Available online at www.sciencedirect.com
ScienceDirect
Computer Vision and Image Understanding 103 (2006) 156–169
Computer Vision and Image Understanding
www.elsevier.com/locate/cviu

Automated camera layout to satisfy task-specific and floor plan-specific coverage requirements

Uğur Murat Erdem *, Stan Sclaroff

Computer Science Department, Boston University, Boston, MA 02215, USA

Received 17 December 2004; accepted 7 June 2006
Available online 1 August 2006
Communicated by Seth Teller

Abstract

In many multi-camera vision systems the effect of camera locations on the task-specific quality of service is ignored. Researchers in Computational Geometry have proposed elegant solutions for some sensor location problem classes. Unfortunately, these solutions use unrealistic assumptions about the cameras' capabilities that make these algorithms unsuitable for many real world computer vision applications. In this paper, the general camera placement problem is first defined with assumptions that are more consistent with the capabilities of real world cameras. The region to be observed by cameras may be volumetric, static or dynamic, and may include holes. A subclass of this general problem can be formulated in terms of planar regions that are typical of building floor plans. Given a floor plan to be observed, the problem is then to reliably compute a camera layout such that certain task-specific constraints are met. A solution to this problem is obtained via binary optimization over a discrete problem space. In experiments the performance of the resulting system is demonstrated with different real indoor and outdoor floor plans.

Keywords: Camera placement; Sensor networks; Visibility; Best view

1. Introduction

Computer vision in video sensor networks has become a popular research topic in recent years. Decreasing cost of associated hardware and increasing practical need for such systems are among the reasons attracting more and more researchers to focus in this area. One of the key questions raised by a camera network is where to position the individual cameras given certain constraints. Different visual tasks have different requirements. For an intruder detection system, complete visual coverage of the region of interest may be needed. For a multi-view reconstruction task, it may be desirable to have a minimum number of video sensors with some given angular separation. For some systems the aggregate video sensor network, depending on the specific system design and architecture, should be made fault-tolerant to camera drop out, e.g., the occasional failures of cameras, temporarily obstructed camera views, etc. As in cellular telephone networks, the aim is to have as much coverage as possible within a predefined region, with an acceptable level of quality-of-service. Similarly in video sensor networks, the layout of sensors should assure a minimum level of image quality needed to satisfy certain task-specific requirements, e.g., sufficient image resolution, depth of field, servo speed for pan-tilt-zoom cameras, etc.

It is important that the camera layout satisfy task-specific requirements since this will improve a vision algorithm's runtime performance. Yet, this remains a relatively underdeveloped area of video sensor networks. The lack of interest may be linked to researchers' tendency to focus on the design of new algorithms and a clear lack of standard test beds to compare them. For instance, a well-known face

* Corresponding author.
E-mail addresses: merdem@cs.bu.edu (U.M. Erdem), sclaroff@cs.bu.edu (S. Sclaroff).

1077-3142/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.cviu.2006.06.005

recognition algorithm might in theory (and perhaps even in practice) perform well in a single camera setup, but perform poorly in a multi-camera setting. It would be unfair to blame solely the algorithm for its poor performance. The real reasons might be the additional requirements inherent to the camera network which are omitted by the constraints of a single camera setup. The lack of a standard test bed to evaluate and to compare the performances of different camera network setups contributes to this oversight. One of the main driving forces for this work is to study and improve the effect of the off-line camera placement on the machine vision system's on-line performance. The rule of thumb is: no matter how good and efficient a vision algorithm might be, it will perform miserably if poor decisions are made in the up-front task of choosing the cameras, setting their parameters, and designing their layout in the region of interest.

2. Related work

In Computational Geometry, extensive progress has been made in solving optimal guard location problems for a polygonal area, e.g., *The Art Gallery Problem* (AGP) and its variants, where the task is to determine the minimum number of guards and their static positions such that all points in a polygon are observed [1–3]. Even though efficient algorithms exist giving a lower bound for AGPs with simple polygons [1], the exact solution is proven to be NP-Hard. A variant of the AGP is known as *Watchmen Tour Problem* where guards are allowed to move inside the polygon [4–6]. The objective is to find the optimal number and route for guards guaranteeing the detection of an intruder with an unknown initial position and unlimited speed. Suzuki et al. introduce another restricted variant of *Watchmen Tour Problem* called *boundary search* where guards are allowed to move only along the boundary of the polygon [7]. In a similar vein, *Floodlight Illumination Problems* deal with the illumination of planar regions by light sources [8,9].

Current solutions to the AGP and its variants employ unrealistic assumptions about the cameras' capabilities that make these algorithms unsuitable for most real world computer vision applications: unlimited field of view, infinite depth of field, infinite servo precision and speed. One main aim of our work is to bridge the gap between the highly theoretical, well-established computational geometry and more realistic requirements of computer vision with real video cameras.

A survey of sensor planning methods that employ more realistic assumptions is given in [10]. In work published contemporaneously with that presented here [11], a probabilistic sensor planning framework with a "visibility analysis" is proposed which evaluates the visibility of potential subjects over possible camera configurations. While their problem definition is very similar to ours in scope, their approach differs in a number of ways. They model environments with a given object density, we model environments with a given set of coverage and cost constraints. They use a local optimization method to solve a highly non-linear constrained optimization problem, as opposed to the global optimization employed here to solve a linear optimization problem over binary variables. Any solution produced by our approach is guaranteed to be optimal up to the current sample space which is not necessarily true for [11]. Our technique, moreover, allows modeling of visibility obstructions, e.g., columns, as holes in the arbitrary polygonal shapes.

There is also related work in robotic motion control for video surveillance, e.g., [12] where gradient descent is employed to compute optimal locations for mobile sensors given some utility function over a convex polygon. In contrast, our work does not currently allow cameras mounted on mobile platforms.

Task-based computer vision and camera control have a long history [13]. One interesting problem is to determine the *next best view*: finding the next optimal camera parameter setting given the acquired visual data history for the scene under exploration [14–16]. A number of active vision methods have also been proposed for surveillance applications. For instance [17,18] use a peripheral sensor to detect the position of moving objects and drive a foveal sensor to gather detailed images of the targets. Mikic et al. [19] build a camera network for an intelligent room with static and active cameras. They also employ orientation-based active camera selection criteria. In [20] human tracking across cameras is accomplished by selecting the next camera that gives the maximum tracking confidence. An interesting application for best view selection can be found in [21] where a central algorithm chooses and combines the best views from a camera network using cinematographic rules. In [22] the task becomes estimating the external parameters of individual cameras in a camera network with non-overlapping field of views. It is important to emphasize that in none of these systems consideration is given to the off-line selection and placement of the cameras to improve the on-line system performance.

3. Problem definition

In this paper, we pose the problem of optimal camera placement for a given region and vision task. We focus on the camera placement problem where the goal is to determine the optimal positioning and number of cameras for a region to be observed, given a set of task-specific constraints and a set of possible cameras to use in the layout. This camera placement takes place off-line for cameras that will be mounted on surfaces in an area of interest to support the task-specific requirements of on-line computer vision systems. In the most general (and the most challenging) case, the region to be observed by cameras might have an arbitrary volumetric shape. It may be an open space, a delimited environment, or a blend of both, e.g., outdoors vs. indoors. The region may include holes that are caused, for instance, by columns, trees, or furniture in a room that can obstruct potential camera views. It may contain an

arbitrary number of static and dynamic objects. The region itself may change in time, i.e., furniture or walls may be added, removed, or relocated in a floor plan. One can, finally, choose from an arsenal of different types of cameras that could be used in satisfying the requirements for the specified video sensing tasks.

3.1. Cameras

For the sake of completeness we first outline some optical camera parameter definitions and their main limitations. We then describe three major video camera types employed in surveillance and compare them with respect to their optical parameters. Three crucial parameters for the current work are:

- Field of View (FoV): the maximum volume visible from a camera. The FoV is determined by the apex angles (azimuth and latitude) of the visible pyramidal region emanating from the optical center of the camera. This pyramid is also known as the *viewing frustum* and can be skewed by oblique projection.
- Spatial Resolution: Spatial resolution of a camera is defined as the ratio between the total number of pixels on its imaging element excited by the projection of a real world object and the object's size. Higher spatial resolution captures more details and produces sharper images.
- Depth of Field (DoF): Depth of field is the amount of distance between the nearest and farthest objects that appear in acceptably sharp focus in an image. It is determined by the f-stop or f-number which shows the relation in between the aperture diameter and the focal length of the lens. For instance, an f-stop of f/16 shows an aperture diameter that is 1/16th of the focal length.

There are many types of video cameras available. They differ in the sensor element type, lens type, servo capabilities, etc. The following three are frequently used in computer vision research and applications[1]:

- Fixed Perspective Camera: once mounted in place, these cameras have a fixed position, orientation, and focal length.
- PTZ (Pan-Tilt-Zoom) Camera: these cameras can rotate around their horizontal (tilt) and vertical (pan) axis using remotely controlled servos. Some also have an adjustable focal length (zoom). They are mounted in a fixed position in the environment.
- Omnidirectional Camera: These cameras have 2π horizontal FoV angle, as opposed to a pyramidal one. Despite their total FoV range, they may suffer from lens abberation effects due to the small focal length and convex mirrors used in the setup [18,17].

