uses a probabilistic occupancy estimation for updating the voxels and to cope with sensor noise. Whereas, an R tree was used in Ref. [24]. However, with higher resolutions, it becomes difficult to integrate all the clouds into the hierarchical data structure, posing problems for real time integration of all the input clouds. Besides the point based methods, an alternative is to use a fully volumetric data structure to implicitly store samples of a continuous function [25–27]. In these methods, depth maps are converted into signed distance fields and cumulatively averaged into a regular voxel grid. The work done in Ref. [28] provides a hashing scheme to overcome the difficulties posed by hierarchical structures and simple voxel grids. Newcombe et al. [29] provide a truncated sign distance function implicit surface representation with a GPU accelerated voxel grid pipeline. Unlike the conventional point, mesh, and voxel-based 3D reconstruction representations, Refs. [30] and [31] employed a 3D geometry-based representation by learning a continuous 3D mapping. Rigid Fusion [32] and Deep Deform [33] create non-rigid 3d models. The existing methods collect pointclouds only at the viewpoints. Due to a small number of viewpoints, the number of pointclouds to be handled is also small. Continuous capturing of pointclouds requires an efficient way to merge the pointclouds without storing all of them in the memory buffer.

3 Background

3.1 Terminology. A reference pointcloud generated using high accuracy metrology instrument is used as a true representation of the part geometry. We denote the reference cloud as $\mathcal{P}$. The reconstructed 3D pointcloud using our algorithm is denoted as $\tilde{\mathcal{P}}$. A pointcloud will contain the set of points seen by the camera lying on any 3D object within view. The point is represented by the coordinates $<x, y, z>$ in the Cartesian space. Accuracy of the 3D reconstruction is defined as the maximum deviation error of the measured output pointcloud $\tilde{\mathcal{P}}$ with respect to $\mathcal{P}$. The deviation is measured in euclidean distance between a pair of corresponding or closest points from both the clouds.

Rigid bodies are located by attaching a local frame O to the body. O defines a pose as the position of the frame origin and orientation of three unit direction vectors along the X, Y, and Z axes. We are concerned with the frames attached to the camera (O_c), part (O_p), robot end-effector (O_e), and robot base (O_b). In this work, we assume O_b to be the global reference. A homogeneous transformation matrix T is used to establish the relationship between frames [34]. For instance, O_c can be defined by using the transformation ${}^{b}T_{c}$ which transforms O_c to O_b. Similarly, the robot end-effector can be located by using ${}^{b}T_{e}$. The system uses several other frames linked in a kinematic tree structure to represent the robot links and environment objects. However, we do not need them for the description of the algorithm.

The robot or a serial-link manipulator is a kinematic tree structure of the different frames attached to rigid links. Revolute joints establish the relationship between links. A N joint robot is also said to have N degrees-of-freedom (DOF). Most industrial manipulators in use have 6 DOF, which corresponds to the manipulator used in our work. The position of all revolute joints is defined by a robot configuration vector $q \in \mathbb{R}^N$. The forward and inverse kinematics relationships are used to find q given O_e and vice-versa. The robot representation and kinematic relationships are stored in a robot object $\mathcal{R}$. Lastly, a robot trajectory τ is a sequence of configurations space points $\tau = \{q_1, q_2, \ldots, q_k\}$, which corresponds to a path followed by the robot end-effector.

3.2 Experimental Setup. The automated robotic inspection system comprises a 6 DOF industrial ABB IRB 2600 robot, an Intel RealSense D415 RGB-D sensor, and a custom 3D printed housing to connect the sensor to the robot end-effector rigidly. We use Hexagon absolute arm with an integrated laser scanner to generate the reference pointcloud $\mathcal{P}$ and assess the performance

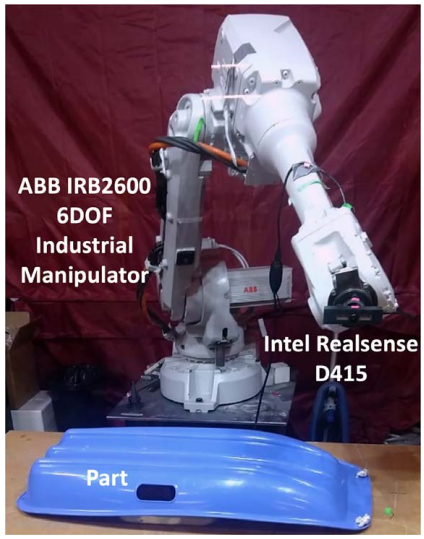

Fig. 1 Robotic inspection cell using a 6 DOF industrial arm with an Intel RealSense D415 RGB-D sensor mounted to the end-effector

of our system. The accuracy of the laser scanner is 0.1 mm. The system's software is based on Robot Operating System (ROS) interface written in c++. We used the ROS-1 and $C++17$ versions for the computations on a core i5 machine with 32 GB RAM. Figure 1 illustrates the setup for our work.

3.3 Acceptable Imaging Configurations. A grid of pixels represents an image of the observable area at a resolution specific to the camera. An RGB-D camera maps the observable area in a 2D RGB image and generates depth information or euclidean distance from the origin of O_c to each pixel visible to the camera. The grid with all euclidean distance values is also known as a depth map. The depth map is then converted to a pointcloud which contains a set of points in the Cartesian coordinate system of the camera O_c. The camera frame points can be converted to the world coordinate frame O_b using appropriate rigid body transformation. A geometric transformation that uses intrinsic K and extrinsic properties of the camera such as optical center and focal length are used for the conversions (refer Fig. 2(*b*)).

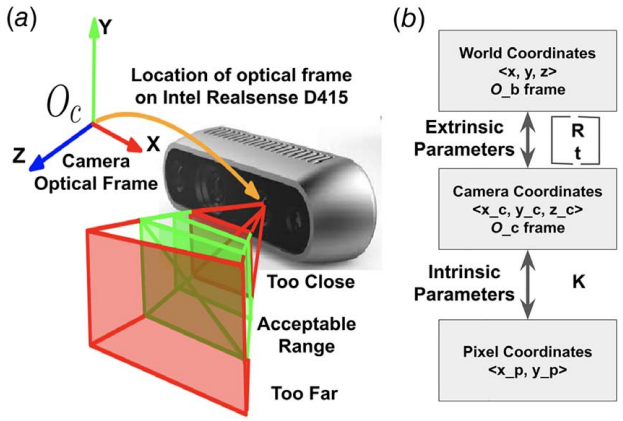

Fig. 2 (*a*) D415 depth camera is illustrated. The camera frame O_c is located at one of the depth sensors as shown. The depth map is a square pixel grid and the frustrum shows the grid extended along the Z direction of O_c. The three segments of the frustrum; too close, acceptable range, and too far illustrate the typical camera field of view and (*b*) a block diagram showing the relationship between conversion of coordinates between world, camera, and pixel frames.

