Dynamic Multi-Target Coverage with Robotic Cameras

Wolfgang Hönig and Nora Ayanian

***Abstract*— When tracking multiple targets with autonomous cameras for 3D scene reconstruction, e.g., in sports, a significant challenge is handling the unpredictable nature of the targets' motion. Such a monitoring system must reposition according to the targets' movements and maintain satisfactory coverage of the targets. We propose an approximate, centralized approach for maximizing the visible boundary of dynamic targets using mobile cameras in a bounded 2D environment. Targets and obstacles translate, rotate, and deform independently, and cameras are only aware of the current position and shape of the targets and obstacles. Using current information, the environment is searched for better viewing positions, then cameras navigate to those positions while avoiding collisions with targets and obstacles. We present a benchmark and metrics to evaluate the performance of our method, and compare our approach to a simple gradient-based local method in several real-time simulations.**

I. INTRODUCTION

Maximizing visual coverage of multiple targets is a necessary component for accurate 3D reconstruction of the scene. For example, one can use a team of drones to track a football team to generate replay footage and individual feedback from any angle, record military training missions for later review, or document animals in the wild. The captured footage can then be replayed in 3D in virtual reality from any angle, to review plays, provide feedback for players or soldiers, or to study articulation in animals. This requires covering a sufficient portion of the boundary of the targets in order to collect enough information for reconstruction.

Having a limited number of cameras, due to safety, cost, or intrusiveness considerations, complicates the task. This requires a monitoring system to reposition according to the targets' movements in order to acquire information about the targets' appearance. In such scenarios, there is often little to no information about the targets' intentions, making it difficult or impossible to predict their motion. This makes tracking and maintaining sufficient coverage of the boundary of the targets a significant challenge.

In this work, we develop a method for covering the boundary of dynamic targets in obstructed environments in 2D. As an example, consider the target coverage problem in Fig. 1. Here, two cameras, C_1, C_2, must cover two targets, T_1, T_2, in an environment with an obstacle O_1. Each camera's angle of view is limited and the goal is to achieve high coverage of the targets, that is, maximize the visible boundary. Boundary segments that are visible are marked in green. The targets and obstacles may move or change shape over time, causing

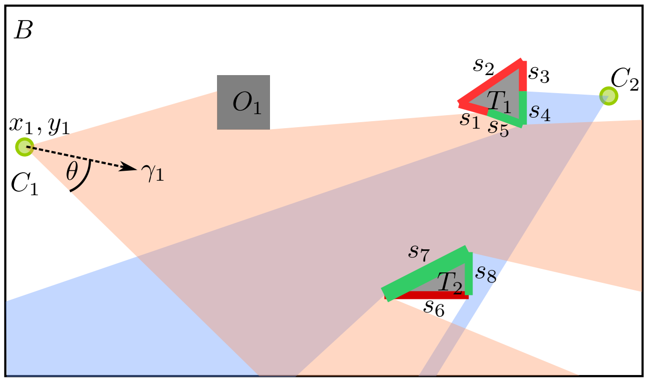


Fig. 1. Example scene bounded by B with two targets $\mathcal{T} = \{T_1, T_2\}$, one obstacle $\mathcal{O} = \{O_1\}$, and two cameras $\mathcal{C} = \{C_1, C_2\}$ at a fixed time step. The boundary of the targets $\partial\mathcal{T}$ is covered by a set of segments $\mathcal{S} = \{s_l | 1 \leq l \leq 8\}$. The segments are covered by the following cameras: $\text{visibleBy}(s_l) = \emptyset$ for $l \in \{1, 2, 3, 6\}$, $\text{visibleBy}(s_5) = \{C_1\}$, $\text{visibleBy}(s_l) = \{C_2\}$ for $l \in \{4, 8\}$, and $\text{visibleBy}(s_7) = \{C_1, C_2\}$. The length of the green segments sum up to $\text{vis}(\mathcal{S})$.

occlusions and changing the optimal observing locations. Since we do not have any a priori knowledge about the path targets and obstacles will take or how they will transform, camera positions must be chosen based on the current state of the environment. Different from previous work in multi-target tracking, where each target is modeled as a point, we purposely choose to model each target as a polygon. Polygon-based modeling is key to our approach. Unlike point-based methods, it allows covering the boundary of the targets, which is necessary for future reconstruction of the scene, and computing how much of the boundary is covered.

Although subproblems or variations thereof have been addressed in both robotics and other fields, in this work we present, to our knowledge, the first end-to-end algorithm that approximately maximizes the coverage of dynamic targets using mobile cameras in a 2D environment that includes dynamic obstacles. More specifically, our novel contributions are three-fold. First, we develop extensions to existing algorithms that make them more suitable for this problem, including computation of visible segments and motion planning for visibility; second, we present an integrated algorithm that approximately solves the dynamic coverage maximization problem; and third, we propose a benchmark and performance metrics to quantify our results and possible future research on this task.

II. PROBLEM FORMULATION

Consider a team of M cameras $\mathcal{C}(t) = \{C_j(t) | 1 \leq j \leq M\}$ which move within an environment containing obstacles $\mathcal{O}(t)$ to maximize the coverage, i.e., the visible boundary, of N independent targets $\mathcal{T}(t) = \{T_i(t) | 1 \leq i \leq N\}$. Each camera has the pose $p_j = [x_j, y_j, \gamma_j] \in SE(2)$ and fixed

Authors are with the Department of Computer Science, University of Southern California, USA {whoenig, ayanian}@usc.edu

This work was supported in part by ARL W911NF-14-D-0005 and ONR N00014-14-1-0734.

field of view θ (half-angle), i.e. $C_j(t) = (p_j(t), \theta)$, with dynamics $\dot{p}_j = u_j$. This simplified model can be realized with camera-equipped ground robots or UAVs that move in a fixed horizontal plane. The state of the environment at any time t can be written as a tuple $\mathbf{X}(t) = (\mathcal{T}(t), \mathcal{O}(t))$. Note that all angles and angular operations are within $[-\pi, \pi]$ in this paper, unless specified otherwise. This work focuses on constructing an algorithmic solution to the centralized, 2D version of this problem.