[1] One additional camera type not included in this list is a mobile camera (mounted on a moving platform or robot). This type is not included because we focus on the off-line sensor layout problem.

Each of the mentioned cameras is defined by a set of parameters. Let the vector π_i represent the parameters that define camera i. It will contain two sub-vectors: π_i^{I}, the intrinsic parameters like focal length, and π_i^{E}, the extrinsic parameters that define the location and orientation of the camera with respect to the world coordinate system.

We will refer to the layout planning phase of the vision system as off-line and to the runtime phase as on-line. For all three camera types the location parameters are variable during the off-line phase, i.e., we can place them freely (excluding positions restricted by the environment). Table 1 shows a more structured comparison of the three camera types. We label as "OFF-LINE" any parameter that is adjustable off-line but must remain fixed during on-line phase. We label as "ON-LINE" any parameter adjustable during the on-line phase. For instance, a PTZ camera's zoom, FoV, DoF and orientation can be changed when the system is up and running but its location cannot.

3.2. Camera placement problem

Camera placement is an optimization problem by definition. Let $\mathbf{V}$ be an arbitrary connected volumetric region. If $\mathbf{V}$ is not connected then its connected parts can be treated as individual regions. Let $\mathscr{T}$ be the given task and let $\mathscr{C}$ be the set containing all the constraints required by $\mathscr{T}$. These may include spatial, i.e., coverage, constraints of $\mathbf{V}$, temporal, i.e., foveation, constraints for active cameras, quality-of-service, i.e., resolution, constraints, etc. The challenge is to find where to place a set of cameras in $\mathbf{V}$ satisfying $\mathscr{C}$ and minimizing a given cost function $G(\cdot)$. This can be stated in a more compact form as:

$$\arg\min_{\Pi} G(\Pi) \text{ subject to } \mathscr{C} \text{ given } \mathbf{V} \tag{1}$$

where $\Pi = \{\pi_1 \dots \pi_N\}$ and N is the optimal number of cameras to be placed. Note that this definition is an abstraction and different problem instances can be created by plugging-in different constraints, objectives, and tasks. Let us give four problem instances that are consistent with the definition in Eq. (1).

Problem 1: given a volumetric area $\mathbf{V}$, find a camera set Π minimizing a given cost $G(\cdot)$ such that $\forall\, p \in \mathbf{V}$ is visible from some camera $\pi_i \in \Pi$:

$$\arg\min_{\Pi} G(\Pi) \text{ s.t. } \bigcup_{i=1}^{|\Pi|} \Delta(\pi_i, \mathbf{V}) = \mathbf{V} \tag{2}$$

where $\Delta(\pi_i, \mathbf{V}) = \{p \in \mathbf{V}: \textit{point } p \textit{ is visible from camera } \pi_i\}$.

If we assume that all the cameras are omnidirectional, i.e., 2π FoV, have unlimited resolution, infinite DoF and unit cost then this problem simply reduces to the *Art Gallery Problem* [1]. This is a simplified version of the surveillance problem, where one needs to assert that it is possible to see all points of interest in $\mathbf{V}$ at all times. A slightly different task is when one needs to be able to foveate some active camera to any point in the region in less than some time threshold. For instance, consider a two-level surveillance system where an

Table 1
A comparison of the three basic video camera types

	Zoom	FoV	DoF	Orient.	Location
Fixed	OFF-LINE	OFF-LINE	OFF-LINE	OFF-LINE	OFF-LINE
PTZ	ON-LINE	ON-LINE	ON-LINE	ON-LINE	OFF-LINE
Omni	BOTH	OFF-LINE	BOTH	OFF-LINE	OFF-LINE

event, e.g., noise from an opening a door, is detected by a low resolution sensor, e.g., a microphone or a proximity sensor, which then sends a request to a high resolution sensor, e.g., a PTZ camera, for detailed investigation. The acceptable time window between the request and foveation depends directly on the task at hand.

Problem 2: given a volumetric area $\mathbf{V}$, find a camera set Π minimizing a given cost $G(\cdot)$ such that $\forall\, p \in \mathbf{V}$ is visible from some camera $\pi_i \in \Pi$ in less than time T. In a more compact form:

$$\arg\min_{\Pi}\ G(\Pi)\ \text{s.t.}\ \forall p \in \mathbf{V}\ \text{exists}\ \pi_i : \Lambda(\pi_i, p) \leqslant T \tag{3}$$

where $\Lambda(\pi_i,p)$ gives the maximum time required to foveate camera π_i on point p. Even though the visibility of a point is guaranteed with these two problem definitions, the image quality is not. Including a minimum spatial resolution constraint addresses this problem.

Problem 3: given a volumetric area $\mathbf{V}$, find a camera set Π minimizing a given cost $G(\cdot)$ such that $\forall\, p \in \mathbf{V}$ is visible from some camera $\pi_i \in \Pi$ with a given minimum required spatial resolution.

$$\arg\min_{\Pi}\ G(\Pi)\ \text{s.t.}\ \bigcup_{i=1}^{|\Pi|} \Omega(\pi_i, r, \mathbf{V}) = \mathbf{V} \tag{4}$$

where $\Omega(\pi_i,r,\mathbf{V}) = \{p \in \mathbf{V}$: *point p is visible from camera* π_i *with spatial resolution greater than* $r\}$.

Consider a computer vision system where the task is person identification by face recognition. Suppose the system is composed of a network of cameras. To increase the success rate and reliability of the recognition system, firstly the whole area must be visible by the camera network. Secondly the resolution of the face image must be sufficient for the specific algorithm employed. For instance in [23] the resolution used was 60×50 (reduced to 30×25). Problem 3 addresses exactly these requirements, minimizing at the same time some cost function $G(\cdot)$, e.g., camera network bandwidth, energy consumption, or total price.

Now consider another scenario where the task is again face recognition in some given region with a camera network. Only this time different regions have different minimum resolution requirements like in an airport where security check point areas may require higher resolution coverage compared to others (which we refer to as *hot spots*). The task is to place a collection of cameras satisfying all coverage constraints and minimizing total cost at the same time. This problem can be defined as follows:

Problem 4: given a volumetric area $\mathbf{V}$, find a camera set Π minimizing a given cost $G(\cdot)$ such that $\forall\, p \in \mathbf{V}$ can be viewed by some camera $\pi_i \in \Pi$ with some minimum spatial resolution required by p.

$$\arg\min_{\Pi}\ G(\Pi)\ \text{s.t. exists}\ \pi_i : R(p, \pi_i) \geqslant \mathrm{d}(p),\ \forall p \in \mathbf{V} \tag{5}$$

where $R(p,\pi_i)$ is the spatial resolution for point p in camera π_i and $\mathrm{d}(p)$ is the required spatial resolution density function.

These are only a few interesting examples of the general camera placement problem given in Eq. (1).

3.3. Problem simplification

Although the discovery of an algorithm that can solve the most general case of the camera layout problem for a given volume of interest is highly desirable, it may prove quite challenging. We therefore focus on a more manageable subclass of this general problem that can be formulated in terms of planar regions that are typical of a building floor plan, e.g., Fig. 1. We then approximate the region by a polygon. This is a valid assumption since most buildings and floor plans consist of polygonal shapes or can be approximated by a collection of polygons. The problem then becomes to reliably compute a camera layout given a floor plan to be observed, approximated by a polygon. A solution to this problem can be obtained via binary optimization over a discrete problem space, as will be shown in the following sections.

Efficient computational geometry algorithms exist for operations involving simple polygons,[2] like convexity determination, area finding, triangulation, etc. We allow, however, the polygon to have cavities (holes) that represent potential visibility occluding entities in the floor plan, e.g., columns, separator wall partitions, etc. The well known linear time visibility algorithms for simple polygons [25] are not applicable for this case. We therefore formulated a radial sweep visibility algorithm that can handle a polygon with holes. Detailed analysis of the algorithm is provided in the next section.

4. Visibility algorithm

Given a simple polygon $\mathbf{P_e}$ and simple polygonal holes $\mathbf{P}_k\ k = 1 \ldots K$ and a point x such that:

- $\mathbf{P}_k \subset \mathbf{P_e}\ \forall\, k$
- $\partial\mathbf{P}_i \cap \partial\mathbf{P}_j = \emptyset : i \neq j\ \forall\, i\ \forall\, j$
- $x \in \mathbf{P_e} \wedge x \notin \mathbf{P}_i\ \forall\, i$

[2] A simple polygon is defined as a region enclosed by a single closed polygonal chain that does not intersect itself [24].

