2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Occlusion Handling for Industrial Robots

Ling Zhu, Meghna Menon, Mario Santillo and Gregory Linkowski

Abstract— **Industrial robots contain minimal sensing capability beyond recognition of their internal state. It is critical that an external vision system should cover the designated robot work space with awareness of blind spots and occlusions. This work presents two mechanisms to handle occlusions in an external multi-robot vision system: occlusion-aware optimal sensor positioning, and event-driven occlusion detection. When deploying sensors to the system, various scenarios are considered during optimization to reduce potential occlusions and increase sensor coverage. These methods are tested on a working cell with three industrial robot arms. The experimental results demonstrate the effectiveness of the proposed scenario-based multi-objective optimization for sensor positioning. Once the sensors are deployed, occlusion detection is actively triggered prior to robot path planning.**

I. Introduction

Industrial robots are heavily used across the manufacturing industry for their speed, accuracy and load-carrying capabilities. For these reasons, however, industrial robots are often bound to operate within caged spaces, physically separated from human interaction and requiring technician training to even perform routine tasks. Additionally, these robots contain little on-board sensing for dynamic path adjustment or recognition of an impending collision.

In circumstances where it is necessary to partially or fully uncage the industrial robots, 2D LiDAR-based safety scanners can be installed to ensure safe operation of the robot. Not only can these sensors be expensive, but also they generally require increasing the work cell footprint to account for human motion entering the guarded space. Additionally, elaborate scanner-field programming may be required to operate the work space as intended. Traditionally, scanner coverage is configured in an ad-hoc basis without regard for minimization of potential occlusions. Any external sensing which generates a 3D point cloud (PCD) or a 2D image is also inevitably hindered by such occlusions.

In line with the vision of Industry 4.0 [13], facilities are turning more toward collaborative robots and multi-robot systems. This necessitates a more robust ability to track and coordinate such robots while minimizing occluded regions from movement of manipulators, material objects, and human counterparts in the scene. An example of this can be seen in Figure 1. In the first image, a robot is fully captured by a two RGB-D cameras. However, as the robot performs movements to complete a task, full coverage of the robot is lost. To ensure coverage of robot work space for advanced safety and path-planning schemes, the combined optimal deployment of sensors is critical.

Authors are with Ford Motor Company, Research and Advanced Engineering, Dearborn, MI 48124 United States `lzhu40, mmenon8, msantil3, glinkows@ford.com`

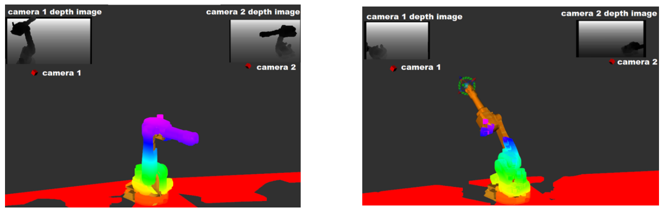

Fig. 1. The robot is fully covered by two RGB-D cameras (top), and inadequate coverage occurs after the robot moves to a different orientation (bottom). Coverage is denoted by rainbow fill on the robot.

In order to effectively minimize occluded regions within a robot work space and allow for safe autonomy and collaboration of industrial robots, we propose a novel framework to handle the occlusions: occlusion-aware optimal positioning for multi-modal sensors, coupled with event-driven occlusion detection. When placing sensors in a defined work space, we perform multi-scenario optimization [8] to determine optimal placement to cover dynamic robot movements in the work space. This is achieved via input of randomized or user-defined robot poses, minimizing potential occlusion through different scenarios. Once the sensors are deployed, occlusion detection is performed during robot path planning to ensure safety.

There have been several works in the area of work space coverage through optimal placement of sensors to generate accurate measurements with minimized propagated error [9], [11], [17], [12]. These papers utilize methods such as greedy search, dual sampling, convex optimization and genetic algorithms with the goal of finding a global optimum across the array of sensors in the space. In [22], the author in particular highlights the optimized placement of light sources in addition to the sensors themselves. However, these works fail to consider occlusions that are dynamically changing in the scenes. For industrial robots, when multiple robots are presented in the work space, the occlusions change based on the movement of each robot. Additionally, work space is often limited and the robots perform routine tasks. Thus, consideration of the most frequent robot poses and reduction of occlusions for these poses would enhance the system performance. As far as the work is concerned, this research is the first in considering different poses with occlusion in the area of optimal sensor positioning.