The following notation is visualized in Fig. 1. We model the targets $\mathcal{T}(t)$ and obstacles $\mathcal{O}(t)$ as time-varying polygons without holes that translate, rotate, and/or deform. We make the simplifying assumption that at any given time, the instantaneous position and shape of targets and obstacles and the pose of each camera are known (however, they are not known in advance). In practice, this might be achieved by explicit sensing (e.g., adding markers to the targets) or implicit sensing using the obtained camera data in real-time. Furthermore, we assume the environment is bounded by a time-invariant simple polygon B, which contains all targets, obstacles, and cameras at all times.

The boundary of all targets $\partial\mathcal{T}(t)$ can be tessellated into line segments, such that each continuous segment is visible by the same (possibly empty) set of cameras. We denote the set of all such line segments as $\mathcal{S}(\mathcal{T}(t), \mathcal{O}(t), \mathcal{C}(t))$ (or for short $\mathcal{S}(t)$). We can also define the set of cameras that see the entire line segment s_l:

$$\text{visibleBy}(s_l) = \{C_j \mid s_l \text{ is visible by camera } C_j\}. \quad (1)$$

Section IV-B describes an efficient algorithm to simultaneously compute $\mathcal{S}$ and $\text{visibleBy}(s_l)$.

The visible surface length can be defined as:

$$\text{vis}(\mathcal{S}(t)) = \sum_{s_l \in \mathcal{S}(t)} \begin{cases} \text{length}(s_l), & \text{visibleBy}(s_l) \neq \emptyset \\ 0, & \text{otherwise} \end{cases}. \quad (2)$$

We can now define coverage of the targets.

Definition 2.1: Coverage on the set of targets $\mathcal{T}(t)$ is the ratio between visible surface length and target surface length:

$$\text{coverage}(\mathcal{T}(t), \mathcal{O}(t), \mathcal{C}(t)) = \frac{\text{vis}(S(t))}{\text{length}(\partial\mathcal{T}(t))}. \quad (3)$$

The value of (3) falls between 0 (nothing of any target is visible) and 1 (all targets are fully visible).

The average coverage over a period of time $[t_1, t_2]$ can then be defined as

$$\frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \text{coverage}(\mathcal{T}(t), \mathcal{O}(t), \mathcal{C}(t)) dt. \quad (4)$$

It is desirable to maximize the average coverage of the targets over an infinite horizon.

Problem 2.2 (Static Coverage Maximization): For some initial state $\mathcal{C}(t_1)$, $\mathbf{X}(t_1) = (\mathcal{T}(t_1), \mathcal{O}(t_1))$, where $\mathcal{T}(t) \equiv \mathcal{T}(t_1), \mathcal{O}(t) \equiv \mathcal{O}(t_1)$ (i.e., targets and obstacles are static), find goal poses $p_j(t_2)$, $j \in \{1, \ldots, M\}$ and corresponding trajectories $p_j(t)$, $t_1 \leq t \leq t_2$ such that the average coverage (4) of the static targets is maximized as $t_2 \to \infty$, subject to the following constraints:

1) $\| [x_{j_1}\ y_{j_1}]^\mathrm{T} - [x_{j_2}\ y_{j_2}]^\mathrm{T} \|_2 \geq \delta\ j_1, j_2 \in \{1, \ldots, M\}\ j_1 \neq j_2, \delta > 0$ (no inter-camera collisions)
2) $\| [x_{j_1}\ y_{j_1}]^\mathrm{T} - q \|_2 \geq \delta\ j \in \{1, \ldots, M\}$, $q \in \mathcal{T} \cup \mathcal{O}$ (no collisions between cameras and targets or obstacles)
3) $\dot{p}_j = u_j, \| [\dot{x}_j\ \dot{y}_j]^\mathrm{T} \|_2 \leq v_j^{\max}, |\dot{\gamma}_j| \leq \dot{\gamma}_j^{\max}, \forall j \in \{1, \ldots, M\}$ (camera velocity is bounded)

For the static Problem 2.2, finding an optimal placement of the cameras and a corresponding optimal trajectory is a well-posed problem since the position of the targets and obstacles are known and static. However, we are interested in covering dynamic targets in an environment with dynamic obstacles.

Problem 2.3 (Dynamic Coverage Maximization): Consider Problem 2.2 with *dynamic*, independent targets that translate, rotate, and deform over time, such that $\mathbf{X}(t) = (\mathcal{T}(t), \mathcal{O}(t))$, $\mathcal{T}(t) \not\equiv \mathcal{T}(t_1), \mathcal{O}(t) \not\equiv \mathcal{O}(t_1)$. For the state $\mathbf{X}(t)$, $t_1 \leq t \leq t_2$, find the corresponding trajectories $p_j(t)$ such that the average coverage (4) is maximized, subject to the constraints defined in Problem 2.2.

Problem 2.3 is impossible to solve optimally since we have no information about the future positions or shapes of the targets. Thus (4) can not be optimized directly. However, (4) can be used as a metric to measure performance afterward. An additional practical consideration is the trade-off between total coverage and quality of coverage. For example, covering the target at long distances can increases coverage, but decreases the number of camera pixels used in doing so. Modeling this trade-off, however, is very application specific, thus it is addressed in our solution but is not included in the problem definition.

III. RELATED WORK

The static version of our problem is related to the Art Gallery Problem, where, given a polygonal description of an environment with holes, the goal is to find the lowest number and position of guards such that the whole polygon is observed. The Art Gallery Problem is known to be NP-hard [1], but approximate solutions have been proposed. These include an algorithm with time-complexity $O(n^5)$ with a guaranteed optimality bound, where n is the number of edges of the input polygon [2]; random sampling based methods with probabilistic optimality bounds [3]; and distributed control algorithms to steer robots to optimal positions [4]. However, those solutions assume omni-directional cameras, which are rarely used in practice, and static, not dynamic, scenes.

Cooperative Multi-Robot Observation of Multiple Moving Targets (CMOMMT) is the problem of maximizing the total number of visible targets for a given time window using omni-directional cameras with limited range. CMOMMT was first introduced by Parker and Emmons [5], and its many extensions include a distributed version [6] and a solution based on machine learning [7]. However, the problem assumes targets as points rather than complex shapes and hence does not optimize for overall coverage of the target objects and does not consider occlusions at all.

