Low-Cost Indoor Location Management for Robots Using IR Leds and an IR Camera

BARIS TAS, University of Texas at San Antonio
NIHAT ALTIPARMAK, University of Louisville
ALI ŞAMAN TOSUN, University of Texas at San Antonio

Many applications in wireless sensor networks can benefit from position information. However, existing accurate solutions for indoor environments are costly. Radio-Frequency (RF)-based approaches are not suitable for some indoor environments such as factory floors where heavy machinery can cause interference. We propose a low-cost and simple location management system using infrared (IR) leds and the Wii Remote Controller (WRC) which has an IR camera. The proposed solution is motivated by the need to find the location of a mobile robot used for data collection in a wireless sensor network. In the proposed schemes, the WRC is placed vertically on the mobile robot pointing upward and IR leds are placed irregularly on the ceiling. The mobile robot determines its position using the relative positions of the IR leds detected by the WRC. The WRC senses a few IR leds at a time, and they are differentiated using the irregularity among them. We analyze the problem theoretically and show that there exist limitations for covering large areas. We also discuss how to overcome these limitations. For small coverage areas, we provide *optimal* solutions using linear programming. The proposed scheme uses the resources efficiently and can cover a large area using a single WRC and multiple IR leds. We have simulation results including nonvertical placements of the WRC. The proposed scheme is easy to implement and requires minimal bandwidth for location management.

Categories and Subject Descriptors: C.2 [**Computer-Communication Networks**]: General—*Location management*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Sensor networks, localization, mobile robot, infrared sensor

ACM Reference Format:
Baris Tas, Nihat Altiparmak, and Ali Şaman Tosun. 2014. Low-cost indoor location management for robots using IR leds and an IR camera. ACM Trans. Sensor Netw. 10, 4, Article 63 (June 2014), 41 pages.
DOI: http://dx.doi.org/10.1145/2536713

1. INTRODUCTION

A wireless sensor network (WSN) consists of potentially thousands of sensor nodes and is deployed in an ad hoc manner for collecting data from a region of interest over a period of time. Even though the technology is new, WSNs received an enthusiastic reception in the science community as WSNs enable precise and fine-grain monitoring of a large region in real time. Some examples of successful large-scale deployments of WSNs to-date are in the context of ecology monitoring (monitoring of microclimate

This work is partially supported by Army Research Office (ARO) under grant W911NF-11-1-0170.
Authors' addresses: B. Tas (corresponding author), Department of Computer Science, University of Texas at San Antonio, TX; email: baristas@gmail.com; N. Altiparmak, Department of Computer Engineering and Computer Science, University of Louisville, KY; A. Ş. Tosun, Department of Computer Science, University of Texas at San Antonio, TX.

DOI: http://dx.doi.org/10.1145/2536713

forming in redwood forests [Tolle et al. 2005]), habitat monitoring (monitoring of nesting behavior of seabirds [Mainwaring et al. 2002]), and military surveillance (detection and classification of an intruder as a civilian, soldier, car, or SUV [Arora et al. 2004, 2005]). Also, an alarm system to be used for safety-critical systems such as fire and burglar alarms designed for indoor environments is studied [Strasser et al. 2007].

To improve the scalability and performance of WSNs, there has been a flurry of work on employing a mobile node for data collection. The data mules [Shah et al. 2003] work exploits random movement of a mobile node to opportunistically collect data from a sparse WSN. Here, the nodes buffer all their data locally and upload the data only when the mobile node arrives within direct communication distance. The Zebranet [Juang et al. 2002] system uses tracking collars carried by animals for wildlife tracking. Data is forwarded in a peer-to-peer manner and redundant copies are stored in other nodes. A shared wireless infostation model [Small and Haas 2003] uses radio-tagged whales as part of a biological information acquisition system. Mobility of the mobile node is not controlled in these approaches. Mobile element scheduling (MES) work [Somasundara et al. 2004] considers controlled mobility of the mobile node to reduce latency and serve the varying data rates in the WSNs effectively. The MES work shows that the problem of planning a path for the mobile node to visit the nodes before their buffers overflow is NP-complete. Heuristic-based solutions are proposed to address this problem [Somasundara et al. 2004; Gu et al. 2005; Zhao and Ammar 2003]. Sencar [Ma and Yang 2007] uses a mobile observer to collect data. Area is divided into regions and the mobile node moves in straight lines in each region. Multihop forwarding is used to relay packets from distant sensors to sencar. Data salmon [Demirbas et al. 2007] constructs a spanning tree and moves the mobile basestation on this tree to optimize the cost of retrieval. To reduce the size of the path that the mobile node travels, rendezvous points are used as regional collection points and the mobile node collects the data from the rendezvous points [Xing et al. 2007]. Mobile nodes are also used for data collection, storage, and retrieval in underwater sensor networks [Vasilescu et al. 2005]. This work assumes a single mobile node. Multiple mobile nodes are also proposed to improve the performance [Jea et al. 2005] using load balancing between the mobile nodes.

Efficient use of mobile nodes requires location information. A mobile robot should know its approximate location to follow predetermined optimal paths and make location-dependent decisions. Individual sensors should also know the location of the mobile to coordinate sleep-wakeup schedules and save resources. Location management schemes using GPS [Getting 1993] are not suitable for indoor environments. Existing approaches include RADAR [Bahl and Padmanabhan 2000] which uses RF to locate and track users inside buildings. RF-based schemes use RSSI (Received Signal-Strength Indicator), which is obtained automatically with the received messages in most sensor radios. Although RF-based schemes provide the cheapest localization technique [Stoyanova et al. 2007], they yield very noisy estimations, especially for indoor systems [Whitehouse et al. 2007]. Also, the RSSI values depend on many factors ranging from the antenna orientation to the environment specifics [Stoyanova et al. 2007]. In a factory setting, heavy machinery causes the RF signal interference, impeding the accuracy of the localization service based on RF signals. To overcome limitations of RF-only schemes, a combination of RF and ultrasound is used to provide location information in some applications [Priyantha et al. 2000; Harter et al. 1999]. In anchor-node-based localization, a subset of nodes called anchors know their location [Bulusu et al. 2000]. Using these anchor nodes, the error from RF signals is mitigated. Anchor nodes can be mobile and broadcast their location periodically so that other sensors are able to calculate their locations using the broadcasts [Ssu et al. 2005].

Robot localization has received a lot of interest recently in robotics. *Robot localization* is the process of obtaining the robot's position using sensor readings. Different types of sensors, such as sonar, inertial, RF, and laser sensors, have been used. Dead Reckoning (DR) is a technique commonly used to predict the current location of a robot using inertial sensors and the previous positions of the robot [Chung et al. 2001; Tsai 1998; Golfarelli et al. 2001]. However, DR cannot be used solely since it is based on the previous locations. If DR is used continuously for a long period of time, a huge deviation from the real position of the robot occurs. A combination of sensor types can be used for robot localization. For example, both RF and inertial sensors are used to acquire a better accuracy on the location of the mobile robot [Zmuda et al. 2008]. Radio-frequency-identification (RFID)-based solutions are proposed for indoors [Choi et al. 2011; Miah and Gueaieb 2010; Hahnel et al. 2004]. The main problem with RFID-based solutions is the accuracy. The most accurate systems that use other sensors in addition to RFID tags have 1.5 to 2.5cm error on the estimation of the location of the robot on average [Choi et al. 2011]. There are also robot localization schemes that use landmarks [Betke and Gurvits 1997]. In these systems, the robot senses the landmarks and figures out its location relatively. Simultaneous localization and mapping (SLAM) algorithms are used for building a map of the environment while estimating the robot pose [Dissanayake et al. 2001; Montemerlo et al. 2002; Eliazar and Parr 2003; Jeong and Lee 2005]. CV-SLAM [Jeong and Lee 2005] uses an upward camera and it requires high computation complexity since it involves image processing to match the landmark images correctly. There are also commercial systems for indoor localization. A new startup, ByteLight [2013], uses special LED bulbs which emit a form of barcode that can be read and interpreted by one's smartphone for indoor localization.

In this article, we propose a framework for indoor location management using an InfraRed (IR) camera, and IR leds. For this purpose, we picked Wii Remote Controller (WRC) as the IR camera, since it is a low-cost device. The goal is to find the location of a mobile element accurately. The WRC has an observation window (OW) with a resolution of 1024×768 pixels and can detect each IR led as one pixel having *Wii.x*- and *Wii.y*-coordinates. Each WRC has the capability of broadcasting the coordinates through bluetooth. The WRC is placed at the center of the robot in an upright position pointing to the ceiling. By carefully placing the IR leds on the ceiling, the location of a mobile robot is calculated using the WRC. The IR leds are placed on the ceiling so that the led pairs generated by the IR leds are distinct as much as possible in terms of their length and slope values. The robot estimates its position by exploring the distinct led pairs in its OW. A proper placement of the IR leds is challenging, so we propose several schemes that produce irregular placements with distinct pairs and compare them. Once a placement is picked, the IR leds are placed on the ceiling. Also, the placement found is installed on the ME (mobile element). The ME uses the led placement and the data from the WRC to figure out its current position. Having irregular placement of the IR leds, an ME is able to distinguish the IR-led pairs and estimate its location. A proper led placement solution should have as many distinct led pairs as possible. In a perfect solution, all the led pairs are unique. However, it is theoretically impossible to acquire a perfect solution for the led placement when the size of the coverage area in consideration increases. We provide some theoretical results showing that it is not possible to guarantee the constraints required for a perfect solution for large coverage areas. The system can be used for both instant localization and tracking. The ME is capable of finding its location by observing the IR leds. Then, the ME uses the location information for location-dependent decisions. Therefore, instant localization is a natural outcome of the system. In case a central system seeks

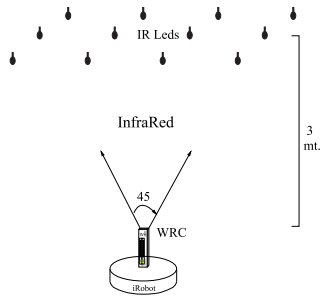


Fig. 1. Tracking system.

to track the mobile element, the mobile element broadcasts its location periodically so that the basestation knows the mobile element's location. Using simulations, we show that, in order to achieve good location estimations, a perfect led placement is not a necessity when dead reckoning is used as an auxiliary technique. In simulations, we also consider erroneous alignment of the WRC. The proposed framework can also be used for multiple robots without any extra resources. In this work, the terms mobile robot, mobile element (ME), mobile device, and robot are used interchangeably. An earlier version of this article appeared in the 28^{th} IEEE International Performance Computing and Communications Conference (IPCCC '09) [Tas et al. 2009].

The rest of the article is organized as follows. The proposed tracking system is described in Section 2. The problem of proper placement of the IR leds on the ceiling is formulated in Section 3. Proposed schemes that generate proper placements are discussed in Section 4. Since accurate location estimations depend on the irregularity of the led placements (the more irregular the led placement, the more accurate the location estimations), we compare the proposed schemes in terms of their irregularity in Section 5. There exist limitations on providing a *perfect* irregularity for the led placement. We analyze these limitations theoretically in Section 6. Section 7 discusses how to differentiate the led pairs having equal slope-length values showing the distances between them as well as the frequency of the led pairs having equal slope-length values. The simulation results on the accuracy of our framework including the misaligned WRC (placed nonvertically on the top of the ME) are shown in Section 8. Finally, we conclude with Section 9.

2. PROPOSED TRACKING SYSTEM

The WRC is placed at the center of the mobile device in an upright position pointing to the ceiling with the tip of its IR camera sensor. IR light sources are placed at a certain height pointing to the floor, as shown in Figure 1. First, we consider the robot moving only vertically and horizontally, without changing its orientation and then we consider rotations of the robot. To expand the coverage area to arbitrary-sized grids, it is enough to increase the number of IR light sources used. No additional WRCs are needed. Therefore, the cost is quite low since the cost of one WRC corresponds to the cost of nearly 200 IR light sources. If a fixed WRC was placed on the ceiling and one IR sensor was placed on the mobile robot, multiple WRCs would be needed to expand the coverage area, resulting in a costly solution.

WRC is the Nintendo Wii game-console's controller, released in November 2006. WRC has an Infrared (IR) camera that provides high-resolution and high-speed tracking of up to four IR light sources at the same time. More than four IR leds cannot be tracked by the WRC. The camera provides location data with a resolution of 1024×768 pixels

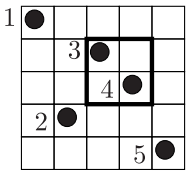
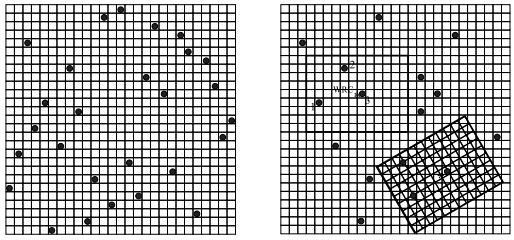


Fig. 2. Irregular placement of IR leds.

with a 100 Hz refresh rate, and a 45-degree horizontal field of view [Lee 2008]. WRC has a suggested retail price of US$40. The IR leds we used are TSAL6400 high-power infrared-emitting diodes, with a peak wavelength of 940 nanometer, radiant power of 35 milliwatt, and half-sintensity angle of $\pm 25^{\circ}$. The average expected lifetime of the leds is 10 or more years [Vishay Semiconductors 2004]. An IR led costs about US$0.20. The mobile robot senses the IR leds using the WRC attached on top of it. Then, it calculates its current position using the sensed information.

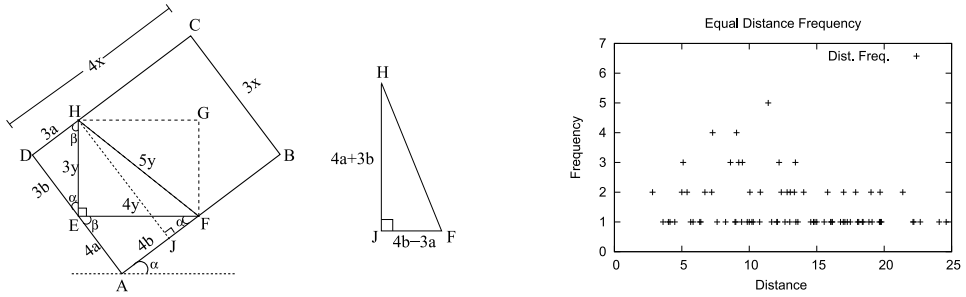
Our goal is to find the location of the mobile robot using the positions of IR leds detected by the WRC. The area covered is large and the WRC is able to see only a fraction of the area and detect only a subset of the IR leds. The sensible area is called observation window (OW). The size of the OW of the WRC depends on the height from the IR camera on WRC to the IR leds on the ceiling. The robot should be able to determine its position using the locations of the IR leds read from the WRC. One challenge is that the IR camera cannot differentiate IR light sources. There are no unique IDs corresponding to IR light sources. In this setting, the IR leds have to be differentiated in order to figure out the location of the mobile robot. One way of solving this problem is examining the slopes and distances of the IR-led pairs. If the IR leds are placed so that each IR-led pair has a unique slope and distance values, then the robot can figure out its location perfectly using this uniqueness. As a result, the key point is *irregular* placement of IR light sources. Once a proper IR led placement pattern is found, the map of the IR led locations is fed to the robot. Using its map and the locations of the IR leds read from the WRC (the relative positions of each IR led with respect to each other), the robot can identify the IR leds detected by the WRC. After identifying the IR leds, it is trivial to find the physical location of the mobile robot. An example of an irregular placement of IR leds is given in Figure 2. The vectors we consider are $\{(1,2); (1,3); \ldots; (4,5)\}$, where a vector is represented as $(startLed, endLed)$. The corresponding slope-and-length pairs of these vectors are $\{(-3, \sqrt{10}); (-\frac{1}{2}, \sqrt{5}); (-\frac{2}{3}, \sqrt{13}); (-1, 4\sqrt{2}); (2, \sqrt{5}); (\frac{1}{2}, \sqrt{5}); (-\frac{1}{3}, \sqrt{10}); (-1, \sqrt{2}); (-\frac{3}{2}, \sqrt{13}); (-2, \sqrt{5})\}$, where a pair is represented as $(slope, length)$ and none of the IR-led pairs has the same slope and length. Assuming the thick rectangle in the figure is the observation window that the WRC sees, then the location of the mobile element is figured out by examining the IR leds in this window. The slope-length pairs in WRC's window are computed. In the example, the WRC sees two IR leds and the slope-length pair is computed as $(-1, \sqrt{2})$. Since all the slope-length pairs are unique, the corresponding vector (3, 4) is found. Once the vector is found, it is trivial to find out the location of the mobile element because we know the global locations of the IR leds that constitute the found vector. As the proper placement of the IR leds plays a crucial role in finding the location of the mobile element, we discuss different schemes on how to place the IR leds on the ceiling in Section 4.

We first consider the axis-aligned movement (horizontally and vertically) of the mobile robot without changing its orientation. Then, this scheme can be extended for the rotations of the mobile robot using the following approach. We handle the rotations by



(a) a Costas array of order 27 (b) our modified system

Fig. 3. Costas-array-based tracking.



(a) finding the maximum $4k \times 3k$ axis-aligned rectangle in ABCD (b) equal distance frequencies

Fig. 4. Rotations.

reducing the problem to only the vertical-horizontal movement case. When a window of size 4m $\times$ 3m (local window) is rotated, it will have an axis-aligned $4k \times 3k$ window in it. If we can guarantee two IR sensors in this $4k \times 3k$ window, then we can guarantee two IR sensors in the rotated 4m $\times$ 3m window. We compute the maximum $4k \times 3k$ axis-aligned rectangle area inside the big rotated 4m $\times$ 3m rectangle for a given rotation angle α as shown in Figure 4(a). The ratio of area of the axis-aligned rectangle to rotated rectangle is $(\frac{3}{4sin(\alpha)+3cos(\alpha)})^2$. The value of alpha that minimizes this is $\alpha = 53^\circ$ and the area ratio is $\frac{9}{25}$, which means that the smallest axis-aligned $4k \times 3k$ rectangle is 36% of the bigger one. If our window size is 1024×768, we have to guarantee two static IR leds in every 614×460 axis-aligned area to make our system work in rotations as well.

