IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 8, 2022, accepted May 12, 2022, date of publication May 23, 2022, date of current version June 15, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3176817

A Multiresolution Approach for Large Real-World Camera Placement Optimization Problems

V. ANIRUDH PULIGANDLA, (Member, IEEE), AND SVEN LONČARIĆ, (Senior Member, IEEE)
Image Processing Group, Department of Electronic Systems and Information Processing, Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia

Corresponding author: V. Anirudh Puligandla (apuligandla@fer.hr)

This work was supported by the ImmerSAFE Project under the European Union's (EU's) H2020-MSCA-ITN-2017 Call and is part of the Marie Sklodowska-Curie Actions—Innovative Training Networks (ITN) Funding Scheme under Project 764951.

ABSTRACT There have been numerous attempts at solving the optimal camera placement problem across multiple applications. Exact linear programming-based, as well as, heuristic combinatorial optimization methods were shown to be successful in providing optimal or near-optimal solutions to this problem. Working over a discrete space model is the general practice when solving the camera placement problem. However, discretized environments often limit the methods' usage only to small-scale datasets due to resource and time constraints that grow exponentially with the number of 3D points collected from the discrete space. We propose a multi-resolution approach that enables the usage of existing optimization algorithms on large real-world problems modelled using high resolution 3D grids. Our method works by grouping together the given discrete set of possible camera locations into clusters of points, multiple times, resulting in multiple resolution levels. Camera placement optimization is repeated for all resolution levels while propagating the optimized solution from low to high resolutions. Our experiments on both simulated and real data with grids of varying sizes show that using our multi-resolution approach, existing camera placement optimization methods can be used even on high resolution grids consisting of hundreds of thousands of points. Our results also show that the strategy of grouping points together by exploiting underlying 3D geometry to optimize camera poses is not only significantly faster than optimizing on the entire set of samples but, it also provides better camera coverage.

INDEX TERMS Image decomposition, integer linear programming, heuristic algorithms, pareto optimization.

I. INTRODUCTION

Advanced Driver Assistance Systems (ADAS) that provide surrounding view or top view from vehicles using multiple in-vehicle cameras have recently found lot of interest and applications in the vehicle industry, [1], [2]. Multiple camera systems for vehicle surround-view generally use four wide-angle or fish-eye lens cameras placed on the vehicle to generate new perspectives from the obtained images. For small vehicles, like cars, it is intuitive to place four cameras on four sides of the vehicle. Assembling a multi-camera surround-view system for larger vehicles, such as, construction equipment, is complicated due to large sizes and irregular shapes of these heavy machines. Calibration of multiple cameras is a complex task. It is known that vehicle surround view systems rely on camera calibration, [3], to estimate the cameras' poses with high accuracy. An optimal camera placement setup proposed using, high-resolution, realistic 3D vehicle models can aid in camera calibration by providing an initial accurate estimate of cameras poses in real-world.

Camera placement optimization or optimal camera placement (OCP) is a decades old problem. OCP problems have been applied to a wide range of applications such as, video surveillance, [4], 3D reconstruction of objects, [5], human behaviour monitoring and motion capture systems, [6], [7], OCP with VR interface, [8], and so on. Problem formulation frameworks range from simple single objective, minimization or maximization problems, [9], to more complex, multi-objective or non-linear problems, [10]. Despite the variety, one thing common to all of them is that the modelled space is discretized. It is an established practice in OCP

The associate editor coordinating the review of this manuscript and approving it for publication was Diego Oliva.

problems to discretize the space and use combinatorial optimization algorithms to find the optimal locations for placing cameras. For example, for surveillance applications, points are sampled randomly or at equal intervals over a floor plan. While it is true that continuous space model would provide better accuracy, continuous models are not used due to their complexity (e.g., estimating camera coverage requires calculating intersection between polygons or volumes). Using sets of points for optimization is not an efficient process when compared with continuous space model, but it has shown to be a successful approach for all the existing OCP problems across various applications. A large number of sampled points are required for an acceptable approximation of the continuous space. However, a greater number of input points imply a larger solution search space for the optimization algorithms resulting in an increase in the time required for finding the solution to the OCP problem.

This work focuses on the use-case scenario of finding optimal locations and directions of multiple cameras that need to be placed on a vehicle to achieve surround-view vision. Vehicle surround-view camera systems produce surround-view outputs by combining the video streams of individual cameras. Slight variation in a camera's orientation may result in serious artefacts in the surround-view output. Therefore, it is necessary to ensure that the OCP problem is applied on high-resolution, realistic 3D models of vehicles so that the resulting optimized camera poses precisely adhere to the real vehicle's structure and surface boundaries. To show applicability in real-world scenarios, we use high-detailed realistic 3D models of vehicles in a space modelled using high-resolution 3D grids where each voxel represents few millimeters on the real vehicle's surface. While OCP problems used in this work consist of tens of thousands of decision variables, the widely used branch-and-bound optimization algorithm (a method that guarantees the global optimal solution), [11], cannot handle more than a few hundred variables due to limitations on the amount of required resources, rendering it impractical for most real-world OCP problems.

To overcome the limitations on the amount of resources and time required by the optimization process, we propose a new multi-resolution (MR) approach that works on only a small subset of the input points at one time, thus, reducing the size of the solution search space for any given optimization algorithm. Our method is an iterative process that works, at each iteration, by grouping the input points into a given number of clusters and optimizing camera poses on the clusters of points. At the end of an iteration, only the selected clusters of points (i.e., solution obtained from optimization) are propagated as input points for optimization in the next iteration, while the rest of the points are discarded. Our proposed multi-resolution representation can be compared to the image representations used in different ''multi-scale'' image processing methods. We choose the term ''resolution'' because, when a number, N, of camera positions are grouped into a number, K, of clusters, where $K < N$, the cluster center points, computed as the mean of all the points belonging to the cluster, are represented as the new set of camera positions for an iteration. This implies that the vehicle's surface is represented by K number of points instead of N resulting in a ''low-resolution'' description of the vehicle's 3D surface. But, as the algorithm progresses through iterations, fewer number of points get grouped together into clusters, resulting in a ''high-resolution'' representation of the vehicle's surface.

Primary advantage of our proposed MR method comes from low-resolution descriptions as they reduce the size of the search space for any optimization method irrespective of the choice of the algorithm. This quality enables us to use branch and bound-optimization methods on large real-world models without encountering the resource constraints that are integral to those methods. While it may be assumed that working with only a subset of the points may result in lower coverage accuracy, our results show that, in fact, our approach improves the coverage accuracy that is obtained when camera poses are optimized on the complete set of sampled points. We believe this improvement arises due to two factors; 1) by clustering the sampled points based on their 3D position and surface orientation, we can effectively capture the geometrical features of the vehicle's surface, thereby identifying important regions on the vehicle for camera placement; 2) by representing the clusters of points by the mean position, we can achieve sub-voxel accuracy on the vehicle's model, thereby adding some degree of continuity to an otherwise discrete optimization problem.

Camera placement optimization on the entire set of samples without clustering will be called as single resolution (SR) optimization in the remainder of this document. The results from SR optimization on the same data are used to establish the efficacy of our proposed MR method. It is important to note that all discrete optimization problems used in different OCP applications require a ''look-up'' table describing the coverage of every point in the dataset. This look-up table, known as *visibility matrix*, is computed beforehand as a pre-processing step. In fact, this step is expensive in terms of time required as it involves millions of geometrical calculations. Existing literature in this field overlooks this problem as it considered to be a pre-processing step, where the visibility matrix can be computed once per dataset and stored in a file, for example. However, our experiments show that, for large real-world data, computing this matrix is impractical as it takes several hours when the number of sampled points range in tens of thousands. Our results show that by reducing the number of input camera points, the MR method significantly reduces the time required for the overall optimization process (including the time required for the pre-processing step of visibility calculations) while improving the camera coverage quality.

The rest of the document is organized as follows: Section II provides an overview of prior relevant research, Section III describes the optimal camera placement problem for vehicle surround-view, Section IV describes the multi-resolution

method and the results and conclusions are discussed in Sections V and VI, respectively.

II. BACKGROUND WORK

The origins of optimal camera placement trace back to the art gallery problem, [12]. Work done by Hörster and Lienhart, [9], and by Erdem and Sclaroff, [13], can be considered pioneering in the field of optimal camera placement for indoor surveillance scenarios. Given an indoor setting with a floor plan, their work aims to optimally place a certain number of cameras in designated regions to cover as much of the floor as possible. They proposed multiple problem statements, such as, to minimize the cost of the multi-camera network, maximize coverage given a desired number of cameras, etc. They use binary decision variables to model possible camera locations and the points that are to be covered by the placed cameras (named *control points*). They use the branch-and-bound method which is classified as non-heuristic as it can provide provable bounds around the optimal solution, [11]. The proposed linear programming framework is aptly named binary integer programming (BIP), and has been proven, over the past decade, to be the most accurate model, although, expensive in terms of time and resource consumption.

Most BIP-based formulations are $\mathcal{NP}$-complete. Branch-and-bound algorithms, due to this reason, often require impractical amounts of time and resources to find exact optimal solutions for even small datasets. To circumvent this limitation, numerous approximate optimization methods have been proposed. Despite being prone to local optima, approximate methods find widespread interest in research and practical applications due to lower algorithmic complexity. Hörster and Lienhart, [9], presented an adaptive greedy algorithm that works by creating a rank-matrix for all the possible camera locations. The rank is calculated as the total number of control points covered by each camera pose. The algorithm iteratively picks a camera pose with the highest rank until the required number of cameras are placed. It is an adaptive greedy heuristic because once a camera is selected, that position and the covered control points are removed and only the remaining points are considered for the next camera. Such a feature, however, removes the combinatorial aspect of the OCP problem, but enables it to provide a feasible solution in significantly less time than branch-and-bound-based optimization. Due to their lower complexity, Greedy algorithms are often used for an upper bound on the solution or as a solution initialization step in a more complex algorithm, [14].

The authors in [15] proposed a search heuristic method called particle swarm optimization (PSO). The method works by initializing a number of particles spread randomly across the search space. The particles are compared in terms of the objective function value and the best one is picked as the solution to the OCP problem. In the category of genetic algorithms, Gupta *et al.*, proposed an algorithm that works in a similar fashion to the genes present in the human body. It starts with an initial solution and makes mutations and crossovers of the initialized solution while keeping a record of the objective function values of each candidate solution. The candidate with the best objective function value is selected as the final solution after a certain number of mutations. Evolutionary algorithms like PSO have been extensively studied in camera placement optimization problems. Wang *et al.* proposed two variants of PSO algorithm in [16] and [17], respectively. While the core algorithm remains same as the standard PSO, in RPSO, they use a weighted particle re-sampling scheme to maintain diversity in the particle population. Similarly, in LH-RPSO, [17], they replace the re-sampling strategy of RPSO with a new *Latin Hypercube*-based particle sampling strategy. Both their methods are claimed to improve solutions obtained by the standard PSO algorithm. Some other approximate optimization techniques, including probabilistic search space sampling techniques and evolutionary algorithms are detailed in [18]–[20].

