CFPP: Collision-free Path Planning for Wireless Mobile Sensors Deployment[†]

Ting-Yu Lin[⋆], Hendro Agus Santoso, Chung-An Lin, and Gui-Liu Wang
Department of Electrical and Computer Engineering
National Chiao Tung University

***Abstract*—With the growing popularity of wireless mobile sensors, automated sensors deployment in a smart sensing environment has become practical and feasible. Once the deployment algorithm determines moving destinations (goals) for all sensors, however, the problem of how to schedule collision-free moving paths to reach the goals safely remains largely unaddressed in the wireless sensor networking (WSN) literature. In this paper, we propose a collision-free path planning (CFPP) mechanism, based on geometric formulations and batched movements, to address the sensors deployment problem. Our proposed CFPP mechanism ensures 100% sensors goal reachability, which is critical for most WSN monitoring applications that require sufficient sensing coverage to operate correctly. Performance results show that our CFPP outperforms other existing path-planning mechanisms in terms of computation latency, energy consumption, and sensors reachability (goals reaching success probability).**

***Index Terms*—Mobile sensors deployment, path planning.**

I. Background

Wireless sensor networks (WSNs) have been thriving and attracting significant attention thanks to the advances of micro-electromechanical system (MEMS), sensing technology, and wireless communication. A WSN is widely used for habitat and environmental surveillance, medical application, agricultural assistance, and as solutions to military problems [6], [10], [14]. For surveillance applications, sufficient sensing coverage is essential for a WSN to operate successfully. With the growing prevalence of wireless mobile sensors, automated sensors deployment has become practical and feasible. Once the deployment algorithm produces moving destinations (goals) for all sensors, however, the problem of how to schedule moving paths to reach the goals without collisions remains largely neglected in the sensor networking field.

Traditional path-planning methods in the area of multi-robot systems can be generally classified into three categories: roadmap-based, performing cell decomposition, and applying the concept of artificial potential field, according to the authors in [5], [13]. The roadmap-based methods construct visibility graphs to assist in the path-planning process [15]–[17], in which the resultant moving paths may touch (fixed) obstacles at the vertices or even edges that are considered unsafe. To address this drawback, approaches based on cell decomposition by computing Voronoi diagrams have been introduced to perform path scheduling in a more precise way at the cost of increased computation latency [5]. The third category of path-planning approaches models the obstacles and targets as electrostatic charges interacting with each other, creating a potential field [18]–[20]. The obstacle exerts a repulsive force while the target location has an attractive effect on the robot position. This method enables path planning to be completed in real time. However, as only local properties are considered, the robots may get trapped at local minima or aimless oscillation without reaching their goals. The aforementioned approaches assume one or more fixed obstacles in a multi-robot system, which differs from our mobile sensor networking scene. In this paper, we investigate a sensing system where all sensors are mobile and considered as dynamic objects (rather than fixed obstacles).

Two more closely related path-planning methods to our envisioned scenario that also deal with dynamic objects include ADO [7] and Super A* [12]. In ADO, all robots are prioritized and equipped with omnidirectional cameras (visual sensors) to perform real-time path-planning computations. ADO suffers from the deadlock problem in which some robots are unable to reach their destinations (goal positions). In addition, extra (non-negligible) energy consumption is required due to the constant usage of visual sensors for path calculations in ADO. On the other hand, Super A* modifies the classical A* search algorithm [9] by taking environmental changes into account while robots move. Super A* successfully resolves some deadlock cases that ADO fails to handle. However, the triangular deadlock problem in ADO remains unsolved in Super A*. Moreover, extensive computation latency is required in Super A* due to the high time-complexity nature of the A* search algorithm.