When the robot rotates, we will not be able to use slope information, which will cause a problem in identification of IR leds if there are more than two leds in the local area and there is no unique distance between any of the pairs. Let's see whether the sole use of distance information suffices for location computation. Figure 4(b) shows the frequencies of different distance values of each led pair in the global window for Figure 3(b). Maximum frequency is 5 and more than 3 is quite rare, which means not having a unique distance pair is a low probability. Besides, in a large number of cases, more than two IR sensors will be in the window, resulting in at least three pairs. If a pair does not produce a unique distance, another pair can be used. If there are less than three leds in the local window and the distance value is not unique, the locations of pairs in the previous window can be kept and distinct distance pairs can be found. By making the calculation for the nearest previous window, the location of the robot can be found.

3. PROBLEM FORMULATION

Proper placement of the IR leds on the ceiling is challenging and it is the goal of the proposed methods. We name this problem Wii Coverage Problem (WCP) and its definition is the following.

Definition 3.1 (*Wii Coverage Problem* [$WCP(N_1, N_2, a, b, k)$]). Given positive integers $N_1, N_2, a \leq N_1, b \leq N_2$, and $k \leq a * b$, create a set of possible led locations $L = \{1, 2, \ldots, N_1 * N_2\}$ in an $N_1 \times N_2$ array. Find a minimum-cardinality subset of led locations $L' \subseteq L$ satisfying the condition that each $a \times b$ subarray of the $N_1 \times N_2$ array has at least k leds in it.

Here, $N_1 \times N_2$ array represents the ceiling and each $a \times b$ array represents an Observation Window (OW). WCP is a nontrivial optimization problem since it is asking the minimum-cardinality subset. In order to show its hardness, we recast WCP as a decision problem and apply the theory of NP-completeness as follows.

Definition 3.2 (*Wii Coverage Decision Problem* [$WCDP(N_1, N_2, a, b, k, K)$]). Given positive integers $N_1, N_2, a \leq N_1, b \leq N_2, k \leq a * b$, and $K \leq N_1 * N_2$, create a set of possible led locations $L = \{1, 2, \ldots, N_1 * N_2\}$ in an $N_1 \times N_2$ array. Is there a subset of led locations $L' \subseteq L$ with $|L'| \leq K$ satisfying the condition that each $a \times b$ subarray of the $N_1 \times N_2$ array has at least k leds in it?

Hitting set is a well-known NP-complete problem [Garey and Johnson 1990] and it is polynomial-time-reducible to the restricted case of WCDP.

Definition 3.3 (*Hitting Set*). Given a collection C of subsets of a finite set S and a positive integer $K \leq |S|$. Is there a subset $S' \subseteq S$ with $|S'| \leq K$ such that S' contains at least one element from each subset in C?

THEOREM 3.4. *WCDP is NP-complete.*

PROOF. S is the possible led locations in an $N_1 \times N_2$ area ($S = L$). Create the collection C of subsets of S using all the $a \times b$ OWs. S' contains the locations of the IR leds that are placed on the $N_1 \times N_2$ area satisfying $|S'| \leq K$ ($S' = L'$). Then, having at least one IR led in each $a \times b$ window (each subset in C) is finding the hitting set in this collection. In WCDP, every subset in the collection C is chosen based on a geometric constraint of $a \times b$ OWs. Such geometric constraints are a natural occurrence of the hitting set problem and for many geometric range spaces including disks and squares, computing the hitting set remains NP-complete [Agarwal et al. 2009; Mustafa and Ray 2009; Ganjugunte 2011; Megiddo and Supowit 1984]. The reduced problem is the restricted case of WCDP since it only guarantees at least one IR led in each $a \times b$ window, where WCDP requires at least k IR leds in each $a \times b$ window. In other words, the hitting set problem under geometric constraints is equivalent to $WCDP(N_1, N_2, a, b, k, K)$ for $k = 1$. Since an NP-complete problem is reduced to the restricted case of WCDP, WCDP is also NP-complete. □

Minimum hitting set is the optimization version of the hitting set problem and it is classified as NP-hard [Ausiello et al. 1980].

Definition 3.5 (*Minimum Hitting Set*). Given a collection C of subsets of a finite set S, find a minimum-cardinality subset $S' \subseteq S$ such that S' contains at least one element from each subset in C.

By Theorem 3.4, it is obvious to see that WCP is equivalent to the minimum hitting set problem with geometric constraints and therefore it is also NP-hard. This means that no polynomial-time algorithm exists to solve WCP. Moreover, we want the vectors

formed by IR leds to be unique, as discussed in Section 2. This requirement adds another constraint: All IR-led pairs should have different length-slope values in order to distinguish them. This additional constraint makes the problem even harder.

Since the WCDP is NP-complete, a given solution to this problem has to be verified in polynomial time. Consider the case where at least one pair of the IR leds ($k = 2$, two IR leds) has to exist in each $a \times b$ window. Counting the total number of leds is easy; however, we need an efficient mechanism to figure out the number of leds in a given $a \times b$ window. Then, verifying the correctness of a given solution is calling this mechanism for all the $a \times b$ windows in the $N_1 \times N_2$ area. The mechanism requires a *matrix structure*, and the verification of the correctness of a solution should take polynomial time.

The basis for the structure comes from set theory. In Figure 5, we have the sets A_1, A_2, A_3, A_4, each covering a specific area of the whole rectangle. Then, the number of points in the area, A_4, is found by the following equation.

$$A_4 = (A_1 + A_2 + A_3 + A_4) - [(A_1 + A_2) + (A_1 + A_3) - A_1] \tag{1}$$

The first step in finding the number of points in a specified rectangle is to create and initialize a matrix, MatrixPoints, with the same size as the size of a given solution (the same number of rows and columns). The existence of a point is represented by a 1 and nonexistence of a point is represented by a 0 in the matrix. The goal is to determine how many points there are in an $X \times Y$ window area at a given time. We need an auxiliary matrix M to achieve the goal. Each entry of this matrix is associated with the number of points in the rectangle defined between (0,0) and the coordinates of the related entry. Lines 1–6 in Algorithm 1 show how to build the auxiliary matrix. Once the auxiliary matrix M is constructed, the number of pairs in a specific window can be determined. Eq. (2) finds the number of points in a window of which the right-bottom coordinates are i, j, width is X, and height is Y using the matrix M.

$$A_4 = M(i, j) - [M(i, j - Y) + M(i - X, j)] + M(i - X, j - Y) \tag{2}$$

Figure 5 shows an example array and WRC window area A_4 inside of our array. Finally, we derive Algorithm 1 that verifies whether all $X \times Y$ windows contain at least k number of points (IR leds) or not. The complexity of this algorithm is $O(mn)$, where m and n are the dimensions of the given matrix.

ALGORITHM 1: Verify(MatrixPoints, k, X, Y, RowSize, ColumnSize)

```
1 M = MatrixPoints;
2 for i = 1 to RowSize do
3     for j = 1 to ColumnSize do
4         M[i, j]+ = M[i − 1, j] + M[i, j − 1] − M[i − 1, j − 1];
5 for i = X to RowSize do
6     for j = Y to ColumnSize do
7         if M[i, j] − M[i, j − Y] − M[i − X, j] + M[i − X, j − Y] < k then
8             return NOT VERIFIED;
9 return VERIFIED;
```

For the array in Figure 5, for example, we need to create a 9×9 matrix and fill each entry of this matrix with the number of points in the rectangle defined between (0,0) and the coordinates of the related entry. For example, the number of points in the area of A_1 is kept in the third row and fourth column of our matrix, $M(3, 4)$. By using our matrix, we can easily check the number of points in our observation window as we show in Eqs. (3) and (4).

$$A_4 = (A_1 + A_2 + A_3 + A_4) - [(A_1 + A_2) + (A_1 + A_3) - A_1] \tag{3}$$

$$A_4 = M(6,8) - [M(3,8) + M(6,4) - M(3,4)] \tag{4}$$

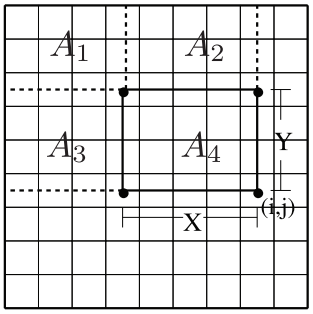


Fig. 5. Finding the number of points in the WRC window.

4. PROPOSED SCHEMES

We propose Costas-based Remove-Slide Linear Programming led placement (CRS-LP), Metric-based Random led placement (MBR) , and schemes based on linear programming in order to solve the WCP(N_1, N_2, a, b, k) problem. We start with a Costas-array-based algorithm since the properties of the Costas arrays match our requirements exactly in terms of unique slope-and-distance pairs as discussed in Section 4.1. Next, we propose an *intelligent* random led placement scheme for comparison purposes. Finally, we propose linear programming schemes for the *optimal* solution to the WCP. The led placements constructed by these schemes can be found at the journal Web page [Led 2014].

We make use of the following metric for CRSLP and MBR methods.

$$metric = 2(W_{0s}) + 1(W_{1s}) \tag{5}$$

In Eq. (5), W_{0s} represents the number of windows having *zero* leds in it within the vicinity of an led location. Similarly, W_{1s} represents the number of windows having *one* led in it within the vicinity of an led location. The *vicinity* of an led with (x,y)-coordinates consists of all the $a \times b$ windows in the set $\Lambda_{(x,y)}$ (the set of Observation Windows (OWs) in the vicinity of the location, (x,y)). Each window in the set, $\Lambda_{(x,y)}$ contains the led location (x,y). For instance, in Figure 25(a), the vicinity of the location (16,13) is depicted for a 30 $\times$ 30 grid with 108 (108 = 12 $\times$ 9) OWs, each sized 12 $\times$ 9. Since our goal is to guarantee at least two leds in each OW, our metric is dependent on the number of OWs having only *zero* and *one* led. The OWs having greater than or equal to two leds in it already satisfy our problem, *WCP*, so we do not consider them in the metric. Also, since eliminating the OWs having *zero* leds in it is more important than eliminating the OWs having *one* led in it, the coefficients for W_{0s}, and W_{1s} are 2 and 1, respectively, in our metric.

4.1. Costas-Based Remove-Slide Linear Programming Led Placement (CRS-LP)

The mobile robot figures out its position using the IR leds that are attached on the ceiling. During this operation, the number of unique length-slope pairs that are generated out of the led set is very crucial. We want to have as many unique length-slope pairs as possible, therefore we start with a Costas array benefiting from its irregularity. The Costas array is first introduced in Costas [1984]. A Costas array is a two-dimensional array consisting of blanks and dots where each row and each column exactly has one

dot in it. Moreover, the pairs generated from the dots have unique length-slope values, which is exactly what we need for the solution of our problem. The second property of Costas arrays provides the irregularity that we are looking for. Since the pairs are unique in terms of length and slope, the mobile element (ME) is able to differentiate all the pairs and it might calculate its location using this irregularity. Figure 3(a) shows an example Costas array of 27.

Costas arrays cannot be used directly for our problem. First, a Costas array might have more than two leds in some OWs. Since it is sufficient to have at least two leds in OWs, those excess leds are removed from the Costas array with the *remove-leds* phase of the CRS-LP algorithm. Second, some OWs might have less than two leds in them. For those OWs, a *slide-leds* step followed by the *LP* step of the CRS-LP algorithm will be applied.

The third step is the *slide-leds* step. After removing the extra leds, if any OWs have less than two leds in them, the IR leds left are slided in order to increase the number of windows that have at least two IR leds in it.

After the slide step, if there still exist some windows having less than two IR leds in it, the union of these windows together with the already-existing IR leds in that union are fed to a linear programming tool (IBM ILOG CPLEX Optimizer is used), which in turn returns the necessary locations for IR leds, guaranteeing that all the windows will have at least two IR leds at most. We call this algorithm Costas-Remove-Slide-with-Linear-Programming (CRS-LP) led placement.

4.1.1. Generating Costas Array. There are systematic constructions for Costas arrays. These constructions make use of primitive elements of finite fields. We use two construction methods, namely, Welch and Lempel construction methods. The Welch construction method is given in a theorem by L. R. Welch [Golomb 1984] as follows.

THEOREM 4.1 [WELCH]. *Let g be a primitive root modulo the prime p. Then the $(p-1)\times(p-1)$ permutation matrix with $a_{ij}=1$ iff $j \equiv g^i \pmod p$, $1 \le i \le p-1$, $1 \le j \le p-1$ is a Costas array.*

In Theorem 4.1, g is a primitive root (element) in the finite field $GF(p)$ for prime p($GF(p)$ is the field of integers modulo p). g in $GF(p)$ is primitive if the successive powers of g are equivalent to one ($g^1, g^2, g^3, \ldots, g^{p-1}=1$) for all the nonzero elements of $GF(p)$. For example, four different 10×10 Costas arrays can be generated for $p=11$ with primitive elements 2, 6, 7, 8 in the finite field, $GF(11)$.

The Lempel construction method is also given in a theorem by A. Lempel [Golomb 1984] as follows.

THEOREM 4.2 [LEMPEL]. *Let α be a primitive element in the field $GF(q)$, for any $q>2$. Then the $(q-2)\times(q-2)$ symmetric permutation matrix with $a_{ij}=1$ iff $\alpha^i+\alpha^j=1$ is a Costas array.*

In Theorem 4.2, q is a prime power. For example, four different 9×9 Costas arrays can be generated for $q=11$ with primitive elements 2, 6, 7, 8 in the finite field, $GF(11)$.

4.1.2. Removing Leds. After Costas array generation, there might exist some extra leds which our system does not need. Some OWs might have more than two leds in it. Therefore, we apply the *removing-leds* step of the CRS-LP algorithm.

In this step, we make use of Voronoi diagrams, heavily used in computational geometry. In addition to computer science, Voronoi diagrams are used in applications from several fields such as physics, biology, archaeology, astronomy, and so on [Okabe et al. 2000]. Voronoi diagrams are considered antiquity, since they were first used by Descartes in 1644. The generalization of Voronoi diagrams (d-dimensional case) was conducted by Georgy Voronoy.

Informally, a two-dimensional Voronoi diagram is the following. Assuming there is a set $P = \{p_1, p_2, \ldots, p_n\}$ of n points which are called sites, each point of the plane containing the point set P is attached to the site closest to it introducing a partition of the plane. Each site owns the part attached to it.

Formal definition of a Voronoi diagram is the following. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane, which are called *sites*. $\nu(p_i)$ is the *Voronoi cell* for p_i, and it contains the set of points q in the plane that are closer to p_i than any other site. So, the Voronoi cell for p_i is defined as $\nu(p_i) = \{q \mid |p_i q| < |p_j q|, \forall j \neq i\}$, where $|ab|$ represents the distance between points a and b. Finally, when the Voronoi cells are removed from the plane, the resulting lines and points constitute the Voronoi diagram of the point set P, Vor(P).

An led is considered as an *extra led* if the *metric* Eq. (5) does not change after the removal of that led. In other words, if the number of OWs having zero or one leds in the *vicinity* of the candidate led for removal does not change after we remove the candidate led, that led is considered as an *extra led*; see the beginning of the section for the definition of vicinity. Intuitively, good candidates for *extra led*s exist in the dense parts of the grid. In order to find a good candidate led for removal, we make use of the Voronoi diagram of the points resulting from the Costas array generation, Section 4.1.1. We pick the led with the smallest Voronoi cell as the candidate led for removal, since the points in denser parts of the grid form smaller Voronoi cells compared to the sparse parts of the grid. If the candidate led is considered as an *extra led* based on the metric, the extra led is removed. Then, a new Voronoi diagram of the points left is constructed in order to find the next candidate led for removal. If the candidate led is *not* considered as an *extra led* based on the metric (the metric changes after the removal of the candidate point), we continue by picking the site having the next smallest Voronoi cell as a candidate for removal. The algorithm terminates when no extra leds are found after traversing all the cells of a Voronoi diagram. Appendix A illustrates an example of the removing-leds step.

Algorithm 2 shows the details of the algorithm. The inputs are M, X, Y, *RowSize*, and *ColumnSize*. M is the two-dimensional bit-array where the locations of the leds resulting from the Costas generation step are set to 1, and other locations are set to 0. X and Y represent the OW's size. *RowSize* and *ColumnSize* represent the size of the whole grid. Actually, since the Costas array is an $|M| \times |M|$ array where $|M|$ is the number of points in M, *RowSize* and *ColumnSize* are equal to $|M|$, $|M| = RowSize = ColumnSize$.

ALGORITHM 2: RemoveLeds(M, X, Y, RowSize, ColumnSize)

```
   // M holds the leds' locations. 2d bit array.
 1 repeat
 2     VorM = ConstructVoronoi(M);
 3     minPQ = ConstructPriorityQueue(VorM);
 4     keepRemoving = false;
 5     while minPQ.size() > 0 do
 6         smallestface = minPQ.extractMin();
 7         CandidateLed = smallestface.site();
 8         if !DoesMetricChange(CandidateLed, M, X, Y, RowSize, ColumnSize) then
 9             M.remove(CandidateLed);
10             keepRemoving = true;
11             break;
12 until keepRemoving == false;
13 return M;
```

In each iteration of the outer loop of Algorithm 2, if there exists an *extra led*, it is found and removed. When there are no *extra led*s left to be removed, the algorithm terminates. In order to find an *extra led*, we first construct the Voronoi diagram, *VorM*, of the points in M. Using *VorM*, the min-priority queue, *minPQ*, with the area of each cell of the Voronoi diagram as key, is created in order to find a good *candidate led*. The first candidate led is the point having the Voronoi cell with the smallest area, lines 6 and 7. The test for the removal of the candidate led is done in function *DoesMetricChange*. If the metric Eq. (5) does not change after the removal of the candidate led, then the candidate led is an extra led and the function returns false. The *extra led* is removed from M in line 9. The variable *keepRemoving* holds the decision about termination. It is set to true in line 10, since the algorithm will continue with the next iteration of the outer loop that finds the next led to remove. If the candidate led is not an *extra led*, then the function *DoesMetricChange* returns true and we continue with the next iteration of the inner loop. The second candidate led is the point having the Voronoi cell with the next smallest area, lines 6 and 7. The search continues until the inner loop finds an *extra led*. When the inner loop cannot find an *extra led*, the value of the variable *keepRemoving* remains false and the algorithm terminates, guaranteeing that all the leds left in M are required for our system.