Similar problems have also been studied in the field of Wireless Sensor Networks. Ding *et al.* present a distributed framework based on game theory to track moving targets using pan-tilt-zoom cameras [8]. Extensions by Morye *et al.* try to maximize the resolution without impacting tracking performance [9]. Yao *et al.* concentrate on optimizing the handover between cameras for path-based object tracking [10]. Fiore *et al.* suggest a similar path-based approach specifically for dynamic scenes [11] . While these solutions can track dynamic targets, they specifically optimize for pan-tilt-zoom cameras only and some of them do not specifically take into account coverage, which is the primary objective in this work.

Target trajectory prediction has been studied especially for surveillance applications (e.g. [12]). Those solutions use past observations of target locations to predict future movement, and have been mainly evaluated for pedestrian movement. However, in the use-cases we consider, little is known about the targets beforehand, and their intentions are not known. Thus, we do not expect predictable behavior and therefore cannot incorporate such solutions. Furthermore, central to the purpose of future scene reconstruction is the ability for the targets and obstacles to vary in shape and orientation, which, to our knowledge, is not included in any existing prediction framework.

Specialized versions of the 3D coverage problem have been researched as well. The notion of viewpoint entropy was defined in [13], which can be used to find the N best views of a scene, e.g., in CAD applications. While that is similar to our formulation, it considers only static scenes and does not take camera resolution into account. Schwager *et al.* present a distributed solution to coverage with downward facing cameras mounted on UAVs [14]. There, the goal is to cover the inside of a given polygon in the plane from above. In the present work, however, the cameras are in the same plane as and try to maximize the visible boundary area of a transforming set of targets in an environment with transforming obstacles. This adds additional challenges such as occlusions and complex motion planning to avoid those occlusions, making these fundamentally different problems.

IV. PRELIMINARIES

Prior to approximately solving Problem 2.3, we must determine the reward function to model the tradeoff between resolution and coverage, and compute the visible boundary.

A. Reward Function

The reward function balances the trade-off between coverage, camera resolution, and collision avoidance. For the resolution, we define *camera utilization* as the number of pixels used to view a target. Assume we have a line segment s_l with first endpoint $s_l^{(1)}$ and second endpoint $s_l^{(2)}$ that is visible by C_j. The line segment and the position of the jth camera form a triangle, where the angle opposite the segment ($\angle s_l^{(1)} p_j s_l^{(2)}$) defines the portion of the field of view which is used to cover s_l. Since the line segment is within the field of view, the triangle must be a subset of the visibility polygon. Assuming that all cameras have the same number of pixels, we can compute a relative utilization measure:

$$\begin{aligned}&\operatorname{utilization}(\mathcal{T},\mathcal{O},\mathcal{C})\\&=\frac{1}{M}\sum_{C_j\in\mathcal{C}}\sum_{s_l\in\mathcal{S}}\begin{cases}\frac{\angle s_l^{(1)}p_j s_l^{(2)}}{2\theta}, & C_j\in\operatorname{visibleBy}(s_l)\\0, & \text{otherwise}\end{cases}.\end{aligned}\tag{5}$$

The result will be between 0 (no pixel of any camera contains parts of any target) to 1 (all pixels cover some target).

For collision avoidance, we use repulsion between the different objects:

$$\begin{aligned}\operatorname{repulsion}(\mathcal{T},\mathcal{O},\mathcal{C}) = &\sum_{O\in\mathcal{T}\cup\mathcal{O}}\sum_{C_j\in\mathcal{C}}\operatorname{invdist}(O,C_j)\\&+\sum_{\substack{(C_j,C_i)\in\mathcal{C}\times\mathcal{C}\\i<j}}\operatorname{invdist}(C_i,C_j)\end{aligned}\tag{6}$$

with $(a\neq b)$

$$\operatorname{invdist}(a,b)=\begin{cases}\frac{1}{(\operatorname{dist}(a,b))^2}, & \operatorname{dist}(a,b)<d_2\\0, & \text{otherwise}\end{cases}.\tag{7}$$

Here $\operatorname{dist}(a,b)$ returns the shortest Euclidean distance between different geometric objects a and b. We only consider objects within a sensing radius d_2 to limit the value of that portion in magnitude.

Finally, the reward function is the weighted sum:

$$\begin{aligned}\operatorname{reward}(\mathcal{T},\mathcal{O},\mathcal{C}) = &\lambda_1\operatorname{coverage}(\mathcal{T},\mathcal{O},\mathcal{C})\\&+\lambda_2\operatorname{utilization}(\mathcal{T},\mathcal{O},\mathcal{C})\\&-\lambda_3\operatorname{repulsion}(\mathcal{T},\mathcal{O},\mathcal{C})\end{aligned}\tag{8}$$

The parameters $\lambda_1,\lambda_2,\lambda_3\in\mathbb{R}^+$ must be tuned to reflect the trade-off between the different components. Both $\operatorname{coverage}(\cdot)$ and $\operatorname{utilization}(\cdot)$ compute values in $[0,1]$. To satisfy the collision constraints 1 and 2 in Problems 2.2 and 2.3, we analyze the boundary case of a single collision while both $\operatorname{coverage}(\cdot)$ and $\operatorname{utilization}(\cdot)$ evaluate to 1. Assuming that $d_2\geq\delta$ we get

$$\lambda_3\geq\delta^2(\lambda_1+\lambda_2)\tag{9}$$

as inequality which must hold to satisfy the constraints.

B. 2D Polygonal Visibility Computation

Each component of our solution requires computing the set of segments $\mathcal{S}$ as well as the set of cameras that are able to observe each segment ($\operatorname{visibleBy}(s_l)\quad\forall s_l\in\mathcal{S}$) for the set of targets $\mathcal{T}$, obstacles $\mathcal{O}$, and cameras $\mathcal{C}$.