In this paper, we regard all sensors as dynamic objects and propose a collision-free path planning (CFPP) mechanism, based on geometric formulations and batched movements, to address the sensors deployment problem. Our proposed CFPP method incurs little computation latency, moderate energy consumption, and ensures 100% goal reachability. The remainder of this paper is organized as follows. Section II prepares the readers with geometric preliminaries. We present the CFPP protocol details in Section III. Section IV provides performance comparisons in terms of computation latency, energy consumption, and goals reaching success probability. Finally, we conclude the paper in Section V.

[⋆]Corresponding author (E-mail: ting@cm.nctu.edu.tw).

[†]This research was co-sponsored in part by the Ministry of Science and Technology (MOST) of Taiwan under grant number 103-2221-E-009-016 and in part by the Delta-NCTU Cooperation Project.

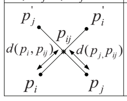
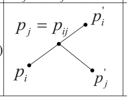
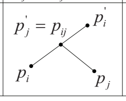
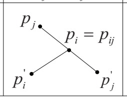
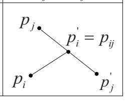
$L_j(p_i)L_j(p_i^{'})<0$ && $L_i(p_j)L_i(p_j^{'})<0$	$L_i(p_j)==0$ && $L_j(p_i)L_j(p_i^{'})<0$	$L_i(p_j^{'})==0$ && $L_j(p_i)L_j(p_i^{'})<0$	$L_j(p_i)==0$ && $L_i(p_j)L_i(p_j^{'})<0$	$L_j(p_i^{'})==0$ && $L_i(p_j)L_i(p_j^{'})<0$
				
Case I	Case II	Case III	Case IV	Case V

Fig. 1. Possible intersection (collision) cases generated by moving paths of any two sensors s_i and s_j, where p_i (p_j) denotes the original position of s_i (s_j) and $p_i^{'}$ ($p_j^{'}$) indicates the physical movement destination for sensor s_i (s_j).

II. Preliminaries

In this paper, we assume the sensor volume is neglected and regarded as a *moving point* on a 2D plane, while every moving path (performed by a sensor) regarded as a *line*. Suppose no two moving paths share the same line (i.e., no path lies in the sub-path of another). We identify the collision cases based on the following geometric theorem.

Theorem 1. *With respect to the line $ax + by + c = 0$ on a 2D plane, points $Q_1(x_1, y_1)$ and $Q_2(x_2, y_2)$ fall in the same side if $(ax_1 + by_1 + c)(ax_2 + by_2 + c) > 0$, in different sides if $(ax_1 + by_1 + c)(ax_2 + by_2 + c) < 0$, while one or both reside(s) exactly on the line if $(ax_1 + by_1 + c)(ax_2 + by_2 + c) = 0$.*

For an arbitrary sensor s_i departing from point p_i (with coordinate (x_i, y_i)) to point $p_i^{'}$ (with coordinate ($x_i^{'}$, $y_i^{'}$)), the moving path can be formulated as a line, denoted as L_i. Similarly, the moving path of another sensor s_j is given as L_j. Define p_{ij} as the *intersection point* of lines L_i and L_j, which can be easily obtained by solving the two line equations. According to Theorem 1, we can now classify five possible intersection (collision) cases for any two sensors s_i and s_j, as illustrated in Fig. 1, where $d(p_i, p_{ij})$ and $d(p_j, p_{ij})$ represent the Euclidean distances from p_i to p_{ij} and from p_j to p_{ij}. **Case I** shows the case in which points p_i and $p_i^{'}$ fall in different sides of line L_j, whereas points p_j and $p_j^{'}$ fall in different sides of line L_i as well. In **Case II**, the departure point p_j of sensor s_j gets in the way of the moving path of s_i, while in **Case IV**, on the contrary, the departure point p_i of sensor s_i blocks the moving path of s_j. **Case III** draws the condition in which the destination point $p_j^{'}$ of sensor s_j lies on the moving path of s_i, whereas **Case V**, on the opposite side, displays the condition that destination point $p_i^{'}$ of sensor s_i falls on the moving path of s_j.