The running-time complexity of the *remove-leds* step of CRS-LP is the following. The outer loop iterates the number of removed leds, $|M_{removed}|$, times. Constructing the Voronoi diagram takes $O(|M|lg|M|)$ time. Constructing the priority queue takes $O(|M|)$. $|M|$ is the number of leds in the array, M. The inner loop iterates $O(|M|)$ times in the worst case. The priority queue operation *extractMin* takes $O(lg|M|)$ and the function *DoesMetricChange* takes $O(RowSize \times ColumnSize)$. Therefore, within the outer loop, the runtime of the function *DoesMetricChange* is $O(RowSize \times ColumnSize \times |M|$ dominating the overall runtime. The total worst-case running time of the algorithm is $O(|M_{removed}| \times (|M|lg|M| + |M| + |M|(lg|M| + (RowSize \times ColumnSize))))$. Since $|M| = RowSize = ColumnSize$ for a Costas array having $|M|$ points in it, the worst-case running time is $O(|M_{removed}||M|^3)$ where $|M|$ represents the number of leds in the Costas array generated in the Costas-array-generation step, Section 4.1.1.

4.1.3. Sliding Leds. After the *removing-leds* step, if there still exist some OW(s) having 0 or 1 leds in it, the *sliding-leds* phase of the CRS-LP algorithm is applied. The goal of this phase is to eliminate the OWs having 0 or 1 led in them without adding any leds to the led set constructed through *generating-Costas-array* and *removing-leds* phases.

We continue with the led set found in the *removing-leds* phase. First, the union of the OWs having 0 or 1 leds in them is found. Then, using the union, some *candidate led*s (if there are any) to slide are identified. The leds which are not involved in the union and which are on the border of the union are considered as *candidate led*s. Once the *candidate led*s are found, they are slided towards the union in order to reduce the number of the OWs having 0 or 1 led.

A *candidate led* has the capability of sliding to one of its neighbor locations as Figures 6(a) through 6(f) depict. In the figures, shaded cells are the elements of the union of the OWs having 0 or 1 led in them. For a *candidate led* to slide successfully, the metric computed after sliding should be less than the metric computed before sliding (see Eq. (5) for the metric). In other words, for a successful sliding, the metric should improve, resulting in less number of OWs having 0 or 1 leds in them.

For each *candidate led*, possible sliding locations are found. Assuming the metric improves for some possible sliding locations, the next step is picking up the best location among these possible ones for each *candidate led*. For each possible location, the number

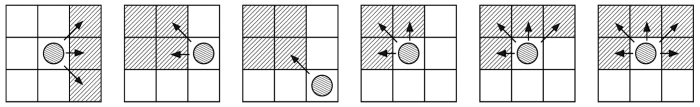


Fig. 6. Some possible sliding movements.

of unique slope-length pairs is calculated as if the led is moved to that location. Then, the location having the maximum number of unique slope-length pairs is picked as the destination location and the led is moved to its new location.

There are two different sliding methods. In *sliding method 1*, a *candidate led* is moved to a sliding location only if the slope-length constraint is preserved. Since the Costas arrays have the unique slope-length property and the *removing-leds* phase preserves the property, the number of pairs is actually equal to the number of unique slope-length pairs. So in method 1, an led is allowed to slide only if the number of unique slope-length pairs does not change after the slide step. In *sliding method 2*, the sliding location that has the greatest number of unique slope-length pairs is picked without considering the slope-length constraint. For large grids, *method 1* has no effect on the led set, since preserving the slope-length property and covering a larger region at the same time is not trivial without adding extra leds. *Method 2* eliminates some of the OWs having less than two leds for large grids. However, the resulting led set does *not* preserve the slope-length property as the downside of *method 2*.

After all the *candidate led*s are processed, the union of the OWs having 0 or 1 led is computed again to see if any one of the *candidate led*s is moved to a new location, and the previous steps are repeated. If none of the *candidate led*s can be slided, or if there are no OWs having 0 or 1 led in them, the algorithm terminates. Appendix B provides an example of the sliding-leds step.

Algorithm 3 contains the details of the sliding-leds step. The running-time complexity of the *sliding-leds* phase of the CRS-LP algorithm is the following. The function *findUnionOWs*, line 2, first finds the OWs having 0 or 1 led. Finding those OWs takes

ALGORITHM 3: SlideLeds(M, X, Y, RowSize, ColumnSize, methodType)

```
   // M holds the leds' locations. 2d bit array.
 1 repeat
 2     union = findUnionOWs(M, X, Y, RowSize, ColumnSize);
 3     candidateLeds = findCandidates(union, M);
 4     if candidateLeds.size()==0 then
 5         break;
 6     keepSliding = false;
 7     foreach candidate in candidateLeds do
 8         slidingLocations = findSlidingLocations(union, candidate);
 9         foreach location in slidingLocations do
10             if !DoesMetricImprove(candidate, location, M) then
11                 slidingLocations.remove(location);
12         if slidingLocations.size() > 0 then
13             bestLocation = getBestLocation(slidingLocations, methodType);
14             slide(candidate, bestLocation, M);
15             keepSliding = true;
16 until keepSliding == false;
17 return M;
```

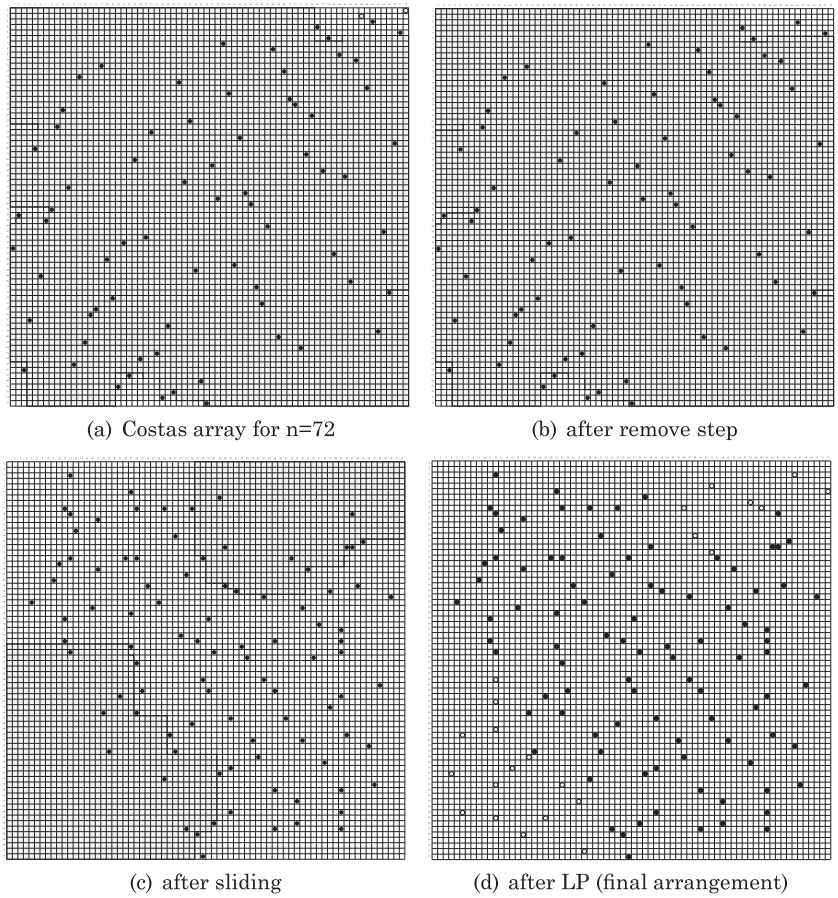
O(RowSize $\times$ ColumnSize) time in the worst case. The union of two polygons P_1, and P_2 can be found in $O(vlogv + klogv)$ time where $v = v_1 + v_2$, v_1 is the number of vertices P_1 has, v_2 is the number of vertices P_2 has, and k is the complexity of the union of P_1 and P_2 using the map overlay algorithm from computational geometry. Let the number of OWs having less than two leds be N_{ow}. Assuming that $RowSize \in O(n)$ and $ColumnSize \in O(n)$, v and k is $O(n+n) = O(n)$ in the worst case. So, the complexity of finding the union of two polygons is $O(nlogn + nlogn) = O(nlogn)$. There are N_{ow} OWs to be united. Since N_{ow} is $RowSize \times ColumnSize$, $N_{ow} \in O(n^2)$. As a result, the complexity of finding the union of all OWs is $O(n^3logn)$ in the worst case. The total runtime of the function *findUnionOWs* is $O(RowSize \times ColumnSize) + O(n^3logn)$, which is equivalent to $O(n^2) + O(n^3logn)$ assuming $RowSize, ColumnSize \in O(n)$ (the first term is the complexity of finding OWs having less than two leds). Finally, the runtime of the function *findUnionOWs* is $O(n^3logn)$. Using the union and the led locations M, the *candidate led*s can be found in $O(RowSize + ColumnSize) = O(n+n) = O(n)$ time, line 3. The upper bound for the number of candidate leds is $O(|M|)$, where $|M|$ is the number of leds. Therefore, the first inner for-loop, line 7 to line 15, iterates $O(|M|)$ times. For each candidate led, corresponding sliding locations can be found in constant time $O(c_1)$, where c_1 is a constant, line 8. The for-loop, line 9 to line 11, iterates the number of sliding locations times which is a constant, c_2. The function *DoesMetricImprove* runs in $O(RowSize \times ColumnSize)$ time. The function *getBestLocation* in Algorithm 3, line 13, returns the best location for the led to slide according to the sliding method type—method 1 or method 2—given as the input. It first finds the number of unique slope-length pairs for each sliding location, so its runtime is dominated by finding the number of unique slope-length pairs, which is $O(|M|^2 log(|M|))$. The outer loop iterates τ times until the algorithm terminates. Recall that if none of the *candidate led*s can be slided, or if there are no OWs having 0 or 1 led in them, the algorithm terminates. As a result, the running time of the sliding-leds phase is $O(\tau(n^3logn + n + c_1|M| + c_2|M||M|^2 log(|M|)))$ which can be rewritten as $O(\tau n^3 logn)$.

The *sliding-leds* phase of the CRS-LP algorithm cannot eliminate all OWs having 0 or 1 led all the time, especially for large grids. Therefore, we use linear programming if there still exist OWs having less than two leds after the *sliding-leds* phase.

4.1.4. Linear Programming (LP). We conclude our *C*RSLP algorithm with the linear programming (LP) step. Recall that we started with a Costas array, then removed the unnecessary leds. If there still exist OWs having less than two leds in them, we proceed with the sliding step (sliding method 1 and 2). After the sliding step, a linear programming step is run to make the OWs have at least two leds in them if some of the OWs still have less than two leds in them. Using this approach, we can reduce a large number of constraints and variables for linear programming.

Assuming the Wii coverage problem could not be solved with the previous steps (Costas-remove-slide), we apply the LP step. At this point, we have OWs having less than two leds in them. The goal of this step is eliminating all those OWs having less than two leds by adding leds on proper locations. In order to achieve this aim, we apply the basic linear programming formulation as discussed in Section 4.3.2 on the union of the OWs having less than two leds in them by feeding the locations of the leds that already reside in the union.

Let the set P contain the variables in the union of OWs having less than two leds, $var_{ij} \in P$. var_{ij} represents the cell located at the i^{th} column and j^{th} row in the grid, and (i, j) is a location in the union. Moreover, an OW has the size $a \times b(4l \times 3l)$. The LP



(a) Costas array for n=72 (b) after remove step

(c) after sliding (d) after LP (final arrangement)

Fig. 7. Illustration of CRS-LP algorithm where n = 72.

formulation for this step is the following.

$$\begin{aligned}
&\text{Minimize: } \sum var_{ij} && ;var_{ij} \in \{0,1\} \\
& && ,\forall var_{ij} \in P \\
&\text{Subject To: } \sum_{i=m}^{m+a-1} \sum_{j=n}^{n+b-1} var_{ij} \geq k && ;\forall m, \forall n(\text{m,n}) \text{ is the upper left position of an OW} \\
& && \text{having less than 2 leds in the union.} \\
& var_{ij} = 1 && ;\text{if the location (i,j) had led from the previous steps} \\
& && \text{of CRSLP algorithm.} \qquad (6)
\end{aligned}$$

We show an example for a grid of 72×72 using 12×9 OWs. We start by a Costas array of 72 which is constructed using the Welch method as depicted in Figure 7(a). After the *remove* step is conducted, two leds located at $(72, 1)$ and $(64, 2)$ are found unnecessary, and removed as shown in Figure 7(b). (In Figure 7(a), the two void circles represent the leds removed by the *remove* step.) The thick lines in Figures 7(a) and 7(b) are the boundaries of the union of OWs having less than two leds. At this point, 52% of the OWs have less than two leds. Then, we apply one of the sliding methods. Sliding

method 1 preserves the slope-length constraint. However, method 1 cannot find any leds to slide in order to eliminate OWs having less than two leds. Let's apply sliding method 2. Figure 7(c) demonstrates led positions and the union of the OWs having less than two leds after the sliding method is applied. At this point, 20% of the all OWs have less than two leds. Figure 7(d) shows the final configuration after LP is conducted according to the formulation (6) (void circles represent the leds added by the LP steps; 23 leds are added by the LP algorithm). All OWs contain two leds or more than two leds in this final arrangement. There are 117 leds in the final led set. The number of all pairs is 6786, and 3939 of these pairs are distinct. Therefore, $\frac{3939}{6786} = 0.58$ of the pairs have unique slope-length values. If only the pairs of which lengths fit in a 12×9 OW are considered, then the number of these pairs is 603, and 213 of them are distinct, making $\frac{213}{603} = 0.35$ of these pairs distinct.

4.2. Metric-Based Random Led Placement (MBR)

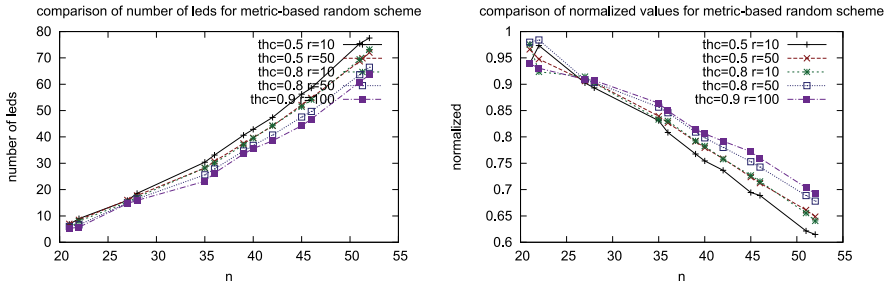
We propose an intelligent random led placement algorithm for comparison purposes. The scheme is based on a metric that we will discuss shortly, and the quality of the outcome depends on two parameters, a *threshold constant* and a *repetition value*. The placement of the IR leds in random locations is based on the metric Eq. (5).

Algorithm 4 shows the details of MBR led placement. The inputs are k, X and Y representing the Observation Window (OW) size, *RowSize* and *ColumnSize* representing the whole grid, *thresConst*, and *repetition*. When the goal is to guarantee at least two IR leds in each OW, the value of k is 2. The arguments *thresConst* and *repetition* are used to generate feasible led placements. In the end, the algorithm returns a set of led locations M guaranteeing at least two IR leds in each OW.

ALGORITHM 4: MBR($k = 2$, X, Y, RowSize, ColumnSize, thresConst, repetition)

```
1  threshold = threshold_original = X × Y;
2  repetitionCount = 0;
3  M[][]={}; // M holds the leds' locations. 2d bit array.
4  while 1 do
5      ledLocation = randomLed();
6      metric_prev = computeMetric(M, ledLocation);
7      M.setLed(ledLocation); // set led location to 1 in 2d bit array
8      metric_current = computeMetric(M, ledLocation);
9      if metric_prev − metric_current ≥ threshold then
           // led added to M.
10         if !has0s1s(M) then
               // termination condition
11             break;
12         repetitionCount = 0;
13         threshold = threshold_original;
14     else
           // do not add the led
15         M.resetLed(ledLocation); // set led location to 0 in 2d bit array
16         repetitionCount++;
17     if repetitionCount ≥ repetition then
18         threshold = threshold × thresConst;
19         repetitionCount = 0;
20 return M;
```

The basis of the MBR algorithm is the infinite loop starting with line 4. In each iteration, an led location is randomly chosen using the function *randomLed*. Then,



(a) number of leds (b) normalized

Fig. 8. Threshold constant and number of repetition comparisons for metric-based random led placement. 12×9 windows.

metric_prev, the metric for that location without the led in the location, is computed. With M.setLed(ledLocation), line 7, an led with ledLocation is added to M. The metric for the location, ledLocation, is computed once more with the led added this time, and it is set to *metric_current*. The metric difference between *metric_prev* and *metric_current* gives an idea on the feasibility of the random location, ledLocation. The higher the metric difference, the more feasible the led location, since having a low number of OWs having one or zero leds results in low metric values. We define a threshold value for this issue. *threshold* is set to $X \times Y$ in line 1. This value is the highest value the metric difference can hold. If the metric difference is higher than the threshold value, then the led is added to the set M, line 9. The threshold value is updated throughout the program lifetime according to the algorithm argument *thresConst*, where $0 < thresConst < 1$. The argument *repetition* binds the number of iterations that use a specific threshold value. After the led is added, the helper function, line 10, *has0s1s* decides if the algorithm should terminate or not. The function returns true if M has any OWs having one or zero leds in it. It returns false, otherwise. So, if the function *has0s1s* returns false, this means that all the OWs have more than or equal to two leds in it, and the program terminates. If not, we reset the *threshold* value to its original value, the *repetitionCount* (counts the number of iterations that a specific threshold value is used) to 0, and continue with the next iteration. If the metric difference is less than the threshold value, line 9, then the led will not be added to the set M and *repetitionCount* is incremented. Line 17 decides if the same threshold value is used for the next iteration. If *repetitionCount* is greater than a user-specified *repetition* value, then the threshold value is updated according to *thresConst* (user specified), and *repetitionCount* is reset.