There also have been many works presented in the area of occlusion detection and occlusion-aware dynamic object tracking. The prior art can be categorized between the use of 2D and 3D vision systems. Several papers utilize parameters such as probabilistic visibility, spatiotemporal data around the target, and Haar-like features to determine occlusions in 2D images [16], [18], [21], [2]. When using 3D data or generating 3D reconstructions through multi-view images, use of segmentation, transformations of image views, and neural network approaches were employed to characterize the occluded regions [20], [15], [14], [1], [19]. For industrial robots, these image-based recognition methods are subject to much higher safety standards, and ultimately not suited for the application. Recent work [10] uses raycasting to configure the free space and occlusions surrounded by self-driving car, and it's applicable for mobile robots not the sensing from fixture.

Our framework addresses limitations in these works through utilization of fused depth data from a multiple depth cameras. This fused data is converted into a 2D matrix (i.e. low-resolution depth images), which allows reduced latency between the cameras and server. Applying the same occlusion detection algorithm, we formulate multiple objective functions considering multiple scenarios in the optimization, and minimize occluded regions in all scenarios using multi-objective optimization. The occlusion detection algorithm is completely separable to each voxel, which can leverage the computation power of GPUs.

II. Occlusion Handling Framework

There are two aspects of occlusion handling that we aim to address in our work. Figure 2 shows the occlusion handling framework.

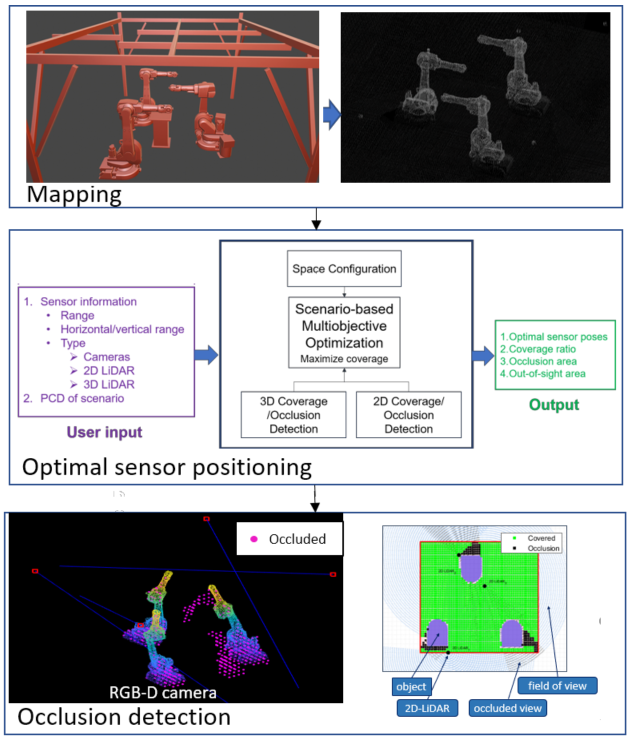

Fig. 2. Framework for occlusion handling

A. Mapping of Robot Work Cell

The first step is to map the robots to various desired poses via random generation of valid joint positions in space. For each pose set, the mapping process generates a corresponding PCD of the work cell from simulated RGB-D cameras positioned around the space (Figure 2, top).

B. Optimization of Sensor Coverage

The PCDs are then used as inputs to the optimization algorithm, with each PCD considered as a scenario. Once all the different PCDs are obtained, the optimization algorithm runs offline.

From the user input, the objective function utilizes the occlusion detection algorithm presented in Section III to establish the covered, out-of-sight, and occluded regions of the work space. A multi-objective optimization is proposed in Section IV maximizes the covered volume for all scenarios (poses), and provides multiple optimal solutions. User is then able to choose a preferred solution that fits the demands of their work space. Running the optimization with specified robot poses (scenarios) and commonly held poses ensures higher coverage.

With the optimal camera positions selected from the framework output, the user can deploy the cameras in their physical work cell, networked to a centralized server. Once all the sensors have been deployed and calibrated, our occlusion detection algorithm can monitor and report warnings regarding occluded regions in critical areas prior to robot path planning.