A naive approach would be to create a polygon with holes, where B is the outer boundary and the targets/obstacles form the holes, then compute the omni-directional visibility polygon for each camera. Each visibility polygon can then be intersected with the respective field-of-view triangle of that camera to obtain directional visibility polygons. The visible boundary surface is the intersection between the target polygons and the union of all directional visibility polygons.The complexity of this approach is limited by the Boolean polygon operations, which can be done efficiently in $O((n+k)\log n)$, where n is is the number of line segments and k the number of intersections [15]. However, implementation requires exact arithmetic, which slows the computation in practice. Therefore, we propose an integrated method to compute the visibility polygon, visible surface, and invisible surface at the same time. The method is based on the polar line sweep paradigm, such as used by Asano [16] to compute visibility polygons.

Let us first consider the problem of computing the visibility polygon for a single camera at p_j with limited field of view θ. Imagine we shoot rays originating from p_j to determine the visible edge in a certain direction. This currently visible edge can only change at each vertex of $\mathcal{T}$, $\mathcal{O}$, and B, therefore we call the vertices *event points*. We create a list of all such event points using polar angles between 0 and 2π with the horizontal axis being zero and the origin at p_j. We sort those event points by polar angles, and traverse them in order. Furthermore, we keep a priority queue of *active edges*, which are edges intersecting with the current ray, sorted by distance to p_j. Only the closest of these edges, the *current edge*, is visible, but the other active edges might become visible at a later point during the sweep. At each event point, the queue is updated, and hence the currently visible edge might change. The resulting visibility polygon is simply the aggregation of all current edges during the angular sweep. Refer to [16] for more details on the approach.

Our first extension addresses the limited field of view of our cameras. We add two additional special event points, where the angle is defined by the boundary of the field of view of the camera. The visibility polygon is now defined by p_j itself, and the event points which were inside the field of view. Note that a full angular sweep of 2π is still required in order to keep track of the invisible segments as well.

Our second extension uses another intermediate step to compute *polar edges* before the creation of the event point list. This allows us to track where each event point was created (boundary, obstacle, or target); we can use that information to update the visible and invisible segments during the sweep.

Finally, we can repeat this algorithm for each camera sequentially. The pseudocode is shown in Algorithm 1.

Algorithm 1 2D Polygonal Visibility Computation

```
Input  $\mathcal{T}, \mathcal{O}, \mathcal{C}, B$
Output  $\mathcal{S}$, visibleBy($s_l$)  $\forall s_l \in \mathcal{S}$
 1: segments ← createSegments($\mathcal{T}, \mathcal{O}, B$)
 2: for $j \leftarrow 1$ to $M$ do
 3:     polarEdges ← createPolarEdges(segments, $p_j$)
 4:     eventPoints ← vertices(polarEdges) $\cup \{\gamma_j - \theta, \gamma_j + \theta\}$
 5:     sortAscendingByPolarAngle(eventPoints)
 6:     activeEdges ← emptyPriorityQueueByDistance()
 7:     for all eventPoint $\in$ eventPoints do
 8:         if eventPoint is firstVertex then
 9:             activeEdges.add(eventPoint.edge)
10:         else
11:             activeEdges.remove(eventPoint.edge)
12:         end if
13:         if firstElementChanged(activeEdges) then
14:             update($\mathcal{S}$)
15:             update(visibleBy($\cdot$))
16:         end if
17:     end for
18:     segments ← $\mathcal{S}$
19: end for
```

In practice, many corner cases must be handled, such as p_j being on an edge or vertex. Furthermore, all compare operations need to be done within an ϵ-boundary to avoid numerical issues caused by the floating point representation.

The time complexity for one iteration is $O(n \log n)$, where n is the number of segments [16] and therefore depends on the number of obstacles and targets and their polygonal complexity. We repeat the computation M times, however, the number of the input segments n possibly grows in each iteration (some segments might split into two segments).

V. ALGORITHM

While it is impossible to solve Problem 2.3 exactly since future positions of the targets and obstacles are unknown, we present an approximate solution that uses current information about the state of the environment. A flowchart that details the components of the solution, which we briefly describe here and expand upon in the rest of this section, is shown in Fig. 2. Initially, we execute *global optimization* to find good camera poses based on the current state. If the global optimizer suggests that a much higher reward than currently achieved is possible (with some threshold percentage τ), then *task assignment* and *motion planning* are executed. The former finds an assignment of goal positions such that the estimated time traveled is minimized; the latter computes a motion plan while avoiding obstacles for each camera. Next, the *motion plan is executed.* Since the scene is dynamic, however, we discuss how the plan can be adjusted during execution. Finally, we refine the current camera poses by executing *local optimization* but check periodically (every T_1 seconds) if the global optimization finds a better result. This control strategy is repeated indefinitely.

A. Global Optimization

The first step in our solution is to compute a close-to-optimal pose for all cameras given the position and shape of the targets and obstacles. Since the underlying problem is NP-hard, we use a heuristic solution based on sampling, as described in [3]. The basic idea is to greedily and incrementally position cameras (as if newly placing them in the space) while ensuring that each new position will cover target boundaries that are not yet covered.

Algorithm 2 Triple sample algorithm

```
Input  $\mathcal{T}, \mathcal{O}, B, K, \theta$
Output  $\mathcal{C}$
 1: invisibleSegments ← boundary($\mathcal{T}$)
 2: for $j \leftarrow 1$ to $M$ do
 3:     $q_1$ ← sampleOnSegments(invisibleSegments)
 4:     bestReward = 0
 5:     for $k \leftarrow 1$ to $K$ do
 6:         poly ← visibilityPolygon($\mathcal{T}, \mathcal{O}, B, q_1$)
 7:         $q_2$ ← sampleInPolygon(poly)
 8:         yaw ← sample($q_2$, $q_1$, $\theta$)      ▷ ensure $q_1$ is visible from $q_2$
 9:         $p_j \leftarrow (q_2, \text{yaw})$
10:         bestReward ← max(bestReward, reward($\mathcal{T}, \mathcal{O}, C_1, \ldots, C_j$))
11:     end for
12:     $p_j$ ← arg(bestReward)
13:     updateInvisibleSegments()
14: end for
```

Algorithm 2 describes the triple sample algorithm to find a close-to-global optimum camera pose for a static scene.