The running-time complexity of the MBR led placement algorithm is $O(RepThresConst \times RowSize \times ColumnSize \times RowSize \times ColumnSize + |M_{final}| \times RowSize \times ColumnSize)$. There are at most $RowSize \times ColumnSize$ different led locations that might be considered in the grid, and *computeMetric* takes $O(RowSize \times ColumnSize)$ time. Also, the repetitions of thresholds add another factor $RepThresConst$ which is user dependent. These three runtimes constitute the first term in the running-time complexity. The second term comes from the function *has0s1s*. Its complexity is $O(RowSize \times ColumnSize)$, and it is computed the number of leds in the final set, $|M_{final}|$, many times.

Figures 8(a) and 8(b) show the effect of *thresConst* and *repetition* on the number of leds and normalized distinct slope-length values (distinct slope-length pairs/all pairs), respectively, for $n \times n$ grids with 12×9 OWs where $n = [21, 22, 27, 28, 35, 36, 39, 40, 42, 45, 46, 51, 52]$. In the figures, *thc* represents *thresConst* and *r* represents *repetition*. The higher the *thresConst* and *repetition* values, the better the led

placement, as the figures suggest. When *thresConst* is high, the algorithm is more likely to catch a good led location with a good metric difference because the decreasing rate of the variable *threshold* is low. When *repetition* is high, the algorithm is again more likely to find a good led location with a good metric difference because the algorithm runs with the same higher threshold value in a greater number of iterations. Appendix C provides an example of the MBR algorithm with the *thresConst* and *repetition* values of 0.9 and 100, respectively.

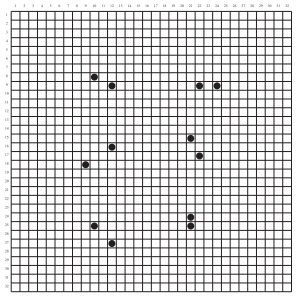
4.3. Linear-Programming-Based Schemes

Linear programming (LP) can be used to solve WCP(N_1, N_2, a, b, k), and *optimal* solutions for small coverage areas are found using the LP techniques. A linear programming problem might be defined as the problem of maximizing or minimizing a linear function subject to linear constraints. For WCP, we consider the ceiling as an $N_1 \times N_2$ grid and x_{ij} represents the cell located at the i^{th} column and j^{th} row in the grid. The OW will have the size $a \times b$ ($4l \times 3l$), and the linear constraints will be formed according to the vertical and horizontal movements of the mobile element. We have four types of formulations: concrete integer linear programming (CILP), basic integer linear programming (BILP), and linear programming with row-column constraints and incremental linear programming.

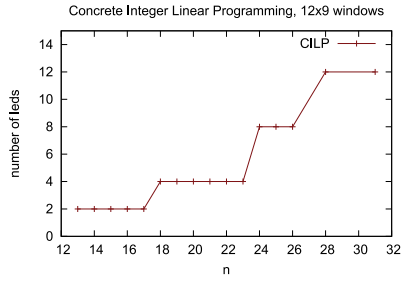
4.3.1. Concrete Integer-Linear-Programming (CILP) Formulation. This method finds the *optimal* solutions for the WCP. The binary-linear-programming formulation for our problem, *Wii Coverage Problem [WCP(N_1, N_2, a, b, k)]*, is the following.

$$
\begin{aligned}
\text{Minimize: } & \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} x_{ij} && ;x_{ij} \in \{0,1\} \\
\text{Subject To: } & \sum_{i=m}^{m+a-1}\sum_{j=n}^{n+b-1} x_{ij} \geq k && ;\forall m, 1 \leq m \leq N_1 - a + 1 \\
& && ,\forall n, 1 \leq n \leq N_2 - b + 1 \\
& x_{ij} + x_{(i+u)(j+v)} + x_{kl} + x_{(k+u)(l+v)} \leq 3 && ;0 \leq u \leq N_1, 0 \leq v \leq N_2 \\
& && ,1 \leq i \leq N_1, 0 \leq j \leq N_2 \\
& && ,1 \leq k \leq N_1, 0 \leq l \leq N_2 \\
& && ,k + u \leq N_1, l + v \leq N_2 \\
& && ,i + u \leq N_1, j + v \leq N_2 \qquad (7)
\end{aligned}
$$

In linear-programming formulation (7), x_{ij} represents the cell located at the i^{th} column and j^{th} row in the grid, and its value represents the existence (1) or nonexistence (0) of an IR led. Our goal is minimizing the number of the IR leds on the ceiling. There are $N_1 \times N_2$ variables in the linear programming formulation. The first set of constraints in Eq. (7), $\sum_{i=m}^{m+a-1}\sum_{j=n}^{n+b-1} x_{ij} \geq k$, is the window constraint where each $a \times b$ overlapping subrectangle of the $N_1 \times N_2$ grid contains at least k leds. Each of these subrectangles should contain at least k leds. This introduces $(N_1 - a + 1) \times (N_2 - b + 1) \in O(N_1 \times N_2)$ many constraints. The second set of constraints in Eq. (7), $x_{ij} + x_{(i+u)(j+v)} + x_{kl} + x_{(k+u)(l+v)} \leq 3$, is the slope-length constraint. It says that, if an led pair having the endpoints (i, j) and $(i+u, j+v)$ takes place, then another pair having endpoints (k, l) and $(k+u, l+v)$ is not allowed in order to preserve the slope-length constraint. Preserving unique slope-length pairs requires $O({N_1}^3)({N_2}^3)$ many constraints, since there is one constraint for each set of integers i, j, k, l, u, v. Due



(a) the optimal solution for 32×32 grids for 12×9 windows



(b) number of leds vs. grid size for 12×9 windows using Concrete Integer Linear Programming (CILP)

Fig. 9.

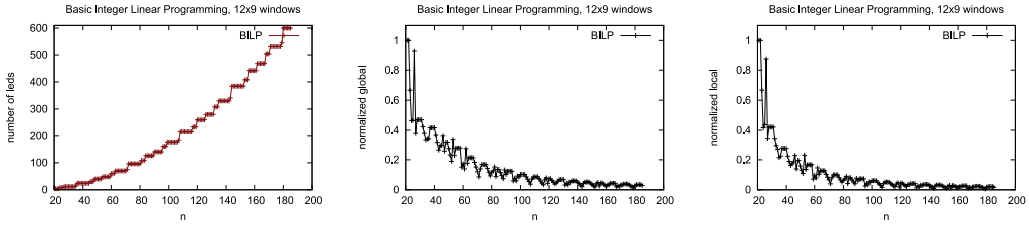
to this high number of constraints, we have results up to only 32×32 grids for 12×9 observation windows (the problems were run on Sun Sparc Enterprise T5440 server which has 256GB memory with 4GB FB-DIMMs. The bottleneck is on the memory). Notice that the solutions of the concrete LP formulation are the optimal solutions for our problem. Figure 9(a) shows the optimal solution for 32×32 grids for 12×9 windows. Moreover, Figure 9(b) shows the number of leds used as the grid size increases. All pairs are distinct in terms of slope and distance when this scheme is used.

4.3.2. Basic Integer-Linear-Programming (BILP) Formulation. The concrete integer-linear-programming formulation introduces too many constraints, so it is not practical to use for big grids. Therefore, we remove the slope-length constraint to get rid of the huge number of constraints caused by the slope-length constraint. Then, the basic linear-programming formulation is the following.

$$\begin{aligned} \text{Minimize: } & \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} x_{ij} && ;x_{ij} \in \{0,1\} \\ \text{Subject To: } & \sum_{i=m}^{m+a-1}\sum_{j=n}^{n+b-1} x_{ij} \geq k && ;\forall m, 1 \leq m \leq N_1 - a + 1 \\ & && ,\forall n, 1 \leq n \leq N_2 - b + 1 \end{aligned} \tag{8}$$

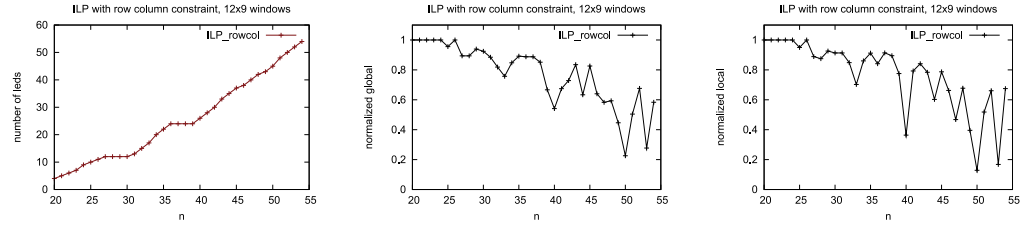
For the formulation (8), the number of constraints is $(N_1 - a + 1) \times (N_2 - b + 1) \in O(N_1 N_2)$, assuming a and b are constants. Similarly, the number of variables is $N_1 \times N_2 \in O(N_1 N_2)$.

Basic integer-linear-programming solutions give a lower bound on the number of required leds for our system. Figure 10(a) shows the number of leds used as the grid size increases. We define two vector types: the *global vector* can have any size as long as the grid size allows, and the *local vector* is a vector that can fit in an $a \times b$ OW window. The definition of normalization in this context is the following. The value found by dividing the distinct global vectors by all of the global vectors that can be generated using the leds placed on the grid is called *global normalization*. Moreover, the value found by dividing the distinct local vectors by all of the local vectors that can be generated using the leds placed on the grid is called *local normalization*. Figure 10(b) shows the quality of the leds placement for the basic integer-linear-programming method using 12×9 OWs for normalization$_{\text{global}}$. The quality in this context is defined as the normalized slope-length uniqueness. Figure 10(c) shows the normalization$_{\text{local}}$ for BILP.



(a) number of leds vs. grid size (b) normalized$_{\text{global}}$ vs. grid size (c) normalized$_{\text{local}}$ vs. grid size

Fig. 10. Basic Integer Linear Programming (BILP) with 12×9 OW.



(a) number of leds vs. grid size (b) normalized$_{\text{global}}$ vs. grid size (c) normalized$_{\text{local}}$ vs. grid size

Fig. 11. Integer linear programming with row-column constraints for 12×9 OWs.

4.3.3. Integer Linear Programming with Row-Column Constraint. Our aim is having unique slope-length pairs in addition to the constraint where each window contains at least k leds in it. In order to increase the number of unique slope-length pairs, we add the constraint where each row and each column in the grid holds at most one IR led to our basic LP formulation. The new LP formulation with extra constraints is the following.

$$\begin{aligned}
\text{Minimize: } & \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} x_{ij} && ;x_{ij} \in \{0,1\} \\
\text{Subject To: } & \sum_{i=m}^{m+a-1}\sum_{j=n}^{n+b-1} x_{ij} \geq k && ;\forall m, 1 \leq m \leq N_1 - a + 1 \\
& && ,\forall n, 1 \leq n \leq N_2 - b + 1 \\
& \sum_{i=1}^{N_1} x_{ij} \leq 1 && ;\forall j, 1 \leq j \leq N_2 \\
& \sum_{j=1}^{N_2} x_{ij} \leq 1 && ;\forall i, 1 \leq i \leq N_1
\end{aligned} \tag{9}$$

The row-column constraint introduces an extra $N_1 \times N_2$ constraints, resulting in infeasible solutions for large N_1 and N_2. Moreover, the LP solver runs longer in order to meet the new constraints. We have the solutions for grids of size up to 54×54 for 12×9 OWs as depicted in Figures 11(a), 11(b), and 11(c).

4.3.4. Incremental Linear Programming. In this method, instead of covering the whole $N_1 \times N_2$ grid at once, we cover the grid incrementally as follows. A solution to our problem for some $N_1 \times N_2$ grid is chosen as the *base solution*. This solution can be

found by using any of the methods we propose. Then, the base solution to $N_1 \times N_2$ is extended by adding new rows and columns satisfying the required constraints using binary integer linear programming. After K many row-column extensions, we replace the base solution with the latest solution we have found, and continue finding new solutions in this manner. Since we make use of the previous solutions, the *incremental linear programming method* requires at least an order of N fewer constraints and variables than the other linear programming methods, as shown in Table I. As a result, this method is faster and scalable since it can be used to find solutions for large N_1 and N_2 values. The formulation for the incremental LP method is presented as formula (10) with $a \times b$ OWs. (The base solution is a solution to an $N_1 \times N_2$ grid.)

$$\begin{aligned}
\text{Minimize: } & \sum_{i=N_1-a+2}^{N_1+k} \sum_{j=1}^{N_2+k} x_{ij} + \sum_{i=1}^{N_1-a+1} \sum_{N_2-b+2}^{N_2+k} x_{ij} && ;x_{ij} \in \{0,1\} \\
\text{Subject To: } & \sum_{i=m}^{m+a-1} \sum_{j=n}^{n+b-1} x_{ij} \geq k && ;\forall m, N_1 - a + 2 \leq m \leq N_1 + k - a - 1 \\
& && ,\forall n, 1 \leq n \leq N_2 + k - b - 1 \\
& \sum_{i=m}^{m+a-1} \sum_{j=n}^{n+b-1} x_{ij} \geq k && ;\forall m, 1 \leq m \leq N_1 - a + 1 \\
& && ,\forall n, N_2 - b + 2 \leq n \leq N_2 + k - b - 1 \\
& x = 1 && ;x \in |S_{LIIR}| \qquad (10)
\end{aligned}$$

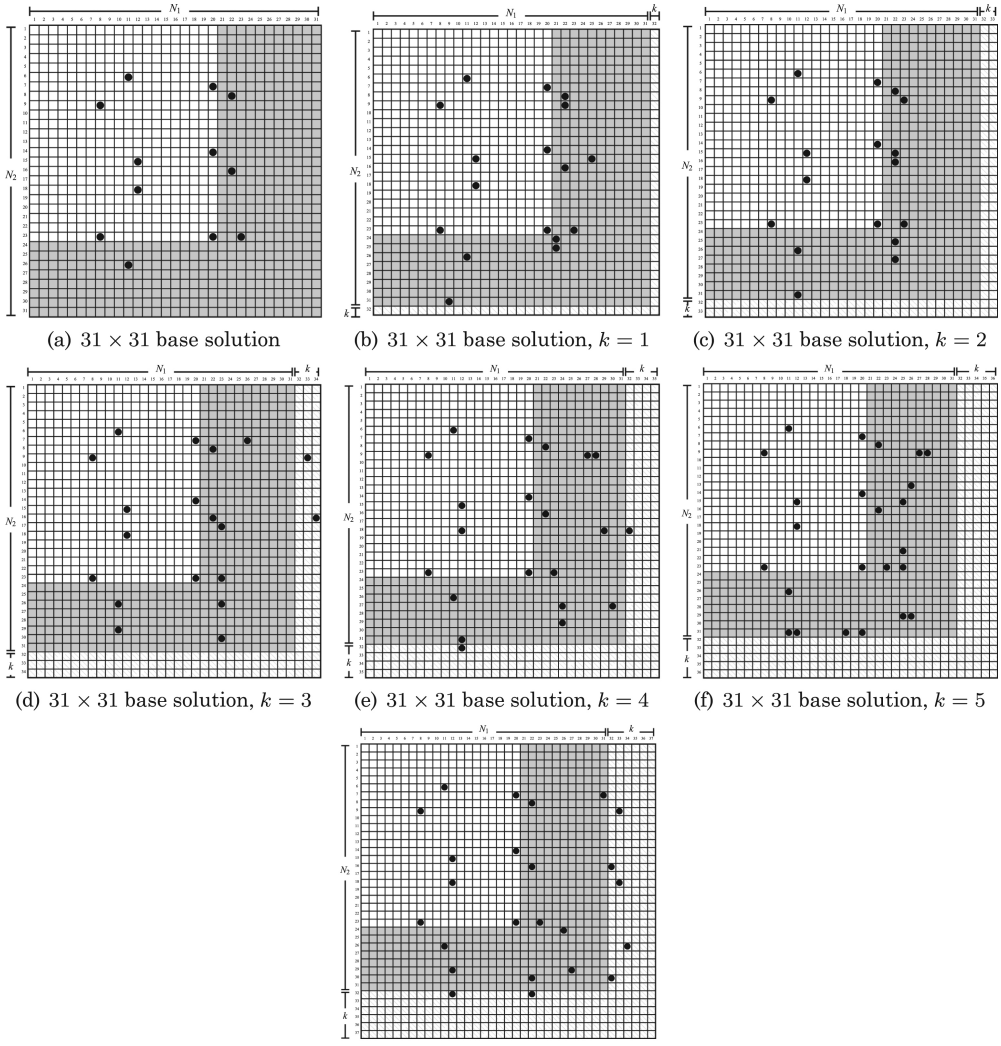
In LP formula (10), S_{LIIR} is the set of led locations from the base solution. The set S_{LIIR} does not contain all the leds from the base solution, but only the leds in the incremental region. The leds in the gray area in Figure 12(a) are the elements of the set S_{LIIR}. The incremental region is the region containing the variables which we minimize in the incremental LP formula (10).

Once the linear programming part is run and new leds are found, all the leds from the base solution and the new leds from the LP run are combined to form a new solution with higher N_1 and N_2 values.