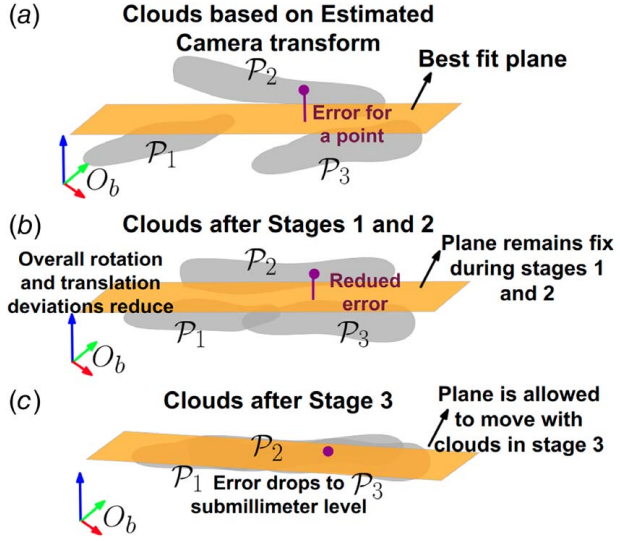

Fig. 3 $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ are example pointclouds collected by using estimate ${}^{b}T_c'$: (*a*) the clouds do not coincide in the beginning, (*b*) optimization stages 1 and 2 keep the plane fixed and reduce the deviations, and (*c*) optimization stage 3 finds optimal ${}^{b}T_c^*$ and by allowing the plane to move

We are interested in minimizing the uncertainty due to inappropriate camera distances from the object. The depth map comprises a fixed number of pixels given by the camera resolution. Larger distances from the objects will lower the density of pixels used to represent the scene resulting in inaccuracy and loss of surface features. Closer objects have a lower uncertainty, but there is a minimum distance below which the camera view is obscured, dependent on the focal length. The range in which the pointcloud obtained has the least amount of uncertainty is an acceptable range ($\vec{R}_a$), and the corresponding location of the camera in the robot base frame O_b is an *acceptable imaging configuration*. The camera performance model that we use accounts for this acceptable range constraint. We found a range of $\vec{R}_a = <30, 45>$ cm for any point on the object to be acceptable. Figure 2(*a*) illustrates the region within which any captured point will have good accuracy.

4 System Overview

The entire process for automating the 3D reconstruction of the part under consideration as a pointcloud $\tilde{\mathcal{P}}$ is performed using the following steps:

(1) The process starts by first estimating the camera frame's transformation O_c with respect to the robot base frame O_b. The camera localization module finds this transform.
(2) We need minimum number of locations around the part through which the camera needs to pass to view the entire surface. The imaging configuration generation module finds such discrete locations.
(3) Robot trajectory generation module connects the camera locations by finding robot configuration space paths. The coverage plan is found by minimizing the execution time and under constraints.
(4) Pointcloud merging module triggers pointcloud capturing continuously when the robot executes the motion and processes them in parallel. A merged pointcloud is the output $\tilde{\mathcal{P}}$.

${}^{b}T_c$ transforms the coordinates of all the points in O_b. The camera localization module is responsible for finding ${}^{b}T_c$. Traditional methods to find this transformation have underlying inaccuracies that propagate to $\tilde{\mathcal{P}}$. We present a fast and accurate camera calibration method in Sec. 5. The next step is to position the camera around the part and capture pointclouds. The system identifies discrete

poses, which are called imaging configurations of the camera represented by frames. The imaging configuration generation module generates a minimum number of such configurations. We will discuss the method in detail in Sec. 6. We then connect the imaging configurations in a sequence. A robot trajectory generation module is responsible for connecting and finding a joint configuration trajectory provided to the robot. We will discuss the method in detail in Sec. 7. The last step constitutes capturing the pointclouds as the robot executes the motion. We present a framework in which we continuously capture the clouds and reduce measurement uncertainty. We will discuss the method in detail in Sec. 8. We derive the theoretical foundations that explain the lower error using our approach in Sec. 9.

5 Camera Localization

Camera localization amounts to determining the transformation ${}^{b}T_{c}$. The robot kinematic model gives the transformation from the robot end-effector to the base ${}^{b}T_{e}$. We assume that ${}^{b}T_{e}$ is determined accurately by a robot calibration procedure and can be used to compute ${}^{b}T_{c}$ as ${}^{b}T_{e}{}^{e}T_{c}$. The conventional approach to calibration is to fix few fiducial markers in the robot workspace. The positions of the markers are known with respect to the robot base O_b. The markers are seen from the depth camera, and their positions are recorded in the camera frame O_c. A homogeneous system of equations is obtained and solved using a singular value decomposition method to find the ${}^{b}T_{c}$. The transformation can then be used to obtain the matrix ${}^{e}T_{c}$. The approach requires modification of the end effector to probe the markers and is prone to human errors. We observe errors of 3–5 mm using this calibration procedure. We have formulated a novel camera calibration approach as a multistage optimization problem in this work. We will now describe the step by step approach for camera calibration from Secs. 5.1 to 5.3.

5.1 Estimate Transformation ${}^{e}T_{c}$. We need an estimate of ${}^{e}T_{c}$ denoted as ${}^{e}T_{c}'$ in order to initialize the calibration algorithm. The estimate ${}^{e}T_{c}'$ is determined by the user and provided as input. Since we design the housing, we know the poses of the camera and housing frames used to find ${}^{e}T_{c}'$. The measurements can be made using rulers or calipers.

5.2 Collect Calibration Pointcloud. The next step is to collect pointclouds of a planar surface (e.g., the surface of the table on which the robot is mounted) from diverse camera poses. The robot is moved such that the poses correspond to the acceptable range $\vec{R}_a$. A pointcloud is captured for each imaging configuration with respect to O_c. Outlier points are removed using depth-based filters and box filters and downsampled. The discretization used in our voxel filter is 5 mm in X, Y, and Z directions.

5.3 Perform Multi-Stage Optimization. The pointclouds should ideally coincide when clouds are transformed using ${}^{e}T_{c}'$ to the robot base as they belong to the same plane (table surface in our case). However, they do not coincide due to errors in ${}^{e}T_{c}'$. Figure 3 illustrates the high deviation for three sample pointclouds when we use ${}^{e}T_{c}'$. The deviation is computed by fitting a least-square plane and finding shortest distance of a point to the plane (see Fig. 3). Therefore, our method outputs optimal ${}^{e}T_{c}*$, which will correspond to accurate camera calibration.

We use a multi-stage optimization approach to find the optimal ${}^{e}T_{c}*$ to reduce error in calibration. The estimated ${}^{e}T_{c}'$ is used as an initial guess for the optimization approach described as follows:

(1) *Objective Value:* We fit a plane P by using least square methods to the pointclouds transformed using ${}^{e}T_{c}'$. The average of shortest distance (deviation) of all the points in all the pointclouds to P is the objective value.

(2) *Decision Variable:* ${}^{e}T_{c}$ is converted to a 6 dimensional vector $\vec{x} = <x, y, z, \alpha, \beta, \gamma>$ where $<x, y, z>$ are the positional coordinates of the camera frame O_c origin and $<\alpha, \beta, \gamma>$ describe the Euler angles for unit X, Y, and Z vectors of O_c. The vector $\vec{x}$ is used as the decision variable.

(3) *Stage 1:* The first stage is a discrete optimization stage. ${}^{e}T_{c}'$ is converted to initial guess $\vec{x}_0$ and fed to the discrete optimizer. The optimizer samples discrete values around the initial point in space for exploitation and a set of random values within pre-defined bounds on $\vec{x}$ for exploration. We end up with a new value of transform $\vec{x}_1$, which becomes the second stage's initial guess.