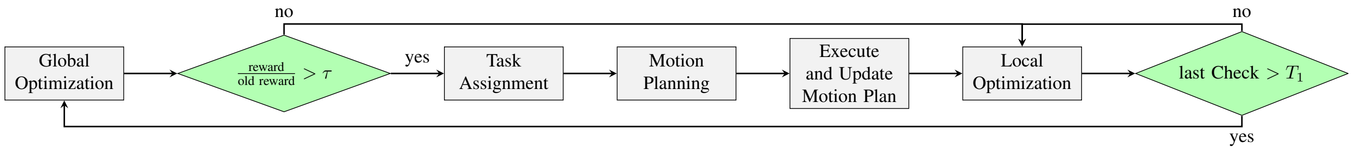


Fig. 2. Flowchart of the proposed algorithm. Local optimization is used continuously. In fixed time intervals, global optimization is performed and, if a better result was obtained, task assignment and motion planning are used to move the cameras to their new pose.

Our approach is similar to [3], however, we have added the use of a reward function to account for the trade-off between coverage, resolution, and collisions (line 10) and a third sampling step for the orientation of the camera (line 8).

Initially, the boundary of all targets is not covered by any camera and thus the list of invisible segments initializes with $\partial\mathcal{T}$. First, a point q_1 is uniformly sampled on the invisible segments. Next, the visibility polygon for an observer with omni-directional vision located at q_1 is computed. This polygon marks the area where a camera could be located at such that it observes q_1. Then, a point q_2 and an angle γ is uniformly sampled within that polygon such that a camera with pose $[q_2, \gamma]$ would observe q_1. This sampling is repeated K times, and the pose that maximizes the reward is chosen. Finally, the list of invisible segments is updated and the entire process is repeated for the next camera.

Note that sampling can be either uniform or use another distribution if different parts of the target boundary are not equally important (e.g., front vs. side coverage). Assuming the number of samples per camera (K) is fixed, we can compute an estimated solution in $O(M \cdot n \log n)$ time, since the visibility polygon must be computed ($O(n \log n)$ [16]) M times.

B. Task Assignment

The result of the global optimization step is not ordered, thus, any camera can be assigned to any goal location. In order to find a good mapping between current camera positions and the goal camera positions, we solve a standard task assignment problem by employing the Hungarian method [17]. We use the estimated time required to travel to the goal locations as the cost, since each robot has different maximum velocities. When estimating time, we ignore possible obstacles or targets on the way, since the targets and obstacles are dynamic, thus calculated geodesic distances or other shortest paths around those objects will become inaccurate.

C. Motion Planning

Once locations are assigned to each camera, a motion plan is necessary to move from the current positions such that coverage during the motion plan execution is maximized. To avoid high-dimensional state spaces, we consider the planning for each camera individually. Using standard algorithms such as RRT* or PRM* [18], we can find a feasible path while avoiding obstacles. We employ PRM* and make use of its multi-query capability. Therefore, we need to build the roadmap only once and can plan a feasible motion for all cameras based on the same roadmap. This allows the motion plan to be updated efficiently during execution so that we can update the goal position using our local optimization strategy.

Since the default uniform sampling does not take visibility into account, we adjust the sampling of states in a similar fashion as in the algorithm for global optimization. In particular, our motion planning sampler first samples a point q_1 on the boundary of any object, computes the visibility polygon of that boundary point, and finally samples a valid state for the camera within that polygon such that q_1 is visible (Lines 3, 6, 7, and 8 in Algorithm 2). Unfortunately, this strategy would never sample any states in regions where obstacles occlude all targets. Such states might be required as intermediate steps for the motion plan. Hence, we sample uniformly with a user-defined probability p and use our custom sampler with probability $1-p$.

We also use a custom cost function to optimize the path itself according to our reward function. The cost to move from p_{j_1} to p_{j_2} with $p_{j_1}, p_{j_2} \in SE(2)$ can be computed as:

$$\begin{aligned}\text{cost}(p_{j_1}, p_{j_2}) = \frac{1}{2}\Bigg(&\frac{1}{\text{reward}(\mathcal{T}, \mathcal{O}, \{(p_{j_1}, \theta)\})} \\ &+ \frac{1}{\text{reward}(\mathcal{T}, \mathcal{O}, \{(p_{j_2}, \theta)\})}\Bigg) \\ &\cdot \left(\left\lVert \begin{bmatrix} x_{j_1} \\ y_{j_1} \end{bmatrix} - \begin{bmatrix} x_{j_2} \\ y_{j_2} \end{bmatrix} \right\rVert_2 + \frac{1}{2}(\gamma_{j_1} - \gamma_{j_2})\right)\end{aligned} \tag{10}$$

Here the last product denotes the distance between states as defined by OMPL and $\text{reward}(\cdot)$ is defined in section IV-A. The idea is based on the `StateCostIntegralObjective` in OMPL [19]. This helps to prefer states with higher rewards, such as states where the camera is closer to the target objects. In cases where the camera can only move slowly relative to the target, it might be better not to employ this cost function, as it tends to pick slightly longer routes and requires more time for the computation.

We limit the amount of time for the planning stage to T_2 seconds and, for collision avoidance, states which are closer than d_1 to any target or obstacle are considered invalid.

D. Execute and Update Motion Plan

The generated motion plan is for a given time, but the target objects can move at any time. Hence, we need to react dynamically to changes so that collisions are avoided. The next intermediate goal is the farthest intermediate point on the trajectory $p_j(t)$, such that it could be reached within the simulated time-step moving at speed $v_j^{\max}$ if no obstacles were present. For the translational component, we use artificial potentials [20], with the following force:

$$F = \lambda_4 F_a - \lambda_5 F_r \tag{11}$$

where F_a is the 2-dimensional vector towards the next intermediate goal (corresponding to the attractive force) and F_r the vector towards obstacles in range weighted by their distance, similar to (6) (corresponding to the repulsive force). We move the camera following the negative gradient of the potential field with the maximum velocity possible. The rotational part is handled separately by moving towards the specified yaw of the next intermediate goal, respecting $\dot{\gamma}_j^{\max}$.

Motion is re-planned during execution based on the state of the environment when the motion plan was first created. This can be achieved online by using multi-query planners such as PRM* [18]. In addition, goal positions are updated using the local optimizer to ensure that once the goal of a motion plan is reached, it is a current local optimum rather than an optimum at the time when the initial motion plan was created.