Figures 12(a) through 12(g) depict how the incremental LP method works when the CILP (31×31) solution is chosen as the initial base solution for 12×9 OWs for the first K extensions. Each of the solutions 32×32 through 37×37 is found using the solution 31×31 as the base solution. Since $K = 6$, for this example, the most recent solution, 37×37, is picked as the next base solution. Figures 13(a) through 13(c) show the subsequent base solutions. For example, in Figure 13(a), the solution for 37×37 is the base solution and the solution for 43×43 is found using 37×37. Since $K = 6$, the solution for 43×43 will be the next base solution as used in Figure 13(b).

Picking the K value is crucial for the performance of the incremental LP, regardless of the type of method used for the initial base solution. When K is a multiple of a (a is the size of the horizontal edge of the $a \times b$ OWs) or greater than a, the performance mimics the performance of BILP (low number of leds, low normalization values), which is not favorable. When a fraction of a is chosen as K—such as $K = 6$ for 12×9 OWs—, the LP run is exposed to a distortion and the normalization values get better.

Since the incremental LP method requires fewer constraints and is more scalable compared to the other LP methods we propose, the incremental LP method could also be applied to the final solution that can be found using the CRSLP method, Section 4.1, in order to achieve solutions for very high $N_1 \times N_2$ grids.



(a) 31×31 base solution (b) 31×31 base solution, $k = 1$ (c) 31×31 base solution, $k = 2$

(d) 31×31 base solution, $k = 3$ (e) 31×31 base solution, $k = 4$ (f) 31×31 base solution, $k = 5$

(g) 31×31 base solution, $k = 6$

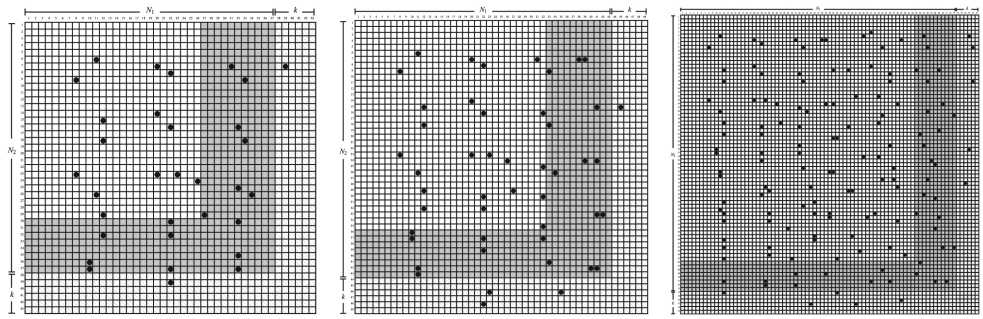
Fig. 12. Illustration incremental LP method on CILP (31×31) as the initial base solution, 12×9 OWs.

Scalability issues. Although integer-linear-programming formulations are simple, integer linear programming is NP-hard. Therefore, the LP approaches CILP and ILP_{rowcol} do not work for large N_1 and N_2 values. Moreover, the number of constraints for our problem increases exponentially as N_1, and N_2 increase linearly, causing scalability problems.

Table I shows the required number of constraints and variables for the integer linear-programming formulas proposed.

5. COMPARISON OF THE PROPOSED SCHEMES

In this section, we compare the methods that find led placements for the WCP. Table II compares the methods presented by the number of leds used and normalized values found by dividing the distinct vectors (pairs) by all of the vectors (pairs) that can be



(a) 37×37 base solution, $k = 6$ (b) 43×43 base solution, $k = 6$ (c) 73×73 base solution, $k = 6$

Fig. 13. Incremental LP method on CILP (31×31), base solutions (12×9 OWs).

Table I. Comparison of the Number of Constraints and Variables among Integer-Linear-Programming Formulas

Method	Number of Constraints	Number of Variables
CILP	$(N_1 - a + 1)(N_2 - b + 1) + \Theta({N_1}^3 {N_2}^3)$	$N_1 \times N_2$
$\text{ILP}_{\text{rowcol}}$	$(N_1 - a + 1)(N_2 - b + 1) + N_1 + N_2$	$N_1 \times N_2$
BILP	$(N_1 - a + 1)(N_2 - b + 1)$	$N_1 \times N_2$
Incremental LP	$k(N_1 + N_2 + 2k - a - b - 1) + \lvert S_{LIIR} \rvert$	$(N_2 + k)(\lvert N_2 - N_1 \rvert + k + a - 1)$ $+ (N_1 - a + 1)(\lvert N_2 - N_1 \rvert + k + b - 1)$

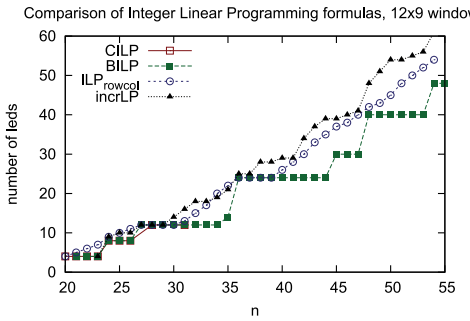
generated using the leds placed on an $N \times N$ grid with N values up to 54. In the table, global normalization values are represented as Norm_G and local normalization values are represented as Norm_L. The methods on the table use 12×9 OWs. The method incremental LP uses the base solution generated by the method BILP, where $n = 23$ and the value of k is 6. Moreover, CRSLP uses the sliding method 2.

The method CILP requires a huge number of linear programming constraints as the coverage area gets larger. As a result, we have the *perfect* solutions up to $n = 32$ for 12×9 OWs. Therefore, CILP is the best method for grids where $n \leq 32$. BILP is a method that gives the lower bound on the number of leds required, however, it does not provide good solutions. $\text{ILP}_{\text{rowcol}}$ produces good results, but has limitations on guaranteeing the constraints for large grids, as we discussed in Section 6. We have the results generated by $\text{ILP}_{\text{rowcol}}$ up to $n = 54$ using 12×9 OWs. Incremental LP is a fast technique producing good results, and can be used for large grids. However, incremental LP requires the greatest number of leds among all LP-based methods. Figures 14(a), 14(b), and 14(c) compare all the LP-based schemes in terms of number of leds used, Norm_G, and Norm_L, respectively, for n values up to 150.

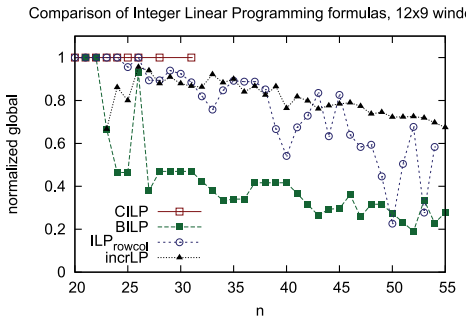
The intelligent random led placing algorithm, MBR, produces solutions almost as good as CRS-LP. However, MBR requires the greatest number of leds among all proposed methods. CRS-LP is the method producing good results without having to have large memory requirements. Moreover, for large areas, CRS-LP is the most promising one since it produces the highest normalization values and uses the lowest number of leds (except BILP, whose solutions are the lower bound on the number of leds required). Some entries in Table II are missing for CRS-LP. The reason is that the Costas construction methods, Lempel and Welch, cannot produce Costas arrays for all n values. However, any other Costas constructions can be used for the missing entries. Figures 15(a), 15(b), and 15(c) compare the methods in terms of number of leds used, Norm_G, and Norm_L, respectively, for n values up to 150.

Table II. Comparison of the Proposed Methods

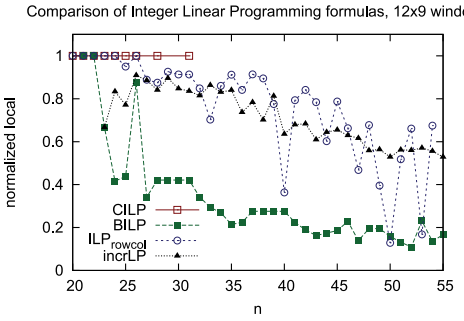
N	CILP			BILP			ILP_{rowcol}			Incremental LP			MBR			CRS-LP		
	Led#	$Norm_G$	$Norm_L$	Led#	$Norm_G$	$Norm_L$	Led#	$Norm_G$	$Norm_L$	Led#	$Norm_G$	$Norm_L$	Led#	$Norm_G$	$Norm_L$	Led#	$Norm_G$	$Norm_L$
21	4	1.0	1.0	4	1.0	1.0	5	1.0	1.0	-	-	-	5.1	0.97	0.96	8	1.0	1.0
24	8	1.0	1.0	8	0.46	0.41	9	1.0	1.0	9	0.86	0.83	9.93	0.93	0.91	-	-	-
27	12	1.0	1.0	12	0.38	0.34	12	0.89	0.88	12	0.93	0.89	14.37	0.90	0.88	14	1.0	1.0
28	12	1.0	1.0	12	0.47	0.42	12	0.89	0.87	12	0.87	0.84	15.56	0.90	0.87	16	0.96	0.94
30	12	1.0	1.0	12	0.47	0.42	12	0.92	0.91	14	0.88	0.85	16.55	0.90	0.87	-	-	-
31	12	1.0	1.0	12	0.47	0.42	13	0.88	0.91	16	0.86	0.84	16.94	0.90	0.86	-	-	-
33	-	-	-	12	0.38	0.29	17	0.76	0.7	18	0.92	0.86	18.2	0.89	0.85	-	-	-
36	-	-	-	12	0.34	0.22	24	0.89	0.84	25	0.84	0.74	25.3	0.84	0.8	28	0.88	0.80
39	-	-	-	24	0.42	0.28	24	0.66	0.77	28	0.87	0.81	32.23	0.81	0.74	33	0.87	0.78
42	-	-	-	24	0.32	0.19	30	0.73	0.84	34	0.80	0.68	36.8	0.80	0.7	-	-	-
45	-	-	-	30	0.3	0.19	37	0.83	0.79	39	0.78	0.65	42.1	0.78	0.68	47	0.77	0.69
48	-	-	-	40	0.32	0.2	42	0.59	0.68	48	0.74	0.56	49.59	0.74	0.64	-	-	-
54	-	-	-	48	0.23	0.14	54	0.58	0.67	61	0.70	0.56	65.74	0.69	0.54	-	-	-



(a) number of leds comparison of integer-linear-programming solutions for 12×9 OWs



(b) $\text{normalization}_{\text{global}}$ values comparison of integer-linear-programming solutions for 12×9 OWs



(c) $\text{normalization}_{\text{local}}$ values comparison of integer-linear-programming solutions for 12×9 OWs

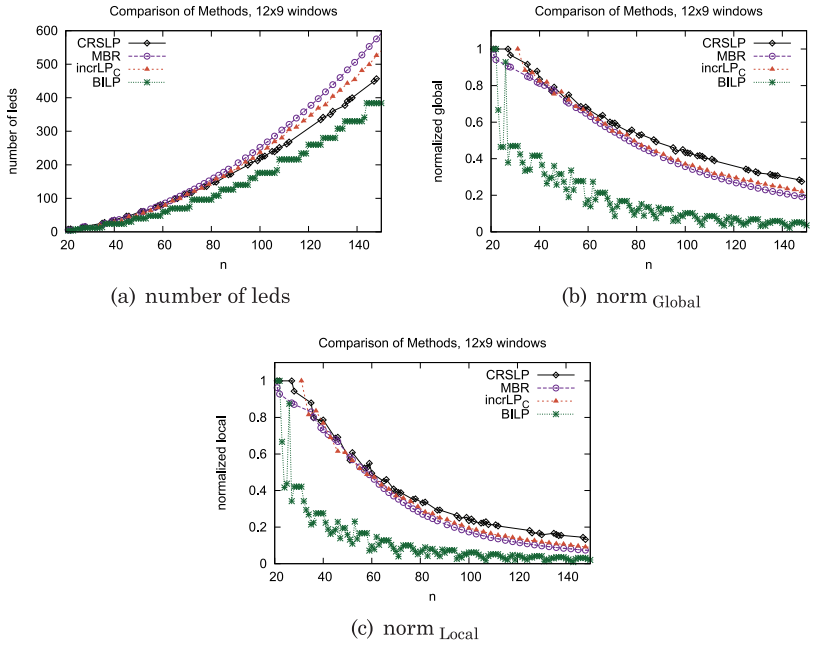
Fig. 14. Comparison of integer-linear-programming solutions (zoomed in).

6. THEORETICAL RESULTS ON THE SATISFIABILITY OF COSTAS AND WINDOW CONSTRAINTS

We provide theoretical analysis on the applicability of three properties in terms of the size of the grid. Two of these properties are the Costas array properties: *row-column constraint* (each row and column has one led) and *unique slope-length constraint* (all the led pairs have unique slope-length values). The third property is the *window constraint* where each OW has at least k IR leds in it. There are limitations on satisfying these properties simultaneously, and it is not possible to satisfy all three properties at the same time for all grid sizes. In order to prove it, we change the WCP slightly. Instead of guaranteeing k leds in each OW, k leds are guaranteed in nonoverlapping OWs. We call this problem $WCP_{non-overlapping}$. It's trivial to observe that the number of leds required for WCP is more than the number of leds required for $WCP_{non-overlapping}$.

THEOREM 6.1. *For an $N_1 \times N_2$ grid and $a \times b$ nonoverlapping windows having k leds in each window, it is not possible to satisfy both the window constraint and the row-column constraint if $\lfloor \frac{N_1}{a} \rfloor \times k > b$, or $\lfloor \frac{N_2}{b} \rfloor \times k > a$.*

PROOF. Proof by contradiction. Assume that for an $N_1 \times N_2$ grid and $a \times b$ nonoverlapping windows having k leds in each window, it is possible to satisfy both the window constraint and the row-column constraint if $\lfloor \frac{N_1}{a} \rfloor \times k > b$. For the first b rows with complete OWs spanning N_1 columns, the number of leds is at least $\lfloor \frac{N_1}{a} \rfloor \times k$ resulting from the windows constraint. The number of leds for the first b rows is at most b according to the row-column constraint. If $\lfloor \frac{N_1}{a} \rfloor \times k > b$, at least one of the b rows has



(a) number of leds

(b) norm $_{\text{Global}}$

(c) norm $_{\text{Local}}$

Fig. 15. MBR, CRSLP (sliding method 2), BILP comparison, 12×9 OWs. n values range up to 150.

more than one led in it, contradicting the row-column constraint. Similarly, assume that it is possible to satisfy both the window constraint and the row-column constraint for the first a columns with complete OWs spanning N_2 rows if $\lfloor \frac{N_2}{b} \rfloor \times k > a$. Then, if $\lfloor \frac{N_2}{b} \rfloor \times k > a$, one of the columns has more than one led in it, contradicting the row-column constraint. □

Since both the window and the row-column constraints cannot be satisfied at the same time for all $N_1 \times N_2$ grids, we remove the row-column constraint and keep the windows constraint because the windows constraint is the heart of our methods for location estimation. Now, we will investigate whether both the window constraint and the unique slope-length constraints hold for all grid sizes or not.

THEOREM 6.2. *For an $N_1 \times N_2$ grid and $a \times b$ nonoverlapping windows having k leds in each window, it is not possible to satisfy both the window constraint and the unique slope-length constraint if $\lfloor \frac{N_1}{a} \rfloor (\lfloor \frac{N_2}{b} \rfloor - 1)k^2 > (2a-1)(2b-1)$.*

PROOF. Proof by contradiction. Assume that, for an $N_1 \times N_2$ grid and $a \times b$ nonoverlapping windows (having k leds in each window), it is possible to satisfy both the window constraint and the unique slope-length constraint if $\lfloor \frac{N_1}{a} \rfloor (\lfloor \frac{N_2}{b} \rfloor - 1)k^2 > (2a-1)(2b-1)$. A specific case regarding two consecutive vertical windows is considered. Figure 16 depicts the consecutive vertical windows. The ends of the pairs considered should be in different windows, as the figure shows. The total number of led pairs corresponding to the vertical consecutive OWs for an $N_1 \times N_2$ grid is found as follows. Both two consecutive nonoverlapping vertical windows can be paired as shown in Figure 16. An ellipse in the figure represents a window pair. A magnified window pair is depicted to the right of the figure. Each window pair contributes k^2 led pairs to the total number of pairs. There are $\lfloor \frac{N_1}{a} \rfloor (\lfloor \frac{N_2}{b} \rfloor - 1)$ vertical consecutive window pairs. Therefore, the total

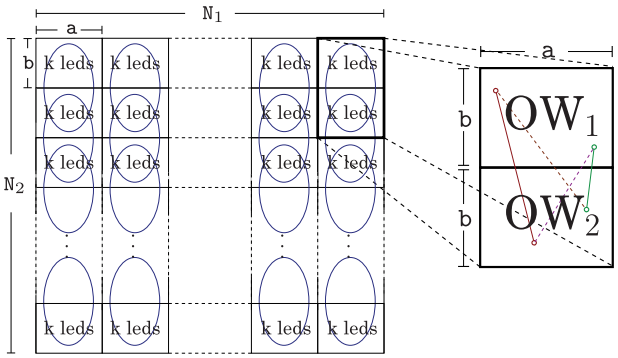


Fig. 16. Total number of unique slope-length pairs using two consecutive vertical windows.

number of pairs corresponding to the vertical consecutive OWs for an $N_1 \times N_2$ grid and $a \times b$ nonoverlapping windows having k leds in each window is $\lfloor \frac{N_1}{a} \rfloor (\lfloor \frac{N_2}{b} \rfloor - 1)k^2$.

The maximum number of *unique* slope-length pairs between two consecutive vertical paired windows is computed as follows. Let $R_v(a, b)$ represent the maximum number of unique slope-length pairs between two consecutive nonoverlapping vertical $a \times b$ windows. Then, $R_v(a, b)$ can be computed iteratively or recursively. For the iterative version, $R_v(a, b)$ is computed as $R_v(a, b) = b(a + \sum_{i=1}^{a-1} 1) + (b - 1)(a + \sum_{i=1}^{a-1} 1) = (2a - 1)(2b - 1)$.