III. Occlusion Detection

Our design of the occlusion detection algorithm focuses on two aspects: accurate detection and event-driven capability. For accurate detection, multiple 3D sensors such as RGB-D or LiDAR are used, which provide depth information with high accuracy comparing to 2D information. However, handling PCD information from multiple sensors in real-time can be computationally expensive. To reduce latency of the detection algorithm, only downsampled depth images from the camera system are required by the central server. In addition, the algorithm applies to completely separable spaces within the multi-camera system to allow parallelization.

A. Detection Algorithm

The first step of performing detection is space configuration of the given area that needs to be examined. This process is performed through voxelization of the given space. After this process is performed, each voxel is checked with a downsampled depth map from each RGB-D camera to determine if the voxel is covered by any of the cameras. Since the depth images can be considered as ray-based, each voxel is calculated based on the location to find if it aligns with the specific ray from each camera. Figure 3 shows the raycasting procedure using depth images. In the figure, the 3D green box within the camera field of view (FOV) is a voxel within the work space. This voxel is evaluated against the corresponding value in the downsampled depth image

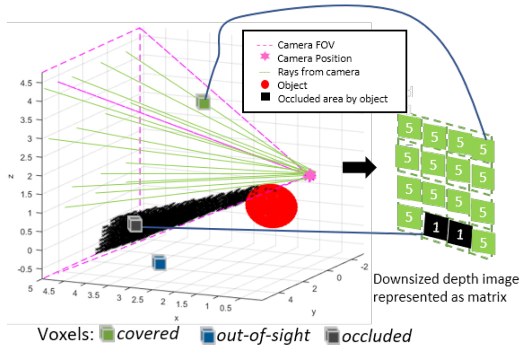
Camera FOV
Camera Position
Rays from camera
Object
Occluded area by object
Downsized depth image represented as matrix
Voxels: covered out-of-sight occluded

Fig. 3. Occlusion detection using raycasting: The voxel is traced to the corresponding ray from the camera, checked against the depth matrix, and marked accordingly.

(numerical values corresponding to distance from the camera) shown at the right, and marked as one of the following flags: [*covered*, *occluded*, *out-of-sight*]. These *covered* and *occluded* voxels change as events occur in the work space (i.e. robot movement or materials entering the cell). *Out-of-sight* voxels remain the same unless the cameras are re-positioned. This allows algorithm to only require checking the out-of-sight voxels once upon initialization, and reduces complexity of detection with regards to *covered* and *occluded* voxels. The time complexity of the detection algorithm for each voxel is $O(M)$, where M is defined as the number of cameras. Voxels can be run through the detection algorithm in parallel. The detailed algorithm is shown below.

Algorithm 1 Occlusion detection algorithm

```
1: for each $V_i \in V$ do
2:     $V_i \leftarrow out-of-sight$
3:     $[x_{world}, y_{world}, z_{world}] = getXYZ(V_i)$
4:     for each $m \leftarrow 1 \, to \, M$ do
5:         $[x_{camera_m}, y_{camera_m}, z_{camera_m}] = Transform(R_m, O_m, [x_{world}, y_{world}, z_{world}])$
6:         $[pitch, yaw] = calcualte\_ray\_angle([x_{camera_m}, y_{camera_m}, z_{camera_m}])$
7:         if $pitch \in [pitch_{min}, pitch_{max}] \wedge yaw \in [yaw_{min}, yaw_{max}]$ then
8:             $[d_j, d_k] = angle2index([pitch, yaw])$
9:             $V_i \leftarrow occluded$
10:            if $D_m[d_j, d_k] < x_{camera_m}$ then
11:                $V_i \leftarrow covered$
12:                Break
13:            end if
14:        end if
15:    end for
16: end for
```

In the algorithm, V is the voxel array, R_m and O_m are the rotation matrix and translation of m^{th} camera to world frame, *pitch* and *yaw* are the angles of the ray from camera origin to the voxel, and D_m is the downsampled depth image of m^{th} camera. Line 3 calculates the location of the voxel in world frame. Then in line 4 - line 14, the algorithm checks whether each voxel is occluded. The outer for loop at line 1 can be run in parallel. Note that after the first run of the algorithm, *out-of-sight* voxels can be removed from the set V to further reduce the computation.