There are two cases where the motion plan execution might not make forward progress. First, if the dynamic scene changes so dramatically that the motion planner can not add new goal or start states because there used to be an object at the requested position. Second, if the repulsive force is in the opposite direction of the attractive force. We detect both cases and abort the execution if they continuously occur for more than T_3 seconds.

E. Local Optimization

Our approach uses gradient ascent [21] on the reward function to optimize locally. In order to comply with constraints 3 in Problems 2.2 and 2.3, we saturate the resulting output to ensure the maximum translational and angular velocities are not exceeded. Since computing visibility (Sec. IV-B) is not an analytical approach, its gradient, and that of the reward function, cannot be directly computed. Thus, the gradient of the reward function is numerically approximated in a small ϵ neighborhood. Discontinuities appear in the reward function when segments become visible or invisible to the camera. Thus, it is possible that a small step along the gradient may result in a lower reward. To avoid this, only motions which yield a higher reward are executed. Different gradient ascent step sizes are used for translation (step size α) and rotation (step size β).

VI. RESULTS

We demonstrate the applicability of our approach by creating a set of 16 static and 16 dynamic scenes, to which we compare results of our solution and a local optimization. Additionally, we simulate moving targets as humans and camera-equipped quadcopters using V-REP [22].

A. Test Scenes

As described in our problem formulation, our primary objective is to maximize the visible boundary length. Hence, a good metric to compare results is given by (4). However, to quantify the variance of the results, the dynamic behavior of the scene itself must be repeatable. To our knowledge, no standard benchmark for the problem exists and therefore we made an effort to create one.

TABLE I
PARAMETERS USED FOR BENCHMARK

Reward		Global Opt.		Local Opt.		Motion Pl.		Art. Pot.	
λ_1	1.0	K	500	ϵ	10^{-6}	p	0.1	λ_4	1.0
λ_2	0.2	T_1	5 s	α	0.5	T_2	0.5 s	λ_5	1.0
λ_3	1.0	τ	1.2	β	0.005	d_1	1 m	T_3	2 s
d_2	2 m								

We publish our test set containing 16 static and 16 dynamic scenes together with a visualization tool[1], which can be used in future research to further improve the method. We describe a scene in 2D in a human-readable `json`-file. The scenes differ significantly from each other in terms of complexity and maximum speed of cameras, targets, and obstacles. The number of cameras varies between one and five, the targets between one and six, and we use up to two obstacles. Much more complex scenes are possible as well. A selection of dynamic example scenes is shown in Fig. 3.

B. Test Scene Simulations

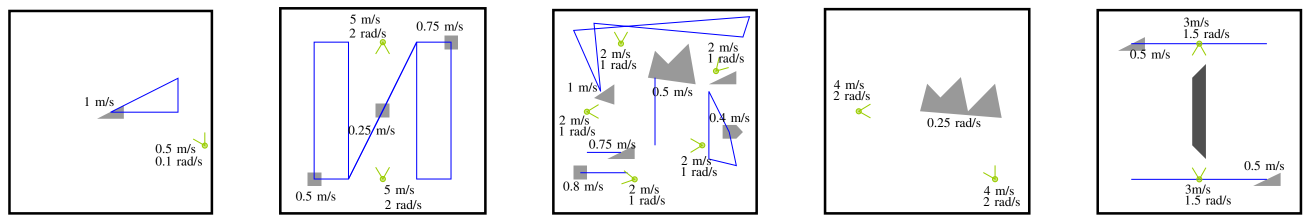
We implement our algorithm using C++ and employ `boost geometry` for low-level geometry computations such as line intersections and OMPL for motion planning. We run our proposed algorithm as well as local optimization only on all test scenes. Both use the same parameters (see Table I) and implementation and execute in real-time on the same hardware (i7-4600U 2.1 GHz, 12 GB RAM). Each run executes for 30 s and therefore simulates the same time-frame of 30 s, and we collect the average coverage (as given by (4)) as well as the average camera utilization.

While the local optimization is deterministic, our algorithm is not because the global optimization step uses random sampling. Hence, we repeat the measurements 20 times and report the variance as well as a Box-and-Whisker plot as shown in Fig. 4. Note that numerical instabilities can create variance even for the baseline, such as in `dynamic7`.

We achieve higher or comparable average coverage in most cases compared to the baseline. In particular, Problem 2.2 is nearly perfectly solved because of the nature of the algorithm: after some time, the global optimization finds a close-to-optimal solution and the camera moves and stays there. Problem 2.3 is much harder in the sense that the optimum changes over time. While we execute the motion plan (which can take some time if the optimum was found far away), the optimal position changes continuously. Figure 5 shows how the coverage and utilization change over time and sample configurations for the `dynamic9` scene using our method and local optimization. Because of the dynamic behavior of the targets, the coverage changes over time in both cases. However, in our case the motion planner causes more jumps in the coverage and we maintain a higher average coverage over time.

However, our algorithm shows lower coverage in some cases as well. The camera is slower than the target in `dynamic1`, thus our algorithm is spending too much time

[1]`https://github.com/USC-ACTLab/coverage_scenes`



(a) `dynamic1` (b) `dynamic7` (c) `dynamic9` (d) `dynamic13` (e) `dynamic16`

Fig. 3. Selection of dynamic test scenes used in the evaluation. The targets and obstacles are annotated with their velocities and the path is marked with a blue line (if they are not static). Each camera is shown at its initial position annotated with its respective $v^{\max}$, $\dot{\gamma}^{\max}$, and initial angle of view.