III. Collision-Free Path Planning (CFPP)

In practical sensors deployment, a collision-free moving path scheduling is essential, so that mobile sensors can reach their destinations without colliding with each other. However, the scheduling strategy is non-trivial since various collision cases need be systematically classified and handled/resolved in different ways.

A. Path Planning Strategy

Given the five potential collision (intersection) cases caused by any two moving paths, we establish a colliding set C_i, which includes all sensors whose moving paths intersect with that of s_i, for each sensor. Instead of performing one-time physical movements, we propose to use *batched movements* such that the scheduling complexity can be reduced at the expenses of increased moving latency. Define $order_i$ as the cardinality of set C_i ($order_i = |C_i|$) for sensor s_i, indicating its moving order. We start from performing movements for sensors with the least $order$ value. All sensors with the currently least (smallest) $order$ value are contained in set M_{min_order}. Intuitively, sensors with $order$ value of zero can move simultaneously since no other sensors pose potential colliding sources to them. For any sensor s_i with non-zero $order_i$ value, potential colliding conditions (on per node-pair basis) caused by all members in its C_i set should be analyzed and handled case by case. Specifically, all sensors are divided into moving groups (batches) based on their $order$ values and processed round by round (batch by batch). Sensors in set M_{min_order} are evaluated in the same round. The evaluation and processing details will be provided later in this section. After the evaluations, a subset of M_{min_order} (or probably the whole M_{min_order} set) is determined and all sensors included in the subset are allowed to move simultaneously in the current round. For sensor s_i that has been evaluated and permitted to move, the $tflag_i$ is set $true$, indicating its moving intention. Once the physical movement has been successfully performed by sensor s_i, moving flag $mflag_i$ is set $true$ and s_i is removed from the consideration list. All $order$ values for the remaining sensors (physical movements not performed yet) should be refreshed, and the batched scheduling procedure starts over accordingly.

Now, we elaborate on the evaluation procedures for determining a set of movable sensors in a single round (batch). Based on the idea of batched movements, we regard all sensors with the currently minimum $order$ value as a potential moving batch and include them in set M_{min_order}. We then analyze all members in set M_{min_order} one by one to determine their moving possibilities. In our design, we start the evaluation from sensor with the smallest ID, say s_1, and identify all possible collision cases caused by members in its colliding set C_1. For any two sensors s_i and s_j with moving orders $order_i$ and $order_j$, the previously five collision cases can be further classified into ten cases according to the relationship of $order_i$ and $order_j$. Suppose $s_i \in M_{min\_order}$, $s_j \in C_i$, and $order_i = order_j$, we term the five collision cases as **Case S-I**, **Case S-II**, **Case S-III**, **Case S-IV**, and **Case S-V**, where 'S' indicates that sensors s_i and s_j are potentially scheduled to move in the "same" round due to equal $order$ value. On the other hand, if $order_i < order_j$ (note that $order_i > order_j$ is not possible since $s_i \in M_{min\_order}$), we define another five collision cases as **Case D-I**, **Case D-II**, **Case D-III**, **Case D-IV**, and **Case D-V**, where 'D' means s_i and s_j are potentially scheduled to move in "different" rounds due to their unequal $order$ values. In each potential collision case, on detecting a colliding possibility, s_i tries to resolve the collision by adjusting/prolonging the waiting time T_j or increasing the moving speed V_j of sensor s_j. Originally all waiting times are set to zero, and moving speeds all set at a constant velocity V. If the adjustment (on either waiting time or moving speed) is successful, the colliding possibility is eliminated and s_i moves on to evaluate collision cases with other members in C_i. To avoid repeated adjustments on a single sensor, in our design, each sensor is allowed to be adjusted (either on waiting time or moving velocity) *once*. In addition, s_i itself cannot be adjusted by other sensors in set M_{min_order} that are evaluated *after* it, if s_i is indeed scheduled to move in the current round. We keep track of the adjustment possibility for sensor s_i by the $dirty_i$ bit, implying adjustable if set $false$ and not adjustable if set $true$. When s_i intends to resolve a collision by adjusting another sensor with $dirty$ bit set $true$, the adjustment is prohibited and s_i is not allowed to move in the current round ($tflag_i$ set to $false$), since the collision remains. Only when all members in C_i with various colliding possibilities are all resolved can sensor s_i be included into the movable set and perform physical movement. Upon receiving the moving instruction from the clusterhead, s_i waits for T_i (possibly adjusted) and then moves with speed V_i (possibly adjusted). In our route scheduling strategy, we try to include *as many sensors as possible* to move simultaneously in the same round (batch).