All the approximate optimization methods try to uniformly search the solution space without having to go through all the combinations. Due to this reason, they are faster than exact BIP-based optimization. But, unlike BIP-based method, they fail to find the global optimal solution and often end up in local optima. In, [21], citing the difficulties in solving real-world OCP problems due to its $\mathcal{NP}$-hard characteristic, Ahn *et al.* proposed a *two-phase* algorithm as an approximate optimization algorithm. Their method is similar to the method we propose in this work, as both try to solve a simplified OCP problem at lower image resolutions first. However, their method works only in two resolutions and is proposed only for two-dimensional data. Additionally, they use different optimization algorithms for the two resolution levels. In contrast, we propose a general method that is applicable to 3D data of any kind and size. The only requirement for our method is that the discretized possible camera locations have an associated camera view-direction vector.

Despite the vast amount of literature in this field, their applicability to real world OCP problems is challenging. One of the algorithms presented by Kritter *et al.* [14] is a *row weighting local search* (RWLS) algorithm that was originally proposed by Gao *et al.*, [22], for the general *unicost set covering problem*. The RWLS optimization method was recently explored in the context of OCP in [23]. Kritter *et al.* mentioned in [24] that RWLS was not yet studied in OCP scenarios. They proved the method's superiority in [14] where they studied the OCP problem for large 3D models of European cities. However, citing the method's complexity, they mention that their models were sampled sparsely at large intervals to obtain a feasible solution in reasonable amount of time. In contrast, we test our method on models with tens of thousands of sampled points. Apart from these application-specific methods, some surveys give a comprehensive overview on various modelling and optimization strategies for the OCP problem, [24], [25].

There also exist some bi-objective methods that aim to maximize coverage while simultaneously minimizing the cost of the multi-camera system, [26]. Although, triangular or pyramidal camera field-of-view models are prevalent, there exist many other types of camera coverage models in usage, [27]. Finally, it is to be noted that the choice of coverage quality metric is also important to achieve reliable results, [28].

III. OCP FOR VEHICLE SURROUND-VIEW

The optimal camera placement problem includes few steps such as: discretizing the space, defining binary decision variables, defining an objective function (e.g., maximizing camera coverage), and finding the optimal value of the objective function while enforcing certain constraints on values taken by the variables. Our proposed formulation of the problem of OCP for vehicle surround-view is detailed in the following sub-sections. Our problem formulation is like the BIP-based framework proposed in [9]. Some modifications were made to the problem to allow greater degrees of freedom for camera poses and extension to three-dimensional space. Because the camera view directions have an additional degree of freedom, these modifications only increase the complexity of defining the variables, whereas, the objective function, constraints and the optimization procedure remain unchanged.

A. MODELLING SPACE

We model the space as a volume using a structured grid. Structured grids are a collection of points forming a symmetric 3D grid, [29]. These points usually represent the centroids of encompassing voxels. Such a representation is discrete in nature, hence, there is no further need to sample the space for discrete data. The space for the optimization problem is defined within the volume. To keep the simulations realistic, we collected 3D models of different heavy machines used in the construction industry, that are freely available on crowd-sourced internet platforms, such as, [30]. The polygonal models were voxelized into a structured grid using an open source tool called *binvox*, [31], [32].

The linear programming formulation for the optimal camera placement problem requires definition of two discrete sets of variables: (1) possible camera poses, and (b) points that are viewed by the placed cameras (commonly known as *control points*). In the problem of OCP for vehicle surround-view, the voxelized vehicle model is placed at the center of the volume, with the slice of voxels at $y = 0$ representing the ground plane. The control points are defined around the vehicle on a spherical cap surface, [33], with a radius of 12 meters from the center of the vehicle. This value of radius was chosen as it gives a level of visibility that is required for surround-view under safety critical conditions, [34]. Fig. 1 shows an example visualization of the modelled space with a voxelized bulldozer vehicle model placed at the center of the volume. The vehicle is shown in red and the control points are shown in blue.

For simplicity, we consider voxels on the entire surface of the vehicle as possible locations where cameras can be placed. To extract the vehicle's surface, first, a morphological dilation operation is performed on the voxels representing the vehicle to grow its boundaries by one unit. Then, the original volume is subtracted from the dilated volume, leaving behind a one-voxel thick boundary on the vehicle's surface. These boundary voxels are defined as possible locations for camera placement, $x_i \; \forall i \in N$, where N is the number of boundary voxels. Each position x_i is associated with a camera view direction vector, $\hat{x_i}$, computed using the marching cubes algorithm, [35], on the vehicle's surface. Each camera is rotated in the pitch and yaw directions about the primary view direction, at equal steps up to 90^o on either side.

If there are a total of Φ rotations for each camera, then the set of all possible camera poses can be written as,

$$x_{i\phi} = \begin{cases} 1 & \text{if a camera is placed at location} \\ & i \text{ with orientation } \phi \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where, $\phi = 1, \ldots, \Phi$. The spherical cap around the vehicle is defined by three parameters: radii of the top and bottom circular segments and the height of the spherical cap. As a spherical cap is defined in continuous space, all the voxels that lie within 0.5 units of the surface of the spherical cap are set as control points. The control points are given as,

$$c_j = \begin{cases} 1 & \text{if } \textit{control point} \text{ j is covered} \\ & \text{by at least one camera} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

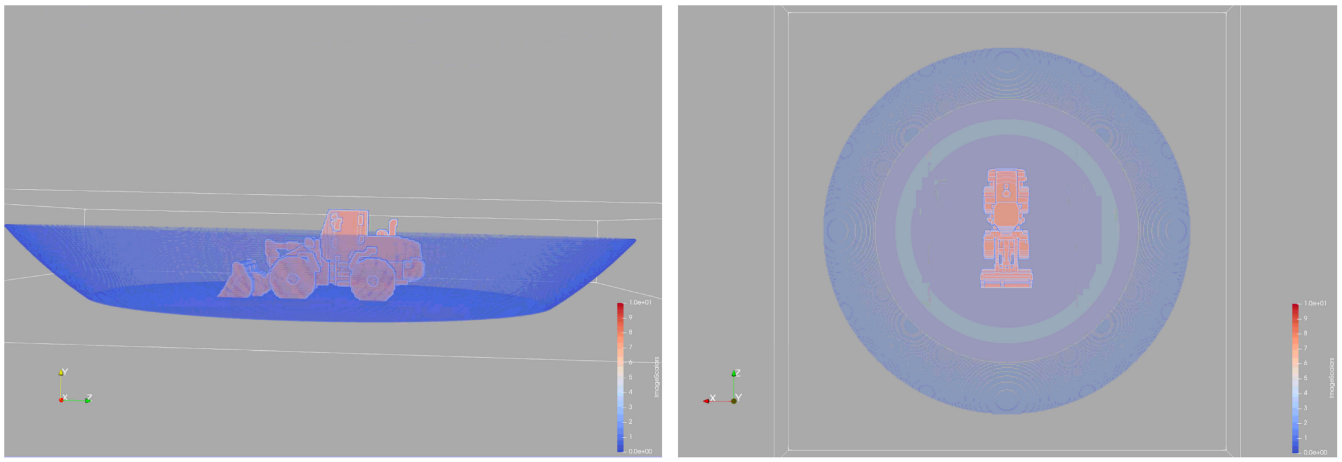
for all $j = 1 : J$, where J is the total number of voxels on the spherical cap surface. The objective function to maximize multi-camera coverage by optimally placing a pre-defined number of cameras, n, is given as,

$$max \sum_j c_j. \tag{3}$$

The poses for these n cameras are selected from the set of all possible camera poses $x_{i\phi}$, such that they maximize the sum of control points c_j that are covered by the n cameras.

B. MODELLING CAMERA's FoV

The camera's field-of-view (FoV) has been previously defined in numerous ways, in both two and three dimensions, for various optimal camera placement problems. Depending on the application, a camera's FoV can have multiple coverage criteria, such as, visibility/occlusion, resolution, etc., [27]. Moreover, some applications define multiple types of cameras (for e.g., wide-angle lens, fish-eye lens, etc.). For simplicity, we define only one type of camera for our optimization problem with only one criteria for visibility/occlusion. Defining and handling multiple types of cameras and criteria is a simple and scalable process, but



(a) Side View (b) Top View

FIGURE 1. Example visualization of modelled space with bulldozer vehicle model. The voxels in red represent the set of possible camera locations, whereas, voxels in blue show the control points.

with a downside of increased complexity of the optimization problem. For our problem, we adopt the commonly used pyramidal FoV model in three dimensions. The pyramidal FoV is described using three parameters, namely, depth of the view frustum, z_f, horizontal field of view angle, α_h and vertical field of view angle α_v. Additionally, the origin p and view direction vector $\hat{\theta}$ associate the FoV pyramid to possible camera poses $x_{i\phi}$.

C. VISIBILITY CHECKS

It is necessary to define an additional variable $g_{i\phi j}$ specifying if a control point c_j is covered by a camera pose $x_{i\phi}$ or not. It can be precomputed for all camera poses $x_{i\phi}$ using simple geometrical calculations and stored in a two-dimensional visibility matrix. The variable is described as,

$$g_{i\phi j} = \begin{cases} 1 & \text{if } \textit{control point } j \text{ is} \\ & \text{covered by a camera placed at} \\ & \text{position } i \text{ with orientation } \phi \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Visibility of a control point c_j by a camera at $x_{i\phi}$ is determined by using point in plane calculations. If a control point lies inside all the five planes of the camera's FoV pyramid, then the corresponding entry in the visibility matrix is marked as 1. This pre-processing step can be expensive when the set X consists of tens of thousands of points. For N camera positions, J control points and Φ orientations for each camera position, the time complexity of our visibility algorithm is $\mathcal{O}(N\Phi J)$. This complexity can be considered to be simpler when compared to other state-of-the-art methods as we do not consider any static or dynamic occlusions (except for self-occlusion) in our model. However, without parallel computations, the time required for these calculations may be impractical for real-world OCP scenarios. As the geometrical calculations are mutually independent, to speed up the process of creating the visibility matrix, we perform these calculations parallel on the GPU using OpenCL.