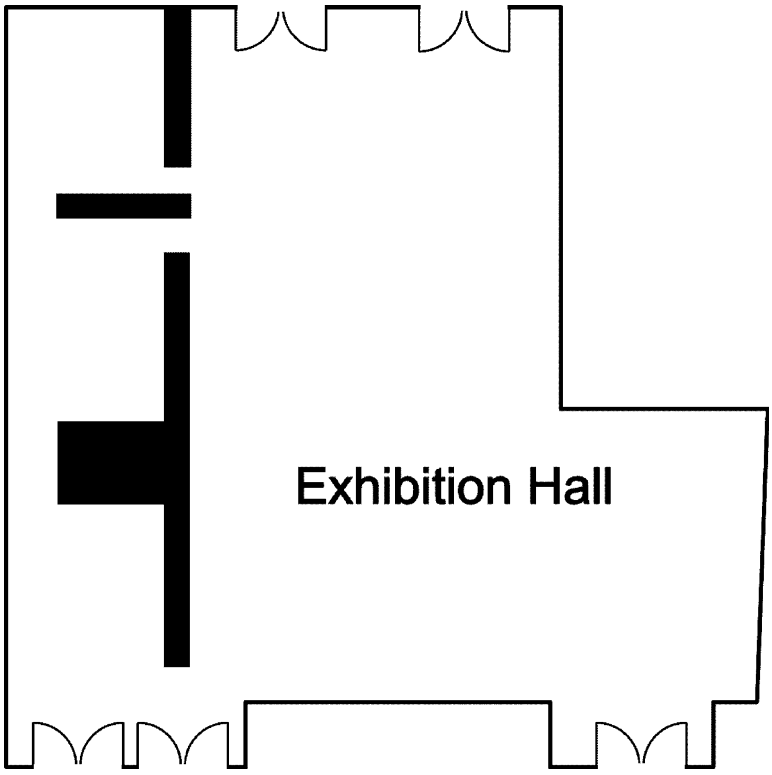

Fig. 1. A typical floor plan.

where ∂ is the boundary operator, find the visibility polygon $\mathbf{P_V}$, i.e., all points p such that,

$$\mathbf{P_V} \triangleq \max\{p : p \in \mathbf{P_e} \wedge p \notin \mathbf{P}_i \forall i \wedge \overline{xp} \subset \{\mathbf{P_e} \setminus \bigcup_{k=1}^{K} \mathbf{P}_k\}\}$$

In essence, the goal is to compute the polygonal region containing all visible points from a given point x inside a simple polygon with simple polygonal holes, e.g., Fig. 2. Let us first present the algorithm using a single simple polygon. The extension to a simple polygon with simple polygonal holes is straightforward and will be explained later.

The polygon $\mathbf{P_e}$ is represented as an edge list in Cartesian coordinates,

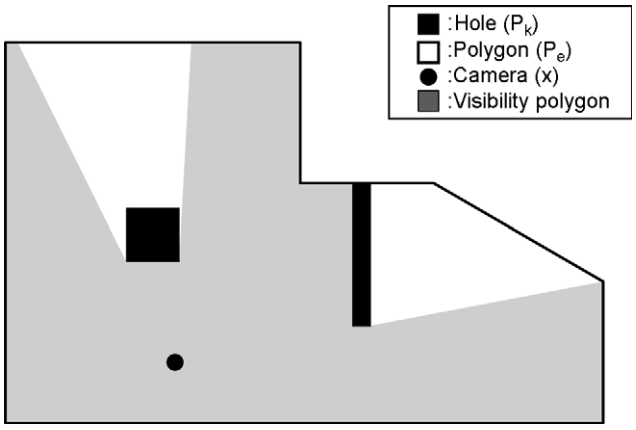

Fig. 2. A visibility polygon example.

$$\mathbf{ELC} \triangleq \{(vc_1^s, vc_1^e), (vc_2^s, vc_2^e), \ldots, (vc_i^s, vc_i^e), \ldots, (vc_E^s, vc_E^e)\}$$

where E is the number of edges, i is the edge index ordered CCW (counter clockwise), $vc_i^s, vc_i^e \in \mathbb{R}^2$ are the start and end vertices of the ith edge in Cartesian coordinate system and $vc_i^e = vc_{i+1}^s$. This representation has a well defined interior and exterior of the polygon. The region of space on the left of any edge contributes to the polygon. The idea of the algorithm is to perform a radial sweep of the polygon with x being the center of the sweep and compute the visible line segments over the range $[0,2\pi]$. The union of all the visible line segments is the visibility polygon $\mathbf{P_V}$ which is also the output of the algorithm. The initial step is to convert **ELC** to its polar coordinate representation,

$$\mathbf{ELP} \triangleq \{(vp_1^s, vp_1^e), (vp_2^s, vp_2^e), \ldots, (vp_i^s, vp_i^e), \ldots, (vp_E^s, vp_E^e)\}$$

where $vp_i \triangleq \{\theta_i, r_i\}$, polar angle and radius of ith edge's vertex respectively. This conversion is necessary for the radial sweep of the polygon. To prevent potential ambiguities for edges crossing $\theta = 0$ each such edge is split into two edges at the intersection point $\theta = 0$ as illustrated in Fig. 3.

An important observation is that only the edges satisfying $\theta_i^s < \theta_i^e$ can be fully or partially visible from x hence no other edge may contribute to $\mathbf{P_V}$.

Theorem 1. *Given an edge list in counter clockwise order of a simple polygon* $\mathbf{P}$ *and a point* $x \in \mathbf{P}$, *let* θ_i^s *and* θ_i^e *represent the polar angles of the start and end vertices of edge* e_i *with* x *being the pole. Let* $\mathbf{P_V}$ *be the visibility polygon from point* x. *If* $\theta_i^e \leqslant \theta_i^s$ *then edge* e_i *cannot be part of* $\mathbf{P_V}$.

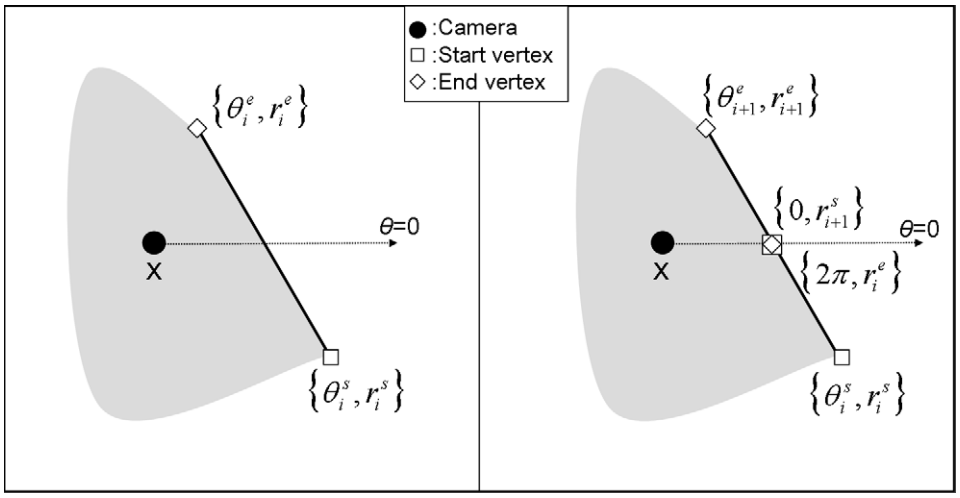

Fig. 3. Illustration of splitting the edge i crossing $\theta = 0$.

Proof. $\mathbf{P_V}$ is star convex by definition.[3] Any edge of $\mathbf{P}$ creates two half-planes. Let e_i be some edge of $\mathbf{P}$. Consider the vector u connecting the start vertex of e_i to the point x. It is a necessary condition for star convexity (by definition) to have the line segment connecting point x with any point of e_i inside the half-plane containing the inward-pointing normal to the edge e_i. Hence the cross product $u \times e_i$ is strictly positive ($>$ because we have the edges in CCW order). This is only possible when $\theta_i^e > \theta_i^s$. □

Therefore all edges where $\theta^e \leqslant \theta^s$ can be eliminated at this step. To sweep the polygon in CCW order, an ordered list of vertex polar angles is necessary. Let,

$$\mathbf{Q} \triangleq \{(\theta_1^s, r_1, \epsilon_1), \ldots, (\theta_E^s, r_E, \epsilon_E), (\theta_1^e, r_1, \epsilon_1), \ldots, (\theta_E^e, r_E, \epsilon_E)\}$$

be the list of polar angles (θ), radii (r) and respective edge pointers (ϵ) of all remaining vertices. In order to sweep the polygon in monotonically increasing radial order, let $\mathbf{Q}$ be sorted in lexicographically ascending order. For vertices with equal polar angles and radii, the end vertex is considered smaller than the start vertex.

Algorithm 1 constructs the visibility polygon by keeping track of the current visible edge during the sweep. It does so by keeping the index of the visible edge for the current sweep angle in the variable *ActiveEdge*. The *ActiveEdge* can only change at a polygon vertex point, hence each vertex is an *event point*. There are three main types of *event points* (Fig. 4):

Type 1. The current vertex is the end vertex of *ActiveEdge*, i.e., Fig. 4(a). The current vertex is part of the visible polygon. To find the next active edge it is necessary to find all the edges intersecting the half line emanating from x in the direction of the current vertex j. There may be more than one such edge. A sorted list (**SL**) is used to contain these edges sorted by their radii in increasing order. The edge with closest intersection point k to x is visible. This intersection point is part of $\mathbf{P_V}$ and its corresponding edge (head of **SL**) becomes the new *ActiveEdge*.

Type 2. The current vertex is the start vertex of some edge other than the *ActiveEdge* and it is farther away from x than *ActiveEdge*, i.e., Fig. 4(b). The edge incident to the current vertex is inserted into the sorted list **SL** since it is a candidate for future visibility.