For each of the ten collision cases identified, we define corresponding actions (**Action D-I**, **Action D-II**, $\cdots$, **Action S-I**, **Action S-II**, $\cdots$) to evaluate respective case and perform necessary adjustments. If colliding possibility remains due to unsuccessful adjustment, physical movement by sensor s_i is not allowed and should be deferred. Thus we additionally define **Action Deferred** to perform corresponding operations. Note that in **Case D-I**, **Case D-III**, and **Case D-IV**, no action is needed since s_i and s_j are scheduled in different rounds (no collision is likely to happen in the three cases despite intersection exists between the two moving paths). For the rest of seven cases, we describe the evaluation principles exercised by respective action as follows (detailed operations are available in Algorithm 1, Section III-B).

Action D-II In this case, since s_j gets in the way of s_i's moving path, the clusterhead instructs s_j to slightly adjust its location along line $\overrightarrow{p_j p_j^{'}}$ to avoid collision. Assume the location adjustment is small enough to have no effect on other moving paths.

Action D-V Sensor s_i is not allowed to move, for its destination point $p_i^{'}$ will block the moving path of s_j in a later round. In this case, the moving order of s_i should be set to be larger than that of s_j ($order_i = order_j + 1$) to postpone s_i's physical movement after s_j. In addition, a fix_order_i flag should be set $true$, indicating no updates on $order_i$ will be performed in later rounds to ensure the delayed movement after s_j, and then **Action Deferred** is invoked for s_i.

Action S-I Define the traveling time from p_i to the intersection point p_{ij} as $t_{p_i \to p_{ij}}$ (obtained from available $d(p_i, p_{ij})$ and V_i), the clusterhead evaluates if $T_i + t_{p_i \to p_{ij}} = T_j + t_{p_j \to p_{ij}}$, where T_i and T_j are the waiting times of s_i and s_j as defined earlier. If equality holds, a collision at the intersection is expected, and the waiting time T_j of s_j should be increased by a small amount of Δt to avoid the collision. However, in case s_j has already been processed with $dirty_j$ set $true$, the adjustment is prohibited and s_i is not allowed to move in the current round. Consequently, moving order of s_i is increased ($order_i = order_i + 1$) and **Action Deferred** is invoked for s_i.

Action S-II If s_i reaches the intersection point p_{ij} no later than s_j's departure time, the clusterhead should instruct s_j to slightly adjust its location along line $\overrightarrow{p_j p_j^{'}}$ to avoid collision.

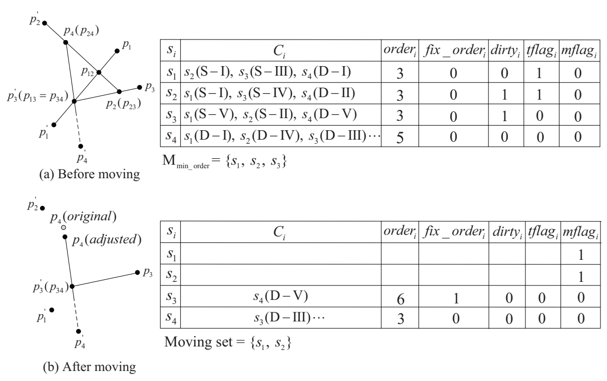
Action S-III If s_j reaches the intersection point p_{ij} no later than s_i, the destination point $p_j^{'}$ of s_j will block the moving path of s_i. In this case, the clusterhead should instruct s_j to increase its waiting time T_j by setting $T_j = T_i + (t_{p_i \to p_{ij}} - t_{p_j \to p_{ij}}) + \Delta t$ to ensure the delayed arrival of s_j at p_{ij} ($p_j^{'}$). If the adjustment of T_j is not successful due to a $true$ flag of $dirty_j$, then s_j is not allowed to move in the current round. Consequently, moving order of s_j is increased ($order_j = order_j + 1$) and **Action Deferred** is invoked for s_j.