In our experiments, the largest model has over 14 million points in the initial set $x_{i\phi}$, and over 1500 sampled control points. As it is not possible to construct a matrix of this size, we introduce two additional conditions to filter out camera poses that do not make any significant contribution to the solution. The first condition is used to remove cameras whose FoV is occluded by the vehicle itself. A threshold value, cov_{self}, defines a tolerance value on the number of vehicle points that are allowed to fall within a camera's FoV. All camera poses with occlusion more than the threshold are not included in the final visibility matrix. Additionally, camera poses that do not cover any control points (e.g., cameras pointing directly up) do not contribute to the solution. Similarly, cameras covering too few control points also do not contribute to the solution of a coverage maximization problem with fixed pre-defined number of cameras. A pre-set parameter cov_{min}, defines a lower bound on the number of control points a camera needs to cover. All cameras that cover less than cov_{min} control points are not included in the final visibility matrix.

D. INTEGER PROGRAMMING MODEL

Having defined the binary variables and the objective function, we need to enforce certain constraints that describe coverage or place some restrictions on camera placement. First, a constraint that expresses coverage defining variable, $g_{i\phi j}$, in terms of other variables defined in (1) and (2), is given as,

$$c_j \cdot \left(\sum_{i\phi} x_{i\phi} \cdot g_{i\phi j} - 1 \right) \geq 0. \tag{5}$$

The above inequality is non-linear as it involves the product of two binary variables. It can be linearized by adding a binary variable, $v_{i\phi j}$, representing the product, $c_j \cdot x_{j\phi}$, and

the following two constraints,

$$c_j + x_{i\phi} \geq 2 \cdot v_{i\phi j} \tag{6}$$
$$c_j + x_{i\phi} - 1 \leq v_{i\phi j}. \tag{7}$$

The following constraint is required to ensure that n camera poses are selected,

$$\sum_{i\phi} x_{i\phi} = n. \tag{8}$$

Lastly, to ensure that one possible camera location does not have more than one camera placed, the following constraint needs to be added,

$$\sum_{\phi} x_{i\phi} \leq 1. \tag{9}$$

The integer programming model for the optimal camera placement problem for vehicle surround-view coverage is defined as,

$$\text{maximize} \quad \sum_{j} c_j$$
$$\text{subject to (5), (6), (7), (8) \& (9).} \tag{10}$$

IV. MULTI-RESOLUTION METHOD

All approximate optimization methods used for optimal camera placement problems build on the core idea of effectively exploring only a subset of the solution search space without having to search all of it for the global optimal solution. Our proposed multi-resolution method, on the other hand, is built to explore only the important regions in the modelled space, therefore, reducing the size of search space before even the optimization procedure begins. In fact, with fewer number of points in the set of possible camera positions, x_i, our method can exponentially decrease the number of calculations required in the pre-processing step of visibility checks. The idea behind the multi-resolution method is to group the initial set of possible camera placement locations, x_i, into K clusters of points, and optimize for n camera poses while considering the cluster centroids as the new set of possible camera locations. The clustering step reduces the size of x_i from N to K, where $K \ll N$, therefore decreasing the total number of combinations of possible solutions (i.e., the solution search space). The two steps of clustering and optimization are repeated iteratively, while propagating the solution from one iteration to the next. It means that only the camera points comprising the n clusters of points, selected in this iteration as the optimal camera poses, are passed onto the next iteration, while the rest of the camera points are discarded. The iterative process stops when the number of camera points comprising the selected n clusters, in a certain iteration, is less than a user-defined limit, l. At this stage, camera poses are optimized, for one last time, without any further clustering of the input set of points, x_i. Figure 2 shows an illustration of the multi-resolution method, and, the next paragraph explains, through an example, the iterative process and the reasoning behind naming the method as *multi-resolution*.

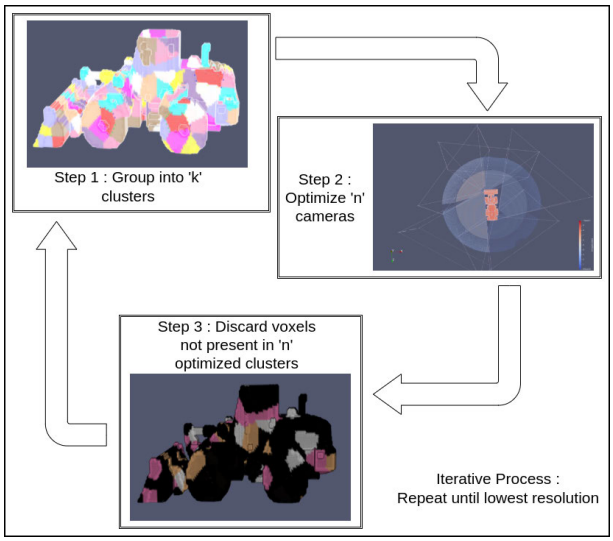


FIGURE 2. An illustration of the multi-resolution optimization method showing an example situation of placing two cameras on the data points. At each resolution the selected camera points are highlighted in solid color. Only the sets of points shown in solid color propagate to the subsequent resolution level, with the rest being discarded.

Consider a 3D model of a bulldozer vehicle. A standard bulldozer vehicle would measure 4.5 meters in its longest dimension (length). When it's 3D model is voxelized into a cube of length 128 voxels, the longest dimension is fit exactly into 128 voxels. It implies that each voxel in the voxelized 3D model would measure $\frac{4.5}{128} \approx 35mm$, i.e., $r = 35mm$ is the 3D image's resolution. Assuming the bulldozer model consists of $N = 11,000$ voxels representing the surface of the vehicle and that the voxel surface is uniformly one voxel thick, the surface area can be calculated as $N \times r^2$. Now, if the surface voxels are grouped together into $K = 100$ clusters, assuming that all the clusters are uniform in shape and size, the resolution of the resulting clustered image will be $\sqrt{\frac{N \times r^2}{K}} \approx 367mm$. Because the vehicle's surface is represented by larger voxels than the original image, we call the clustered data as low-resolution image. Say we want to optimize poses of $n = 5$ cameras. Given that the approximate size of each cluster in this case is 110 voxels, we will have $N = 550$ points, belonging to the 5 clusters that were selected as the solution in this iteration, that will be passed as input to the next iteration. Assuming $l = 200$, the input points will be clustered again into $K = 100$ clusters because $550 > l$. This results in a resolution of $82mm$ for this iteration. It is to be noted that this resolution is higher than previous iteration but it is still lower than the original resolution of $35mm$. The clustering and optimization steps are repeated until when N becomes lower than l. When $N < l$, the input points are not clustered any further and the set x_i for optimization at this stage is a small subset of the original set of points with the resolution, $r = 35mm$. Because the resolution of the last iteration is same as the resolution of the original voxelized

model, we call it as the *highest* resolution, while, the first iteration is called *lowest* resolution.

A. CLUSTERING BASED ON POINT ORIENTATION

3D image clustering is an extensively studied topic in the field of computer vision. Well-known, unsupervised clustering methods such as the K-means and the Gaussian mixture model algorithms, [36], [37], are proven to be efficient on general 2D/3D point data. There also exists a vast variety of application specific clustering methods, such as, for 3D point cloud data, [38], [39], 2D images, [40], 3D image (RGB+D) data, [41], etc. The data we use here for OCP is like 3D point clouds, i.e., each camera position is represented using 3D spatial coordinates along with an associated 3D vector representing the camera's view direction. However, to our knowledge, clustering methods tailored specifically for OCP problems do not exist. We propose a new clustering method adapted from the *SLIC* algorithm proposed by Achanta *et al.*, [42].

Our clustering algorithm is an iterative process, where all the points are assigned to a set of cluster centers, while at each iteration, new clusters centers are added or existing ones are removed depending on the number of outlier points. The input to the algorithm is a set of N 6D vectors, $a_i = [x_i\ y_i\ z_i\ u_i\ v_i\ w_i]^T$, that are a merger of the set of possible camera positions, $P_i = (x_i, y_i, z_i)$, and their associated primary orientation vectors, $\hat{P}_i = (u_i, v_i, w_i)$. It is to be noted that clustering is done based only on the primary orientations of the camera points. At this stage, the rotations, Φ, for the camera positions do not yet come into play as those rotations about the primary view direction vector are computed after clustering, and before optimization, to form the set, $x_{i\phi}$, of possible camera poses. The input set of points are grouped together into a set of K clusters, where the value of K is chosen by the user and passed as input to the algorithm. Initially, a set of clusters seeds, $C_k = [x_k\ y_k\ z_k\ u_k\ v_k\ w_k]^T$, $k = 1, \ldots, K$ are selected from the input set of points, a_i. The output of the algorithm is a *labels* vector of values k for each point a_i indicating the cluster it belongs to, and a set of cluster centers C_k that are estimated as the average of all the points a_i belonging to each cluster k. The clustering process can be broadly categorized into three steps: 1) initialization 2) assignment and 3) update.

The initial seeds are selected through a strategy that exploits the orientations associated with the camera positions. Assuming that all the clusters are uniform in size, the approximate size of each cluster is given as, $c_{size} = \frac{N}{K}$. Under another assumption that the clusters are uniform in shape, the length of each cluster can be given as, $S = \sqrt[3]{\frac{N}{K}}$. The input points a_i are first voxelized using the *voxel grid* filter provided as part of the Point Cloud Library API, [43], with S as the length of the resulting voxels. The voxel grid filter essentially downsamples the points using a 3D grid, where the centers of each cell represent the mean of all the points that fall within that cell. Selecting a length of S for the voxel grid ensures that the resulting points are all spaced at least S units apart. Downsampling with a voxel grid results in a number of points that is usually larger than the number of clusters, K. We randomly select K points, from the downsampled set of points, to represent the initial set of clusters centers, C_k.

In the assignment step, the points a_i are, each, assigned a label k corresponding to the cluster center C_k that it belongs to, based on a distance metric D. D is computed as a combination of the Euclidean distances between the position and the normal vectors of a point a_i and a cluster center C_k. If the Euclidean distance between the position vectors of a camera point and a cluster center is given as, $d_s = \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2 + (z_i - z_k)^2}$, and the Euclidean distance between their normal vectors is given as, $d_n = \sqrt{(u_i - u_k)^2 + (v_i - v_k)^2 + (w_i - w_k)^2}$, then the distance metric is written as,

$$D = \sqrt{d_n^2 + \frac{m}{S} \cdot d_s^2}, Changes \tag{11}$$

where, m, is a user-defined parameter that acts as a relative weight between d_s and d_n. The spatial distance has to be normalized by the search radius S because, d_n is usually under 1 as it is computed using unit normal vectors, whereas, d_s varies from model to model. During assignment of points to clusters, only the points lying within a radius of S units from a cluster center are searched and compared using the distance metric. It implies that the farthest point assigned to a cluster center cannot be more than S units away from it, spatially. Therefore, normalizing by S brings the value of d_s similar to that of d_n.