B. Depth Image Resolution

Depth image downsampling is optional, but it becomes critical when sensors communicate information to a server in a wireless manner. If each sensor is equipped with its own

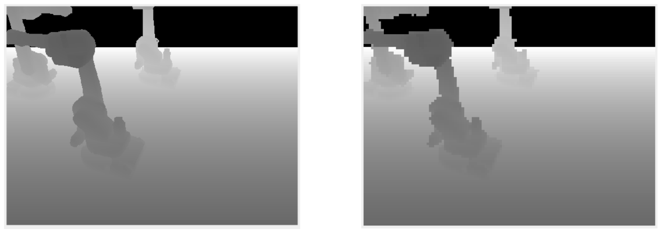

Fig. 4. Depth images of robot work space: original (left, 480x640), downsampled (right, 87x116)

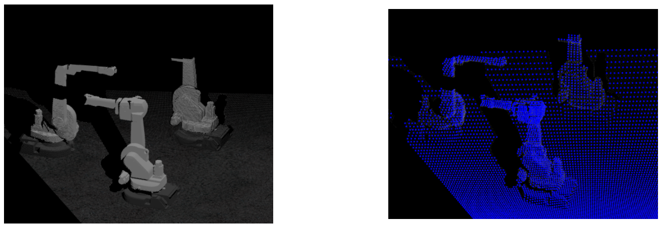

Fig. 5. Verification of downsampled depth matrix by re-generating PCD

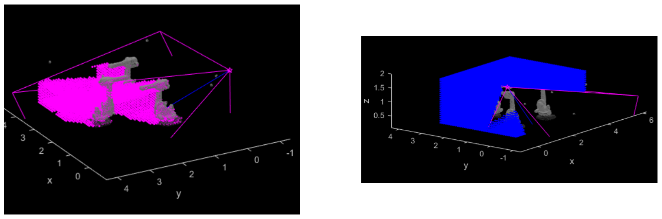

Fig. 6. Occluded (left) and out-of-sight areas (right) for a single camera

processor, transmitting downsampled depth matrix highly improves system performance. This downsampling can also smooth the depth image to reduce sensor noise without removing valuable information. From the experiment, Figure 4 shows the original depth image downsampled from one camera viewing the work space.

After reducing the resolution, the maximum voxel size within the camera sight is $2cm$. From the visualization, the downsampled depth matrix is still able to adequately capture the surface of the robots. To further verify that information is not lost through this process, we re-generated the PCD using the same downsampled depth matrix shown in the right image from Figure 4. In Figure 5, the left image is a full resolution PCD from one RGB-D camera, and the right image is the PCD generated from downsampled depth matrix. When comparing re-generated PCD with the original, though it is generated from a low-resolution depth matrix, the surfaces of the robot are still captured. Figure 6 shows the occluded area and *out-of-sight* area of a single camera found by the occlusion detection algorithm. It also uses the downsampled depth matrix shown in Figure 4.

IV. Scenario-based Multi-objective Optimization for Sensor Positioning

To generate optimal sensor poses, we perform optimization targeting to maximize the volume covered by the 3D sensors. In performing this optimization, the *out-of-sight* volume and *occluded* volume are minimized. For a given sensor position, the *occluded* volume is non-deterministic and incorporates the robot's poses to determine the volume.

To effectively evaluate this volume, we consider multiple sets of poses (scenarios) in the multi-robot setup, and optimize the covered volume in all of these scenarios, resulting in a multi-objective optimization problem. Traditionally, to handle multiple scenarios in optimization, all scenarios are converted into a single objective optimization problem by aggregation such as through a weighted sum. However, the use of a single objective optimization only provides a single optimal solution. In the case of varied robot poses, scenarios will conflict and the single optimal solution will not provide a satisfactory trade-off between scenarios. Instead, we utilize multi-objective optimization for multi-scenario problems presented in [7], [23], [8] which provides a set of solutions that represent different trade-offs among objectives. From this set of solutions, a user can choose one of the solutions based on priority and necessity with respect to the evaluated task. Primarily, user-specified poses that might be performed throughout the defined task are added as possible scenarios to evaluate the quality of the solution. For example, if user evaluates multi-robot part picking from specified bins, the user can specify scenarios related to the picking poses such that the optimization of sensor positioning minimizes occlusions throughout the task performed. The user might provide scenarios such as robot being stationed in a ready pose, or one robot performing the part picking task while the others have remained in a ready-state pose.