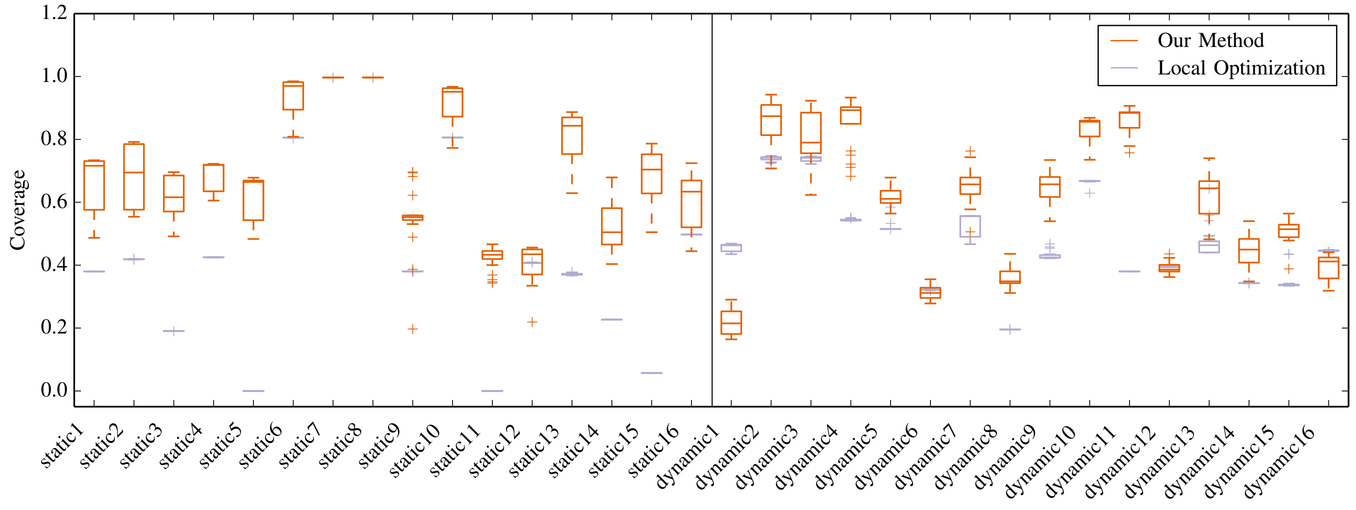
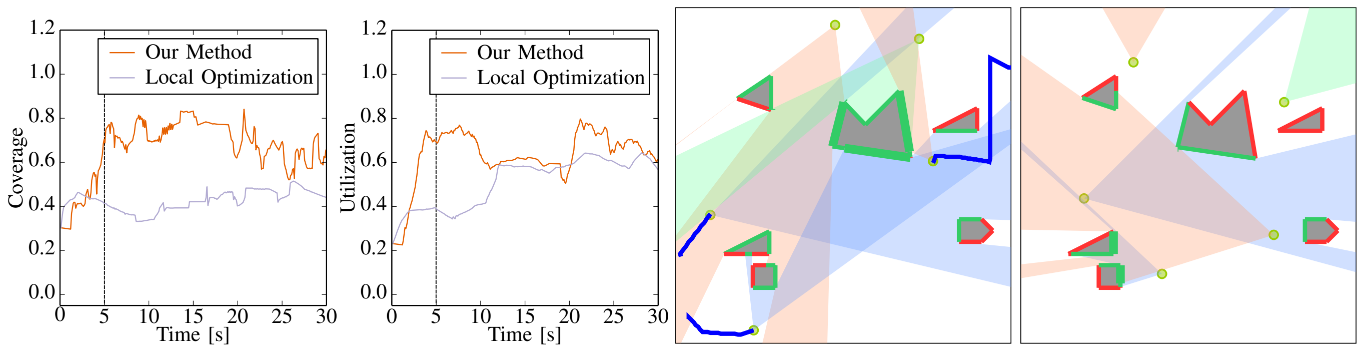


Fig. 4. Box-and-Whisker plot for the created test scenes for our proposed algorithm (orange) vs. local optimization (blue). Our algorithm achieves significantly higher coverage in most cases. The rectangles mark 25 and 75 percentile respectively, outliers are shown using the "+"-symbol. If only our method is visible (e.g. `static7`), both methods achieve the same results.



(a) Coverage over time (b) Utilization over time (c) Our method at 5 s (d) Local optimization at 5 s

Fig. 5. Results for `dynamic9` using our algorithm vs. local optimization. Blue lines mark the planned trajectory. Our method outperforms the baseline for both coverage and utilization.

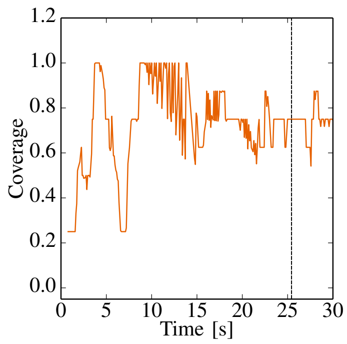
executing the motion plan. In `dynamic16`, the global optimum is only slightly better than the local optimum, and some coverage is lost while trying to move to the global optimum. Increasing τ here might help achieve the same results as a pure local optimizer, but may also reduce the performance in other test cases.

The supplemental video presents example runs of the following scenes: `static11`, `static15`, `dynamic9`, `dynamic13`, and `dynamic16`.

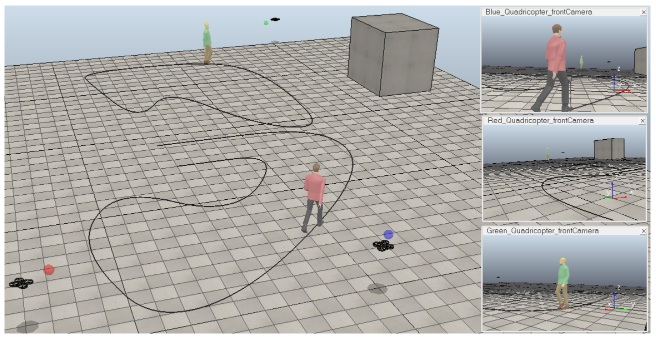
Higher coverage has the disadvantage that the camera utilization is often reduced. To achieve higher coverage, a camera often needs to move farther away from the targets. This behavior causes the camera utilization to drop, because there are typically gaps between different targets. Nevertheless, the trade-off between the two properties can be adjusted by changing the ratio between λ_1 and λ_2.