We go through each cluster center sequentially, search a spherical neighbourhood of radius S around it, and assign the encountered points to the cluster center if it is the smallest distance, D, the point has seen so far. There can be some overlap between different clusters as the search radius around a cluster center is same as the distance between two cluster centers (recollect that both are equal to S). Due to the overlap, it might happen that some points initially assigned to one cluster may get reassigned to another. This, however, provides some flexibility to the size of clusters, allowing some clusters to be larger or smaller to better fit the vehicle's 3D structure. Once the neighbourhoods of all the initialized cluster centers are checked, the outlying points that have not been assigned to any of the cluster centers are collected (Let us call the count of outlying points as N_{ol}). At the same time, the size of each cluster, i.e., the number of points assigned to each cluster center are also counted. The number of unassigned points together with sizes of exiting clusters give us an estimate on how many cluster centers to be added or removed. This feature makes the clustering process dynamic in nature, helping to accurately cover the vehicle's entire surface.

In the update step, first the cluster centers that do not have any assigned points are removed. If K_r cluster centers are removed, the remaining $K - K_r$ cluster centers, C_k, are updated as the mean of all the points, a_i, assigned to the each cluster center. An estimate of cluster centers that need to be added is obtained by diving the number of

outlying points by the, previously calculated, approximate cluster size, c_{size}, i.e., $c_{add} = \frac{N_{ol}}{c_{size}}$. The set of outlying points is processed in the same way as the initialization step, to initialize c_{add} number of new cluster centers. This process of adding and/or removing cluster centers changes the number of clusters (say, the final number of clusters after completion is K') that was initially selected by the user. The approximate cluster size, calculated previously from the user-inputted number of clusters, K, however, remains constant throughout the clustering, making c_{size} (or S) as the primary parameter that controls the clustering process. Therefore, the two parameters, K and S, are interchangeable as input to the clustering algorithm. We run the clustering process with k as input.

The assignment and update steps are repeated iteratively until, either the number of outlying points changes by less than 10% from the previous iteration, or if there are no more cluster centers required to be added. For example, when $N_{ol} < c_{size}$, c_{add}, as an integer division, becomes zero, implying that no additional cluster centers can be initialized. Our experiments show that the algorithm usually runs until there is no possibility of adding any more clusters. Hence, when the stopping criteria is satisfied, the existing cluster centers are updated as the mean of all the points assigned to the cluster, and the outlier points (if any) are checked against all the cluster centers using the distance metric, D, and assigned to the closest cluster center, as a last step. At the end, the total number of clusters may be different than the value of K initially chosen by the user, but the dynamic process ensures that the surface of the vehicle is effectively captured by uniform clusters. The complete algorithm is detailed in Algorithm 1. The clustering algorithm has a time complexity of $\mathcal{O}(N)$.

Algorithm 1: Clustering Based on Point Orientation

Input: K,$a_i = [x_i\ y_i\ z_i\ u_i\ v_i\ w_i]^T\ \forall\ i = 1 : N$;
Result: $labels_i = k\ \forall\ i = 1 : N$, $C_k\ \forall\ k = 1 : K'$
$S = \sqrt[3]{\frac{N}{K}}$;
Initialize: $C_k = [x_k\ y_k\ z_k\ u_k\ v_k\ w_k]^T\ \forall\ k = 1 : K$;
$D_i = \inf$, $labels_i = -1\ \forall\ i = 1 : N$;
$K' = K$, $t = 0$, $N_{ol}(t) = N$, $c_{size} = \frac{N}{K}$;
while *(1)* **do**
  for $k = 1$ **to** K' **do**
    for a_i *in neighbourhood of radius* S **do**
      D = D(a_i, C_k) as in equation 11;
      if $D < D_i$ **then**
        $D_i = D$;
        $labels_i = k$;
      end
    end
  end
  outliers = collect points with label == -1;
  $N_{ol}(t+1)$ = size(outliers);
  remove all centers with $size(C_k) < 1$;
  $c_{add} = \frac{N_{ol}(t+1)}{c_{size}}$;
  if $(\frac{N_{ol}(t) - N_{ol}(t+1)}{N_{ol}(t)} < 0.1)\ ||\ (c_{add} < 1)$ **then**
    break;
  end
  Re-estimate C_k as mean of all a_i with $labels_i == k$;
  Initialize c_{add} clusters and append to C_k;
  $K' = K$ after adding and/or removing clusters;
  $t = t + 1$;
end
Force assign outlying points to nearest cluster;
Re-estimate C_k as mean of all a_i with $labels_i == k$;

B. ALGORITHM

The initially collected set of voxels representing the vehicle's surface, P_i, and their primary orientation vectors, $\hat{P}_i$ (together represented as a set of 6D points, a_i), and the set of control points, $b_j = [x_j, y_j, z_j]^T$ are passed as input to the multi-resolution (MR) algorithm. Additionally, the required number of cameras to be optimally placed, n, is also passed as input to the algorithm. As illustrated at the beginning of this section, the MR method involves two steps: 1) Clustering of input points, and; 2) Optimizing for camera poses considering the cluster centers as the new set of possible camera locations. These two steps are repeated iteratively until the total number of variables in the set a_i is more than the pre-defined limit, l. The algorithm starts with a loop where the two steps of clustering and optimization are repeated.

Initially, the set of points a_i are clustered into K' clusters. Following this step, the set of K' cluster centers, C_k, represent the new set of possible camera locations, a_i, thus, decreasing the size of input from N to K'. These points are then rotated, about the associated primary direction vector (remember that $C_k[x_k\ y_k\ z_k\ u_k\ v_k\ w_k]^T$ is 6D vector representing both position and orientation) to form the set of variables $x_{i\phi}$. Together with Φ rotations in the four directions, the resulting set of possible camera poses consists of $((\Phi + 1) \times K')$ number of points (instead of $(\Phi + 1) \times N$ points when considered without clustering). Followed by the rotation step, visibility checks are performed on the sets of variables, $x_{i\phi}$ and c_j, to create the visibility matrix, as detailed in Section III-C. Post the visibility checks step, the accepted camera poses (the exact number depends on parameters cov_{self} and cov_{min} and varies from model to model) comprise only a small subset of the original set of points, as most of the possible camera poses that do not fulfill visibility criteria are discarded. Binary variables, $x_{k\phi}$ and c_j, are then initialized over the accepted camera poses and the complete set of control points, and passed as input to the user-chosen optimization method, along with the binary variables $g_{k\phi j}$ that represent the visibility matrix.

This algorithm can work with any optimization method, as all discrete combinatorial optimization methods take the same input (integer decision variables, set of constraints and a pre-computed visibility matrix) and produce the same output (set of optimal camera poses). The optimization process

then selects n (where n is a number chosen by the user representing the number of cameras to be optimally placed) optimal camera poses from the input set, by optimizing the objective function while enforcing the constraints. After completion of the optimization process, the obtained solution is processed for the next iteration. The points belonging to the n optimally selected clusters are extracted to represent the new set of possible camera locations a_i. The rest of the points belonging to the remaining $K' - n$ clusters are discarded. If the number of points, N, in the new set a_i are more than l, then the steps of clustering, creating rotated direction vectors, visibility checks, and optimization, are repeated. Otherwise, the program exits the loop and optimization is done on the remaining set of points a_i, for one last time, without clustering. As these points are not clustered any further, this step of final optimization is called the *highest resolution* level. It might sometimes happen that the solution obtained at the highest resolution is worse than a solution obtained in one of the previous iterations. Therefore, the solution at each iteration is saved and the best of all solutions is presented as the final result of the MR algorithm. The output of the algorithm is a set of n 6D vectors, $camSol_n = [x_n\ y_n\ z_n\ u_n\ v_n\ w_n]^T$ representing the poses of the optimally placed cameras.

The multi-resolution optimization algorithm is detailed in Algorithm 2. Initialization of parameters m, l, Φ, cov_{min} and cov_{self} is discussed later in Section V-A. In the algorithm, the functions *cluster()*, *rotate()*, *visibilityChecks()* and *optimize()* each perform a specific task as implied by their names, such as; the function *cluster()* performs the clustering operation as described in Algorithm 1; the *rotate()* function produces a set of Φ rotated direction vectors about each camera pose's primary orientation, as described in Section III-A; *visibilityChecks()* performs all the necessary calculations described in Section III-C and sets up the visibility matrix, and; function *optimize* passes all the variables and the parameter n to the user-selected optimization method (such as, branch-and-bound, greedy heuristic, etc.) to obtain the optimal camera poses. In the *rotate()* function, the notation "$\hat{\cdot}$" refers to the direction vector ($\hat{P} = [u, v, w]^T$) part of the 6D variable. Lastly, the *optimize()* function in the loop returns a vector of labels, k_n, of the n solution clusters. k_n is shown in the while loop, whereas it is omitted from the call to *optimize()* function in the *highest resolution* because, the input points at this stage correspond to points from the original voxel image without any clustering. By saving the clusters' labels, we can easily track and retrieve all the camera points that belong to those clusters.

The time complexity of our proposed MR algorithm depends on three factors, i.e., the individual complexities of clustering, visibility checks and optimization steps. Time required for the clustering step, with a linear complexity depending on the number of input points in set x_i, varies from one resolution level to another. It requires highest computational time for the lowest resolution level when the size x_i is equal to N, whereas, the computational time for subsequent resolution levels decreases with each level as the number of points in input set x_i at a resolution level $t + 1$ is always less than the number of points in x_i at level t. The complexity of visibility checks step is given as $\mathcal{O}(K'\Phi J)$, as after the clustering step, the MR method works with only K' points. This is significantly lower than the corresponding complexity for SR method ($\mathcal{O}(N\Phi J)$) as $K' \ll N$. The complexity of the optimization step depends on the chosen optimization algorithm. For example, the linear programming based-branch-and-bound algorithm has an exponential complexity whereas the particle swarm optimization method has a linear complexity in the number of particles. The number of points passed as input to the optimization algorithm, after visibility checks operation, represent only a small subset of the points in $x_{i\phi}$. Consider the number of input points after visibility checks for to be N' for SR method and N'' for MR method. Then for the simplest of all cases, if we assume that all the chosen optimization methods have linear time complexity, the complexity of MR method can be given as, $\mathcal{O}(N'')$, which is again significantly lower than the complexity for SR method ($\mathcal{O}(N')$) because, N'' is a subset of the set with $K' \times \Phi$ points whereas, N' is a subset of the much larger set with $N \times \Phi$ points. If the values of parameters cov_{min} and cov_{self} are kept same for MR and SR methods, then N'' will always be much lesser than N'.