A. Multi-objective Optimization

In multi-objective optimization problems, the set of solutions obtained are considered to be equally good. These solutions, defined as non-dominated solutions or Pareto optimal solutions [5], provide trade-offs between objectives.

1) Problem formulation: The scenario-based multi-objective sensor positioning problem is formulated as follows:

$$\begin{array}{ll} \text{Maximize} & f_1(\mathbf{x}) = C(\mathbf{x}, PCD_1), \\ \text{Maximize} & f_2(\mathbf{x}) = \sqcup_{k=2}^{N} C^{(k)}(\mathbf{x}, PCD_k) \\ \text{subject to} & G_i = 1, i \in [1, N], \\ & x_i^{min} \le x_i \le x_i^{max}, i \in [1, M] \end{array} \quad (1)$$

This is a constrained bi-objective maximization problem. The constant G_i is the coverage ratio of the critical area. The user can add other constraints as well. The objective function is evaluated in two possible forms:

- The covered volume of one scenario is directly assigned to the objective function (e.g., f_1).
- A set of covered volumes from multiple scenarios are aggregated, and aggregated value is assigned to the objective function (e.g., f_2).

With a significant scenario, the first form should be used to emphasize the scenario. For the aggregation form, we use a worst-case aggregate function which evaluates the worst value for all K scenarios:

$$\sqcup_{k=1}^{K} f^{(k)}(\mathbf{x}) = \min_{k=1}^{K} f^{(k)}(\mathbf{x}). \quad (2)$$

This is by far the most widely used aggregate function in practice. It takes the most pessimistic case and results "over-designed" solutions for some scenarios.

The covered volume for a given scenario (robot poses represented as PCD) is given below. First, the depth matrix is generated from the PCD of k^{th} scenario for each sensor m. With all depth matrices, the occlusion detection algorithm finds the volume covered by sensor, and the covered ratio of all working area is returned to calculate objective function. This covered ratio of the given critical area is assigned to the constraint function.

Algorithm 2 Objective and constraint function (Scenario k)

```
1: $Initialize(V)$
2: for each $m \leftarrow 1\,to\,M$ do
3:     $D_m \leftarrow PCD2DepthMatrix(PCD_k, R_m, O_m)$
4: end for
5: $V \leftarrow occlusion\_detection(V, D, R, O)$ (see Algorithm 1)
6: $C \leftarrow coverageRatio(V)$
7: $G \leftarrow coverageRatio(V_{critical})$
8: return $C, G$
```

2) NSGA-II: In this work, a fast elitist multi-objective genetic algorithm, known as NSGA-II [3], is used for solving the above multi-objective optimization. This method sorts the population and preserves the good individuals while maintaining diversity of the search. Using non-dominated sorting and crowding distance, NSGA-II quickly converges to the optimum front as well as maintaining the diversity among the individuals in the front. Since NSGA-II performs better when there are two or three objectives, we keep the number of objectives less or equal to three.

V. Experiments

Experimental study is carried out using a work space with three manipulators shown in Figure 2(top two figures). The volume of the work space evaluated is $4m$ x $3.35m$ x $2m$. First, we deployed multiple RGB-D cameras to cover the work space that the robot arms can reach. To obtain the PCD of different poses, we used the GAZEBO simulation environment setup with RGB-D cameras to construct a PCD of the scene.

For application of the multi-objective optimization, we used 4-6 RGB-D cameras with the same specification as the Intel Realsense D435 camera whose field of view is $[80^o, 58^o, 5m]$ (horizontal, vertical, range) to cover the work space. The height of camera position is fixed to $2.28m$, and the roll is set to 0. The remaining four parameters (position: x and y, yaw and $pitch$) are decided by the optimization. In testing, results from manual setup are also given as a baseline. The camera poses for manual process are set empirically. For the four-camera case, we put cameras at the top four corners of the work space respectively, and positioned the cameras toward the work space at between 30° to 45° angles. In the five-camera case, we put the fifth camera at the center-top of the work space facing a vertically downward direction. With the addition of the sixth camera, we positioned the camera at the center of the edge where two of robots are located.