(4) *Stage 2:* The second stage consists of further minimizing the error for the transformation vector using gradient descent algorithm in continuous space using $\vec{x}_1$ as an initial guess. The algorithm converges to the local minimum and exploits the neighborhood of $\vec{x}_1$. Figure 3 shows the output from Stages 1 and 2 of the algorithm. The error is minimized, but the clouds still do not coincide with the plane P as P was generated from original clouds having a high error and is not the true P. Therefore, we allow P to move to the final stage of optimization. The optimal value $\vec{x}_2$ from this stage is used as an initial guess in the final stage.

(5) *Stage 3:* In addition to the vector $\vec{x}_2$, we also use the plane parameters as the decision variable. This stage relaxes the plane with which point distances were being evaluated. The optimum value $\vec{x}^*$ is converted to homogeneous transform ${}^{e}T_{c}*$. Figure 3 shows the output generated where deviation error is minimized, and clouds coincide with the plane.

The result of the multi-stage optimization is the best estimate ${}^{e}T_{c}*$, which projects all the pointclouds to a single plane. The rationale is that ${}^{e}T_{c}*$ is the transformation that concurs with the reality that all pointclouds with respect to O_b should appear coplanar as they belong to the planar surface (table in our case). For simplification, we use the notation ${}^{e}T_{c} = {}^{e}T_{c}*$ for rest of the paper.

6 Imaging Configurations Generation

We capture the pointclouds from diverse locations around the part to reduce the uncertainty during pointcloud merging. Sections 6.1 to 6.3 discuss the steps involved in image configurations selection process.

6.1 Voxelize Part Surface. A method to generate locations around the part where the camera will be moved is by sampling throughout the robot workspace. However, the cycle time optimality is not assured, and camera imaging configurations are generated inefficiently. We focus on a framework to generate the imaging configurations by utilizing the part geometry and focusing imaging points. The CAD $\mathcal{C}$ model of the part in the form of a stereolithography (STL) format, inaccurate pointcloud, or a coarse representation like a hemisphere enclosing the part can be used to represent the part location. All formats are converted into an approximate pointcloud denoted as $\mathcal{P}_a$. $\mathcal{P}_a$ is then transformed to the robot base frame O_b by estimating the part's location in the robot base ${}^{b}T_{p}$. The estimation could be as coarse as measuring the part frame pose using rulers and angle gauges. The estimation only helps us get a rough idea of where the part is with respect to the robot. We will use $\mathcal{P}_a$ as the pointcloud transformed into robot base coordinates in further discussion.

A minimum size bounding box around $\mathcal{P}_a$ is fit, and dimensions are determined. The bounding box is then voxelized. Voxelization involves creating a discrete 3D grid that represents the bounding box. We use a voxel side length of 50 mm for voxelization. The voxel size was chosen according to part geometries used for the experiments. A higher resolution can lead to an increase in computation time and vice versa. The resolution was chosen such that

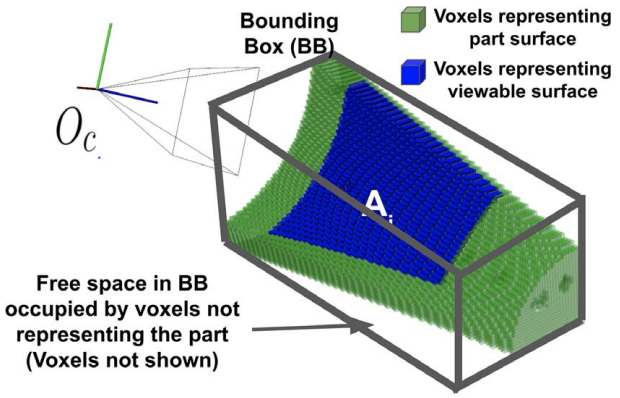

Fig. 4 Minimum size bounding box around the part is illustrated. Free space within the bounding box has empty voxels and are not shown. The voxels representing the part surface are shown. Highlighted voxels denoted as A_i represent the area viewed by the camera at location specified in the image.

computation time was tractable, and enough imaging configurations were present to obtain a feasible set-cover solution. An occupancy grid is generated from this voxelization by highlighting the voxels that enclose at least one point from $\mathcal{P}_a$. The highlighted voxels in the underlying data structure can be viewed as a discrete voxelized part surface. Figure 4 shows a minimum size bounding box around the part and the highlighted voxels within the bounding box representing the part surface. We then use this occupancy grid to sample potential imaging configurations after imposing constraints.

6.2 Generate Feasible Imaging Configuration. Our framework to use the part geometry allows us to introduce acceptable range $\vec{R}_a$ constraint while finding imaging configurations. The feasible imaging configurations are represented by frames O_i, and the set is denoted as $\mathcal{W}$. Position and orientation (pose) specification define O_i in the robot workspace. The pose specification is computed under constraints in the following manner:

(1) *Position Constraint:* The centroid of voxels that belong to the part surface are used as $\mathcal{W}$. So for a grid with M occupied voxels, we have M imaging configurations. Normals at these centroids are computed by taking their neighbors from $\mathcal{P}_a$ into account. The normal direction is chosen such that it comes out of the surface of the part. Each centroid is moved by a distance along corresponding normals by the acceptable camera distance (30 cm in our case).
(2) *Orientation Constraint:* Selecting appropriate orientation for O_i is a challenging task as it involves robot reachability as well. The unit Z-direction vector of frame O_i is chosen to be opposite of the corresponding centroid normal. Unit X and Y direction vectors are chosen such that the X-axis is parallel to the principal axis obtained after computing principal components of points in $\mathcal{P}_a$. Unit Y is obtained by Cartesian axes right-hand thumb rule.

The pose computed under constraints from the steps above cannot be directly used since the robot might not be reachable at these locations. We solve a constrained inverse kinematics problem given by Eq. (1) to find the optimal pose specifications for which the robot is also reachable. The strategy is similar to one we have used in our previous work in Refs. [35,36].

$$q^* \leftarrow \arg\min_q (PE(q, O_i, \mathcal{R})) \tag{1}$$

$$e_{pos} = \| < x_c, y_c, z_c > - < x_i, y_i, z_i > \|_2 \tag{2}$$

$$e_{ori} = \sum_j \exp(b_c^{jT} b_i^j - \psi), \quad j \in \{x, y, z\}, \quad \psi \in \mathbb{R} \tag{3}$$

where q is the robot configuration and $PE(\cdot)$ is the pose error given by summing the position (Eq. (2)) and orientation (Eq. (3)) errors. e_{pos} is the second norm of position errors between the origin of O_c and origin of O_i. e_{ori} is found by allowing some room for the robot to reorient the tool about unit X, Y, and Z axes of O_i. An exact alignment demands the dot product between unit vectors b_c and b_i to be 1. The tolerance ψ takes a scalar value equal to 1 plus a constant equal to inverse cosine of angular tolerance. So, if the robot cannot exactly align O_c with O_i, it can do the best possible alignment within its reachability. The optimization problem is solved for each $O_i \in \mathcal{W}$, and all robot configurations are found. The new frames determined by forward kinematics for each q_i replace the initial $O_i \in \mathcal{W}$.