Algorithm 2: Multi-Resolution Optimization

Input: $a_i = [x_i\ y_i\ z_i\ u_i\ v_i\ w_i]^T\ \forall\ i = 1 : N$,
$c_j = [x_j, y_j, z_j]^T\ \forall\ j = 1 : J$, K, n;
Result: $camSol_n = [x_n\ y_n\ z_n\ u_n\ v_n\ w_n]^T\ \forall\ 1 : n$
Initialize parameters m, l, Φ, cov_{min} and cov_{self};
$cov_{best} = 0$;
/* lower resolutions */
while $N > l$ **do**
  $[C_k,\ labels_i] = \text{cluster}(a_i, K)$;
  $C_{k\phi} = \text{rotate}(\hat{C}_k)$;
  $x_{k\phi} = \text{VisibilityChecks}(C_{k\phi}, c_j)$;
  $[sol_n, k_n, cov] = \text{optimize}(x_{k\phi}, c_j, n)$;
  if $cov > cov_{best}$ **then**
    $camSol_n = sol_n$;
    $cov_{best} = cov$;
  end
  a_i = all points with $labels_i = k\ \forall\ k = k_n$;
  $N = size(a_i)$;
end
/* higest resolution */
$a_{i\phi} = \text{rotate}(\hat{a}_i)$;
$x_{i\phi} = \text{VisibilityChecks}(a_{i\phi}, c_j)$;
$[sol_n, cov] = \text{optimize}(x_{i\phi}, c_j, n)$;
if $cov > cov_{best}$ **then**
  $camSol_n = sol_n$;
  $cov_{best} = cov$;
end

V. RESULTS

Experiments were conducted on synthetic data as well as on 3D models of real vehicles used in the construction industry. The need for synthetic data arises due to large sizes of high detail 3D models of real construction equipment. The main problem arises with the binary integer programming formulation-based branch-and-bound optimization algorithm that has high memory or time requirements. Due to this shortcoming OCP environments with large sizes cannot be handled using the available resources. Therefore, we created simulated data using simple geometrical structures, roughly resembling vehicles. As, this data has small number of samples, we present a detailed comparison of all the considered optimization methods. However, tests on real data further validate the efficacy of our method. We created four models resembling a car, a van, a truck and a bus in four different sizes, S, M, L and H. All the models in the four size categories have about, 75, 200, 650 and 1300 voxels, respectively (or possible camera locations). The number of control points remains same for the four vehicle models at about 320, 530, 1650 and 3360, control points, for the four sizes, respectively. In total, the simulated data consists of 16 instances. Similarly, there are 16 instances of real data, i.e, four vehicle models, namely, bulldozer, JCB, mining truck and tractor scraper, in four sizes labelled as 32, 64, 128, and 256. The real data was obtained as 3D polygon models which were voxelized into cubes of lengths 32, 64, 128 and 256 voxels. The JCB vehicle model, being the smallest of the four has $\sim$ 2300 voxels on the surface of the model (possible camera locations) when voxelized into a cube of 32 voxels, and $\sim$ 115000 camera locations when voxelized into a cube of 256 voxels. Whereas, the mining truck vehicle model, being the largest of the four has $\sim$ 6600 and $\sim$ 360000 surface voxels when voxelized into cubes of 32 and 256 voxels, respectively. The tractor scraper and bulldozer models have similar size with $\sim$ 170000 surface voxels at *size-256*. It is to be noted that for every instance, the total number of variables, for each model, before visibility checks is given by the number of boundary voxels multiplied by $\Phi = 97$. This implies that for the largest model (Mining truck-256), the total number of input variables before visibility checks is $\sim$ 35 million. The polygon and voxelized models of real vehicles have been made publicly available.[1]

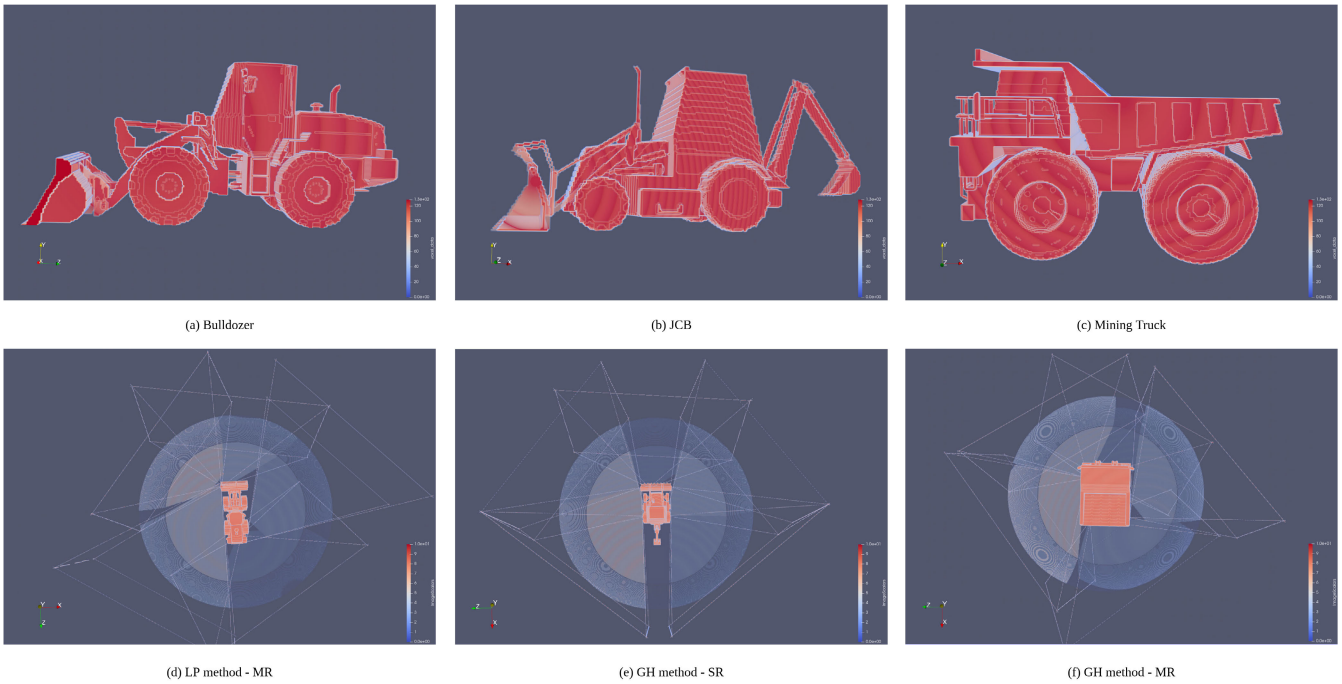
On simulated data, for each of the 16 instances, we test five optimization methods using the single resolution (SR) and multi-resolution (MR) optimization strategies. To recollect, SR is when we optimize for camera poses on all the camera locations (voxels) that are part of the vehicle's surface, without any clustering or sub-sampling. Whereas MR is when we cluster the sets of vehicle's surface voxels multiple times and optimize camera poses on the set of cluster centers. To validate our proposed MR optimization strategy, using C++ programming language, we implemented the Greedy Heurisitc (GH) proposed in [9], Metropolis Sampling (MS) proposed in [19], row weighting local search heuristic (RWLS) algorithm proposed in [22] and the LH-RPSO algorithm proposed in [17]. The LP-based branch-and-bound algorithm (LP) provided by CPLEX, [44], is the fifth optimization algorithm that we used for validation. As the branch-and-bound algorithm is known to provide provable bounds on the optimal solution, it is a good choice to compare against the results from approximate algorithms on the data that we used for this work. However, on the 16 instances of real data, we chose to test only one of the two evolutionary algorithms (MS and LH-RPSO). In consequence, we dropped MS method as coverage results on simulated data showed that LH-RPSO is better and consistent than the MS method. Control point coverage for simulated data and real data are shown in Table 1 and Table 3, respectively. Whereas, the total optimization times for tests on simulated and real data are presented in Tables 2 and 4, respectively. Figure 3 shows an example visualization of the bulldozer, JCB and mining truck vehicle models at *size-256*. The second row in the figure shows visualizations of three OCP solutions. The next two sub-sections detail the values for all the parameters and some modifications to SOTA algorithms, respectively. Detailed analysis of the results is presented in Section V-C.

A. PARAMETERS

The value of the limit is decided based on the system's hardware, i.e., the number of variables and constraints the LP-based branch and bound algorithm can handle without running into the *out-of-memory* error. All the experiments were run on an Intel Core i7-8700K CPU with 12 processing cores and 32GB of RAM. Parallel processing was done using the 12 processing cores of the same CPU. Given our system's hardware, and through trial-and-error, we chose a value of $l = 200$ for all our experiments. In all our experiments spatial distance was given twice the importance as compared against d_n, therefore, we chose $m = 2.0$ for the clustering process. The primary direction vector at each possible camera location was rotated at steps of 3.75° in each direction, producing a total of 96 rotations, i.e., for each possible camera position, $\Phi = 97$, including the primary camera orientation. $cov_{self} = 0$, for all experiments ensured that camera poses occluded by the vehicle were excluded from the visibility matrix. In all cases, we optimized to place five cameras, i.e., $n = 5$ for all experiments.

Choosing a uniform value for cov_{min} is tricky as, a small value may allow too many variables in the visibility matrix, leading to a possible *out-of-memory* error when using the branch-and-bound algorithm, whereas, a lager value may lead to too few variables in the visibility matrix, thereby removing the combinatorial aspect of the optimization problem. To maintain uniformity across all the experiments, we chose a value for cov_{min} in such a way that an accepted camera pose would cover at least 70% of an expected number of control points for one camera, under ideal circumstances.

[1] https://github.com/AnirudhPuligandla/multi-resolution-optimal-camera-placement.git



(a) Bulldozer (b) JCB (c) Mining Truck

(d) LP method - MR (e) GH method - SR (f) GH method - MR

FIGURE 3. Figure showing 3D models of (a) Bulldozer, (b) JCB, and, (c) Mining truck, heavy vehicles, voxelized into cube of length 256 voxels. The bottom row shows visualizations of OCP results with five optimally placed cameras, obtained using (d) LP method at multiple resolutions for bulldozer model, (e) GH method at single resolution for JCB model, and, (f) GH method at multiple resolutions for mining truck model.

For example, if a space model of a particular size category contains J number of control points, the expected coverage for one camera in the ideal scenario will be $\frac{J}{n}$. A value $cov_{min} = 0.7 \times \frac{J}{5}$ was set which remains constant across all vehicle models for one size category. The maximum time limit for each test, i.e., the total time for the individual steps of clustering, visibility checks and optimization, was set at 48 hours.