Type 3. The current vertex is the start vertex of some edge other than the *ActiveEdge* and it is closer to x than

Algorithm 1: FindVisibilityPolygon(**ELC**,x)

input : Edge list **ELC** and a point x inside the polygon.

output: **PV**, the list of vertices of the visibility polygon.

begin

- Convert **ELC** to its polar coordinates, **ELP** ;
- Prune all backward facing edges;
- Construct the edge list **Q** ;
- Sort **Q** in lexicographically ascending order;
- **SL** ← NULL;
- **PV** ← NULL;
- CurrentVertex ← **pop** (**Q**);
- ActiveEdge ← **incidentedgeof** (CurrentVertex);
- **push** (CurrentVertex,**PV**);
- **while Q** ≠ NULL **do**
  - CurrentVertex ← **pop** (**Q**);
  - `HandleEventPoint` (CurrentVertex,ActiveEdge,**PV**,**SL**);
  - 1.1 **while** *The top three vertices in* **PV** *are collinear* **do**
    - **delete** (The 2^{nd} vertex in **PV**);
  - **endw**
- **endw**

end

[3] A subset $\mathbf{P}$ of $\mathbb{R}^n$ is star convex if there is an $x_0 \in \mathbf{P}$ such that the line segment from x_0 to any point in $\mathbf{P}$ is contained in $\mathbf{P}$ [26].

ActiveEdge, i.e., Fig. 4(c). The intersection point k of the half line emanating from x in the direction of the current vertex with *ActiveEdge* and current vertex are parts of $\mathbf{P_V}$. The edge incident to the current vertex becomes the new *ActiveEdge*.

The *while* loop at line 1.1 of Algorithm 1 is necessary to prune out collinear vertices of $\mathbf{P_V}$. These vertices are generated by the algorithm when a special case, illustrated in Fig. 4(d), is encountered. The *event point* $j+1$ of Type 3 after an *event point* j of Type 1 will generate a collinear vertex k.

The blocks of code handling *event points* are labeled accordingly in the function `HandleEventPoint`. Function **insert** at line 2.2 handles the insertion of visibility candidate edges into the list **SL** sorted by their distances from x.

```
Function HandleEventPoint(CurrentVertex,ActiveEdge,PV,SL)
    input  : The current vertex CurrentVertex, the active edge
             ActiveEdge, the visibility polygon PV and the sorted
             list SL.
    output: The function operates on the input parameters.
    begin
TYPE1   if (CurrentVertex is an end vertex) AND (incidentedgeof
        (CurrentVertex) == ActiveEdge) then
            push (CurrentVertex,PV);
            while SL ≠ NULL do
                e ← head (SL);
                if endvertexof (e) > CurrentVertex then
                    k ← intersect (e, CurrentVertex);
                    push (k,PV);
                    ActiveEdge ← e;
                    break;
                endif
                pop (SL);
            endw
        endif
        if CurrentVertex is a start vertex then
            k ← intersect (ActiveEdge, CurrentVertex);
            e ← incidentedgeof (CurrentVertex);
TYPE2       if radiusof (k) < radiusof (CurrentVertex) then
2.2             insert (e,SL);
            endif
TYPE3       if radiusof (k) ≥ radiusof (CurrentVertex) then
                push (k,PV);
                push (CurrentVertex,PV);
                ActiveEdge ← e;
            endif
        endif
    end
```

4.1. Extension to polygons with holes

Note that only the inside region of $\mathbf{P_e}$ and the outside regions of the polygonal holes may contribute to $\mathbf{P_V}$. The inside/outside of the polygons is defined by their edge orderings, i.e., CCW for $\mathbf{P_e}$ or CW (clockwise) for the polygonal holes. For this reason, the aforementioned radial sweep algorithm will work correctly in the presence of holes given that their edges are ordered CW. Hole edges are then simply appended to the edge list **ELC**.

4.2. Runtime analysis

Converting the vertices from Cartesian to polar coordinates takes $O(E)$ time. Sorting **Q** takes $O(E \log (E))$ time. Insertion of edges into **SL** takes $O(\log E)$ time. Since each edge can be added to and removed from **SL** only once, the total running time of `HandleEventPoint` is also bounded by $O(\log E)$. Since `HandleEventpoint` is called $O(E)$ times, the total running time of the Algorithm 1 is $O(E \log(E))$.

5. Camera placement algorithm

The optimal camera placement problem is usually intractable in the continuous domain. To make it easier, we will attack the problem by converting it into the discrete domain. We represent each coverage region as a grid. Choice of the cell shape, e.g., square, hexagonal etc., is implementation dependent. It is also possible to use a multi-level approach for the grid resolution. In our current implementation, we use a square cell shape at a single resolution without loss of generality. An illustration of a polygonal area represented as a single resolution square-cell tiling is given in Fig. 5(a).

The next step is to create a set of candidate cameras (Π) by sampling the camera parameters (π) which are relevant to the task at hand. The goal is to find a subset of Π which optimizes the given cost function and satisfies the constraint set $\mathscr{C}$. For the sake of illustration suppose that the task is face recognition. The task relevant constraints are (but not limited to) the camera location, FoV and the spatial resolution. Each candidate camera has an associated coverage region which we call the *feasible region*. The *feasible region* is defined as the collection of points satisfying all the constraints for a single camera candidate $\pi_i \in \Pi$. The *feasible region* can be computed by intersecting the individual constraints' coverage regions, each represented as a grid. The visibility polygon, computed by Algorithm 1, is converted to the grid representation with some given resolution, i.e., Fig. 5(b). Note that the visibility constraint is always a member of the constraint set $\mathscr{C}$. The spatial resolution constraint can be expressed as a disc centered at the candidate camera location, i.e., Fig. 5(c). Each point inside the disc is guaranteed to satisfy the required minimum spatial resolution constraint. Finally, the FoV constraint can be expressed in terms of the apex angle of the camera

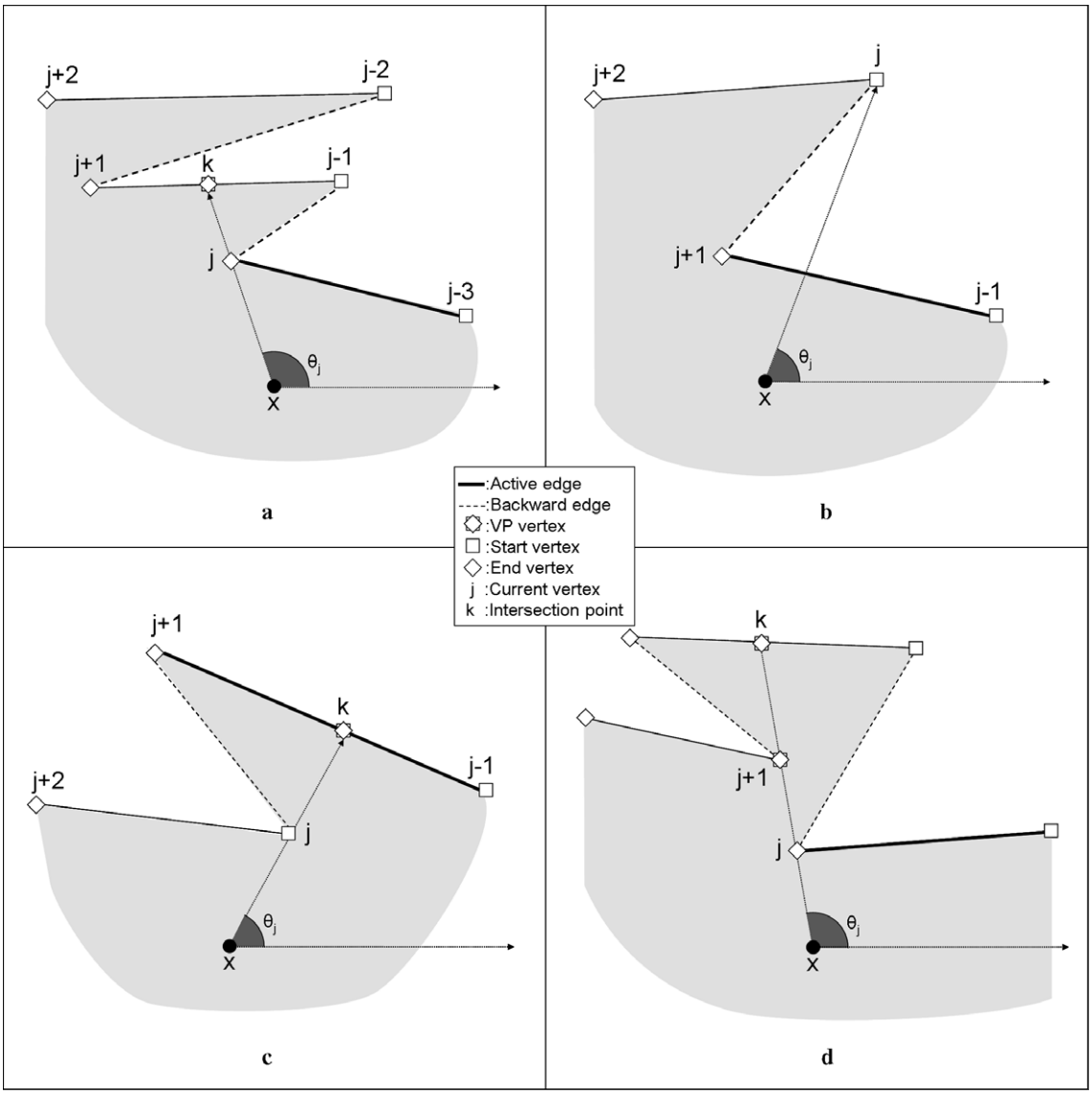

Fig. 4. Event point types and a special case (a) Type 1, (b) Type 2, (c) Type 3, (d) Special case.

frustum and the minimum and maximum distance of the region with acceptable focus from the apex, i.e., Fig. 5(d). The intersection of these three constraints gives the *feasible region*, i.e., Fig. 5(d).