6.3 Select Near-Optimal Imaging Configurations. The set of imaging configurations $\mathcal{W}$ contains several frames. Using all the frames is redundant as there is overlap between surface areas viewed from the camera. So, we need to determine a reduced set $\mathcal{W}_R \subset \mathcal{W}$ which only selects the minimum number of O_i needed to cover the entire part. The problem can be viewed as a set cover problem. A solution to the set cover problem is found by using a greedy search method that uses a coverage heuristic. The selection of frames is made in a specific order based on this heuristic.

The coverage heuristic determines how much area would be left if seen from an imaging configuration. Consider the total surface area of the part to be A. Solution to the set cover problem gives us $\mathcal{W}_R = \{O_1, O_2, \ldots, O_m\}$, where $A_1 \cup A_2 \cup \cup A_m = A$. Figure 4 illustrates an area that can be viewed from the camera over the part when it is positioned at an imaging configuration O_i. The visible area A_i is highlighted with blue voxels. The coverage heuristic value for each O_i is the area remaining A_i'.

We order all the imaging configurations with a preference $O_i > O_j$, such that $A_i' \leq A_j'$ for any two consecutive i, j configurations in the ordered list. We then start selecting the imaging configurations from the top and move greedily through the ordered list to the end until m configuration frames are selected. The m configuration frames are selected such that $A_1 \cup A_2 \cup \ldots \cup A_m = A$ and m is the minimized. The algorithm is greedy and may not be optimal. However, the heuristic produces a solution very close to the optimal solution. Figure 5(*a*) illustrates m imaging configurations over an example part.

We will now discuss how we compute the area covered A_i. We evaluate the area covered by the camera when positioned at a camera configuration such that $O_c^{-1} I_i = I$ by using a depth map. The occupancy grid is reconstructed from $\mathcal{P}_a$ with a discretization of 5–10 mm for a high fidelity measurement. We can view the centroid of all the voxels in the occupancy grid as a reduced pointcloud. The points are transformed into a 2D depth map of the camera using the extrinsic and intrinsic parameters. The depth map values of the points not visible from the camera are negative and are marked as invalid in the underlying data structure. The points outside the

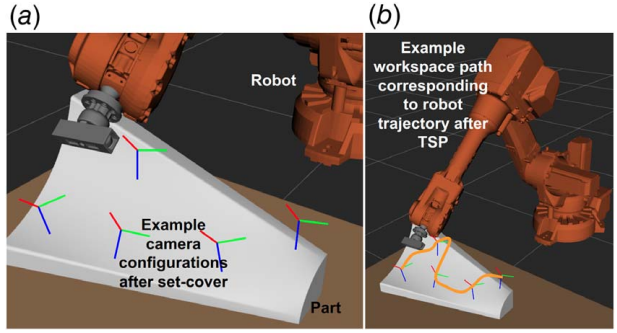

Fig. 5 (*a*) Example optimal imaging configurations that are output of set cover method and (*b*) a workspace path found by TSP connecting all optimal imaging configurations

acceptable range $\vec{R}_a$ are also marked as invalid. The invalid points are unviewable from the camera and constitute A_i'.

7 Robot Trajectory Planning

A robot trajectory $\tau = \{q_1, q_2, \ldots, q_m\}$ needs to be generated to automate the motion through all the imaging frames $\{O_i \in \mathcal{W}_R\}$. Our objective is to minimize the cycle time. The robot's cycle time is computed using a function $\mathcal{T}(\tau)$, which uses the robot joint velocities to compute the time required for traversing the entire path. The non-linear optimization problem of robot trajectory generation under general end-effector constraints is formulated as follows:

$$\tau^* \leftarrow \arg\min_{\tau} \mathcal{T}(\tau), \quad \tau \in Q_f, \quad Q_f \subset Q \tag{4}$$

$$s.t.\ FK(\tau) \rightarrow \mathcal{W}_R \tag{5}$$

$$H(q)_{lb} \leq H(q_i) \leq H(q)_{ub}, \quad q_i \in \tau \tag{6}$$

$$G(q_i) = 0 \tag{7}$$

Q_f is the collision-free robot configuration space which is a subset of the configuration space Q. $FK(\tau)$ is a forward kinematics function which enforces the condition $O_c^{-1} O_i = I$. $H(\cdot)$ defines the lower and upper bounds imposed by robot motors (joint and velocity limits). We represent two constraints as general end-effector constraints ($G(q_i) = 0$); (i) a velocity constraint and (ii) camera acceptable range constraint on origin of the frame O_c. The velocity constraint enforces Cartesian velocity of origin of O_c during the motion to be less than or equal to a maximum value V. The constraint ensures that we move at a pace where pointclouds can be captured without blurs. Sections 7.1 and 7.2 discuss the trajectory generation algorithm in detail.

7.1 Find Workspace Path. We approach the problem of finding a shortest trajectory τ by finding a sequence $\mathcal{S} = \{O_1, O_2, \ldots, O_m\}$ in the robot workspace. Each frame in $\mathcal{S}$ is treated as a node between which edge connections are made by solving a path constrained trajectory generation problem. The problem converts to a TSP version where we need to travel each node only once, and edges have a cost given by the execution time. An adjacency matrix is generated by evaluating all the edge costs. A solution to the TSP is found by using the 2-opt algorithm [37], which eliminates cross-overs by rerouting the path.

A crucial component of solving the TSP is finding a robot trajectory that connects every two waypoints O_i and O_j that the TSP solver queries for cost evaluation. However, the trajectory generation process is computationally expensive. We use an approach that uses a surrogate for evaluating the cost. We use the euclidean distance between every pair of imaging configurations as a surrogate for the true trajectory cost. The surrogate cost is used to create the adjacency matrix, and the TSP is solved. The output of the TSP is the sequence of imaging configurations $\mathcal{S} = \{O_1, O_2, \ldots, O_m\}$. Figure 5(*b*) shows the workspace path connecting the imaging configurations.

7.2 Find Robot Trajectory. We have a robot configuration q_i corresponding to the image configuration O_i that was found when solving the set cover problem. We connect each consecutive pair q_i and q_{i+1} by using the optimization method. A discrete parametric optimization routine uses parametric curves to represent the position versus time profile of each joint in the robot. We use cubic B-spline given by Eq. (8) for j^{th} joint variable. In Eqs. (9) and (10), γ_i are the knots. The knots of a B-spline are the points where the piece-wise polynomials of the B-Spline meet. The map between the arc-length parameter (s) and time (t) is the following, $s = 0$ when $t = t_i$ and $s = 1$ when $t = t_f$. N_{cp} is the number of control points for the one-dimensional spline curve representing q_j, x^j is the vector of control points for q_j, and $R_{i,k}(s)$ are the basis functions parameterized with arc-length parameter s.

$$q_j(s, x^j) = \sum_{i=1}^{N_{cp}} R_{i,k}(s) x_i^j, \ s \in [0, 1] \tag{8}$$

$$R_{i,k+1}(s) = \frac{s - \gamma_i}{\gamma_{i+k} - \gamma_i} R_{i,k}(s) + \frac{\gamma_{i+k+1} - s}{\gamma_{i+1}} R_{i+1,k}(s) \tag{9}$$

$$R_{i,1}(s) = \begin{cases} 1, & \text{if } \gamma_i \leq s < \gamma_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

The i control points denoted as x_j^i for each joint j are used as decision variable in the optimization problem. Overall dimension of the decision variable is $N \times N_{cp}$, where N is the DOF of the robot. The objective function of the optimization routine is a weighted sum of the trajectory time (Eq. (4)) and pose error (Eqs. (2) and (3)). The constraints represented by Eqs. (5)–(7) are also included during optimization. We use a sequential quadratic program (SQP) solver to find a solution to the problem.