Considering the larger sizes of real data, the values of parameters of individual optimization algorithms were different for simulated and real data. Parameter $l = 50$ was set for simulated data as the smallest model has only 75 camera poses. We set the same value of l for instances of simulated data to maintain uniformity. The LP method provided by CPLEX has multiple parameters that decide on optimality, feasibility, choice of optimization algorithm, etc. For tests on both real and simulated data, we chose to emphasise on feasibility over optimality and allowed a tolerance of upto 6% in the gap of the feasible solution from the optimal solution. The GH method does not have any parameters. The MS algorithm was run until either all the available camera poses were visited, or, until a number of iterations equal to the available camera poses were completed. The RWLS algorithm was run for 250 iterations on simulated data and for 500 iterations on real data. The LH-RPSO algorithm was run with 30 particles for both data, and, for 1000 iterations on real data and 500 iterations on simulated data. These values were chosen as a trade-off between coverage quality and time required for optimization. All these values ensure that optimization can be completed in reasonable amount of time without noticeably compromising on coverage. As the parameter values are kept same for both MR and SR methods, it allows for a fair comparison between the two optimization strategies.

B. IMPROVEMENTS TO SOTA METHODS

For the RWLS method, it is required to compute a matrix describing neighbourhood relationships between all the camera poses. A camera pose is considered a neighbour to another if at least one control point is covered by both the cameras. For an instance with α camera poses, the time complexity for this step is given as, $\mathcal{O}(\alpha^2)$. Although this is lower than the complexity of our visibility checks algorithm, this step takes as much time, particularly for the SR method, as in most cases the number of camera poses selected after visibility checks is higher than the number of control points. We perform these calculations parallel on our multi-core CPU using OpenCL kernel code. Similarly, the GH method requires re-computation of the visibility matrix after each iteration. As we already perform parallel computations for visibility checks, we re-use the same OpenCL kernel code for the GH method to achieve an overall speedup.

C. ANALYSIS

In the presented tables, each row shows results for one instance, with alternate columns presenting the results for SR and MR methods, for the selected four or five optimization methods. For each row in Tables 1 and 3, one entry is highlighted in bold to represent the best obtained control point coverage for that instance. Some entries in the tables

TABLE 1. Table showing objective function value (in % of control points covered), obtained from the five optimization algorithms (LP, GH, MS, RWLS and LH-RPSO) using both MR and SR methods for the four simulated vehicle models in four size categories (S,M,L,H) for each model.

Vehicle	Size	LP		GH		MS		RWLS		LH-RPSO	
		SR	MR	SR	MR	SR	MR	SR	MR	SR	MR
Car	S	**84.2**	**84.2**	83.0	83.0	81.0	80.8	83.5	83.3	82.9	83.9
	M	86.2	**86.6**	85.3	85.5	81.4	81.2	84.9	86.2	84.1	85.9
	L	–	**86.9**	84.3	84.3	80.6	81.6	85.6	85.3	84.8	83.3
	H	–	85.2	81.5	84.8	79.1	82.5	85.8	**87.3**	86.2	86.7
Van	S	82.2	83.0	82.9	82.8	79.9	79.6	83.3	84.2	**84.5**	**84.5**
	M	85.6	**86.3**	85.5	84.4	78.5	79.6	85.3	85.7	85.5	85.9
	L	–	**86.7**	84.3	86.2	79.5	81.2	85.4	84.2	84.1	84.3
	H	–	86.4	81.5	86.4	80.4	82.0	85.7	**87.6**	84.0	84.6
Truck	S	**84.2**	83.6	82.9	82.9	78.0	79.3	82.9	83.0	83.2	83.2
	M	**85.7**	**85.7**	85.4	85.4	79.8	81.3	84.9	85.3	85.5	85.5
	L	–	**87.7**	84.3	87.2	81.7	80.6	84.3	85.8	85.1	85.7
	H	–	**88.2**	81.5	87.6	80.1	85.7	85.7	87.5	85.1	85.0
Bus	S	84.2	84.2	83.0	83.9	78.7	80.8	83.0	**84.5**	84.2	83.7
	M	85.6	87.6	85.5	**88.3**	77.4	80.5	85.1	86.2	85.1	84.9
	L	–	89.7	84.3	**89.9**	80.8	81.5	85.5	88.4	84.2	87.5
	H	–	**88.0**	81.4	**88.0**	82.1	82.4	85.1	85.7	84.2	83.4

TABLE 2. Table showing total optimization time (in seconds) taken by the five optimization algorithms (LP, GH, MS, RWLS and LH-RPSO) using both MR and SR methods for the four simulated vehicle models in four size categories (S,M,L,H) for each model.

Vehicle	Size	LP		GH		MS		RWLS		LH-RPSO	
		SR	MR	SR	MR	SR	MR	SR	MR	SR	MR
Car	S	61.52	81.51	1.24	1.47	1.18	1.24	9.88	8.95	4.71	9.27
	M	309.91	178.65	2.84	1.92	2.42	1.53	51.81	16.73	8.91	14.64
	L	–	1113.77	11.61	4.11	15.04	3.73	526.99	83.96	36.76	70.77
	H	–	2629.39	36.24	4.77	57.08	4.98	2174.6	75.47	73.16	145.21
Van	S	75.95	63.35	3.20	6.13	1.61	1.34	12.8	9.64	5.53	9.13
	M	568.84	171.62	3.04	1.83	2.85	1.75	56.86	19.04	9.25	14.44
	L	–	1625.48	12.96	3.52	15.44	3.53	548.15	57.92	32.57	70.08
	H	–	3358.98	41.98	7.39	52.99	9.04	2215.73	254.75	78.27	150.48
Truck	S	69.47	58.36	1.37	1.55	1.19	1.32	11.01	7.85	5.38	9.26
	M	315.63	235.84	2.65	2.06	2.57	1.74	52.75	20.69	8.78	14.84
	L	–	1036.62	11.57	3.3	13.44	3.42	508.40	86.88	31.18	47.49
	H	–	2521.24	36.08	7.01	45.96	7.28	2124.72	166.26	70.97	148.62
Bus	S	68.9	95.36	1.59	1.5	1.44	1.39	12.17	9.02	5.30	8.81
	M	433.13	272.03	3.32	1.82	3.28	1.69	61.98	22.17	9.55	14.86
	L	–	821.96	15.71	3.6	17.29	3.47	585.91	60.27	33.71	47.14
	H	–	2490.46	50.74	5.48	60.34	6.78	2166.97	96.59	85.07	145.15

are marked as '–' because, they could not be completed either because of the *out-of-memory* error or because, the entire optimization process could not be completed in under 48 hours. In the case of simulated data, experiments on any of the four vehicle models for 'L' and 'H' sizes could not be completed using LP method as the branch-and-bound algorithm from CPLEX ran out of available RAM due to high number of variables and constraints. From our trials, we observed that with our available RAM the LP method can be run till completion, only when the total number of variables, i.e., the number of camera poses selected after visibility checks plus the number of control points, is less than ~ 3000. Similarly, for real data we omitted the entire column for SR method using LP optimization because, none of those tests were completed as all the sixteen instances had more than 3000 variables. This shows that using the MR strategy we can use LP optimization on large data where it was not possible otherwise.

From Tables 1 and 3, we can see that optimization process using LP method was completed for only eight of the thirty two instances when using SR method. Whereas, with MR method, optimization using LP method was completed for thirty one out of the thirty-two instances. The only exception is the *tractor scraper 256* model where the total number of variables exceeded 3000. Although this optimization can be completed by lowering the value of parameter l, we did not do that to maintain uniformity in parameter values across all tests. By lowering the value of l we can set an upper bound on the number variables that are passed to LP optimization to thereby, avoid the *out-of-memory* error. It is to be noted that changing the value of l will also change the number of resolution levels, i.e., the number of iterations of the MR method. We believe that a lower or higher number of iterations changes the solution at that iteration but, it does not have any effect on the overall best solution obtained using the MR method.

From Table 1, it can be seen that for simulated data, MR strategy provides the best control point coverage for twelve out of the sixteen instances. For three instances, both MR and SR methods provided the same best coverage while the SR method provided the best coverage in only one instance. These results show that the MR strategy can provide either the same or better coverage in 93% of the cases. Table 2 shows that, in fact, the MR method needs less computational time than state-of-the-art methods to provide better control point coverage. The same quality is re-iterated in the results on real data (see Table 3) where, the MR method provides the same or better control point coverage in 81% of the cases (13 out of 16 instances). Overall, the best coverage was obtained when using the LP optimization method in 18 out of the 32 instances, i.e., in more than half of the cases. This shows the importance of complex methods which can guarantee global optimal solution or provide provable bounds around the global optimum. Moreover, in some cases, where other approximate optimization methods provide better control point coverage, the LP method may also result in a better solution if allowed to run until it finds the optimal solution albeit at the cost of increased computational time.

If we observe on a case to case basis, including results for all the instances from all the optimization methods, we can see that the MR method provided better coverage accuracy than the SR method in 55 out of the total 80 cases (68.75%) on simulated data whereas, the SR method provides better coverage accuracy in only 15 cases (18.75%) while the results from both SR and MR methods are same in the remaining 10 cases (12.5%). In the cases where the result from MR method is better than SR method (excluding the cases where the SR method does not have a result), the average gap between the two corresponding solutions is 1.86%. In cases, where the SR method provided better coverage, the result from MR method is worse by an average of 0.56%. Similarly, in results on real data, the MR method provided better coverage than SR method in 54 out of a total 64 cases with an average gap of 0.82% while it provided the same result as SR method in one case, and, lower coverage in 9 cases with an average gap of -0.86%. This implies that the MR method provided either the same or better coverage than the SR method in at least 80% of all the experiments on both simulated and real data. Moreover, in some cases when it provided lower coverage than SR method, the percentage error between the two solutions is less than 1% in all cases. On the whole, looking at these results, we can say that the MR method provides a solution that is within -1% to $+2\%$ of the solution from SR optimization in significantly less computational time.

From Tables 1 and 3, it may be observed that camera coverage decreases with the increasing number of voxels. While it is intuitive to believe that camera coverage should increase with the number of voxels, we would like to emphasise that there are various parameters that influence the overall coverage for OCP problems. When the size of the vehicle model increases, the camera view frustum parameters also change, so do the number of control points and their sampling rates. Therefore, we emphasise that the coverage values are to be compared per instance (per row in the tables) and between different optimization methods but, not between different sizes or vehicle models. Although the coverage for the 32 size and 256 size for any model are similar, the vehicle models at 32 size have severely distorted shapes and low detail. At least the 128-size model is required for a good approximation of the real-world vehicles. For the 32 and 64 sizes, we sample the set of control points at one in every 20 points. This sampling frequency, however, results in a substantial number of control points for the 128 and 256 sizes that is more than the limit of 3000. To test the LP method, we decreased the control point sampling frequency for 128 and 256 sizes down to one in a hundred points and one in 250 points, respectively. As we can see from coverage values for 128 and 256 sizes, sampling lesser number of control points does not have any impact on the result if control points are sampled uniformly from the original set.

TABLE 3. Table showing objective function value (in % of control points covered), obtained from the four optimization algorithms (LP, GH, RWLS and LH-RPSO) using both MR and SR methods for the four real vehicle models in four sizes (32, 64, 128, 256) for each vehicle model.