TABLE I

SOLUTIONS (CAMERA POSES) [X,Y, PITCH, YAW] AND UNCOVERED AREA IN EACH SCENARIO

Method	Solutions		Scenario 1		Scenario 2		Scenario 3		Random	
	#	Pose	m^3	%	m^3	%	m^3	%	m^3	%
Manual	4	[0, 0, 60°, 45°], [4, 0, 60°, 135°], [4, 3.35, −60°, 135°], [0, 3.35, −60°, 45°]	0.50	2.2	0.61	2.7	0.76	3.4	0.52	2.3
	5	[0, 0, 60°, 45°], [4, 0, 60°, 135°], [4, 3.35, −60°, 135°], [0, 3.35, −60°, 45°], [2, 1.675, 0°, 90°]	0.32	1.4	0.44	1.9	0.39	1.7	0.40	1.8
	6	[0, 0, 60°, 45°], [4, 0, 60°, 135°],[4, 3.35, −60°, 135°], [0, 3.35, −60°, 45°], [2, 1.675, 0°, 90°], [2, 0, 45°, 90°]	0.27	1.2	0.28	1.3	0.26	1.2	0.35	1.6
Single Scenario Optimization	4	[0.41, 3.32, −43°, 65°], [3.64, 3.34, −48°, 120°], [0.08, 0.16, 46°, 43°], [4.00, 0.04, 48°, 135°]	**0.19**	**0.8**	0.31	1.4	0.33	1.5	**0.33**	**1.5**
	5	[0.01, 3.28, −52°, 57°], [3.90, 3.33, −37°, 153°], [0.01, 0.03, 11°, 35°], [3.93, 0.10, 53°, 137°], [2.75, 0.83, 32°, 77°]	**0.14**	**0.6**	0.25	1.1	0.22	1.0	0.30	1.3
	6	[0.04, 2.84, −46°, 49°], [1.49, 3.21, −23°, 96°],[3.99, 3.26, −50°, 153°], [0.00, 0.07, 55°, 39°], [1.69, 0.00, 47°, 43°], [3.86, 0.24, 35°, 121°]	**0.13**	**0.6**	0.21	0.9	0.21	0.9	0.28	1.3
Multi Scenario Optimization	4	[0.07, 3.33, −41°, 59°], [3.68, 3.35, −50°, 119°], [0.04, 0.10, 54°, 42°], [3.84, 0.00, 40°, 134°]	**0.19**	**0.8**	**0.24**	**1.0**	**0.24**	**1.0**	**0.33**	**1.5**
	5	[0.01, 3.35, −48°, 56°], [3.77, 3.25, −36°, 146°], [0.00, 0.06, 6°, 35°], [3.90, 0.01, 48°, 137°], [2.69, 0.82, 31°, 76°]	**0.14**	**0.6**	**0.19**	**0.9**	**0.19**	**0.9**	**0.29**	**1.3**
	6	[0.02, 2.98, −44°, 28°], [1.52, 3.33, −37°, 101°], [3.98, 3.28, −49°, 153°] [0.12, 0.10, 53°, 48°], [1.46, 0.00, 41°, 36°], [3.93, 0.24, 20°, 124°]	0.15	0.7	**0.17**	**0.8**	**0.17**	**0.7**	**0.27**	**1.2**

For each fixed number of cameras, the optimization runs independently. We used NSGA-II and single objective GA with a population size of 40, the SBX recombination operator [6] with $p_c = 0.9$ and index $\eta_c = 15$, and the polynomial mutation operator [4] with $p_m = 0.33$ and index $\eta_m = 20$. The algorithms are run for a maximum of $[100-500]$ generations.

A. Optimization Setup

1) Scenarios and objectives:

a) Scenario 1: All robots in ready pose state: The ready pose shown in Figure 2(top two figures) is the pose which occurs most frequently. Every time a robot finishes a task, it returns to ready-pose state. Before planning any task, each robot moves to the ready-state and starts planning. To emphasize this pose, we set this pose as one objective.

b) Scenario 2: One robot moves, other two are in ready pose state: In most cases, one of the robots starts planning and moving. Then given the status of the first robot, the other robots plan accordingly. To initiate this scenario, we mapped a set of poses that one robot moves to various positions, while the other two robots are in ready-state. The objective function uses worst-case aggregation given in Section IV, and takes the worst value of coverage of all these poses, and assigns it the second objective.

c) Scenario 3: All three robots move together: In this scenarios, we set all three robots to different poses. Similar to scenario 2, worst-case aggregated value is assigned to objective function.