8 Pointcloud Merging

The pointclouds are captured continuously as the robot executes the coverage trajectory τ. The camera's frame rate is 30 FPS, and $30 * \mathcal{T}(\tau)$ is the number of pointclouds. Our method is superior to the existing methods because of reduced uncertainty due to the many pointclouds collected over time. We do not restrict the pointcloud capturing to only when the robot is at an imaging configuration O_i. Additionally, the set $\mathcal{W}_R$ from the set cover module is generated such that the imaging configurations are diversely spread around the part surface.

The massive number of clouds collected can quickly consume the memory and become computationally expensive to handle. The time and space complexity is one of the reasons why continuous capturing of clouds has been avoided. We present a framework that can efficiently handle all the pointclouds and merge them into the output cloud $\tilde{\mathcal{P}}$. We use a strategy that processes the pointclouds in real-time when the robot executes τ and only stores a limited number of clouds in the memory. Our system maintains a set of clouds in two separate buffers; (i) live buffer and (ii) upcoming buffer. The clouds in the buffers are captured over a fixed time interval t. We maintain one thread that runs at an interval of 5 s. The pointclouds captured within 5 seconds are used for one round of updates (live buffer). During these 5 s, another set of clouds is stored in the upcoming buffer for the next round of updates. The process keeps repeating in real-time until all pointclouds are processed.

All the information from the live buffer clouds is used to update the output cloud $\tilde{\mathcal{P}}$ using an underlying data structure and an *update rule*. The clouds are then discarded, and the set of clouds from the upcoming buffer is transferred to the live buffer. The upcoming buffer then starts filling again with new clouds being collected by the robot. We will first discuss the underlying data structure in which clouds are processed and then discuss the update rule in this section. Sections 8.1 and 8.2 provide details on the pointcloud merging method.

8.1 Create Voxel Based Data Structure. We discretize the workspace to make it computationally easy to keep track of the pointclouds being captured. The discretization is obtained by using a voxel representation of the minimum size bounding box that surrounds the part. The bounding box is obtained by using the part geometry information. The voxelized box is transformed into the robot base frame using the estimated transformation of the part with respect to the robot base ${}^bT_p$, which the user provides. The process is similar to the one used during set cover-based

imaging configuration generation. Since ${}^{b}T_{p}$ is an estimate, all the captured points may not necessarily lie within the bounding box. We add adequate margins around the bounding box during voxelization so that all the captured points will always be mapped to a unique voxel.

Each voxel will keep track of a point called the centroid, which will be updated using the points mapped to the corresponding voxel. We represent the set of voxels as $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$, where n is the maximum number of voxels with the bounding box. n will depend on the discretization value used and the part dimensions. Accessing voxels is made computationally inexpensive by the use of appropriate data structures like a hash map.

8.2 Map Points to Voxel Structure. Each voxel V_i stores the following information: (i) a centroid c_i, (ii) a surface normal n_i corresponding to c_i, (iii) standard deviation σ_i associated with c_i, (iv) a list of neighboring voxels $\mathcal{N}_i$ which are used in our update rule, (v) set of points P_i corresponding to the live buffer clouds that are contained inside the voxel, and (vi) other algorithm-related variables. The centroid is a key part of the algorithm as all the centroids collectively represent the points in the pointcloud $\tilde{\mathcal{P}}$. Surface normal n_i information and other variables are used as a part of the update rule in our algorithm. The process of mapping a pointcloud to $\mathcal{V}$ and updating the voxel consists of the following steps:

(1) The points in the clouds that belong to the live buffer are first checked for the acceptable range $\vec{R}_a$ of the depth sensor. If a point was collected outside the acceptable range, it is eliminated. Then the transformation ${}^{b}T_{p}$ is applied to all the points to convert them into robot base coordinates.
(2) A closest voxel v is found for each point obtained after filtration. The point is then appended to the corresponding voxel point set P_i.
(3) The algorithm then iterates through all the voxels in V and updates the c_i, n_i, and σ_i.
(4) The clouds in the live buffer are erased, and upcoming buffer clouds are transferred to the live buffer.

It is essential to understand that the update rule discussed in this section is related to an already computed c_i, σ_i, and n_i. We will discuss how these values are computed during the initialization phase later in this section. For simplicity, we assume these values are known and first discuss the update rule.

The inherent uncertainty in the pointcloud coordinates needs to be captured. We use the information given by the neighborhood of a centroid c_i in order to estimate this uncertainty. A coordinate will have the highest uncertainty along the direction from which the camera views it. Uncertainty along a plane perpendicular to this direction (lateral uncertainty) is significantly low. A cylindrical geometry is suitable to provide uncertainty representation in space. We generate a cylinder with the normal n_i as the principal axis. The cylinder is centered at c_i. The radius and height of this cylinder are represented as r and h. A neighborhood size is selected based on the resolution of the camera to account for lateral uncertainty. We choose $r = 1$ mm based on this neighborhood. The height $h =$ 20 mm in our approach is selected based on camera standard deviation of $\sigma = 5$ mm along the direction in which the camera views the point. We obtain the value as $3 \times \sigma + 5 = 20$ mm as the height. All the points that lie within the cylinder defined for V_i are used to update the centroid using the following Eq. (11).

$$c_i^{k+1} = c_i^k + (x_i^j - c_i^k)/n \tag{11}$$

Here, k is the iteration number, c_i is the centroid that corresponds to voxel V_i. n is the number of points in a set P^i_{neigh}. P^i_{neigh} is all the points that are contained within the cylinder generated for V_i. P^i_{neigh} is created by append all the points in sets P_k, $k \in \mathcal{N}_i$. j is the index of a point $x \in P_i$.

The first set of pointclouds collected over 5 s is used to initialize the algorithm variables c_i, σ_i, n_i, and $\mathcal{N}_i$. We compute c_i as an average of points in P_i that have been mapped during the first iteration. PCA of the covariance matrix of a local neighborhood of c_i is used to estimate surface normal n_i. The eigenvalues λ_0, λ_1, λ_2 of the covariance matrix is computed using PCA, where λ_0 is the greatest eigenvalue. The eigenvector associated with λ_0 is taken as the surface normal n_i. To make the process computationally faster, we store the keys to hash the neighboring voxels that will contain the points within the i^{th} voxel cylinder in the set $\mathcal{N}_i$. The keys for voxels along the positive and negative direction of the normal centered at c_i and at a distance $h/2$ are stored in $\mathcal{N}_i$. $\mathcal{N}_i$ remains constant throughout the algorithm runtime. The updates are stopped as the robot execution stops, and no more pointclouds are present in the upcoming buffer.

9 Estimation of Uncertainty

The camera uncertainty is present along the line connecting the origin of O_c and the point being viewed. Consider the direction to be represented as the Z-axis. Error along Z is significantly larger than lateral errors in a plane perpendicular to Z. This occurs because the lateral coordinate reading is computed directly using the pixel location. Simultaneously, the depth values are obtained by projecting a pattern on the surface and measuring the pattern's distortion. We are interested in estimating the uncertainty in depth, i.e., error δz, and develop a theory on why our approach reduces this error for the point.