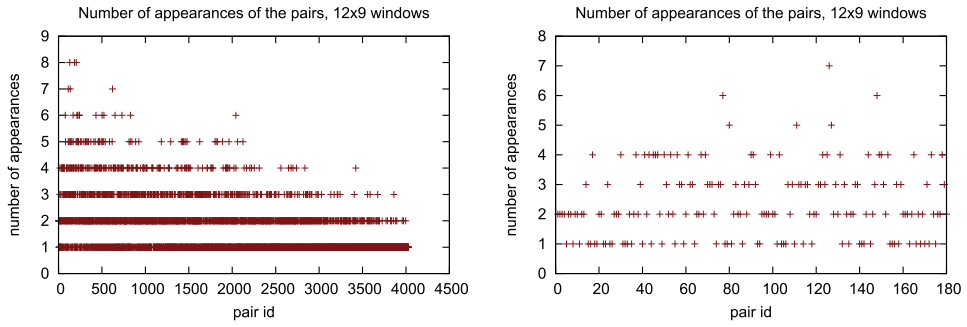
If $\lfloor \frac{N_1}{a} \rfloor (\lfloor \frac{N_2}{b} \rfloor - 1)k^2 > (2a - 1)(2b - 1)$, then there have to be led pairs having the same slope-distance values, contradicting the unique slope-distance constraint. □

For 12×9 windows and $k = 2$, the relation $\lfloor \frac{N_1}{a} \rfloor (\lfloor \frac{N_2}{b} \rfloor - 1)k^2 > (2a-1)(2b-1)$ computes to $N_1 N_2 > 10557$ $\lfloor \frac{N_1}{12} \rfloor (\lfloor \frac{N_2}{9} \rfloor - 1) > 97.75$. Therefore, grids of size of 108×108 and larger cannot have unique slope-length pairs considering vertical consecutive OW pairs. The bound we found is not the lower bound and it can be tightened further considering other arrangements such as nonconsecutive vertical OW pairs, and so on. Moreover, as the grid size increases, the normalization values decrease, since the number of unique led pairs also decreases.

We also investigate the effect of the density of the IR leds placed in an $N_1 \times N_2$ grid on the uniqueness of the led pairs. Assuming m leds are used to cover an $N_1 \times N_2$ grid, it is not possible that all pairs can be unique when the value of m is large. First, the possible number of distinct vectors in an $N_1 \times N_2$ grid, $DV_{N_1N_2}$, needs to be found. Considering $i \times j$ axis-aligned rectangles, there are three different orientations of axis-aligned rectangles that can be generated using i and j unit line segments for the cases where $i \neq 0 \wedge j \neq 0$, $(i = 0 \wedge j \neq 0) \vee (i \neq 0 \wedge j = 0)$, and $i = 0 \wedge j = 0$. Notice that each diagonal of the rectangles can be thought as a distinct vector. Let $p(i, j)$ be the number of unique vectors for an axis-aligned rectangle whose x and y components are either i or j, and $1 < i < max(N_1, N_2)$, $1 < j < min(N_1, N_2)$ are integers. Then $p(i, j)$ is the following.

$$p(i,j) = \begin{cases} 2 & \text{if } i \neq 0 \wedge j \neq 0 \\ 1 & \text{if } (i = 0 \wedge j \neq 0) \vee (i \neq 0 \wedge j = 0) \\ 0 & \text{if } i = 0 \wedge j = 0 \end{cases}$$

Using $p(i, j)$, the number of distinct vectors in an $N_1 \times N_2$ grid ($DV_{N_1N_2}$) is found as $DV_{N_1N_2} = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} p(i, j) = 2N_1N_2 - N_1 - N_2$.



(a) frequency of the global pairs (CRSLP, 72×72 grid, 12×9 OWs) (b) frequency of the local pairs (CRSLP, 72×72 grid, 12×9 OWs)

Fig. 17.

THEOREM 6.3. *For an $N_1 \times N_2$ grid and m leds deployed in the grid, it is not possible to satisfy the unique slope-length constraint if $4N_1N_2 - 2N_1 - 2N_2 < m(m-1)$.*

PROOF. Proof by contradiction. Assume that, for an $N_1 \times N_2$ grid and m leds deployed in the grid, it is possible to satisfy the unique slope-length constraint if $2N_1N_2 - N_1 - N_2 < \binom{m}{2}$. The maximum number of distinct led pairs in an $N_1 \times N_2$ grid is $2N_1N_2 - N_1 - N_2$. Moreover, m leds can generate $\binom{m}{2}$ many led pairs. Then, if $2N_1N_2 - N_1 - N_2 < \binom{m}{2}$, more than one led pairs have to have the same slope-length values, contradicting the unique slope-length constraint. □

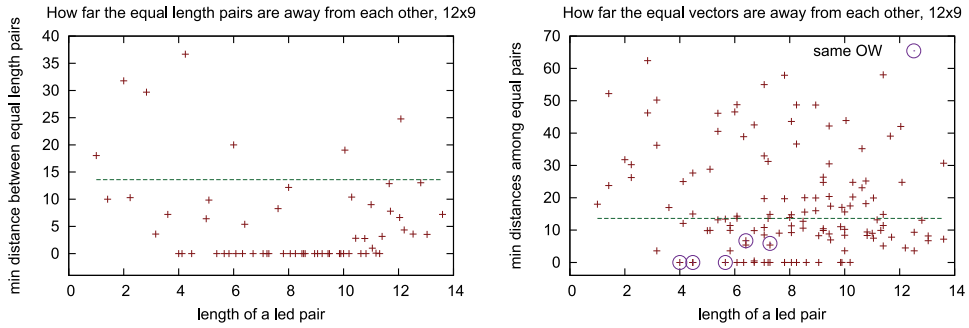
Theorem 6.3 shows that, for $N \times N$ grids, when $m \geq 2N$, it is not possible to have distinct led pairs generated from m leds covering the grid.

7. DISCUSSION

In this section we first show how to work around the theoretical limitations for large coverage areas. Then we discuss the practical aspects of the system. Let *ledSet* be the set of the leds from the placement, and also let led_i, an element of the *ledSet*, have coordinates (x_i, y_i) where $1 \leq i \leq |\text{ledSet}|$. Then, *pairSet* contains led pairs $\text{P}_k(\text{led}_l, \text{led}_m)$ where $1 \leq k \leq \binom{|\text{ledSet}|}{2}$, $1 \leq l \leq |\text{ledSet}|$, $1 \leq m \leq |\text{ledSet}|$, and $l \neq m$. Let Length_{P_k} be the length of the pair, P_k. In other words, Length_{P_k} is the distance between led_l and led_m, $|\text{led}_l, \text{led}_m|$. Also, let Slope_{P_k} be the slope of the pair, P_k. In other words, Slope_{P_k} is the slope between led_l and led_m. Within the scope of these definitions, a perfect solution has distinct Length_{P_k} and Slope_{P_k} values for all k. Recall that the ME uses the length and slope values to differentiate the IR leds. However, for a solution of a large coverage area, Length_{P_k} and Slope_{P_k} values might be equal for some k. We will analyze a led placement produced by CRS-LP for a 72×72 grid with OW size 12×9 to see the effect of equal length and slope values of different led pairs on our system. First, we inspect the frequency of the led pairs having equal length and slope values. Then, we will see how far the led pairs having equal length and slope values are from each other. These observations help us examine the goodness of the led placements.

7.1. Frequency of Led Pairs Having Equal Length and Slope Values

The frequency of the led pairs having equal length and slope values should be analyzed to see the effectiveness of our system. If there exist too many led pairs having equal length and slope values, then it is harder to differentiate them. The led placement produced by CRSLP ($n = 72$) has 117 leds, so there exist 6786 led pairs. These led pairs constitute 4028 different length and slope values. Figures 17(a) and 17(b) show



(a) minimum distances among led pairs having the same lengths (CRSLP, 72×72 grid, 12×9 OWs) (b) minimum distances among the same led pairs (vectors); CRSLP, 72×72 grid, 12×9 OWs

Fig. 18.

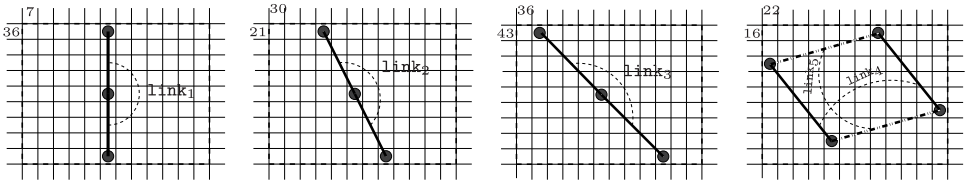
the frequencies of the led pairs having equal length and slope values produced by our example led placement globally and locally, respectively. For Figure 17(a), the x-axis classifies the 4028 distinct length and slope values, and the y-axis shows the number of pairs having the same length-slope value. Maximum frequency is 8 and more than 3 is quite rare, which means having pairs with equal length-slope values in the OW all the time is a low probability. Besides, in a large number of cases, more than two IR sensors will be in the OW, resulting in at least three pairs. It is enough to have one led pair with distinct length-slope value in the OW for the location estimation.

The maximum number of led pairs having distinct length-slope values for a 72×72 grid is 10224 ($2N_1N_2 - N_1 - N_2$; Section 6). Therefore, 6196 (10224–4028) of them are unused. Since the WRC is capable of only tracking the area within its OW, local led pairs are important for our system. There exist 423 led pairs that fit in an OW (local pairs). Specifically, 180 of them have distinct length-slope values. The maximum number of led pairs having distinct length-slope values for a 12×9 grid is 195 ($2N_1N_2 - N_1 - N_2$; Section 6). Therefore, 15 (195–180) of them are unused. The ratio of the number of unused pairs to the maximum number of led pairs having distinct length-slope values is $15/195 = 0.07$. This shows that the led placement produces local pairs having distinct length-slope values almost close to the theoretical limit.

7.2. Distances between the Led Pairs with Equal Length and Slope

Having analyzed the frequency of the led pairs having equal length and slope values, let's see how far these led pairs are from each other. In all led placement methods for large grids, there exist led pairs having equal lengths. When the slope information is also used as another attribute, a greater number of pairs can be differentiated. The distances among the led pairs having equal slope-length values are important for the accuracy of our localization system. If these led pairs are far from each other, they can be easily differentiated using other leds around those pairs, resulting in more accurate location awareness. It turns out that the number of the led pairs having equal slope-length values fitting in the same OW is low.

We analyze the distances between the led pairs having equal slope-length values with an example. Figure 18(a) shows the minimum distances among led pairs having the same length that fit in a 12×9 OW with the led placement produced by the CRS-LP method for a 72×72 grid. In the figure, the led pairs are classified by their lengths and the x-axis shows these classes. Then, within each class, the *minimum distance* among the distances produced by all the pairs in the same class is plotted as a point in the figure. The maximum length that fits a 12×9 OW, 13.6 (the length of the diagonal of a



(a) pair 1 (b) pair 2 (c) pair 3 (d) pairs 4 and 5

Fig. 19. Led pairs having equal slope-length values within an OW for solution of a 72×72 grid, CRSLP, 12×9 OWs.

12×9 OW $= (12-1)^2+(9-1)^2 = 13.6$), is depicted with the horizontal line. Figure 18(b) includes slope as a second attribute that helps to further differentiate the led pairs. This figure can be interpreted as follows. All the led pairs are classified by their lengths, and the x-axis shows these classes. Then, within each class, the led pairs having equal slope values are clustered, and the minimum distance among the distances produced by all the led pairs having equal slope-length values is drawn as a point in the figure.

Each point in the figures can be considered a representation of a link which connects two led pairs. A *link* is the line segment whose end-points are the points on each led pair that are the closest, and the length of a link is the distance from the end-points. There are 350 links (not only the links with minimum lengths but all the links) where the length of the links is smaller than 13.6 when only length is used in order to differentiate led pairs. In addition to the length of an led pair, when slope is also used as a second attribute to differentiate them, there are 90 links connecting two led pairs having equal slope-length values where the length of the links is smaller than 13.6. There are some led pairs having equal slope-length values which can fit in the same OW. The circles in Figure 18(b) represent the links connecting two led pairs having equal slope-length values both fitting in the same OW. For three of these five links, they are on different regions of the grid, and the led pairs constituting each link share a common point so their distances are in between 0 (the first three circles from left to right in Figure 18(b)). The other two links can both actually fit in the same OW. These led pairs are the edges of a parallelogram. Figures 19(a) through 19(d) depict the same pairs that can fit in an OW. Using techniques such as dead reckoning, these pairs can be differentiated easily, as shown in a simulation in Section 8.

7.3. Practical Hurdles

Our system is capable of overcoming the practical hurdles including faulty leds, imprecise led placement, misplaced WRC, and skew ceiling. Since we guarantee *at least* two IR leds in the OW at any time, three or more leds exist in most of the OWs. As a result, if a few leds stop working, there would still exist sufficient leds in the OW. In case there are not at least two leds in the OW as a result of failed leds, the dead reckoning technique is used with the latest robot location computed and the inertial sensors. In this case, the error propagates until the WRC detects enough leds in its OW to estimate the robot location. The error propagation would be minimal unless a significant amount of leds fail.

The system is not required to have a precise led placement to some degree. A data structure used for efficient lookup for the real robot position is constructed. The IR pairs are first sorted according to their length, and then sorted according to their slope values. Assume that the led pairs are not placed precisely. Then, the pairs with imprecise slope and length values read from the IR camera are looked up in the data structure. Since the values in the data structure are precise, whereas the values read from the IR camera are not, the lookup process involves an approximate search within a meaningful interval.

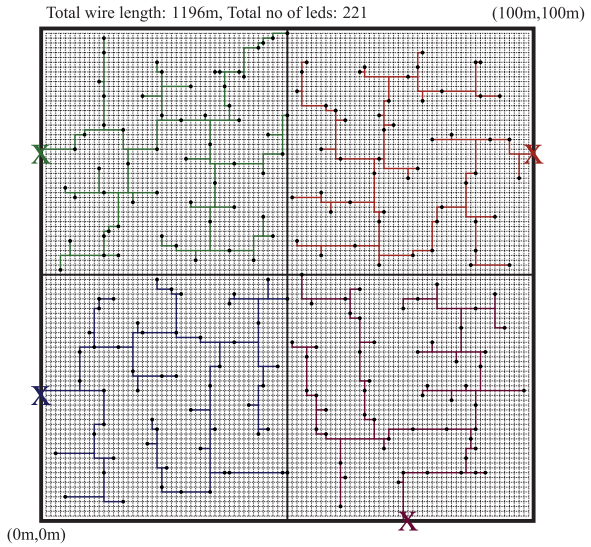


Fig. 20. IR led wiring: four regions.

Instead of looking up the exact values, proper interval values for lengths and slopes are introduced for the approximate search. This approach might increase the number of matching candidates. However, the IR camera detects more than two IR leds most of the time, which means more than one led pair is detected in the OW. Each pair is looked up in the data structure. The matching pair candidates are filtered according to their relative locations and orientations. For example, if the location of a candidate pair is far away from the latest computed location of the robot, that pair is eliminated. Moreover, the relative orientations of the matching pairs help to reduce the candidate pair set. As a result, the real location of the robot is resolved through analyzing multiple pairs.

A misplaced WRC or a skewed ceiling causes deformations on the led locations read from the WRC. The use of the data structure and the approximate search overcome the deformations, resulting in feasible robot location estimations. We include erroneous placements of the WRC in our simulations and show that the system still yields satisfactory robot location estimations.

An array of IR leds can be used for the deployment of the leds. Once the leds are placed on the appropriate cells in the array, it can be mounted on the ceiling. We estimate the length of the required cable roughly using the Steiner Minimal Tree (SMT), where a given set of points is to be connected using minimum total wirelength. Since an array of leds is used, rectilinear distance (the sum of the horizontal and vertical distance between two points) is considered. For example, Figure 20 shows the case where the whole area, which is larger than a football field, is divided into four regions and the IR leds residing in a region are energized using the wall outlet in the corresponding region (uses CRS-LP $n = 100$ placement). The dots represent the IR leds and the "X" character represents the wall outlet. The total wirelength required is 1196m, costing \$461 (RadioShack rates), and the cost of electricity is roughly \$0.48 for a month when the rates of the utility company in San Antonio Texas (cps energy) are used.

8. SIMULATION

We conducted simulations in order to see the effectiveness of our system. The simulation is written in *C++* and we have used the Computational Geometry Algorithms Library, CGAL [CGAL 2014]. The random waypoint model is used for the ME (Mobile

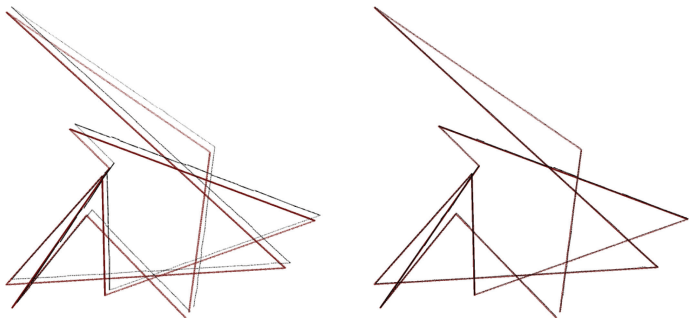
Element)'s path. Ns2 (network simulator) [McCanne and Floyd 2014] has a built-in random waypoint model generator, and the paths are generated using this tool. In simulations, the frequency of the WRC is 50 Hz. In other words, the WRC reads the led locations 50 times a second. Moreover, the orientation of the WRC is axis aligned and does not change over the course of the simulations.