Repeating the above process for all candidate cameras in Π, produces a collection of discrete *feasible regions* represented on the same grid.

To compute the optimal subset of cameras out of Π is a combinatorial problem and may be very expensive if not dealt with carefully. Fortunately, a special optimization model, called 0–1 programming, provides a convenient way to represent this problem in matrix notation in terms of a *Set Coverage Problem* [27]:

$$\min \mathbf{cx} \text{ s.t. } \mathbf{Ax} \geqslant \mathbf{b}, \mathbf{x} \in \{0,1\} \tag{6}$$

where $\mathbf{A}$ is an $M \times N$ matrix where M is the total number of cells and N is the total number of camera parameter samples. $\mathbf{A}$'s ith row elements are coefficients of the ith linear inequality constraint. The vector $\mathbf{b}$ is an $M \times 1$ vector whose ith element is the right-hand-side coefficient of constraint i, $\mathbf{c}$ is a $1 \times N$ vector whose ith element is the cost associated with ith element of $\mathbf{x}$ which is an $N \times 1$ vector containing N decision variables.

In the most general sense, the constraints given by $\mathbf{A}$ and $\mathbf{b}$ define a convex polytope *Poly* in N dimensional space that contains all the feasible solution points for the given optimization problem. The 0–1 programming tries to find a binary vector $\mathbf{x}^*$ which yields the minimum cost function value over *Poly*. Let Q be the set containing all possible binary combinations of $\mathbf{x}$. Let $Q^* \subset Q$ containing only the elements of Q found inside *Poly*. Then 0–1 programming can be explained as looking for $\mathbf{x}^* \in Q^*$ giving the minimum cost function value.

The duality between our problem and Eq. (6) is easily noted when we vectorize each candidate camera's associated discrete feasible region and let it be a column of $\mathbf{A}$, and the

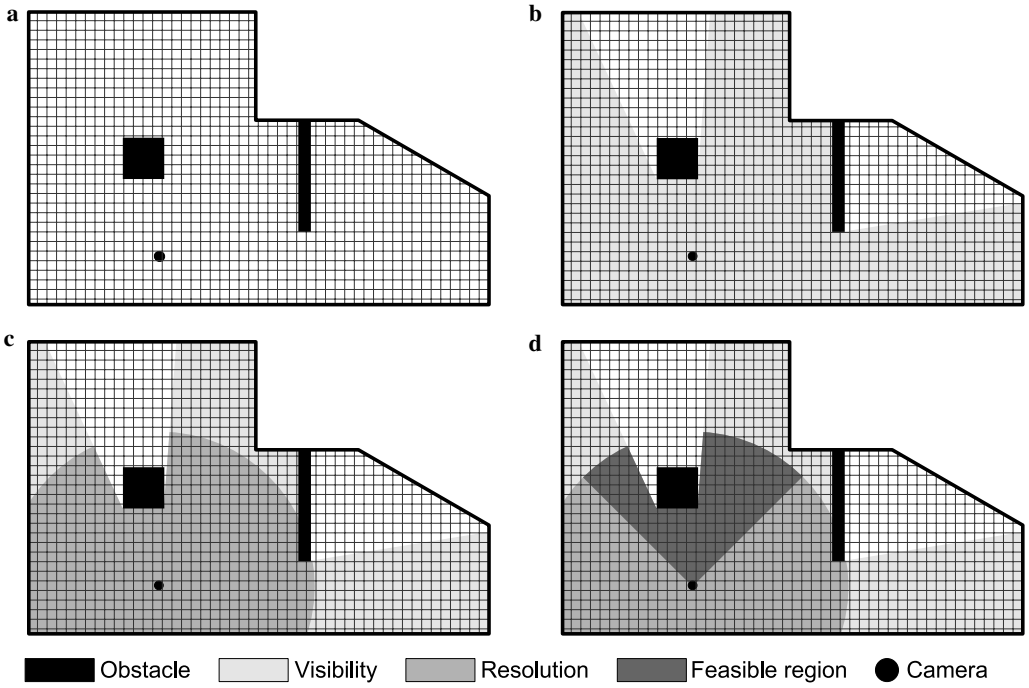

Fig. 5. Illustration of the *feasible region* generation (a) Occupancy grid, (b) Visibility constraint, (c) Resolution constraint, (d) Field of view constraint.

vectorized grid of $\mathbf{P}$ be $\mathbf{b}$. The *Poly* defined by $\mathbf{Ax} \geqslant \mathbf{b}$ will then contain all the feasible combinations of candidate cameras satisfying full coverage of $\mathbf{P}$. We may then apply 0–1 programming to find the optimal set of cameras giving the minimum cost value. In other words the solution to the 0–1 model (Eq. (6)) constructed becomes the solution to our original camera location problem.

We obtain the 0–1 model representation from a given camera location problem following these steps:

Step 1. Find a spatial coverage representation of the constraint set $\mathscr{C}$. This is the most crucial phase of the solution. If there exists a way to represent $\mathscr{C}$ as a spatial coverage problem then it is also possible to solve it using the proposed method. Solutions for Problem 1 and Problem 2 differ mainly in this representation. These will be given in detail in Section 5.

Step 2. Represent the polygonal region $\mathbf{P}$ as an *occupancy grid*, i.e., Fig. 5(a). Let $\mathbf{OG}(\mathbf{P})$ be the $h \times w$ binary matrix whose (i,j)th entry is 1 if grid cell (j,i) is inside $\mathbf{P}$ and 0 otherwise. Let us call $\mathbf{OG}(\mathbf{P})$ the *occupancy grid* of $\mathbf{P}$. Note that $h \times w$ is the resolution of the occupancy grid and it is an input parameter to the algorithm.

Step 3. Sample π N times. Let $\mathbf{s}_j$ be the jth sample. Note that N is an input parameter to the algorithm. For instance, depending on the task constraints, π can be sampled over different focal lengths, i.e., multiple camera lenses, different camera orientations, locations, apertures, etc.

Step 4. For each $\mathbf{s}_j$ find the *occupancy grid* of its spatial coverage representation given $\mathscr{C}$. Since we are dealing with cameras, visibility is the top most constraint. Therefore the first step in computing the spatial coverage is the computation of the visibility polygon. Then the spatial coverage can be found by taking the intersection of the visibility polygon and $\mathscr{C}$, e.g., resolution constraints, FoV constraints, DoF constraints etc. Let $\mathbf{S}_j$ be the $h \times w$ binary matrix whose (i,j)th element is 1 if cell grid (j,i) is inside the spatial coverage of $\mathbf{s}_j$ and 0 otherwise.

Step 5. Construct the 0–1 model (Eq. (6)). Let $\mathbf{A}$ be:

$$\begin{aligned}\mathbf{A}_{(M=h\times w,N)} &= \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_i, \ldots, \mathbf{a}_N\}\\ \mathbf{a}_i &= \{\mathbf{q}_1^i, \mathbf{q}_2^i, \ldots, \mathbf{q}_j^i, \ldots, \mathbf{q}_w^i\}^{\mathrm{T}}\\ \mathbf{q}_j^i &= \{j^{\mathrm{th}} \text{ column of } \mathbf{S}_i\}^{\mathrm{T}}\end{aligned}$$

Let $\mathbf{b}$ be:

$$\begin{aligned}\mathbf{b}_{(M=h\times w,1)} &= \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_j, \ldots, \mathbf{e}_w\}^{\mathrm{T}}\\ \mathbf{e}_j &= \{j^{\mathrm{th}} \text{ column of } \mathbf{OG}(\mathbf{P})\}^{\mathrm{T}}\end{aligned}$$

Let $\mathbf{c} = \{c_1, c_2, \ldots, c_j, \ldots, c_N\}$ where c_j is the cost associated with the camera $\mathbf{s}_j$. This may be the price, consumed bandwidth, consumed energy etc. of the camera sample. If all cameras are given equal cost, e.g., $\mathbf{c} = \mathbf{1}_{1\times N}$, then the solution is for the minimum number of cameras.

We now have all the necessary components of Eq. (6). The last step is to solve this model using one of the well-known methods such as the branch-and-bound algorithm [27]. Let us denote the optimal solution of the model with $\mathbf{x}^*$. Note that the decision variable vector $\mathbf{x}^*$ is also an indicator vector, i.e., if $x_i^* = 1$ then the camera location $\mathbf{s}_i$ is one of the optimal camera locations for the given problem instance. If $\mathbf{x}^*$ is infeasible then there is no camera location configuration satisfying $\mathscr{C}$ given the current sample set $\mathbf{s}$.