In our approach, we measure the surface pointcloud with different camera configurations. We get different δz errors for an actual point on the surface. Based on the camera used, these error values can be in the form of a uniform distribution or a Gaussian distribution. For our case, it is assumed in the form of a Gaussian distribution with a zero mean and σ variance. For a Gaussian distribution, σ indicates the distribution width in which approximately 67% of the samples lie. Since we are interested in only the magnitude of the error $|\delta z|$ and 50% samples on either side of the bell curve will have an approximate variance of $0.68 \times \sigma$ [38]. Hence, the average error for a point captured by the camera is given by $0.68 \times \sigma$.

Let us create a new distribution from the existing Gaussian distribution by drawing n samples. The average and standard deviation of the new distribution is denoted by Eqs. (12) and (13), respectively, where s_i is a drawn sample. Based on the properties of the Gaussian distribution, the new distribution will also be a Gaussian distribution. However, the standard deviation σ will be divided by the square root of n.

$$s_{new} = \frac{1}{n}\sum_{i=1}^{n} s_i \tag{12}$$

$$\sigma_{new} = \frac{\sigma}{\sqrt{n}} \tag{13}$$

This is analogous to our approach, where we are collecting pointclouds by sampling from camera distribution. The merged pointcloud can be thought of as a new distribution. Thus based on our discussion, we can estimate the average error of the output cloud δz_o with standard deviation σ_o with respect to the average error δz of the camera distribution with standard deviation σ using Eq. (14).

$$\delta z_o = 0.68 \times \sigma_o = 0.68\frac{\sigma}{\sqrt{n}} \tag{14}$$

We can therefore quantify the error with n. n is the number of points corresponding to a given actual point on the part generated by pointclouds captured during execution. So, n denotes the number of pointclouds collected. We can see that the error is inversely proportional to the square root of n. As the value of n increases, the error in estimating the actual point decreases but not linearly. The gains in reduction of δz_0 obtained will nullify for bigger n. Therefore, we do not need an infinite number of pointclouds to minimize the error. A

finite number of clouds can be determined for which the error is within the acceptable range.

The errors in camera calibration and robot kinematic chain also contribute towards the overall accuracy. We denote these errors by ω. The average error in the output cloud is then corrected for calibration errors by adding ω to obtain Eq. (15) which is the aggregated errors of the process.

$$\delta z_o \leftarrow \delta z_o + \omega \tag{15}$$

10 Results

We use four test cases for testing the performance of our algorithm. The test cases are inspired by parts used in the industry and have complex geometric surfaces. Figure 6 illustrates the four test parts A, B, C, and D used in our work. Part C is inspired by the work done in Ref. [39]. The parts are manufactured using wood or plastic material. We use different color combinations of the surfaces with different reflectivity to evaluate performance over a wide variety of environment and object conditions. The bounding box dimensions of parts A, B, C, and D are $300 \times 600 \times 300$ mm, $440 \times 870 \times 125$ mm, $570 \times 830 \times 95$ mm, and $140 \times 340 \times 85$ mm, respectively.

The coverage planning algorithm includes solving the set cover to find imaging configurations and TSP to find robot trajectory that minimizes the cycle time. We show the significance of optimizing the number of imaging configurations in execution time in Table 1. The total number of imaging configurations (I.C.) generated over the parts before and after applying the set cover method is recorded. We run TSP to find coverage path for both the cases and execution time when using all the imaging configurations is higher than using a minimum number of configurations. The reduction in cycle time is for a single scan. However, the gains are significant when considering a manufacturing process where several parts are involved during production.

Our approach consists of executing the robot trajectory and continuously capturing and processing pointclouds. We benchmark our approach with three baseline methods. The baselines are chosen to provide insights into how proper selection of imaging configurations can reduce the errors. The comparison with baseline methods will also provide insights into the importance of using acceptable camera distance constraint, continuous cloud capturing, and uncertainty reduction used in our approach. We implement a method to capture pointclouds at random imaging configurations and gradually introduce minimum set, acceptable range constraint, and continuous capturing elements to provide insights into how each element improves accuracy.

Baseline-1 (B1) is a method to capture the pointclouds at random imaging configurations around the part without accounting for the camera performance model. Baseline-1 does not capture the clouds within acceptable imaging distance (30–45 cm in our work). Therefore, some of the points would have high uncertainty and produce erroneous measurements. The captured clouds are downsampled and appended together. This baseline will generate the maximum errors and is a very coarse approximation of the part.

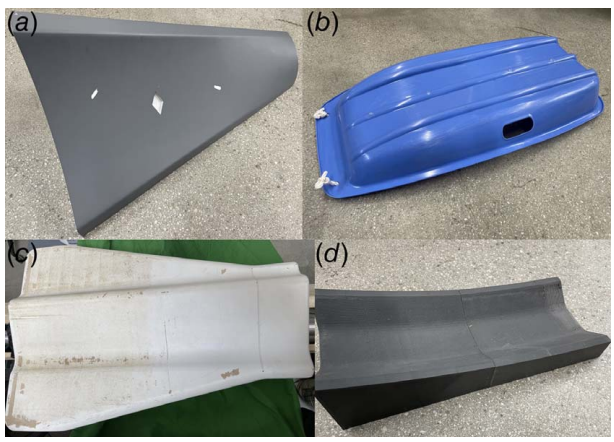

Fig. 6 The four industry inspired parts labeled as A, B, C, and D are shown. The parts are used for performance evaluation of our method. The parts cover a range of surface reflectivity, texture, and complex geometry properties.

Table 1 Total number of imaging configurations (I.C.) generated and minimum number of I.C. required are provided

Part	Total number of I.C. generated	Minimum number of I.C. computed	Cycle time (seconds) for minimum number of I.C.
A	320	18	35
B	288	14	41
C	246	14	33
D	205	10	18

Baseline-2 (B2) captures the pointclouds at the minimum number set of imaging configurations without accounting for the camera performance model. This baseline will signify the importance of using fewer but exhaustive configurations to obtain faster and better results. However, acceptable camera range is not enforced during this step. So, erroneous measurements are expected from the method. All the pointclouds captured at the discrete imaging configurations are then merged, similar to what we do in baseline-1.

Baseline-3 (B3) captures the clouds at the same optimized camera configurations as baseline-2. However, this time we apply the acceptable configuration range constraint to account for camera performance model and eliminate the points not lying at an acceptable range from the camera. This improves measurement accuracy since the points which have a high uncertainty are removed.

Evaluation of the performance of Our Approach (OA) is measured by computing how accurate $\tilde{\mathcal{P}}$ is with respect to $\mathcal{P}$. We overlay $\tilde{\mathcal{P}}$ from each approach on $\mathcal{P}$ and find the deviation error as euclidean norm between corresponding pairs of points. Figure 7 illustrates the deviation errors for all the approaches overlaid on the parts as a colormap. We also provide results showing the maximum and average errors for all the methods in Table 2.