C. Simulation

We implement a test scene in the V-REP simulator using quadcopters equipped with forward-facing cameras and virtual humans which walk on a predefined path (not known to the quadcopters) (cf. Fig. 6b). Since our problem description is limited to 2D, the quadcopters fly at waist height. Because our method requires polygonal targets, humans are modeled as 2D rectangles of fixed size. Note that due to the polygonal approximation, the coverage value presented is only an



(a) Coverage over time



Blue_Quadricopter_frontCamera
Red_Quadricopter_frontCamera
Green_Quadricopter_frontCamera

(b) Screenshot of the V-REP simulation

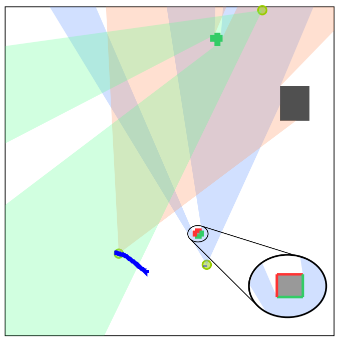

(c) Mapping to 2D scene

Fig. 6. The coverage over time for our V-REP simulation is shown in Fig. 6a. A screenshot and mapping to a 2D scene at $t = 25\,\mathrm{s}$ are shown in Fig. 6b and Fig. 6c, respectively. The path of the simulated humans in Fig. 6b is unknown to the cameras. In Fig. 6c one polygon modeling the human in the red shirt is magnified.

approximate value. Refer to the video for a sample run of the simulation. Future extensions of our approach will make it usable for 2.5D (fixed but different heights for the quadrotors), or even full 3D.

VII. CONCLUSION AND FUTURE WORK

We present an efficient algorithm to maximize the coverage of N dynamic targets in a dynamic 2D environment. Our algorithm uses random sampling and achieves real-time performance despite the difficulty of the problem. We also present a newly developed benchmark containing 32 test scenes together with performance metrics for evaluation, and evaluate our approach with these scenes. We use that benchmark to compare our solution with a local optimizer and achieve higher average coverage in most cases.

In the future, we plan to extend our solution to the 3D version of the problem. While none of the presented components is limited to 2D, in practice the underlying computation of visible and invisible surface areas is complex in the 3D case. Another direction is the development of a decentralized solution. Local optimization, motion planning, and artificial potentials can be executed in a decentralized way, but the global optimization step is challenging. Simple decentralization may cause cameras to be trapped in local optima, which we tried to avoid explicitly. Predictive- or learning-based methods might be helpful in specific target tracking cases in order to improve both coverage and resolution quality over longer time horizons. Finally, it would be interesting to have regions of varying importance assigned to the target boundaries, for example, to prioritize the front of a football player.

REFERENCES

[1] J. O'Rourke, *Art Gallery Theorems and Algorithms*. New York, NY: Oxford University Press, 1987.

[2] S. K. Ghosh, "Approximation algorithms for art gallery problems in polygons," *Discrete Applied Mathematics*, vol. 158, no. 6, pp. 718 – 722, 2010.

[3] H. González-Banos and J. Latombe, "A randomized art-gallery algorithm for sensor placement," in *Proc. Annual Symposium on Computational Geometry*, 2001, pp. 232–240.

[4] A. Ganguli, J. Cortes, and F. Bullo, "Distributed deployment of asynchronous guards in art galleries," in *American Control Conference*, 2006, pp. 1416–1421.

[5] L. Parker and B. Emmons, "Cooperative multi-robot observation of multiple moving targets," in *IEEE Intl Conf Robotics and Automation*, vol. 3, 1997, pp. 2082–2089.

[6] L. Parker, "Distributed algorithms for multi-robot observation of multiple moving targets," *Autonomous Robots*, vol. 12, no. 3, pp. 231–255, 2002.

[7] C. Touzet, "Robot awareness in cooperative mobile robot learning," *Autonomous Robots*, vol. 8, no. 1, pp. 87–97, 2000.

[8] C. Ding, B. Song, A. A. Morye, J. A. Farrell, and A. K. Roy-Chowdhury, "Collaborative sensing in a distributed PTZ camera network," *IEEE Trans. on Image Processing*, vol. 21, no. 7, pp. 3282–3295, 2012.

[9] A. A. Morye, C. Ding, A. K. Roy-Chowdhury, and J. A. Farrell, "Distributed constrained optimization for bayesian opportunistic visual sensing," *IEEE Trans. Contr. Sys. Techn.*, vol. 22, no. 6, pp. 2302–2318, 2014.

[10] Y. Yao, C. Chen, B. R. Abidi, D. L. Page, A. F. Koschan, and M. A. Abidi, "Can you see me now? sensor positioning for automated and persistent surveillance," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 40, no. 1, pp. 101–115, 2010.

[11] L. Fiore, G. Somasundaram, A. Drenner, and N. Papanikolopoulos, "Optimal camera placement with adaptation to dynamic scenes," in *IEEE Intl Conf Robotics and Automation*, 2008, pp. 956–961.

[12] V. Akbarzadeh, C. Gagné, and M. Parizeau, "Kernel density estimation for target trajectory prediction," in *IEEE Intl Conf on Intelligent Robots and Systems*, 2015, pp. 3449–3456.

[13] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich, "Viewpoint selection using viewpoint entropy," in *Proc. Vision Modeling and Visualization Conference*, 2001, pp. 273–280.

[14] M. Schwager, B. J. Julian, M. Angermann, and D. Rus, "Eyes in the sky: Decentralized control for the deployment of robotic camera networks," *Proc. IEEE*, vol. 99, no. 9, pp. 1541–1561, 2011.

[15] F. Martínez, A. J. Rueda, and F. R. Feito, "A new algorithm for computing boolean operations on polygons," *Comput. Geosci.*, vol. 35, no. 6, pp. 1177–1185, 2009.

[16] T. Asano, "An efficient algorithm for finding the visibility polygon for a polygonal region with holes," *IEICE Transactions*, vol. 68, no. 9, pp. 557–559, 1985.

[17] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83 – 97, 1955.

[18] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[19] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[20] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *IEEE Intl Conf Robotics and Automation*, vol. 2, 1985, pp. 500–505.

[21] A.-L. Cauchy, "Méthode générale pour la résolution des systèmes d'équations simultanées," *Compte Rendu des S'eances de L'Acad'emie des Sciences XXV*, vol. S'erie A, no. 25, pp. 536–538, 1847.

[22] M. F. E. Rohmer, S. P. N. Singh, "V-REP: a versatile and scalable robot simulation framework," in *IEEE Intl Conf Intelligent Robots and Systems*, 2013, pp. 1321–1326.