Vehicle	Size	LP	GH		RWLS		LH-RPSO	
		MR	SR	MR	SR	MR	SR	MR
Bulldozer	32	92.2	90.1	92.2	**93.1**	92.0	87.4	85.6
	64	92.9	90.9	**93.2**	92.8	92.6	88.8	90.8
	128	**93.9**	88.9	90.8	93.2	93.5	87.1	88.3
	256	92.6	–	**93.0**	–	91.9	–	88.3
JCB	32	**93.6**	90.9	91.8	**93.6**	92.4	86.7	90.0
	64	**93.7**	90.6	91.2	**93.7**	92.9	84.4	85.4
	128	**93.3**	88.8	90.6	**93.3**	92.7	85.8	90.9
	256	**92.8**	–	90.2	–	91.3	–	89.6
Mining Truck	32	88.4	85.2	87.1	**89.6**	89.4	85.1	87.6
	64	89.4	88.6	88.6	**91.5**	90.9	87.0	88.7
	128	**90.0**	89.4	89.5	–	89.7	87.3	87.7
	256	**90.3**	–	88.0	–	89.9	–	86.9
Tractor Scraper	32	90.5	89.3	90.8	91.8	**92.2**	88.4	90.1
	64	89.7	89.1	89.5	91.9	**92.0**	89.9	88.6
	128	**91.8**	89.8	90.2	91.2	91.4	87.9	88.4
	256	–	–	88.9	–	**91.2**	–	86.5

As mentioned previously, the time complexity of MR method is lower as, firstly, the size of input to the pre-processing step of visibility checks is lower when compared to SR method. The smaller input size to visibility checks step implies that the size of input to optimization algorithm is also, low. Therefore, even though we use the same optimization methods, the overall computational time required for pre-processing and optimization steps is lower for MR method by several factors. the clustering step has low complexity when compared against the steps of visibility checks and optimization. For simulated data, the time required for clustering varied from $1 - 10ms$ for 'S' size category to $\sim$ $500ms$ for 'H' size category. For the mining truck vehicle model in the highest size category (256), computational time for the clustering step was $\sim$ $10s$. This is only a fraction of the total computational time that includes times for clustering, visibility checks and optimization (see the corresponding entry in Table 4). From

TABLE 4. Table showing total optimization time (in seconds) taken by the four optimization algorithms (LP, GH, RWLS and LH-RPSO) using both MR and SR methods for the four real vehicle models in four sizes (32, 64, 128, 256) for each vehicle model.

Vehicle	Size	LP	GH		RWLS		LH-RPSO	
		MR	SR	MR	SR	MR	SR	MR
Bulldozer	32	138.95	42.03	5.56	1035.90	67.79	45.06	21.80
	64	834.49	584.73	16.45	11273.17	227.34	573.84	140.71
	128	1426.68	13410.00	163.46	47665.15	469.09	13116.02	239.98
	256	3655.41	–	1797.59	–	2133.70	–	1778.98
JCB	32	246.09	36.44	4.64	736.65	71.88	39.00	19.02
	64	1806.86	469.25	20.99	8735.27	307.37	450.68	119.18
	128	1753.34	8300.07	142.37	35591.02	601.00	7934.27	200.45
	256	3869.25	–	1388.99	–	2162.99	–	1702.46
Mining Truck	32	79.28	229.89	11.30	576.52	38.25	223.29	30.7
	64	736.89	4804.36	56.38	9711.18	200.69	4469.13	172.09
	128	1538.45	1.42 e+05	849.32	–	1225.67	1.19 e+05	1015.02
	256	14347.50	–	11084.80	–	12079.00	–	11440.37
Tractor Scraper	32	233.91	30.81	3.89	631.55	70.51	29.57	17.36
	64	1668.14	613.31	18.74	9573.47	374.17	600.47	117.49
	128	834.01	12487.27	120.64	26889.26	439.61	12435.59	217.91
	256	–	–	1811.24	–	2463.68	–	1879.59

Tables 2 and 4 we can see that the total computational time for MR method is always less than the corresponding entry for SR method. The only exceptions are the times shown for LH-RPSO method for all size categories and all methods for 'S' size in Table 2. For 'S' size category, for example, the car model has 75 camera poses. By clustering them into 20 clusters we do not gain any noticeable speedup. In fact, when using MR method by optimizing twice (i.e., at two resolutions), computational time only increases as the complexity of visibility checks step or optimization step does not change noticeably when the number of variables is so low. Computational time for most of our selected optimization algorithms increases exponentially with an increase in the number of input variables. Therefore the gap between computational times, for SR and MR methods, becomes visible with increasing size of the OCP instance.

Only the LH-RPSO method has constant complexity as it depends on the number of particles rather than the number of input variables. Therefore, in all cases for simulated data, the times shown for LH-RPSO method using MR optimization is higher than the corresponding entry for SR method, primarily due to optimizing for camera poses multiple times. The results for LH-RPSO method are however, different on real data. From the last two columns in Table 4, we can see that LH-RPSO (MR) method is 1.7 times (tractor scraper-32) to 117 times (mining truck-128) faster than LH-RPSO (SR). Despite doubling or tripling (depending on the number of resolutions) the time required for optimization, the MR method when using LH-RPSO takes overall less time computational time on real data. This is because of the significant amount of time saved during the visibility checks step. High complexity of the pre-processing step, when the size of input is large, overshadows the computational complexity of the optimization step, particularly when using GH or LH-RPSO optimization methods. LP and RWLS are however, complex methods, and owing to this complexity, they can produce better coverage when compared to GH or LH-RPSO optimization methods. SR Optimization for 256 size category on real data could not be completed as the visibility checks step took more than 48 hours for all the vehicle models. Optimizations using RWLS method on real data for 128 size category, also were not completed as the total time for visibility checks and optimization took more than 48 hours. Therefore, for the remaining 24 instances, i.e., all instances excluding 'S' size category in simulated data and 256-size category on real data, the MR method is at least 1.3 times (Truck-M (GH)) and up to 167 times (Mining Truck-128 (GH)) faster than the SR method.

TABLE 5. Table showing computational time (in seconds) for the individual steps of pre-processing and optimization for the tractor scraper model using SR optimization and the bulldozer model using MR optimization. The shown values are for the 32 and 128 size categories.

Instance	pre-processing	optimization		
		GH	RWLS	LH-RPSO
scraper-32(SR)	22.33	8.48	609.22	7.27
scraper-128(SR)	12422.20	65.07	14467.06	13.39
bulldozer-32(MR)	0.75	4.81	67.04	21.05
bulldozer-128(MR)	140.05	23.41	329.04	99.75

Table 5 shows the time taken for the individual steps of visibility checks and optimization. In the table, we show the computational times for the tractor scraper-32 and tractor scraper-128 instances using SR optimization strategy and bulldozer-32 and bulldozer-128 instances using MR method, only as an example to highlight the share of computational times of the pre-processing and optimization steps in the total time. These trends, however, apply across all instances with minor variations in the actual times as per the size category and the number of variables contained in the models. We did not present individual times for all instances and methods to keep the results concise, and also because, our aim is to highlight gain in total computational times, i.e., time taken for all steps including pre-processing, optimization and clustering steps. The time required for the pre-processing step of visibility checks at 128 size category is over

500 times more than that at 32 size category when using SR optimization, whereas, it is only 185 times when using MR optimization. Similarly, the required time for optimization using RWLS method at 128 size category is 23.7 times more than the time at 32 size category when using SR method, whereas, the increase is only 4.9 times when using MR method. Finally, it can be said that by optimizing for camera poses on smaller subsets of data at multiple resolution levels, the overall computational time can be reduced significantly. Moreover, as the MR method produces coverage within an average gap of $[-1, 2]\%$ of the coverage obtained with SR method, it can be said that our proposed clustering method provides a good approximation of the vehicle's surface.

Although it might be interesting to run t-tests on the computational times to see if the difference in times are statistically different, we did not perform those tests as it is evident from Tables 2 and 4 that the computational times for MR method are significantly less than those of SR method. With the help of MR method, optimization was completed in reasonable time on large instances where the SR method took more than 48 hours, or, where the LP method could not be used due to a large number of input variables. However, the MR method also has a limitation when using the LP method. Optimization for tractor scraper-256 (MR) using LP optimization was not completed because, at the last resolution, the number of variables after visibility checks was high, leading to the *out-of-memory* error. While the tests on the tractor scraper model were run with $K = 110$, the optimization on tractor scraper-256 (MR) using LP method was run until completion when the parameters were altered as, $l = 100$ and $K = 85$ to obtain a coverage of 88.5%. Looking at the results from other optimization methods on this instance, it can be said that a lower value of K degraded the result. As this optimization could not be completed on this instance, it may be that this is the limit beyond which the LP optimization cannot be used even with MR strategy.

Moreover, this brings us to the importance of the role played by parameter K in the MR method. For the experiments on real data, we selected the optimal value of K for each vehicle model by running the GH method for $K = 90, 95, \ldots, 105, 110$ and picking K that provided best coverage. On simulated data, we set $K = 30$, for all size categories of car, van and truck models and $K = 50$ for the bus model. In the case of real data, $K = 95$ produced the best coverage for bulldozer and JCB models while $K = 110$ produced best coverage for mining truck and tractor scraper models. We run GH optimization using MR method on the tractor scraper-64 instance with the values of K from 90 to 110 at steps of one. The mean control point coverage obtained in these tests was 89.3 with a variance of 0.98. In general, the optimal value K varies from model to model and there does not exist a general strategy to select an optimal value of K. The user must manually choose a value K that is appropriate to the data. However, it is possible that a small perturbation around the chosen value of K may produce better results.

VI. CONCLUSION AND FUTURE WORK

We proposed a new clustering-based multi-resolution optimization method for the optimal camera placement problem for vehicle surround-view. It was shown that with our proposed method the OCP problem can be solved for large real world vehicle models in significantly less time. Moreover, with the MR method, linear programming-based branch-and-bound method can be used for large data where otherwise, SR optimization strategy does not work due to the optimization method's high resource requirements. Results from experiments on eight simulated and real vehicle models of various sizes show that MR method produces the same or even better camera coverage than the SR method, in only a fraction of the time. Results show that our proposed method is over 150 times faster than state-of-the-art. The fact that the coverage values obtained using MR method lie within $1 - 2\%$ difference of the coverage values obtained with SR method, shows that our proposed clustering method effectively captures the 3D geometry of the vehicle's surface. With the help of clustering into different resolution levels, we can decrease size of the input and the number of variables, thereby, decreasing the computational times of the pre-processing step, as well as, the optimization step by a big factor.