In the perfect system, the WRC should be placed vertically pointing out to the ceiling. We considered erroneous placements of the WRC (the WRC might have been misplaced so that there are a few degrees between the normal of the floor and the WRC). A 5° error can be realized by the human eye. The WRC is straightened if one realizes that it is crooked. Therefore, we conducted the simulations with WRC error up to 5°.

A map of IR leds, which was generated using one of the schemes we discussed, is fed to the robot. A data structure that is used for efficient lookup for the real position is constructed. The pairs are first sorted according to their length and then sorted according to their slope values. The lengths of the pairs are sorted and the pairs having the same length are merged according to their slope values. We assume that the initial position of the ME is known, although it is not required (if the given map has unique led pairs, the ME finds its initial position immediately; otherwise, if there is more than one candidate for the initial position, that position is resolved eventually as the ME moves). The ME reads the led locations in its OW. Then, for each pair in its OW, it finds the corresponding length and slope values. Using these length and slope values, the data structure is searched for the real locations of the leds that constitute the pair. Since the led locations read by the IR camera are deformed due to the visual angle in 3D space, an approximate search within a meaningful interval is applied. Instead of looking up the exact values, proper interval values for lengths and slopes are introduced for the approximate search. This approach might increase the number of matching candidates. However, the IR camera detects more than two IR leds most of the time, which means that more than one led pair is detected in the OW. Each pair is looked up in the data structure. The matching pair candidates are filtered according to their relative locations and orientations. For example, if the location of a candidate pair is far away from the latest computed location of the robot, that pair is eliminated. Moreover, the relative orientations of the matching pairs help to reduce the candidate pair set. After processing all the led pairs in the OW, the location that is the closest to the initial location is chosen as the current location.

The setup for the simulation considers the 3D space as well as the visual angle. The vision of the IR camera is set as a rectangular pyramid where the angles between the opposite triangular faces are 45° and 34.5°. The height of the ceiling is a parameter of the simulation. An imaginary 1024×768 unit-square-sized OW is calculated using the value of the height of the ceiling and the angles between the opposite triangular faces. Depending on the height of the ceiling, the imaginary OW may reside either below or above the ceiling. To mimic the possible deformation, the location of the intersection of the OW plane and the line passing through the IR camera and an IR led is used as the return value of the IR camera. We further analyze the case where the WRC is misplaced on the robot. The misplacement of the WRC causes more deformation. Our simulation results show that the system provides feasible accuracy in all cases.

When the WRC is placed perfectly (vertical to the floor), the robot finds its position with an error of less than a centimeter (cm). When an error of $\alpha > 0$ degrees in the x direction and $\beta > 0$ degrees in the y direction in 3D space is introduced to the placement of WRC, two problems will arise. First, the deformation caused by the visual angle increases. The approximate search finds the matching led pairs most of the time. Dead Reckoning (DR) is applied when the approximate search does not return any matching pairs. DR is the process of calculating the position of a moving object based on its previous location, velocity, and acceleration. DR introduces cumulative errors;



(a) E_x:5°, E_y:5° (uncalibrated) (b) E_x:5°, E_y:5° (calibrated)

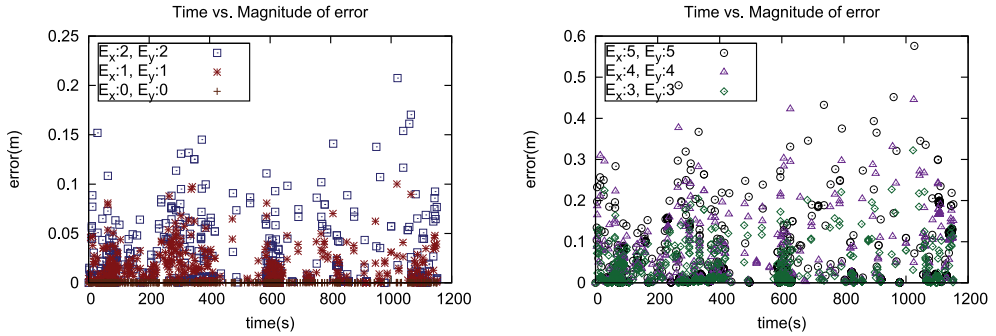
Fig. 21. CRSLP 100×100.

however, the number of DR calls is minimal in our system, and the error resulting from DR is reset whenever the ME is able to find its location using the IR leds. The number of DR calls is minimal, as we discuss later. The second problem caused due to the misplacement of the WRC comes from the error in the placement of the WRC. An absolute error of magnitude of about $h \times \sqrt{tan^2(\alpha) + tan^2(\beta)}$, where h is the height of the ceiling, is introduced to the location that the ME finds. Figure 21(a) shows the real path the ME takes and the path that ME figures out. The figure is the output from a simulation which uses the IR map generated by the CRSLP method with $n = 100$ (100×100). The height of the ceiling is 15m. The simulation time is 20 minutes. The speed of the ME is not constant, and it is at most 2m/s. The coverage area is 100m × 100m. The WRC is placed with an error of 5° in the x direction and with an error of 5° in the y direction. The absolute error can be easily seen in the figure.

The absolute error, $h \times \sqrt{tan^2(\alpha) + tan^2(\beta)}$, is not acceptable for α and β values such as 5° (the magnitude of the error is about 185cm when $\alpha = 5°$, and $\beta = 5°$). Therefore we need a *calibration scheme*. We applied the simplest calibration method assuming that the placement of the WRC does not change during the course of the ME's travel (α and β values are constant). The ME is placed at a known position P_k. The ME calculates its position, P_{ME}. Then, the ME finds the vector in between P_k, and P_{ME}. Using this vector and the height of the ceiling, the ME computes the virtual α, V_α, and the virtual β, V_β values trivially. V_α, and V_β values do not have to match the exact values of α and β, although they are very close to the α and β values. After finding V_α and V_β values, the error vector $(h \times tan(V_\alpha), h \times tan(V_\beta))$ is added to the final location that is computed by the robot for the rest of the simulation. Figure 21(a) shows the calibrated version of the simulation.

The error on the location of the ME is minimized using the preceding calibration method. Figures 22(a) and 22(b) show the magnitude of the errors induced throughout the simulation with CRSLP map($n = 100$), $h = 15m$, and simulation time 1200s (20 minutes). Figure 22(a) compares the error on the location for α and β values from 0° to 2° and Figure 22(b) compares the error on the location for degrees from 3° to 5°. When the error on the WRC increases, the error on the location of the ME increases, as expected. For example, the *average* errors on the location are about 1.1mm, 1.7cm, 3.5cm, 5.3cm, 7.5cm, and 10cm when the α and β values considered are (0°, 0°), (1°, 1°), (2°, 2°), (3°, 3°), (4°, 4°), and (5°, 5°), respectively.

Table III demonstrates the percentage of the number of the DR calls to the number of all location calculations. As seen from the table, the percentages have low values, meaning that our system does not require much DR. Another result from the table is



(a) CRSLP 100x100: calibrated, errors 0-1-2 (b) CRSLP 100x100: calibrated, errors 3-4-5

Fig. 22. Accuracy of our system.

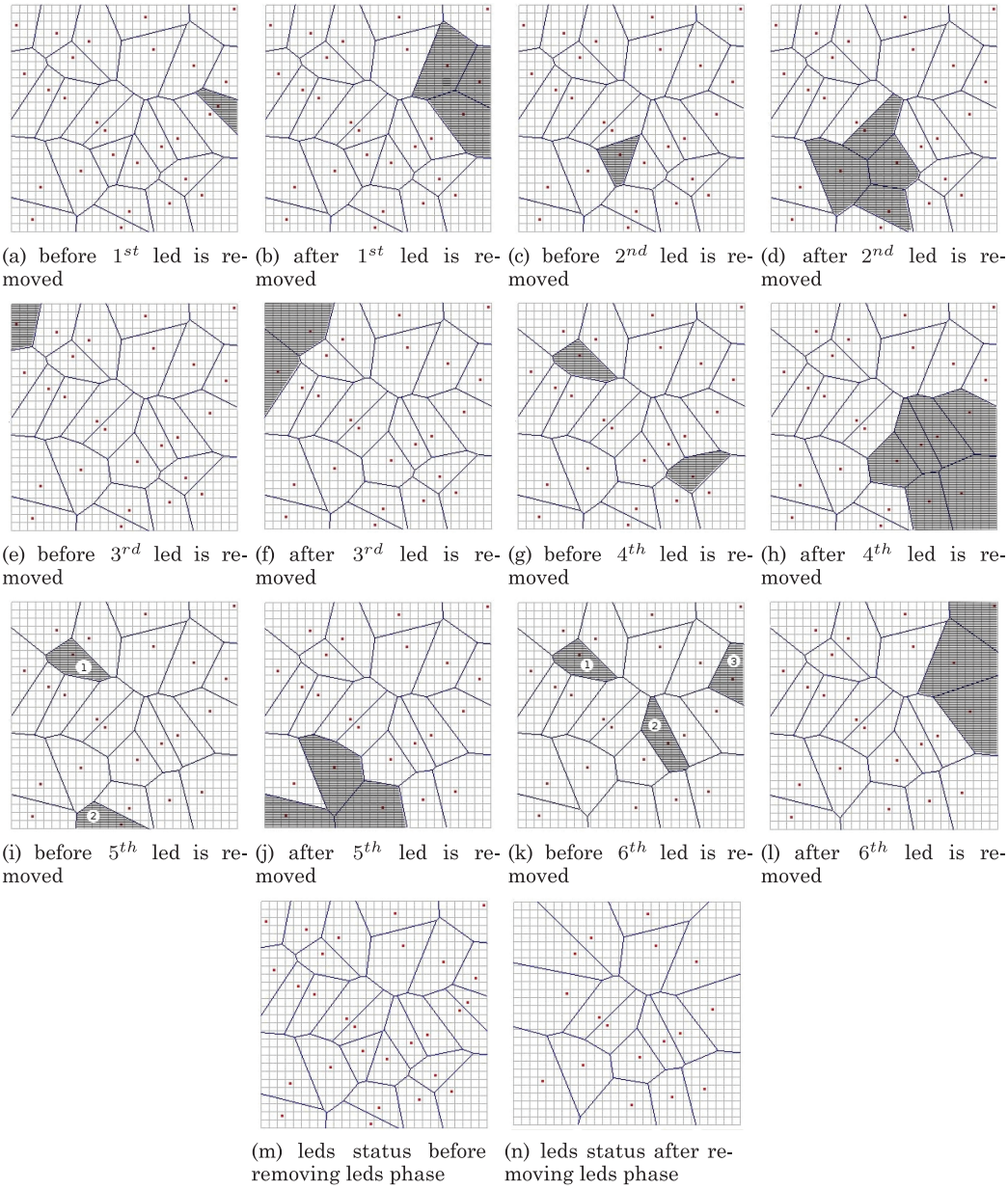
Table III. Percentage of Dead Reckoning Calls, Frequency: 50; Calibrated

Error in degrees	DR calls percentage (CRSLP100×100)	DR calls percentage (CILP31×31)
$E_x : 0, E_y : 0$	$0/57636 = 0$	$0/44995 = 0$
$E_x : 1, E_y : 1$	$11/57636 = 19 \times 10^{-3}$ %	$9/42903 = 21 \times 10^{-3}$ %
$E_x : 2, E_y : 2$	$41/57636 = 71 \times 10^{-3}$ %	$28/42903 = 65 \times 10^{-3}$ %
$E_x : 3, E_y : 3$	$75/57636 = 130 \times 10^{-3}$ %	$42/42903 = 98 \times 10^{-3}$ %
$E_x : 4, E_y : 4$	$126/57636 = 218 \times 10^{-3}$ %	$73/42903 = 170 \times 10^{-3}$ %
$E_x : 5, E_y : 5$	$189/57636 = 327 \times 10^{-3}$ %	$93/42903 = 216 \times 10^{-3}$ %

that the number of the DR calls increases as the error on the WRC gets larger. This is reasonable, since the ME will have difficulty matching the erroneous IR led positions with the IR map it has. The higher the error, the greater the number of DR calls would be required. With the help of other sensors such as inertial sensors, the DR algorithm would work better in the real implementation of our system, and the accuracy on the location would be higher in a real implementation.

9. CONCLUSION

We propose a low-cost and simple location management system for robots using the WRC and IR leds. In the proposed schemes, the WRC is placed on the mobile robot and multiple infrared leds are placed on the ceiling irregularly. The mobile robot finds its current position using the IR leds in its current window. Since the WRC can track up to four infrared leds at any time, the patterns observed are limited. The relative position of the infrared leds with respect to each other is used to find the global position. We analyze the problem theoretically and show that there exist limitations for covering large areas. We also present how to overcome these limitations. The methods for irregular placement of the leds consist of LP based methods, a Costas-array-based method, and an intelligent metric-based random method. The LP-based method, CILP, provides optimum solutions for small areas where $n \leq 32$, but it is not scalable. Another LP-based method, $\text{ILP}_{\text{rowcol}}$, produces good results up to $n = 54$; however, it is not scalable either. BILP gives a lower bound on the number of leds required. Incremental LP is a fast and scalable method producing good results, but its solutions contain the greatest number leds among all LP-based approaches. The Costas array-based method, CRS-LP, is the most promising one for large coverage areas since it is scalable. The proposed schemes can handle rotations and can compute the current location of the robot efficiently and accurately using the distances among the detected IR leds. Finally, we have simulation results supporting the accuracy of the system.



(a) before 1^{st} led is removed (b) after 1^{st} led is removed (c) before 2^{nd} led is removed (d) after 2^{nd} led is removed

(e) before 3^{rd} led is removed (f) after 3^{rd} led is removed (g) before 4^{th} led is removed (h) after 4^{th} led is removed

(i) before 5^{th} led is removed (j) after 5^{th} led is removed (k) before 6^{th} led is removed (l) after 6^{th} led is removed

(m) leds status before removing leds phase (n) leds status after removing leds phase

Fig. 23. Illustration of removing-leds phase of CRS-LP algorithm.

APPENDIXES

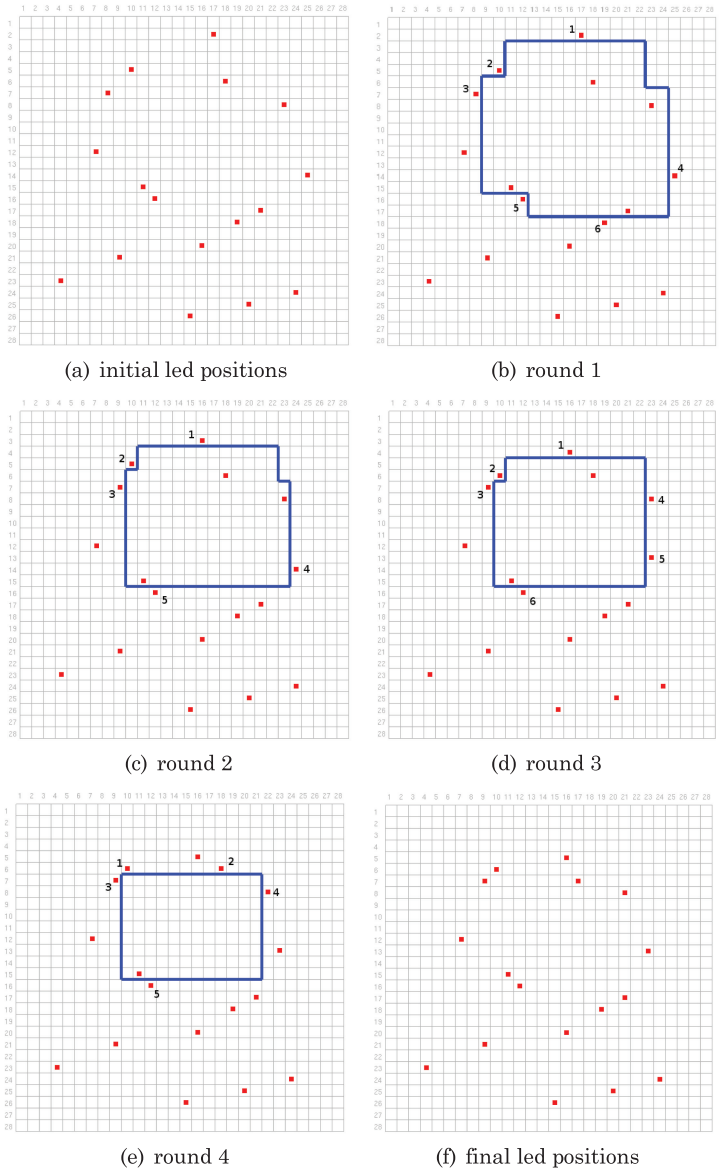
A. EXAMPLE OF REMOVING LEDS STEP OF CRS-LP ALGORITHM

Figures 23(a) through 23(n) demonstrate an example of the *removing-leds* step. A 28-Costas array generated in the Costas generation step is used. The Costas array is constructed using the Welch method for $p = 29$ with the primitive element 3 in the finite field, $GF(29)$. The Costas array augmented with its Voronoi diagram is shown in Figure 23(m). In Figure 23(a), the Voronoi diagram of the initial point set is shown

together with the Voronoi cell having the smallest area; see shaded region. The point in the shaded region is the candidate led to be removed. Line 8 of Algorithm 2 checks whether the candidate led is an *extra led* or not. The metric doesn't change after the removal, so the candidate led is an *extra led*, and it is removed. Figure 23(b) shows the new Voronoi diagram after the led is removed. The shaded area demonstrates the change in the Voronoi diagram. The second and the third leds are removed similarly, as shown in Figures 23(c) through 23(f). Before the forth led is removed, two Voronoi cells have the same smallest area, shown as shaded regions in Figure 23(g). The function *DoesMetricChange* returns true for the candidate led that is towards the left upper part of the grid since the value of the metric is 0 before the removal and 3 after the removal. Therefore, this candidate led cannot be removed. However, the other candidate led having the same area as the first one is an extra led and it is removed. Figure 23(h) shows the new Voronoi diagram after the fourth led is removed. The shaded area demonstrates the change in the Voronoi diagram. For the fifth led to be removed, the smallest Voronoi cell is demonstrated with the number 1 in Figure 23(i), and the point in the cell is the first candidate point. However, it is not an *extra led* and we continue with the next smallest Voronoi cell, which is the shaded region represented by the number 2 in Figure 23(i). The second candidate is an *extra led*, since the metric doesn't change before and after its removal, and it is removed. Finally, this step of the CRS-LP algorithm terminates after removing a total of 11 leds, resulting in an IR led placement pattern having 17 leds, as shown in Figure 23(n). The slope length among the pairs still remains unique.