User can choose specific poses or a series of poses that mimic the movement of robots. Also, other objects such as table, bins, and workers can be included in the scenario. The scenario-based optimization provides flexibility for the user to design any scenarios including complex or corner-case scenarios.

B. Results

In the single scenario optimization, only scenario 1 is considered in optimization. Hence, only a single objective is presented, and one optimal solution is found. Then, we compared the results with the optimal solution from multi-objective optimization using three scenarios. Table I shows optimal solutions and calculated uncovered area for each scenario. The best values are shown in bold font. For scenario 2 and 3, the worse-case aggregated values are shown in the table. For multi-objective optimization, since the optimization will output multiple solutions, the table only shows one selected solution. For further verification, all solutions shown in table are tested on 25 random robot poses that are not included in the given scenarios, the average value on 25 random poses are shown in the rightmost column of the table.

1) Single scenario optimization: From the table, results from optimization methods outperform a manual setup of sensors in all cases. We found that manual setting generates scattered occlusions and uncovered area. These are very difficult to capture by human eye, and numerous adjustments are required with no guarantee of finding optimal positions. From the table, single objective optimization finds the best solution for scenario 1, and performs slightly worse in the other two scenario compared to multi-objective optimization. However, the slight difference in the value can cause significant impact if the occluded area is in an undesired location. Figure 7 shows the optimal solution for four-camera cases. The left figure shows the optimal poses in scenario 1, and the right figure shows the same positioning in scenario 2. In the figure the blue line is the camera orientation and the red box at the edge of each line is the camera location, and the area in magenta color is occlusion. It shows that the optimal positioning creates occluded area as one robot moves away from ready-state, and the robot cannot complete its task any further due to the occluded area in the path.

2) Multi-objective optimization with multiple scenarios: In multi-objective optimization we use three scenarios, and optimize all scenarios simultaneously using NSGA-II. NSGA-II finds optimal front, and the user can select one of the preferred solutions. The selected optimal solutions from NAGA-II have a comparable performance in scenario 1, and outperforms in scenario 2 and 3.

For random poses, the optimal solutions from NSGA-II

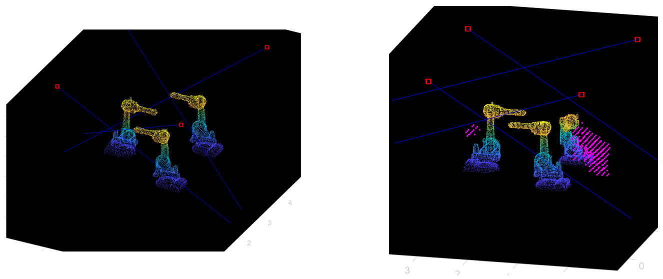

Fig. 7. The optimal solution of single scenario optimization creates occluded area in other scenarios

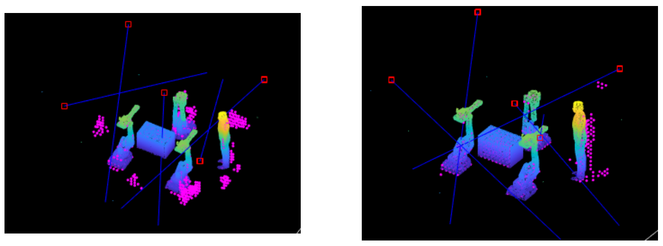

Fig. 8. Manual solution (left) and optimal solution (right) from multi-objective optimization is tested on unexpected objects in the scene

perform better than manual positioning as well as solutions from single scenario optimization. We also tested on unexpected scenario shown in Figure 8. The occlusion detection algorithm is run on a manual positioning of the cameras (left) and the optimal positioning of the camera from NSGA-II (right). The scene includes unexpected objects such as human and table, showing that the optimized positioning provides more robust occlusion aware coverage.