Action S-IV If s_j reaches the intersection point p_{ij} no later than s_i's departure time, the clusterhead should increase the waiting time of s_j by setting $T_j = T_i - t_{p_j \to p_{ij}} + \Delta t$. In case the adjustment is not allowed due to a $true$ value of $dirty_j$, the clusterhead instructs s_i to slightly adjust its location along line $\overrightarrow{p_i p_i^{'}}$ to avoid collision.

Action S-V If s_i reaches the intersection point p_{ij} no later than s_j, the destination point $p_i^{'}$ of s_i will block the moving path of s_j. In this case, the clusterhead should instruct s_j to increase its moving speed V_j by setting $V_j = \frac{V_i \cdot d(p_j, p_{ij})}{d(p_i, p_{ij}) + V_i (T_i - T_j)} + \Delta v$, where Δv is a small amount of speed increment to ensure s_j's earlier arrival at p_{ij} ($p_i^{'}$) than s_i. However, if the adjusted V_j is larger than the maximum possible moving speed V_{max} or the adjustment of V_j is prohibited due to a $true$ value of $dirty_j$, then s_i is not allowed to move in the current round. Moving order of s_i is increased ($order_i = order_i + 1$) and **Action Deferred** is invoked for s_i.

Action Deferred Since s_i (s_j) is not allowed to move in the current round, $tflag_i$ ($tflag_j$) is set $false$. In addition, the clusterhead should confirm if this not-moving decision leads to moving path blocking of any sensor in M_{min_order} set that is already allowed to move in the current round (with $tflag$ set $true$), and do necessary slight location adjustment to resolve the blocking.

Fig. 2 illustrates a snapshot of the CFPP operations. Note that s_4 has more intersections with other sensors, which are not shown in the figure (omitted for brevity). In the current round, potential moving set M_{min_order} includes s_1, s_2, and s_3, all having the currently smallest $order$ value of 3. For s_1, colliding conditions caused by all members in C_1 are analyzed and handled case by case. In this example, since s_1 and s_2 are evaluated to reach intersection p_{12} simultaneously,



s_i	C_i	$order_i$	fix_order_i	$dirty_i$	$tflag_i$	$mflag_i$
s_1	s_2(S−I), s_3(S−III), s_4(D−I)	3	0	0	1	0
s_2	s_1(S−I), s_3(S−IV), s_4(D−II)	3	0	1	1	0
s_3	s_1(S−V), s_2(S−II), s_4(D−V)	3	0	1	0	0
s_4	s_1(D−I), s_2(D−IV), s_3(D−III)⋯	5	0	0	0	0

$M_{min\_order} = \{s_1, s_2, s_3\}$

s_i	C_i	$order_i$	fix_order_i	$dirty_i$	$tflag_i$	$mflag_i$
s_1						1
s_2						1
s_3	s_4(D−V)	6	1	0	0	0
s_4	s_3(D−III)⋯	3	0	0	0	0

Moving set = $\{s_1, s_2\}$

Fig. 2. Every sensor s_i in the potential moving set M_{min_order} should be analyzed by identifying its intersection (collision) relationship with each member in C_i, in which intersection cases D-II, D-IV, S-I, S-II, S-III, S-IV, and S-V require further consideration/processing, before including s_i into the moving set (allowed to move in the current round).