B. EXAMPLE OF SLIDING-LEDS STEP OF CRS-LP ALGORITHM

Figure 24(a) shows the leds' positions after the *removing-leds* step in the example in Section 4.1.2. Since a 28×28 grid is fairly small, sliding method 1 works perfectly without requiring the next phase of the CRS-LP algorithm. Figures 24(a) through 24(f) show the steps for the *sliding leds* algorithm using sliding method 1. Figure 24(b) demonstrates the union of OWs having 0 or 1 led with the polygon, and the candidate leds to slide with number annotations just before sliding the first led. There are 13 OWs having less than two leds in the union. After the first iteration, the candidate led annotated with number 1 is moved to the location $(16, 3)$ from the location $(17, 2)$; the candidate led annotated with number 3 is moved to the location $(9, 7)$ from the location $(8, 7)$; and the candidate led annotated with number 4 is moved to the location $(24, 14)$ from the location $(25, 14)$. After the first iteration, seven OWs having 0 or 1 led left, as well as the union and the new candidate leds, are shown in Figure 24(c). In second round, Figure 24(c), the candidate led annotated with number 1 is moved to the location $(16, 4)$ from the location $(16, 3)$; the candidate led annotated with number 2 is moved to the location $(10, 6)$ from the location $(10, 5)$; and the candidate led annotated with number 4 is moved to the location $(23, 13)$ from the location $(24, 14)$. After the second iteration, four OWs having 0 or 1 led left, as well as the union and the new candidate leds, are shown in Figure 24(d). In the third round shown in Figure 24(d), the candidate led annotated with number 1 is moved to the location $(16, 5)$ from the location $(16, 4)$; the candidate led annotated with number 4 is moved to the location $(22, 8)$ from the location $(23, 8)$. After the third iteration, one OW having 1 led left as well as the union and the new candidate leds are shown in Figure 24(e). In the fourth round depicted in Figure 24(e), the candidate led annotated with number 2 is moved to the location $(17, 7)$ from the location $(18, 6)$; and the candidate led annotated with number 4 is moved to the location $(21, 8)$ from the location $(22, 8)$. Finally, Figure 24(f) shows the final led positions without requiring further action, since every OW has at least 2 leds, and all the led pairs have unique slope-length values.



(a) initial led positions (b) round 1

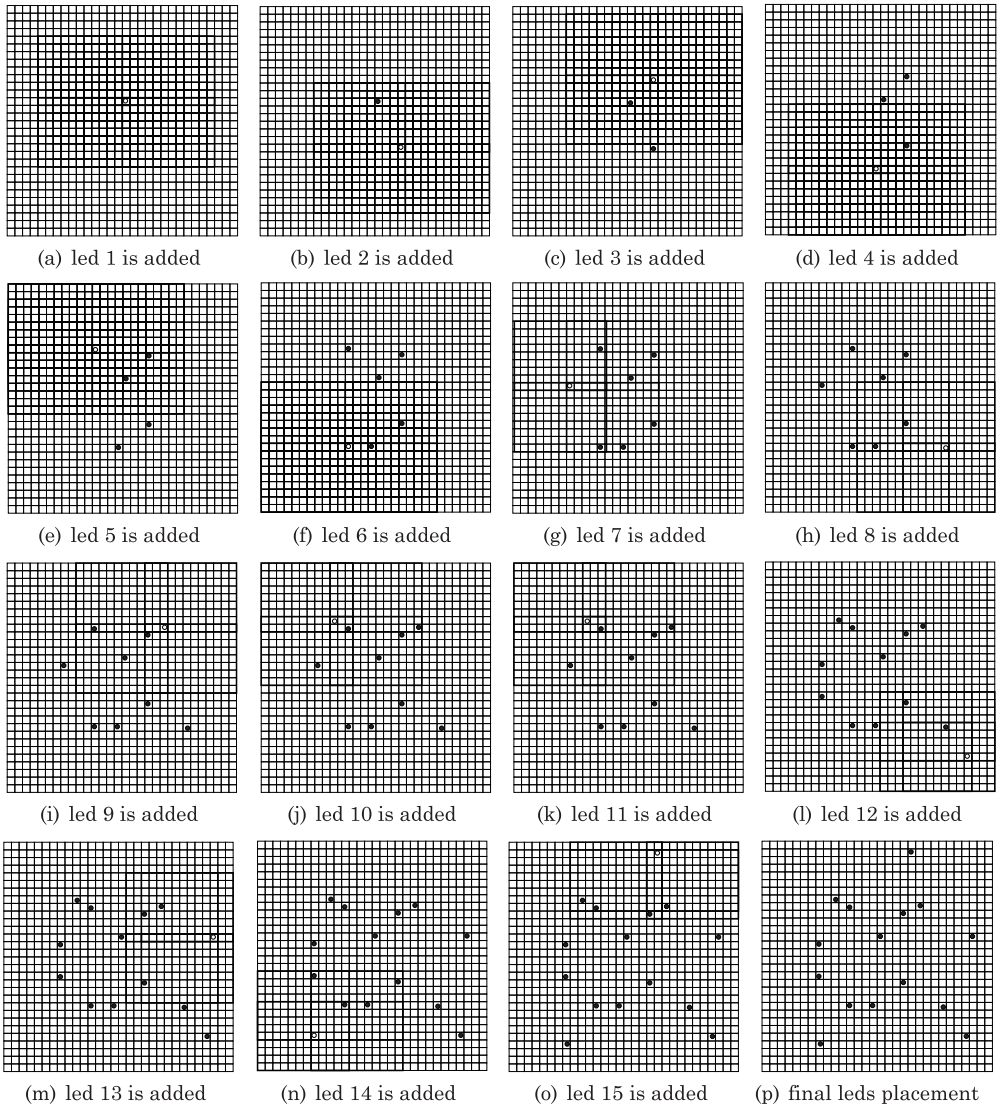
(c) round 2 (d) round 3

(e) round 4 (f) final led positions

Fig. 24. Illustration of sliding-leds (method 1) phase of CRS-LP algorithm.

C. EXAMPLE OF MBR ALGORITHM

Figures 25(a) through 25(o) show the state of the system after each IR led is added, and Figure 25(p) shows the final led placement when MBR($k = 2$, X=12, Y=9, RowSize=30, ColumnSize=30, thresConst=0.9, repetition=100) is run. MBR returns a set of leds guaranteeing two IR leds in each OW of size 12×9 in a 30×30 grid with a threshold constant value of 0.9 and a repetition value of 100. The first led is added to the location (16, 13). The vicinity of (16, 13) is shown in Figure 25(a). There are 108 OWs in the vicinity of (16, 13) and all of them had zero leds in them before adding led 1. Therefore, $metric\_prev = 2 \times 108 + 1 \times 0 = 216$, according to Eq. (5). After adding led 1, there



(a) led 1 is added (b) led 2 is added (c) led 3 is added (d) led 4 is added

(e) led 5 is added (f) led 6 is added (g) led 7 is added (h) led 8 is added

(i) led 9 is added (j) led 10 is added (k) led 11 is added (l) led 12 is added

(m) led 13 is added (n) led 14 is added (o) led 15 is added (p) final leds placement

Fig. 25. Illustration of metric-based random led placement (MBR).

will be no OWs having zero IR leds in them and all of the 108 OWs have one led in them, resulting in $metric\_current = 2 \times 0 + 1 \times 108 = 108$. The value of *threshold* and *threshold_original* is 108. The metric difference is 108 (216 − 108), so the *if statement* in line 9 of Algorithm 4 holds and the IR led is added to the location, (16, 13). The location of the next point added is (19, 19). Figure 25(b) demonstrates the vicinity of (19, 19). The empty circle in the figure represents the most recently added led, and solid circles represent the previously added leds. In Figure 25(b), there are 108 OWs in the vicinity of (19, 19). Here, 81 of the OWs contain zero and 27 of the OWs have 1 led in them before led 2 is added. Therefore $metric\_prev = 2 \times 81 + 1 \times 27 = 189$. After adding led 2, there will be no OWs having zero IR leds in them, 81 of the OWs in the vicinity have one led in them, and 27 OWs have two IR leds in them, resulting in

metric_current $= 2 \times 0 + 1 \times 81 = 81$. The value of *threshold* and *threshold_original* is 108. The metric difference is 108 (189 − 81), so the *if statement* in line 9 of Algorithm 4 holds and the IR led is added to the location (19, 19). When the led to be added has a location closer to the edges of the grid, the number of OWs in the vicinity of the led's location is less than $X \times Y$. For example, when the twelfth led is added at location (27, 26) as shown in Figure 25(l), there are 20 OWs in the vicinity of (27, 26). Before the led is added, 12 of the OWs in the vicinity have one led and none of them has zero leds in it, resulting in *metric_prev* $= 2 \times 0 + 1 \times 12 = 12$. After the led is added, there are no OWs containing zero or one leds in them, resulting in *metric_current* $= 2 \times 0 + 1 \times 0 = 0$. The value of *threshold* is 12 at this point. The metric difference is 12 (12 − 0), so the *if statement* in line 9 of Algorithm 4 holds and the IR led is added to the location (27, 26). The final point is added to the location (20, 2) as shown in Figure 25(o). After this final point is added, there are no OWs having less than two IR leds in them left. Therefore, the function *has0s1s* in line 10 of Algorithm 4 returns false and the algorithm terminates, presenting the system shown in Figure 25(p) guaranteeing at least two IR leds in each 12 × 9 OW in a 30 × 30 grid.

REFERENCES

P. K. Agarwal, E. Ezra, and M. Shair. 2009. Near-linear approximation algorithms for geometric hitting sets. In *Proceedings of the 25th Annual Symposium on Computational Geometry (SCG'09)*. ACM Press, New York, 23–32.

A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, et al. 2004. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Comput. Netw.* 46, 5, 605–634.

A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, et al. 2005. Exscal: Elements of an extreme scale wireless sensor network. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 102–108.

G. Ausiello, A. Datri, and M. Protasi. 1980. Structure preserving reductions among convex optimization problems. *J. Comput. Syst. Sci.* 21, 1, 136–153.

P. Bahl and V. Padmanabhan. 2000. Radar: An in-building RF-based user location and tracking system. In *Proceedings of the 19th Annual joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*. 775–784.

M. Betke and L. Gurvits. 1997. Mobile robot localization using landmarks. *IEEE Trans. Robot. Autom.* 13, 2, 251–263.

N. Bulusu, J. Heidemann, and D. Estrin. 2000. GPS-less low-cost outdoor localization for very small devices. *IEEE Person. Comm.* 7, 5, 28–34.

Bytelight. 2013. http://www.bytelight.com/.

CGAL. 2014. Computational geometry algorithms library. http://www.cgal.org.

B.-S. Choi, J.-W. Lee, J.-J. Lee, and K.-T. Park. 2011. A hierarchical algorithm for indoor mobile robot localization using RFID sensor fusion. *IEEE Trans. Industr. Electron.* 58, 6, 2226–2235.

H. Chung, L. Ojeda, and J. Borenstein. 2001. Accurate mobile robot dead-reckoning with a precision calibrated fiber-optic gyroscope. *IEEE Trans. Robot. Autom.* 17, 1, 80–84.

J. P. Costas. 1984. A study of a class of detection waveforms having nearly ideal range-doppler ambiguity properties. *Proc. IEEE* 72, 8, 996–1009.

M. Demirbas, O. Soysal, and A. S. Tosun. 2007. Data salmon: A greedy mobile basestation protocol for efficient data collection in wireless sensor networks. In *Proceedings of the 3rd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'07)*. 267–280.

M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. 2001. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Trans. Robot. Autom.* 17, 229–241.

A. Eliazar and R. Parr. 2003. Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*. Morgan Kaufmann, San Fransisco, 1135–1142.

S. K. Ganjugunte. 2011. Geometric hitting sets and their variants. Ph.D. thesis, Duke University.

M. R. Garey and D. S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NPCompleteness*. W.H. Freeman and Co., New York.

I. Getting. 1993. The global positioning system. *IEEE Spectrum* 30, 12, 36–38.

M. Golfarelli, D. Maio, and S. Rizzi. 2001. Correction of dead-reckoning errors in map building for mobile robots. *IEEE Trans. Robot. Autom.* 17, 1, 37–47.

S. W. Golomb. 1984. Algebraic constructions for Costas arrays. *J. Comb. Theory, Ser. A*, 13–21.

Y. Gu, D. Bozdag, E. Ekici, F. Ozguner, and C. Lee. 2005. Partitioning based mobile element scheduling in wireless sensor networks. In *Proceedings of the IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON).* 386–395.

D. Hahnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose. 2004. Mapping and localization with RFID technology. In *Proceedings of the International Conference on Robotics and Automation (ICRA'04),* vol. 1. 1015–1020.

A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. 1999. The anatomy of a context-aware application. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99).* 59–68.

D. Jea, A. Somasundara, and M. Srivastava. 2005. Multiple controlled mobile elements (datamules) for data collection in sensor networks. In *Proceedings of the 1st IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'05).* 244–257.

W. Jeong and K. M. Lee. 2005. CV-slam: A new ceiling vision-based slam technique. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05).* 3195–3200.

P. Juang, H. Oki, and Y. Wang. 2002. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Proceedings of the 10th International Conference on Architectural Support for Programming languages and Operating Systems (ASPLOS'02).* 96–107.

Led Locations. 2014. http://www.my.cs.utsa.edu/.btas/wiijournal.html.

J. C. Lee. 2008. Hacking the Nintendo wii remote. *IEEE Pervas. Comput.* 7, 3, 39–45.

M. Ma and Y. Yang. 2007. Sencar: An energy-efficient data gathering mechanism for large-scale multihop sensor networks. *IEEE Trans. Parallel Distrib. Syst.* 18, 10, 1476–1488.

A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. 2002. Wireless sensor networks for habitat monitoring. In *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02).* 88–97.

S. McCanne and S. Floyd. 2014. NS network simulator. http://www.isi.edu/nsnam/ns/.

N. Megiddo and K. J. Supowit. 1984. On the complexity of some common geometric location problems. *SIAM J. Comput.* 13, 1, 182–196.

M. Miah and W. Gueaieb. 2010. Indoor robot navigation through intelligent processing of RFID signal measurements. In *Proceedings of the International Conference on Autonomous and Intelligent Systems (AIS'10).* 1–6.

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. 2002. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*. 593–598.

N. H. Mustafa and S. Ray. 2009. Ptas for geometric hitting set problems via local search. In *Proceedings of the 25th Annual Symposium on Computational Geometry (SCG'09).* ACM Press, New York, 17–22.

A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. 2000. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* 2nd Ed. Wiley.

N. Priyantha, A. Chakraborty, and H. Balakrishnan. 2000. The cricket location support system. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom'00).* 32–43.

R. C. Shah, S. Roy, S. Jain, and W. Brunette. 2003. Datamules: Modeling a three-tier architecture for sparse sensor networks. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*. 30–41.

T. Small and Z. Haas. 2003. The shared wireless infestation model - A new ad hoc networking paradigm (or where there is a whale, there is a way). In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03).* 233–244.

A. Somasundara, A. Ramamoorthy, and M. Srivastava. 2004. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*. 296–305.

K.-F. Ssu, C.-H. Ou, and H. Jiau. 2005. Localization with mobile anchor points in wireless sensor networks. *IEEE Trans. Vehic. Technol.* 54, 3, 1187–1197.

T. Stoyanova, F. Kerasiotis, A. Prayati, and G. Papadopoulos. 2007. Evaluation of impact factors on RSS accuracy for localization and tracking applications. In *Proceedings of the 5th ACM International Workshop on Mobility Management and Wireless Access (MobiWac'07)*. 9–16.

M. Strasser, A. Meier, K. Langendoen, and P. Blum. 2007. Dwarf: Delay-aware robust forwarding for energy-constrained wireless sensor networks. In *Proceedings of the 3rd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'07)*. Springer, 64–81.

B. Tas, N. Altiparmak, and A. Tosun. 2009. Low cost indoor location management system using infrared leds and wii remote controller. In *Proceedings of the 28th IEEE International Performance Computing and Communications Conference (IPCCC'09)*. 280–288.

G. Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, P. Buonadonna, S. Burgess, D. Gay, W. Hong, T. Dawson, and D. Culler. 2005. A macroscope in the redwoods. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys'05)*. 51–63.

C.-C. Tsai. 1998. A localization system of a mobile robot by fusing dead-reckoning and ultrasonic measurements. *IEEE Trans. Instrument. Measur.* 47, 5, 1399–1404.

I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. 2005. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)*. ACM Press, New York, 154–165.

Vishay Semiconductors. 2004. http://www.vishay.com/docs/81108/81108.pdf.

K. Whitehouse, C. Karlof, and D. Culler. 2007. A practical evaluation of radio signal strength for ranging-based localization. *ACM Mobile Comput. Comm. Rev.* 11, 1, 41–52.

G. Xing, T. Wang, Z. Xie, and W. Jia. 2007. Rendezvous planning in mobility-assisted wireless sensor networks. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS'07)*. 311–320.

W. Zhao and M. Ammar. 2003. Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks. In *Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FDTCS'03)*. 308–314.

M. Zmuda, A. Elesev, and Y. Morton. 2008. Robot localization using RF and inertial sensors. In *Proceedings of the IEEE National Aerospace and Electronics Conference (NAECON'08)*. 343–348.

Received October 2012; revised June 2013; accepted July 2013