5.1. Sampling π

Since the candidate selection is performed on continuous camera parameters and since the resulting candidate set should be a fair representation of these domains, we refer to this process as sampling of π. The choice of a sample set has a direct effect on the solution, i.e., it becomes the domain of the optimization model. The negative effects of an ad hoc selection of the sample set may range from an infeasible solution to longer run-times. Some camera parameters may be restricted to specific values or range of values due to the physical constraints, i.e., some locations in the region of interest may not be accessible for camera placement, orientation may be constrained depending on the location of the camera, environmental factors like vibration or temperature may prove unsuitable for some camera types, or potential camera locations may be restricted to specific locations due to the aesthetic considerations. To get an acceptable optimal solution, such issues must be taken into account during the sampling process.

Usually the sampling is carried out first on the planar locations of the cameras in the polygonal area. Then other parameters like orientation, focal length, aperture are sampled over their allowed ranges to reflect possible selections of different camera types, lens types and settings. For example, if our camera arsenal contains only fixed and omnidirectional cameras and three types of lenses differing in focal length, then we could first sample the potential locations. If we only have a limited number of mounting bracket orientations, the orientation would be sampled over these values for the fixed cameras for each sampled location (note that sampling over orientation does not apply for the omnidirectional camera). Finally we would sample over the three different focal lengths for each potential location and orientation.

It is important to note that the optimality of the solution will depend on the density of samples. In other words, *the solution is optimal up to the current sample set*. Usually the larger the number of sample points is, the closer is the solution of the discrete optimization to the continuous (or global) optimal. Empirical results, however, suggest that relatively lower density is generally sufficient to obtain a solution which is acceptably close to the optimal solution for the continuous problem.

5.2. Correctness

The ith constraint in Eq. (6) is:

$$\mathbf{A}_{i,1} \times x_1 + \mathbf{A}_{i,2} \times x_2 + \cdots + \mathbf{A}_{i,n} \times x_n \geqslant b_i \tag{7}$$

Note that $\mathbf{A}_{i,j}$ represents the coverage status of the grid cell by jth sample: it is 1 if occupied, 0 otherwise. The right-hand-side b_i represents the coverage requirement of the *same* grid cell in the same way. So, the constraint in Eq. (7) guarantees the satisfaction of the coverage requirement of the grid cell represented by b_i. Since this is true for all M constraints, Eq. (6)'s feasible region is the subset of the possible combinations $C(N,j)$ $j = 1 \ldots N$ of $\mathbf{x}$ which satisfy $\mathscr{C}$. Then the solution to the model is a feasible combination of $\mathbf{x}$ which minimizes the objective function $\mathbf{cx}$.

5.3. Complexity

The complexity of the presented optimal placement algorithm can be factorized into the complexity of the *feasible region* generation and the complexity of the binary optimization.

The complexity to generate the coverage areas differs for different optical camera constraints. For instance, the resolution constraint can be represented as a disc with a single radius parameter, the FoV constraint can be represented as a vertical cross-section of the visual frustum with a single azimuth or elevation parameter. Both lead to a constant complexity. On the other hand the complexity of the visibility constraint is $O(E \log E)$ where E is the number of edges of the polygonal area to be covered. Finding the cells covered by each constraint area for a single camera sample is $O(Mq)$ where every single cell (M of them) has to be checked if it is inside the area (q of them) or not. Finally, the intersection of the grids is performed by applying the logical AND operator which takes $O(M \log q)$ steps. Hence the overall complexity of computing the *feasible region* for a single camera sample will be dominated by $O(Mq)$ assuming the number of cells is significantly bigger than the number of edges of the polygonal area to be covered. The running time of the *feasible region* computation, however, is almost negligible compared to the running time of the optimization process.

The binary 0–1 optimization belongs to the class of NP-Hard problems unless the constraint $\mathbf{A}$ is a totally unimodular matrix, which unfortunately is not always the case for the problem at hand. We chose the branch-and-bound algorithm because of its relative simplicity and popularity [27]. The branch-and-bound algorithm is an implicit enumeration method which in the worst case visits all possible enumerations, 2^N to be exact, of the binary $\mathbf{x}$ vector. The efficiency of the branch-and-bound algorithm also depends on the clever choice of the algorithmic parameters. In our experiments, the maximum runtime of the optimization process was around 3 min (Experiment 3) with 10,000 cells and 164 camera samples. Nevertheless, we are not concerned too much with the total runtime of the complete optimization procedure since it is off-line by problem definition.

6. Experiments

In this section we implement Problems 2, 3, and 4 defined previously and give solutions using the proposed approach. We test the system using real floor plans and use real camera specifications to sample π. More specifically we show how to convert the constraints of the problem definitions to area coverage constraints. The remaining steps are the same for all experiments.

For each experiment we use a grid resolution of 100×100 cells. The exact scale of the floor plans was unavailable. Assuming standard size door openings and given the grid resolution, the grid cell size is approximately 100 cm^2. Increasing the grid resolution will also increase the number of constraints in the Eq. (6). This is a tradeoff between the speed and accuracy of the discrete approximation. The algorithm is implemented using MATLAB 6.5 R13 on a Dual Athlon 1 GHz computer with 1 GB memory.

6.1. Experiment 1

In this experiment we solve an instance of Problem 2. Suppose the cameras are PTZ. Let us denote the maximum angular pan speed of a given PTZ camera with ω_{pan}. Recall that T is the time constraint (Eq. (3)). Assume that cameras can only be placed along the perimeter of **P**. Consider the worst case scenario. Suppose at some point in time the camera is foveated towards its minimum pan angle given its location. If the camera is located along an edge e, then its orientation corresponding to its minimum pan angle will be along e. Let the minimum pan angle be $-\theta$, i.e., Fig. 6. The camera can then foveate up to orientation $\beta = T \cdot \omega_{\text{pan}} - \theta$. Now consider the other extreme, the camera pointing to its maximum pan angle, θ. It can foveate down to orientation $-\beta$ in time T for the same reason. The intersection of the two regions, formed by orientations spanning $[-\beta,\beta]$ is the *reachable region* for the worst case scenario, as shown in Fig. 6.

Now we are able to represent the constraint defined by the time threshold T as a coverage constraint. The camera model to be placed is Sony EVI-D30 PTZ camera with $\omega_h = 80°/s$. We restrict the camera locations on the boundary of the floor plan. The minimum pan angle is $-90°$. Let $T = 1.5$ s, then we have $\beta = T \cdot 80°/s - 90 = 30°$. The *reachable region* then becomes $[-30°,30°]$. We sample five uniform camera locations per edge on the polygon **P**. The solutions for two floor plans are shown in Fig. 7.

6.2. Experiment 2

In this experiment, we solve an instance of Problem 3. Given a region, the task is to setup a camera network such that every point in the region can be seen from some camera with at least a given spatial resolution. Suppose the

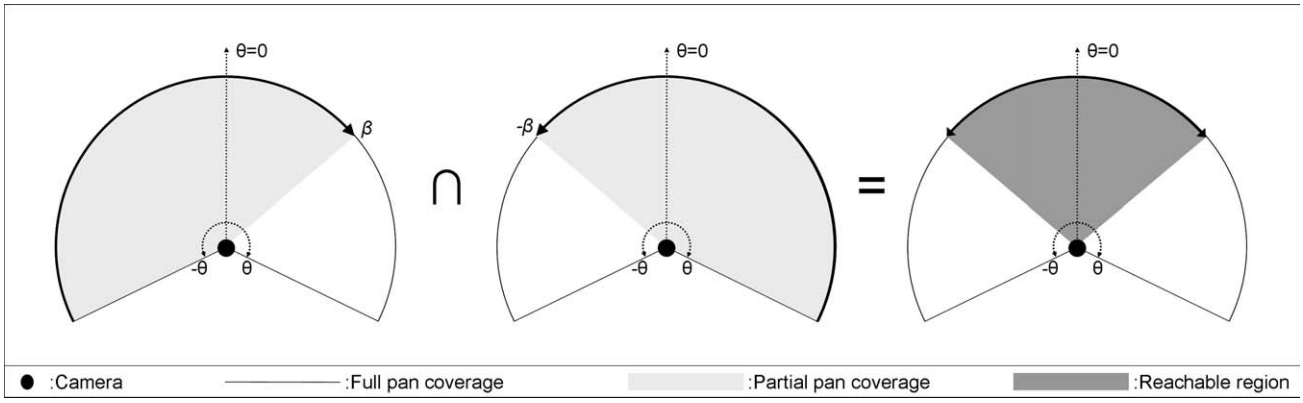

Fig. 6. Illustration of the reachable region.