the clusterhead adjusts the waiting time of s_2 by setting $T_2 = T_2 + \Delta t$ to resolve the collision. Next, since s_3 is found to reach intersection p_{13} earlier than s_1, blocking s_1's moving path, the clusterhead instructs s_3 to increase its waiting time by setting $T_3 = T_1 + (t_{p_1 \to p_{13}} - t_{p_3 \to p_{13}}) + \Delta t$. As to s_4 (scheduled to move in a later round), no action is required since no collision is likely to happen between s_1 and s_4. Consequently, the clusterhead includes s_1 into the moving set. Similar operations apply to s_2. In our example, s_2 has no colliding possibilities with s_1 and s_3. However, since the departure location p_4 of s_4 blocks s_2's moving path, the clusterhead instructs s_4 to slightly move from p_4 (original) to p_4 (adjusted), as shown in Fig. 2 (b). As a result, s_2 is also included into the moving set. For s_3, in our example, both s_1 and s_2 do not pose colliding sources to s_3. Unfortunately, since the destination point p_3' of s_3 will block the moving path of s_4 in a future round, s_3 is not allowed to move before s_4 (not included into the moving set), and $order_3$ should be updated to 6 ($order_4 + 1$) with fix_order_3 set $true$. After the evaluations, sensors included in the moving set (i.e., s_1 and s_2) perform physical movements simultaneously, and $order_4$ and set C_4 are updated accordingly.

B. CFPP Algorithm Summary

TABLE I
SUMMARY OF NOTATIONS USED IN THE CFPP ALGORITHM

Notation	Description
C_i	Set of potential colliding sensors against s_i
$order_i$	Moving order of s_i, where $order_i = \lvert C_i \rvert$
fix_order_i	Indicates the $order$ value of s_i is henceforth fixed
$dirty_i$	Indicates whether s_i has been processed in the current round
$tflag_i$	Indicates whether s_i is allowed to move in the current round
$mflag_i$	Indicates whether s_i has moved from p_i to p_i'
M_{min_order}	Set of sensors with minimum $order$ value in current round

Table I summarizes the notations used in CFPP, and Algorithm 1 provides the pseudocode for CFPP operations.