We can observe that the errors in baseline 1 and 2 are high since the camera distance is allowed to vary and not constrained within a small acceptable range. The points farther or closer to the camera sensor have higher uncertainty leading to erroneous readings. Baseline-2 errors reduce compared to baseline-1 since we selectively use imaging configurations instead of capturing erroneous pointclouds from multiple locations. Baseline-3 further improves accuracy as uncertainty drops when we restrict camera distance within an acceptable range and capture clouds from optimized configurations only. We then add efficient data structures to process continuously captured pointclouds and also account for uncertainty by using cylinder geometry during merging. The accuracy significantly improves, and errors drop to acceptable values.

σ for the z-depth error δz is estimated to be 5 mm. The number of pointclouds captured from the camera for parts have a rounded off average of $n = 900$. The error in camera calibration and robot kinematics is estimated as $\omega = 0.7$ mm based on the value of errors from the camera calibration module and robot accuracy values from the manufacturer. The error δz_o is computed using Eq. (15) as 0.811 mm. The average error is a conservative estimation of the average error of 0.725 mm on four test cases we are getting and explains the sub-millimeter accuracy levels. We can notice that the accuracy improvements will not be significant if more pointclouds are collected since most of the error comes from camera calibration and robot inaccuracies.

11 Discussions

The camera captures pointclouds at a rate of 30 frame per minute. We move the robot at a constant speed of 10 mm s^{-1}. Increasing the

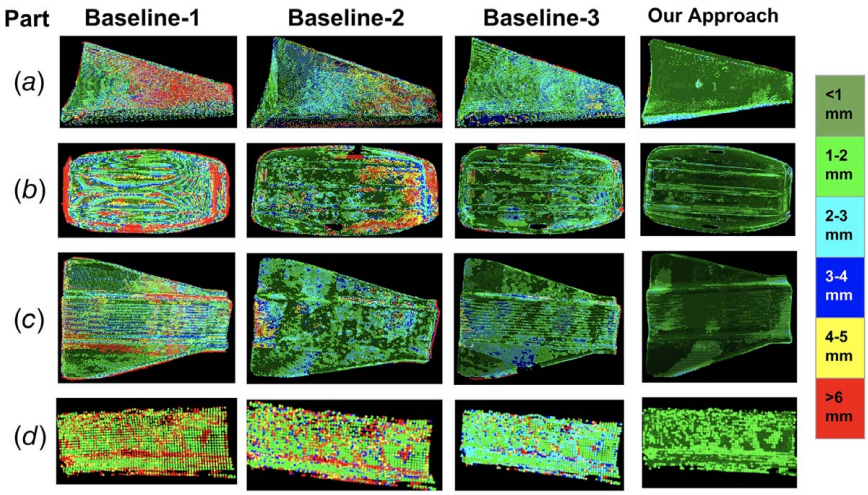

Fig. 7 Deviation errors as a Euclidean norm of distance between corresponding pair of points in $\tilde{\mathcal{P}}$ and $\mathcal{P}$ are shown as a colormap overlaid on the four test cases. The color bar representing the error ranges in mm is shown to the right for reference.

Table 2 Maximum and average errors in mm of the measurements are shown

	Maximum Error (mm)				Average Error (mm)			
Part	B1	B2	B3	OA	B1	B2	B3	OA
A	49.3	22.4	10.9	6.1	3.4	1.6	1.5	0.6
B	110.7	74.1	10.2	5.6	6.7	3.3	2.1	0.7
C	25.2	15.9	11.2	5.7	3.5	2.9	1.5	0.7
D	40.5	29.6	9.5	4.5	14.1	6.7	2.5	0.9

Note: The notation B1, B2, B3, and OA elaborate to baseline-1, baseline-2, baseline-3, and our approach, respectively.

number of imaging configurations in the minimum set will lead to more pointclouds being captured during trajectory execution. However, the execution time will also increase. The decrease in error with an increasing number of pointclouds diminishes over time. Therefore, using a minimum set allows us to keep the execution time low with enough diverse pointclouds captured by the camera to obtain an acceptable accuracy scan.

The uncertainty in the X-Y plane perpendicular to the depth direction is not characterized for the camera. The X-Y coordinates are obtained from the depth image, which is a pixelated grid of z-depth values. The points seen by the camera are mapped to this grid and assumed to belong to a single pixel than be anywhere within a neighborhood of pixels. In our calculations, we consider only the Z-depth uncertainty as the root cause of inaccuracies.

12 Conclusions

We developed memory and time-efficient 3D reconstruction algorithm to construct a pointcloud for a physical object using low-cost depth sensor. We made advances in camera calibration, imaging configuration generation, robot trajectory generation, and pointcloud merging modules of the overall system. The advances in camera calibration resulted from using a multi-stage optimization routine over estimated camera transformation to drive down the error. We introduced the acceptable camera range as a constraint in imaging configuration and trajectory generation algorithms. Discrete parameter optimization with camera distance as a constraint allowed us to generate trajectories where pointclouds can be continuously captured instead of discretely capturing them. We presented theoretical foundations on how a continuous pointcloud capturing introduces coverage redundancy and drives down the overall measurement error. Results were provided on four test cases to evaluate the performance of our algorithm. We found the average measurement accuracy to be 0.725 mm, which meets our sub-millimeter objective.

We plan to develop efficient imaging configuration and trajectory generation algorithms to achieve optimal cycle times in the future. We will develop a method that can further improve camera calibration accuracy since it contributes to most of the error. We will evaluate the performance of our algorithm over a broader range of parts and depth sensors.

Acknowledgment

This work is supported in part by National Science Foundation Grant No. 1925084. Opinions expressed are those of the authors and do not necessarily reflect opinions of the sponsors.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The datasets generated and supporting the findings of this article are obtainable from the corresponding author upon reasonable request.

References

[1] Malhan, R., Joseph, R. J., Bhatt, P., Shah, B., and Gupta, S. K., 2021, "Fast, Accurate, and Automated 3D Reconstruction Using a Depth Camera Mounted on An Industrial Robot," ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2021, Virtual, Aug. 17–19, American Society of Mechanical Engineers.

[2] Glorieux, E., Franciosa, P., and Ceglarek, D., 2020, "Coverage Path Planning With Targetted Viewpoint Sampling for Robotic Free-Form Surface Inspection," Rob. Comput.-Integr. Manuf., **61**, p. 101843.

[3] Bircher, A., Kamel, M., Alexis, K., Oleynikova, H., and Siegwart, R., 2018, "Receding Horizon Path Planning for 3D Exploration and Surface Inspection," Auton. Rob., **42**(2), pp. 291–306.

[4] Vasquez-Gomez, J. I., Sucar, L. E., Murrieta-Cid, R., and Lopez-Damian, E., 2014, "Volumetric Next-best-view Planning for 3D Object Reconstruction With Positioning Error," Int. J. Adv. Rob. Syst., **11**(10), p. 159.

[5] Raffaeli, R., Mengoni, M., Germani, M., and Mandorli, F., 2013, "Off-Line View Planning for the Inspection of Mechanical Parts," Int. J. Interact. Des. Manuf. (IJIDeM), **7**(1), pp. 1–12.