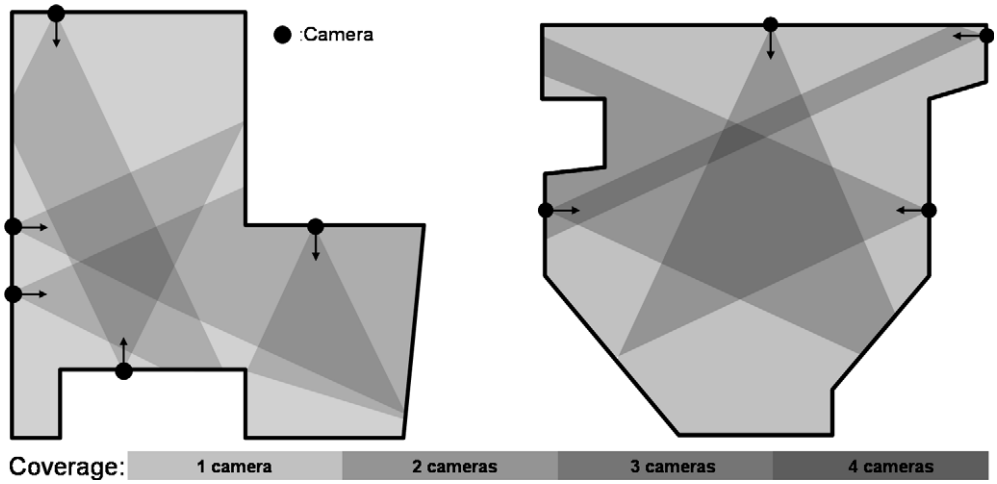

Fig. 7. Solution for Problem 2 with $T = 1.5$ s and $\omega_h = 80°/s$ using two real floor plans.

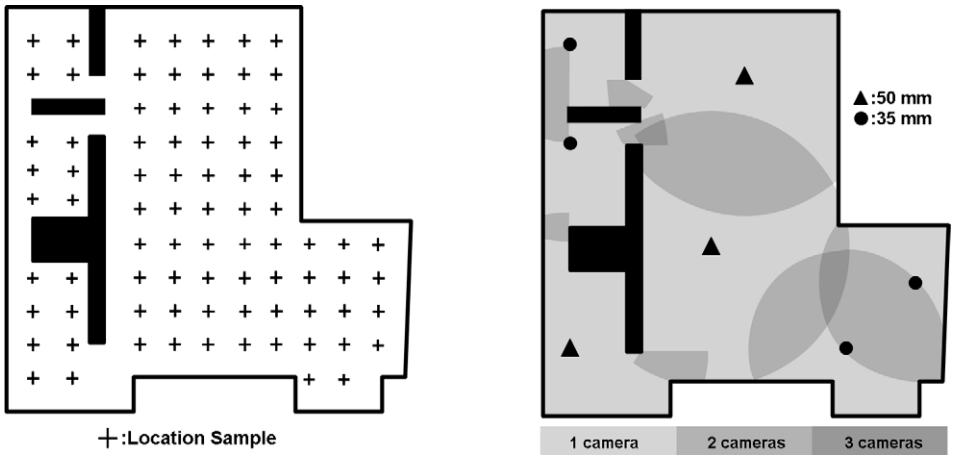

Fig. 8. Solution for problem 2. (Left) Location samples. (Right) Optimal solution.

region of interest is under surveillance and the task of the vision system is to recognize people using the face images captured by some camera of the network. It is clear that in order for the system to work reliably, sufficient discriminatory details in the face image should be visible. These details should also be visible with at least some degree of spatial resolution to ensure accurate recognition. For example, in [23] face images from the FERET database [28] with 50×60 resolution are used for face recognition. Assuming that the average face width is around 200 mm, it would be conceivable to impose a minimum spatial resolution requirement of $\frac{50}{200}$ pixels/mm for the face recognition system under consideration. For the sake of this experiment suppose we have only one type of omnidirectional camera available (i.e., FullView FC-1005.[4]) with two focal length lenses: 35 and 50 mm. Different lens types will have different prices. For the purpose of the experiment, assume the 35 mm lens costs \$100 and the 50 mm lens costs \$150. First we have to compute the feasible region of a single omnidirectional camera with a fixed focal length. Using the equation for the length of the projection of a real world object on the image plane of a camera [29] and the FC-1005 CCD specifications we get the maximum distance guaranteing $\frac{50}{200}$ pixels/mm horizontal resolution to be $\sim$1291 cm for 35 mm lens and $\sim$1844 cm for 50 mm lens.

The resulting optimal layout consists of three 50 mm cameras and four 35 mm cameras, i.e., Fig. 8. Optimal total cost is $(3 \times \$150) + (4 \times \$100) = \$850$.

6.3. Experiment 3

In this experiment we assume different areas of the floor plan have different spatial resolution requirements, e.g., Fig. 9. A real world example would be a casino where game areas and cashiers may require more detailed coverage than the other areas. For this specific example, we set the detailed area spatial resolution constraint to $\frac{50}{200}$ pixels/mm and non-detailed areas to $\frac{5}{200}$ pixels/mm. All other settings and specifications are the same as the ones used in the Experiment 2. The solution consists of one 50 mm camera and five 35 mm cameras, i.e., Fig. 10. High resolution areas are chosen to be around doors and to reflect special exhibition items if the floor plan is of a museum or game tables and cashiers if it is a casino. Optimal total cost is $(1 \times \$150) + (5 \times \$100) = \$650$.

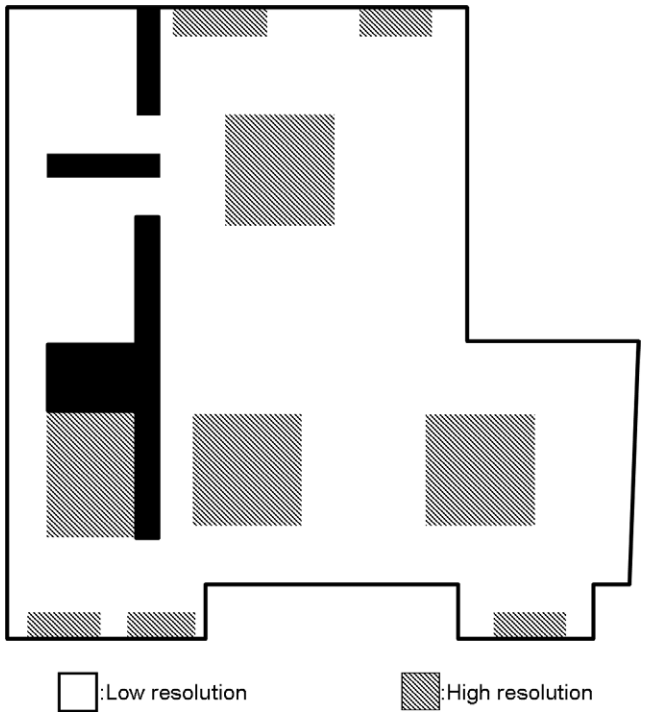

Fig. 9. Resolution constraint map. High resolution is $\frac{50}{200}$ pixels/mm and low resolution $\frac{5}{200}$ pixels/mm.

6.4. Experiment 4

The floor plan is a parking lot. The constraint is to cover the lot with at least $\frac{50}{200}$ pixels/mm resolution using the same

[4] http://www.fullview.com/.

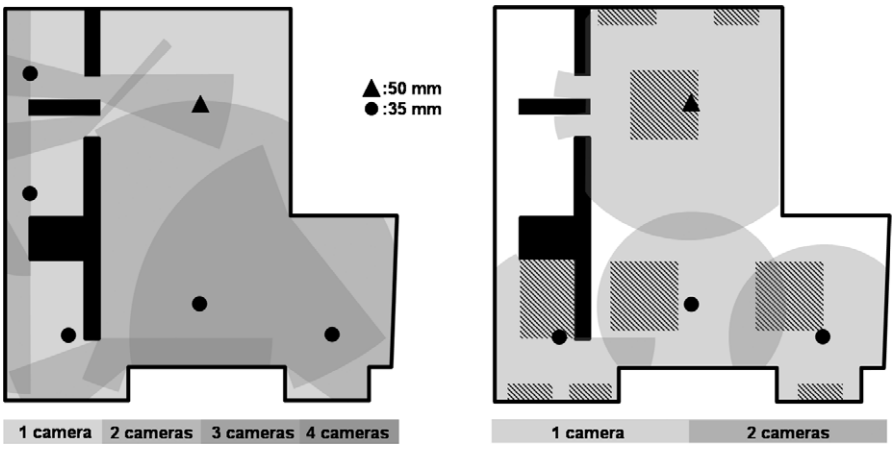

Fig. 10. Solution for problem 3. (Left) Optimal low resolution coverage. (Right) Optimal high resolution coverage.

camera and lens types as in Experiment 3. An additional constraint in this case is that the person's face must be in the field of view of at least one camera. The situation is illustrated in Fig. 11. Even though a region of the parking lot is within a camera's field of view, the person's face can only be visible from that camera within an annulus, which is determined by the height of the camera from the ground, its tilt angle, its vertical field of view coverage, and the expected height and face dimensions of a person. The loci of minimum and maximum distance points (B and T in Fig. 11) define the annulus. The intersection of the resolution constraint and this new constraint produces another annulus. Furthermore, the cameras can be mounted on the wall of the building or on a post depending on the sample location. For the sake of the experiment assume the wall mount costs \$50 and the post costs \$1500. The optimal solution consists of three 50 mm cameras mounted on posts and five 50 mm cameras mounted on walls, i.e., Fig. 12. The total cost is $3 \times (\$150 + \$50) + 5 \times (\$150 + \$1500) = \$8850$.