Algorithm 1 Collision-free Path Planning (CFPP)

```
include all sensors in set $S$;
establish set $C_i$ for $\forall s_i \in S$ ; // $i = 1, \cdots, k$
evaluate $order_i$ for $\forall s_i \in S$;
clear $fix\_order_i$, $dirty_i$, $tflag_i$, $mflag_i$ for $\forall s_i \in S$; // all set to $false$
while ($S$ !empty) do
  re-establish set $C_i$ for $\forall s_i \in S$;
  re-evaluate $order_i$ for $\forall s_i \in S$ with $fix\_order_i == false$;
  reset $T_i = 0, V_i = V, dirty_i = false, tflag_i = false$ for $\forall s_i \in S$;
  include all $s_i$ with the minimum $order_i$ value into the $M_{min\_order}$ set;
  for (each $s_i \in M_{min\_order}$) do
    set $tflag_i = true$;
    for (each $s_j \in C_i$) do
      classify the intersection (collision) $case$ for $s_i$ and $s_j$;
      switch ($case$)
          Case D-II: do Action D-II;
          Case D-V: do Action D-V;
          Case S-I: do Action S-I;
          Case S-II: do Action S-II;
          Case S-III: do Action S-III;
          Case S-IV: do Action S-IV;
          Case S-V: do Action S-V;
    end for
  end for
  perform simultaneous physical movements for $\forall s_i$ with $tflag_i == true$;
  set $mflag_i = true$ for such sensor $s_i$; // physical movement performed
  remove all $s_i$ with $mflag_i == true$ from sensors set $S$;
end while
procedure Action D-II
   slightly adjust location of $s_j$ from $p_j$ (original) to $p_j$ (adjusted);
procedure Action D-V
   set $order_i = order_j + 1$; set $fix\_order_i = true$;
   invoke Action Deferred ($s_i$);
procedure Action S-I
   if $T_i + t_{p_i \to p_{ij}} = T_j + t_{p_j \to p_{ij}}$ then
     if $dirty_j == false$ then set $T_j = T_j + \Delta t$; $dirty_j = true$;
     else set $order_i = order_i + 1$; invoke Action Deferred ($s_i$);
procedure Action S-II
   if $T_i + t_{p_i \to p_{ij}} \leq T_j$ then
     slightly adjust location of $s_j$ from $p_j$ (original) to $p_j$ (adjusted);
procedure Action S-III
   if $T_i + t_{p_i \to p_{ij}} \geq T_j + t_{p_j \to p_{ij}}$ then
     if $dirty_j == false$ then
       set $T_j = T_i + (t_{p_i \to p_{ij}} - t_{p_j \to p_{ij}}) + \Delta t$; set $dirty_j = true$;
     else set $order_j = order_j + 1$; invoke Action Deferred ($s_j$);
procedure Action S-IV
   if $T_i \geq T_j + t_{p_j \to p_{ij}}$ then
     if $dirty_j == false$ then
       set $T_j = T_i - t_{p_j \to p_{ij}} + \Delta t$; set $dirty_j = true$;
     else slightly adjust location of $s_i$ from $p_i$ (original) to $p_i$ (adjusted);
procedure Action S-V
   if $T_i + t_{p_i \to p_{ij}} \leq T_j + t_{p_j \to p_{ij}}$ then
     if $dirty_j == false$ then
       set $V_j = \frac{V_i d(p_j, p_{ij})}{d(p_i, p_{ij}) + V_i (T_i - T_j)} + \Delta v$; set $dirty_j = true$;
       if $V_j > V_{max}$ then
         set $order_i = order_i + 1$; invoke Action Deferred ($s_i$);
     else set $order_i = order_i + 1$; invoke Action Deferred ($s_i$);
procedure Action Deferred ($s_i$)
   set $tflag_i = false$;
   do necessary slight adjustment of $s_i$'s departure location to resolve
    moving path blocking possibly caused by this not-moving decision;
```

IV. Performance Evaluation

In this section, we validate our CFPP mechanism by comparing the performance with two other path-planning approaches: ADO (introduced in [7]) and Super A* (introduced in [12]). Fig. 3 illustrates a monitored 200×200 area with

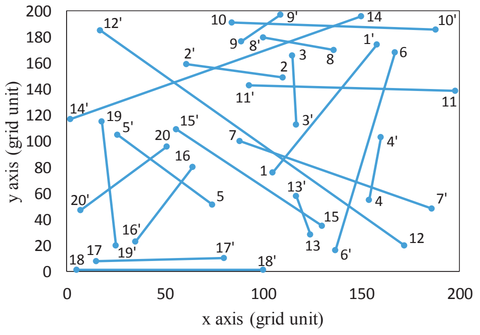


Fig. 3. Random path configuration in a monitored 200 × 200 area with sensors population up to 20 (here i represents the departure position of sensor s_i whereas i' indicates the destination (goal) position of s_i.

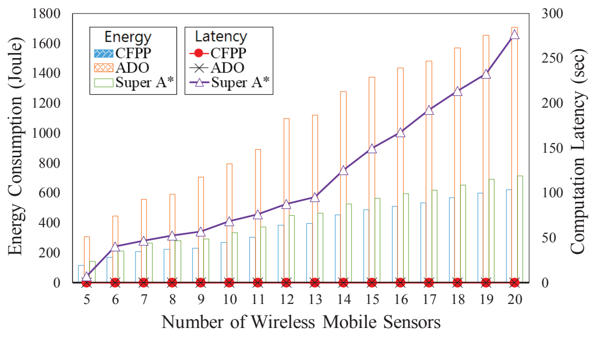


Fig. 4. Computation latency and energy consumption incurred by CFPP, ADO, and Super A* path-planning strategies under various amounts of sensor nodes in a monitored 200 × 200 area.