[6] Jing, W., 2017, "Coverage Planning for Robotic Vision Applications in Complex 3D Environment," Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.
[7] González-Banos, H., 2001, "A Randomized Art-Gallery Algorithm for Sensor Placement," Proceedings of the Seventeenth Annual Symposium on Computational Geometry, Medford, MA, June 3–5, Association for Computational Machinery, pp. 232–240.
[8] Devrim Kaba, M., Gokhan Uzunbas, M., and Nam Lim, S., 2017, "A Reinforcement Learning Approach to the View Planning Problem," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, July 21–26, Institute for Electrical and Electronics Engineers, pp. 6933–6941.
[9] Landgraf, C., Meese, B., Pabst, M., Martius, G., and Huber, M. F., 2021, "A Reinforcement Learning Approach to View Planning for Automated Inspection Tasks," Sensors, **21**(6), p. 2030.
[10] Almadhoun, R., Taha, T., Seneviratne, L., Dias, J., and Cai, G., 2016, "A Survey on Inspecting Structures Using Robotic Systems," Int. J. Adv. Rob. Syst., **13**(6), p. 1729881416663664.
[11] Dong, S., Xu, K., Zhou, Q., Tagliasacchi, A., Xin, S., Nießner, M., and Chen, B., 2019, "Multi-robot Collaborative Dense Scene Reconstruction," ACM Trans. Graph., **38**(4), pp. 1–16.
[12] Papadopoulos, G., Kurniawati, H., and Patrikalakis, N. M., 2013, "Asymptotically Optimal Inspection Planning Using Systems with Differential Constraints," International Conference on Robotics and Automation, Karlsruhe, Germany, May 6–10, IEEE, pp. 4126–4133.
[13] Janoušek, P., and Faigl, J., 2013, "Speeding Up Coverage Queries in 3D Multi-Goal Path Planning," International Conference on Robotics and Automation, Karlsruhe, Germany, May 6–10, IEEE, pp. 5082–5087.
[14] Englot, B., and Hover, F., 2012, "Sampling-Based Coverage Path Planning for Inspection of Complex Structures," Proceedings of the International Conference on Automated Planning and Scheduling, Atibaia, São Paulo, Brazil, June 25–29, AAAI Publications, Vol. 22, p. 392.
[15] Englot, B., and Hover, F. S., 2012, "Sampling-Based Sweep Planning to Exploit Local Planarity in the Inspection of Complex 3D Structures," International Conference on Intelligent Robots and Systems, Vilamoura, Algarve, Portugal, Oct. 7–12, IEEE/RSJ, pp. 4456–4463.
[16] Helsgaun, K., 2000, "An Effective Implementation of the Lin–Kernighan Traveling Salesman Heuristic," Eur. J. Oper. Res., **126**(1), pp. 106–130.
[17] LaValle, S. M., Kuffner, J. J., and Donald, B., 2001, "Rapidly-Exploring Random Trees: Progress and Prospects," Algorithmic and Comput. Rob.: New Directions, **5**, pp. 293–308.
[18] Kavraki, L. E., Svestka, P., Latombe, J. -C., and Overmars, M. H., 1996, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," Trans. Rob. Autom., **12**(4), pp. 566–580.
[19] Bai, S., Chen, F., and Englot, B., 2017, "Toward Autonomous Mapping and Exploration for Mobile Robots Through Deep Supervised Learning," International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, Sept. 24–28, IEEE, pp. 2379–2384.
[20] Chen, F., Bai, S., Shan, T., and Englot, B., 2019, "Self-Learning Exploration and Mapping for Mobile Robots Via Deep Reinforcement Learning," AIAA Scitech Forum, San Diego, CA, Jan. 7–11, AAAI Publications, p. 0396.
[21] Jain, R., 1996, "Building An Environment Model Using Depth Information," Computer, **22**(06), pp. 85–88.
[22] Moravec, H., 1996, "Robot Spatial Perceptionby Stereoscopic Vision and 3d Evidence Grids," Perception.
[23] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W., 2012, "Octomap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," Auton. Rob., **34**, pp. 189–206.
[24] Khan, S., Dometios, A., Verginis, C., Tzafestas, C., Wollherr, D., and Buss, M., 2014, "RMAP: a Rectangular Cuboid Approximation Framework for 3D Environment Mapping," Auton. Rob., **37**(3), pp. 261–277.
[25] Hilton, A., Stoddart, A. J., Illingworth, J., and Windeatt, T., 1996, "Reliable Surface Reconstruction From Multiple Range Images," European Conference on Computer Vision, Cambridge, UK, Apr. 15–18, Springer, pp. 117–126.
[26] Curless, B., and Levoy, M., 1996, "A Volumetric Method for Building Complex Models From Range Images," Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, New York, ACM, pp. 303–312.
[27] Wheeler, M. D., Sato, Y., and Ikeuchi, K., 1998, "Consensus Surfaces for Modeling 3D Objects From Multiple Range Images," Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271), Bombay, India, January, IEEE, pp. 917–924.
[28] Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M., 2013, "Real-Time 3D Reconstruction At Scale Using Voxel Hashing," ACM Trans. Graph. (ToG), **32**(6), pp. 1–11 .
[29] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. W., 2011, "Kinectfusion: Real-Time Dense Surface Mapping and Tracking," 10th International Symposium on Mixed and Augmented Reality, Basel, Switzerland, IEEE, pp. 127–136.
[30] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A., 2019, "Occupancy Networks: Learning 3D Reconstruction in Function Space," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, June 16–20, pp. 4460–4470.
[31] Jiang, C., Sud, A., Makadia, A., Huang, J., Nießner, M., Funkhouser, T., 2020, "Local Implicit Grid Representations for 3D Scenes," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, June 13–19, pp. 6001–6010.
[32] Wong, Y.-S., Li, C., Nießner, M., and Mitra, N. J., 2021, "Rigidfusion: Rgb-D Scene Reconstruction With Rigidly-Moving Objects," Comput. Graph. Forum, **40**(2).
[33] Božič, A., Zollhöfer, M., Theobalt, C., and Nießner, M., 2020, "Deepdeform: Learning Non-Rigid Rgb-D Reconstruction With Semi-Supervised Data," Proceedings of the Computer Vision and Pattern Recognition (CVPR), Seattle, WA, June 13–19, IEEE, pp. 7002–7012.
[34] Spong, M. W., Hutchinson, S., and Vidyasagar, M., 2006, Robot Modeling and Control.
[35] Malhan, R. K., Kabir, A. M., Shah, B., and Gupta, S. K., 2019, "Identifying Feasible Workpiece Placement with Respect to Redundant Manipulator for Complex Manufacturing Tasks," 2019 International Conference on Robotics and Automation (ICRA), Montreal, Canada, IEEE, pp. 5585–5591.
[36] Kabir, A. M., Kanyuck, A., Malhan, R. K., Shembekar, A. V., Thakar, S., Shah, B. C., and Gupta, S. K., 2019, "Generation of Synchronized Configuration Space Trajectories of Multi-Robot Systems," International Conference on Robotics and Automation (ICRA), Montreal, Canada, IEEE, pp. 8683–8690.
[37] Croes, G. A., 1958, "A Method for Solving Traveling-Salesman Problems," Oper. Res., **6**(6), pp. 791–812.
[38] Lane, D., 2003, *Online Statistics Education: A Multimedia Course of Study*, Association for the Advancement of Computing in Education (AACE).
[39] Malhan, R. K., Shembekar, A. V., Kabir, A. M., Bhatt, P. M., Shah, B., Zanio, S., Nutt, S., and Gupta, S. K., 2021, "Automated Planning for Robotic Layup of Composite Prepreg," Rob. Comput.-Integr. Manuf., **67**, p. 102020.