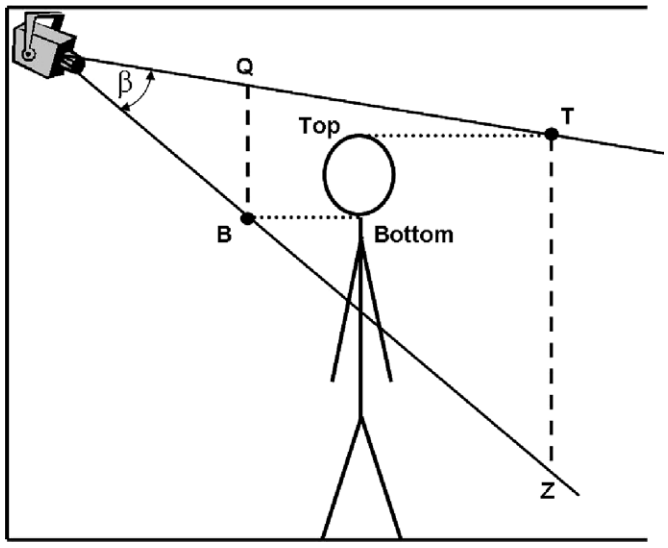

Fig. 11. Coverage limitation due to the camera tilt and a person's height.

7. Conclusion

In this paper, we formulated a solution to the general task-based camera placement problem, in order to satisfy coverage constraints while minimizing the total cost. An efficient radial sweep algorithm was proposed for computing the visible region for each camera. The algorithm works for polygonal regions of interest with polygonal holes. We represent the *feasible regions* and the areas to be covered as grids. This mapping allows us to reduce the original

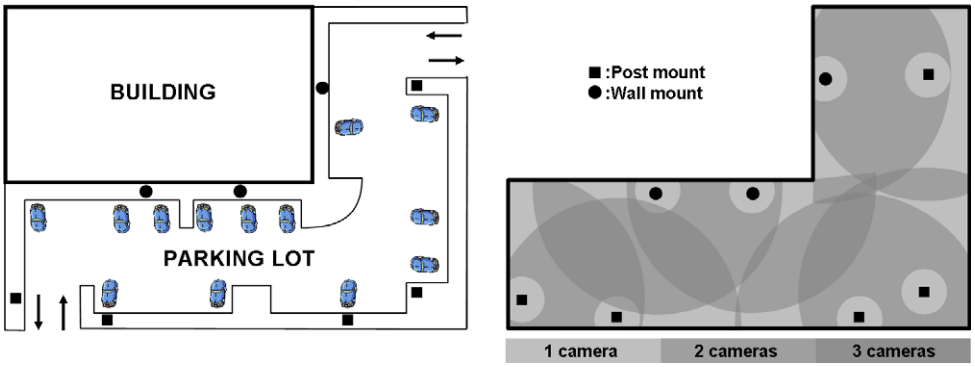

Fig. 12. Left, parking lot and optimal camera locations. Right, coverage map.

problem to the *Set Cover Problem* and solve it using 0–1 programming. We gave four specific, real world problem instances along with a solution based on the discretization. If the task-based constraints of the vision system are reducible to area coverage, then these constraints may also be satisfied by the solution of our proposed method.

The formulation given in this paper is general, and its use is not limited to the layout of video camera networks. We believe that coverage problems found in other applications, such as in layout of sprinkler systems, illumination problems, wireless networks, etc., can be attacked using the proposed method. Even though the presented experiments only optimize the total price, it is also possible to include other cost measures, like network bandwidth, energy consumption, etc. as long as the cost can be expressed in terms of a linear function. It should also be possible to incorporate a budget constraint into the model and find a layout that conforms to this budget if a feasible solution exists.

To gain a tractable solution, we converted a general continuous optimization problem to a discrete domain. Thus, the solution gained—while optimal up to the current sampling—is only approximately optimal for the continuous form. If the density of the grid approximation and the cardinality of the camera sample set are increased, then the solution of the discrete optimization tends to better approximate the continuous optimal solution. In our experiments, however, we found that relatively lower density sampling is generally sufficient to obtain a solution which is acceptably close to the true optimal layout. Nonetheless, in future work, we hope to pursue solutions to the optimization in the continuous space as opposed to the discrete one.

References

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

[2] V. Chvatal, A combinatorial theorem in plane geometry, J. Comb. Theory Series 18 (1975) 39–41.

[3] S. Fisk, A short proof of Chvatal's watchman theorem, J. Comb. Theory Series 24 (1978) 374.

[4] S. Carlsson, B. J.Nilsson, S. C.Ntafos, Optimum guard covers and m-watchmen routes for restricted polygons, in: Workshop on Algorithms and Data Structures, 1991, pp. 367–378.

[5] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. B. Mitchell, T. M. Murali, Sweeping simple polygons with a chain of guards, in: Symposium on Discrete Algorithms, 2000, pp. 927–936.

[6] L.J. Guibas, J.C. Latombe, S.M. LaValle, D. Lin, R. Motwani, A visibility-based pursuit-evasion problem, Int. J. Comput. Geom. Ap. 9 (4/5) (1999) 471–493.

[7] I. Suzuki, Y. Tazoe, M. Yamashita, T. Kameda, Searching a polygonal region from the boundary, Int. J. Comput. Geom. Ap. 11 (5) (2001) 529–553.

[8] P. Bose, L.J. Guibas, A. Lubiw, M.H. Overmars, D.L. Souvaine, J. Urrutia, The floodlight problem, Int. J. Comput. Geom. Ap. 7 (1/2) (1997) 153–163.

[9] V. Estivill-Castro, J. O'Rourke, J. Urrutia, D. Xu, Illumination of polygons with vertex lights, Inform. Process. Lett. 56 (1) (1995) 9–13.

[10] P.K.A. Tarabanis, R.Y. Tsai, A survey of sensor planning in computer vision, in: IEEE Transactions on Robotics and Automation, 1995, pp. 86–104.

[11] A. Mittal, L.S. Davis, Visibility analysis and sensor planning in dynamic environments, in: European Conference on Computer Vision (ECCV), 2004.

[12] J. Cortes, S. Martinez, T. Karatas, F. Bullo, Coverage control for mobile sensing networks, in: IEEE Conference on Robotics and Automation, 2002, pp. 1327–1332.

[13] R. Bajcsy, Active perception, in: Proceedings of the IEEE, 1988, pp. 996–1005.

[14] T. Arbel, F.P. Ferrie, Entropy-based gaze planning, Im. Vis. Comput. 19 (11) (2001) 779–786.

[15] J. Maver, R. Bajcsy, Occlusions as a guide for planning the next view, IEEE Trans. Pattern Anal. Machine Intell. 15 (5) (1993) 417–433.

[16] R. Pito, A solution to the next best view problem for automated surface acquisition, IEEE Trans. Pattern Anal. Machine Intell. 21 (10) (1999) 1016–1030.

[17] Y. Cui, S. Samasekera, Q. Huang, M. Greiffenhagen, Indoor monitoring via the collaboration between a peripheral sensor and a foveal sensor, in: IEEE Workshop on Visual Surveillance, 1998, pp. 2–9.

[18] J. Batista, P. Peixoto, H. Araujo, Real-time active surveillance by integrating peripheral motion detection with foveated tracking, in: IEEE Workshop on Visual Surveillance, 1998.

[19] I. Mikic, K. Huang, M.M. Trivedi, Activity monitoring and summarization for an intelligent meeting room, in: Workshop on Human Motion, 2000, pp. 107–112.

[20] Q. Cai, J.K. Aggarwal, Tracking human motion in structured environments using a distributed-camera system, IEEE Trans. Pattern Anal. Machine Intell. 2 (1999) 1241–1247.

[21] P. Doubek, I. Geys, T. Svoboda, L. VanGool, Cinematographic rules applied to a camera network, in: Proceedings of Omnivis, The fifth Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras, 2004, pp. 17–30.

[22] A. Rahimi, B. Dunagan, T. Darrell, Simultaneous calibration and tracking with a network of non-overlapping sensors, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004), 2004, pp. 187–194.

[23] K. Etemad, R. Chellappa, Discriminant analysis for recognition of human face images, J. Opt. Soc. Am. A 14 (1997) 1724–1734.

[24] M. Berg, M. Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry, Springer, New York, 2000.

[25] H. Gindy, D. Avis, A linear algorithm for computing the visibility polygon from a point, J. Algorithm. 2 (1981) 186–197.

[26] A. Humphreys, E.W. Weisstein, Star convex, From MathWorld—A Wolfram Web Resource, <mathworld.wolfram.com/StarConvex.html>.

[27] L.A. Wolsey, Integer Programming, Wiley-Interscience, USA, 1998.

[28] M.P. Rauss, P.J. Philips, A.T. DePersia, Feret (face recognition technology) program, in: 25th AIPR Workshop: Emerging Applications of Computer Vision, 1996, pp. 253–263.

[29] S. Abrams, P.K. Allen, K. Tarabanis, Computing camera viewpoints in an active robot cell, Int. J. Robot. Res. 18 (1999) 267–285.