C. Discussion

From the above results, it is clear that using a multi-objective optimization method provides multiple solution with higher coverage and less occlusions under given scenarios. With the optimal location and orientation of cameras, the user can reduce the number of cameras with optimal placement. This can significantly reduce the cost without losing quality of coverage.

VI. CONCLUSIONS

In this paper, we presented two mechanisms to handle occlusion for the industrial robots. Occlusion-aware optimal sensor positioning and event-driven occlusion detection. The occlusion-aware sensor positioning considers multiple robot pose scenarios during the optimization and optimizes sensor coverage throughout all scenarios. Scenario-based multi-objective optimization was adopted to solve the problem. Once the sensors were deployed, event-driven occlusion detection is shown to greatly enhance the safety of the vision system. From the experimental study, the optimal sensor positioning can reduce the number of sensors with high coverage under various scenarios. The work will extend to applications in larger scale industrial environments for dynamic multi-robot path planning and task execution.

REFERENCES

[1] A. Ayvaci, M. Raptis, and S. Soatto, "Sparse occlusion detection with optical flow," *International journal of computer vision*, vol. 97, no. 3, pp. 322–338, 2012.

[2] M. A. Baumann, D. C. Dupuis, S. Léonard, E. A. Croft, and J. J. Little, "Occlusion-free path planning with a probabilistic roadmap," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 2151–2156.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[4] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[5] K. Deb, "Multi-objective optimization," in *Search methodologies*. Springer, 2014, pp. 403–449.

[6] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 2, pp. 115–148, 1995.

[7] K. Deb, L. Zhu, and S. Kulkarni, "Multi-scenario, multi-objective optimization using evolutionary algorithms: Initial results," in *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2015, pp. 1877–1884.

[8] K. Deb, L. Zhu, and S. Kulkarni, "Handling multiple scenarios in evolutionary multiobjective numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 6, pp. 920–933, 2017.

[9] E. Hörster and R. Lienhart, "On the optimal placement of multiple visual sensors," in *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, 2006, pp. 111–120.

[10] P. Hu, J. Ziglar, D. Held, and D. Ramanan, "What you see is what you get: Exploiting visibility for 3d object detection," *CVPR*, 2020.

[11] S. Joshi and S. Boyd, "Sensor selection via convex optimization," *IEEE Transactions on Signal Processing*, vol. 57, no. 2, pp. 451–462, 2008.

[12] J. Kim, Y. Ham, Y. Chung, and S. Chi, "Systematic camera placement framework for operation-level visual monitoring on construction jobsites," *Journal of Construction Engineering and Management*, vol. 145, no. 4, p. 04019019, 2019.

[13] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[14] A. Li and Z. Yuan, "Symmnet: a symmetric convolutional neural network for occlusion detection," *arXiv preprint arXiv:1807.00959*, 2018.

[15] M. Li, B. Guo, and W. Zhang, "An occlusion detection algorithm for 3d texture reconstruction of multi-view images," *International Journal of Machine and Computing*, vol. 7, no. 5, 2017.

[16] A. Mittal and L. S. Davis, "A general method for sensor planning in multi-sensor systems: Extension to random occlusion," *International journal of computer vision*, vol. 76, no. 1, pp. 31–52, 2008.

[17] G. Olague and R. Mohr, "Optimal camera placement for accurate reconstruction," *Pattern Recognition*, vol. 35, no. 4, pp. 927 – 944, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320301000760

[18] J. Pan and B. Hu, "Robust occlusion handling in object tracking," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2007, pp. 1–8.

[19] C. Rauch, T. Hospedales, J. Shotton, and M. Fallon, "Visual articulated tracking in the presence of occlusions," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 643–650.

[20] S. Schneider, M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Fusing vision and lidar-synchronization, correction and occlusion reasoning," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 388–393.

[21] Y. Xu, L. Qin, G. Li, and Q. Huang, "An efficient occlusion detection method to improve object trackers," in *2013 IEEE International Conference on Image Processing*. IEEE, 2013, pp. 2445–2449.

[22] S. Yi, R. M. Haralick, and L. G. Shapiro, "Optimal sensor and light source positioning for machine vision," *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 122–137, 1995.

[23] L. Zhu, K. Deb, and S. Kulkarni, "Multi-scenario optimization using multi-criterion methods: A case study on byzantine agreement problem," in *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2014, pp. 2601–2608.