While, the MR method extends the applicability of LP optimization method to large data, it still faces limitations, when the number of clusters is kept high. It will be interesting to study other clustering strategies to see if this problem can be circumvented. We test the method only for the use-case scenario of OCP for vehicle surround-view. The method must be studied on other types of data (floor plan surveillance, for example) to examine the generality of MR optimization strategy. Establishing a relationship between the number of clusters, K, and the overall camera coverage may also help to generalize the method for general applications.

REFERENCES

[1] D. Buljeta, M. Vranjes, Z. Marceta, and J. Kovacevic, "Surround view algorithm for parking assist system," in *Proc. Zooming Innov. Consum. Technol. Conf. (ZINC)*, May 2019, pp. 21–26.

[2] V. Appia, H. Hariyani, S. Sivasankaran, S. Liu, K. Chitnis, M. Mueller, U. Batur, and G. Agarwa, "Surround view camera system for ADAS on TI's TDAx SoCs," White Paper, 2015.

[3] A. Hedi and S. Lončarić, "A system for vehicle surround view," *IFAC Proc. Volumes*, vol. 45, no. 22, pp. 120–125, 2012.

[4] F. Angella, L. Reithler, and F. Gallesio, "Optimal deployment of cameras for video surveillance systems," in *Proc. IEEE Conf. Adv. Video Signal Based Surveill.*, Sep. 2007, pp. 388–392.

[5] X. Zhang, X. Chen, J. L. Alarcon-Herrera, and Y. Fang, "3-D model-based multi-camera deployment: A recursive convex optimization approach," *IEEE/ASME Trans. Mechatronics*, vol. 20, no. 6, pp. 3157–3169, Dec. 2015.

[6] P. Mantini and S. K. Shah, "Camera placement optimization conditioned on human behavior and 3D geometry," in *Proc. 11th Joint Conf. Comput. Vis., Imag. Comput. Graph. Theory Appl.*, 2016, pp. 227–237.

[7] P. Rahimian and J. K. Kearney, "Optimal camera placement for motion capture systems," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 3, pp. 1209–1221, Mar. 2016.

[8] B. Bogaerts, S. Sels, S. Vanlanduit, and R. Penne, "Interactive camera network design using a virtual reality interface," *Sensors*, vol. 19, no. 5, p. 1003, Feb. 2019.

[9] E. Hörster and R. Lienhart, "On the optimal placement of multiple visual sensors," in *Proc. 4th ACM Int. Workshop Video Surveill. Sensor Netw.*, 2006, pp. 111–120.
[10] N. Kirchhof, "Optimal placement of multiple sensors for localization applications," in *Proc. Int. Conf. Indoor Positioning Indoor Navigat.*, Oct. 2013, pp. 1–10.
[11] S. Boyd and J. Mattingley, "Branch and bound methods," Stanford Univ., Stanford, CA, USA, Tech. Rep., EE364b, 2007, pp. 2006–2007.
[12] F. Hoffmann, "On the rectilinear art gallery problem," in *International Colloquium on Automata, Languages, and Programming*. Cham, Switzerland: Springer, 1990, pp. 717–728.
[13] U. M. Erdem and S. Sclaroff, "Automated camera layout to satisfy task-specific and floor plan-specific coverage requirements," *Comput. Vis. Image Understand.*, vol. 103, no. 3, pp. 156–169, 2006.
[14] J. Kritter, M. Brevilliers, J. Lepagnot, and L. Idoumghar, "On the real-world applicability of state-of-the-art algorithms for the optimal camera placement problem," in *Proc. 6th Int. Conf. Control, Decis. Inf. Technol. (CoDIT)*, Apr. 2019, pp. 1103–1108.
[15] Y. Morsly, N. Aouf, M. S. Djouadi, and M. Richardson, "Particle swarm optimization inspired probability algorithm for optimal camera network placement," *IEEE Sensors J.*, vol. 12, no. 5, pp. 1402–1412, May 2011.
[16] X. Wang, H. Zhang, S. Fan, and H. Gu, "Coverage control of sensor networks in IoT based on RPSO," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3521–3532, Apr. 2018.
[17] X. Wang, H. Zhang, and H. Gu, "Solving optimal camera placement problems in IoT using LH-RPSO," *IEEE Access*, vol. 8, pp. 40881–40891, 2019.
[18] P. Liu, Q. Hu, K. Jin, G. Yu, and Z. Tang, "Toward the energy-saving optimization of WLAN deployment in real 3-D environment: A hybrid swarm intelligent method," *IEEE Syst. J.*, early access, Apr. 5, 2021, doi: 10.1109/JSYST.2021.3065434.
[19] J. Zhao, R. Yoshida, S.-C.-S. Cheung, and D. Haws, "Approximate techniques in solving optimal camera placement problems," *Int. J. Distrib. Sensor Netw.*, vol. 9, no. 11, Nov. 2013, Art. no. 241913.
[20] M. Brévilliers, J. Lepagnot, L. Idoumghar, M. Rebai, and J. Kritter, "Hybrid differential evolution algorithms for the optimal camera placement problem," *J. Syst. Inf. Technol.*, vol. 20, no. 4, pp. 446–467, Nov. 2018.
[21] J.-W. Ahn, T.-W. Chang, S.-H. Lee, and Y. W. Seo, "Two-phase algorithm for optimal camera placement," *Sci. Program.*, vol. 2016, pp. 1–16, Sep. 2016.
[22] C. Gao, X. Yao, T. Weise, and J. Li, "An efficient local search heuristic with row weighting for the unicost set covering problem," *Eur. J. Oper. Res.*, vol. 246, no. 3, pp. 750–761, Nov. 2015.
[23] W. Lin, F. Ma, Z. Su, Q. Zhang, C. Li, and Z. Lü, "Weighting-based parallel local search for optimal camera placement and unicost set covering," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2020, pp. 3–4.
[24] J. Kritter, M. Brévilliers, J. Lepagnot, and L. Idoumghar, "On the optimal placement of cameras for surveillance and the underlying set cover problem," *Appl. Soft Comput.*, vol. 74, pp. 133–153, Jan. 2019.
[25] J. Liu, S. Sridharan, and C. Fookes, "Recent advances in camera planning for large area surveillance: A comprehensive review," *ACM Comput. Surveys*, vol. 49, no. 1, pp. 1–37, Jul. 2016.
[26] M. Rebai, M. L. Berre, F. Hnaien, and H. Snoussi, "Exact biobjective optimization methods for camera coverage problem in three-dimensional areas," *IEEE Sensors J.*, vol. 16, no. 9, pp. 3323–3331, May 2016.
[27] A. Mavrinac and X. Chen, "Modeling coverage in camera networks: A survey," *Int. J. Comput. Vis.*, vol. 101, no. 1, pp. 205–226, 2013.
[28] A. Mavrinac, X. Chen, and Y. Tan, "Coverage quality and smoothness criteria for online view selection in a multi-camera network," *ACM Trans. Sensor Netw.*, vol. 10, no. 2, pp. 1–19, Jan. 2014.
[29] J. F. Thompson, Z. U. Warsi, and C. W. Mastin, *Numerical Grid Generation: Foundations and Applications*. Amsterdam, The Netherlands: Elsevier, 1985.
[30] C. Trader. *CGtrader*. Accessed: Aug. 10, 2021. [Online]. Available: https://www.cgtrader.com/
[31] F. S. Nooruddin and G. Turk, "Simplification and repair of polygonal models using volumetric techniques," *IEEE Trans. Vis. Comput. Graph.*, vol. 9, no. 2, pp. 191–205, Apr./Jun. 2003.
[32] P. Min. *BinVox*. Accessed: Feb. 5, 2021. [Online]. Available: http://www.patrickmin.com/binvox and https://www.google.com/search?q=binvox
[33] J. W. Harris and H. Stöcker, *Handbook of Mathematics and Computational Science*. Cham, Switzerland: Springer, 1998.
[34] S. J. Ray and J. Teizer, "Computing 3D blind spots of construction equipment: Implementation and evaluation of an automated measurement and visualization method utilizing range point cloud data," *Autom. Construct.*, vol. 36, pp. 95–107, Dec. 2013.
[35] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, Jul. 1987.
[36] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognit.*, vol. 36, no. 2, pp. 451–461, Feb. 2003.
[37] T. Hastie, R. Tibshirani, and J. Friedman, "The elements of statistical learning: Data mining, inference, and prediction, 20 springer series in statistics,".
[38] K. Klasing, D. Wollherr, and M. Buss, "A clustering method for efficient segmentation of 3D laser data," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2008, pp. 4043–4048.
[39] H. Kisner and U. Thomas, "Segmentation of 3D point clouds using a new spectral clustering algorithm without a-priori knowledge," in *Proc. 13th Int. Joint Conf. Comput. Vis., Imag. Comput. Graph. Theory Appl.*, 2018, pp. 315–322.
[40] P. Scheunders, "A comparison of clustering algorithms applied to color image quantization," *Pattern Recognit. Lett.*, vol. 18, nos. 11–13, pp. 1379–1384, Nov. 1997.
[41] J. Papon, A. Abramov, M. Schoeler, and F. Worgotter, "Voxel cloud connectivity segmentation–supervoxels for point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 2027–2034.
[42] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, May 2012.
[43] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 1–4.
[44] C. U. Manual, "Ibm ilog cplex optimization studio," *Version*, vol. 12, pp. 1987–2018, May 1987.

V. ANIRUDH PULIGANDLA (Member, IEEE) was born in Hyderabad, Telangana, India, in 1992. He received the B.Tech. degree in electronics and communications engineering from Amity University, Jaipur, Rajasthan, India, in 2014, the B.Sc. degree in computer vision and robotics from the University of Burgundy, France, in 2016, and the M.Sc. degree in computer vision and robotics from the University of Burgundy, the University of Girona, Spain, and Heriot Watt University, U.K., in 2018, as part of the Erasmus Mundus Joint Master's Degree Program. He is currently pursuing the Ph.D. degree with the University of Zagreb, Zagreb, Croatia, under the Marie-Curie Actions ITN Fellowship. His research interests include discrete and continuous optimization, signal and image processing, and 3D reconstruction from multiple camera systems using multi-view stereo.

SVEN LONČARIĆ (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Cincinnati, OH, USA, in 1994, as a Fulbright Scholar. From 2001 to 2003, he was an Assistant Professor at the New Jersey Institute of Technology, USA. He is currently a Professor of electrical engineering and computer science at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. He is the Director of the Center for Computer Vision, University of Zagreb; and the Head of the Image Processing Group. He is the Co-Director of the Center of Excellence in Data Science and Cooperative Systems. He was the principal investigator on a number of research and development projects. He has coauthored more than 250 publications in scientific journals and conferences. His research interests include image processing and computer vision. He is a member of the Croatian Academy of Technical Sciences. He has received several awards for his scientific and professional work. He was the Chair of the IEEE Croatia Section.