up to 20 randomly configured moving paths. We observe the computation latency and energy consumption incurred by the three approaches. As shown in Fig. 4, both CFPP and ADO manifest little computation latency, while Super A* requires significantly longer time to compute path-planning solutions. Moreover, computation latency with Super A* increases drastically as number of mobile sensors (moving paths) grows.

To model the moving energy, we estimate the energy consumed by the motion device moving for one grid unit by performing real measurements on the sensor robot used in our implementation testbed with grid size equal to 1 cm. The robot assembles six 1.2 V 2000 mAh rechargeable NiMH batteries with measured 200 $\sim$ 290 mA moving current and average moving speed at 0.06 m/sec. Consequently, the average moving energy consumption per grid (unit distance) can be obtained by $0.29 \times 7.2 \times (\frac{0.01}{0.06}) = 0.348$ Joule. We then compute the total moving energy consumption based on the traveling distance accordingly [11]. To model the energy consumed by visual sensors (cameras) in ADO (recall that ADO algorithm depends on the presence of omnidirectional cameras for calculating moving paths, as described in Section I), we perform measurements on a commercial video camera M30 Series [1]. An estimated 3.4 Watt at average (ranging from 2.2 to 4.6 Watt) facilitates our calculation of the total camera energy required by an individual sensor in ADO, which can be obtained by $3.4 \times moving\ time\ (sec)$ Joule. Fig. 4 displays the aggregate energy consumption for CFPP, ADO, and Super A* respectively. Our CFPP mechanism consumes the least aggregate energy among three, whereas ADO produces the most power cost due to extra energy consumed by visual sensors (omnidirectional cameras). Meanwhile, since the generated Super A* paths are not always in straight lines (shortest paths), Super A* consumes more moving energy as compared to our CFPP.

In order to observe the capability of path-planning algorithms on resolving deadlocks, we add potential deadlock situations as mentioned in [8][1]. Fig. 5 depicts a new configuration consisting of 20 random moving paths with three deadlock situations. Potential deadlocks #1 and #2 simulate the situation when a goal position blocks another sensor's moving path, whereas potential deadlock #3 demonstrates a triangular deadlock situation, in which all involved sensors are prevented from moving.

Fig. 6 shows the sensors goal reachability accomplished by CFPP, ADO, and Super A* as time advances. We observe that ADO suffers from deadlocks #1 and #3 at time 20, which stop sensors $s_{11}, s_{16}, s_{17}, s_{18}$ from moving to their destinations (goals). At time 40, ADO encounters deadlock #2, which prevents sensor s_{14} from reaching its goal position. After time 40, ADO is unable to make any further progress with eventually 75% final goal reachability. Super A* is capable of resolving deadlocks #1 and #2, but unable to handle the triangular deadlock #3, which occurs at time 20 and traps sensors s_{16}, s_{17}, s_{18} from departing toward their destinations. Super A* stops making progress after time 40, leading to 85% final goal reachability. In contrast, our CFPP is capable of resolving all deadlock situations. For deadlocks #1 and #2 (classified as **Case D-V** in our CFPP algorithm), sensors s_4 and s_{10} execute **Action D-V** by deferring their movements (scheduled in a later batch after s_{11} and s_{14} reach their goal positions). For deadlock #3 (a triangular deadlock), since sensors s_{16}, s_{17}, s_{18} will be scheduled in the same moving order, CFPP naturally resolves this deadlock situation by allowing three sensors to move simultaneously without blocking each other. Interestingly, CFPP reachability grows slowly (due to the batched movements applied by CFPP) and outperforms the other two approaches after time passes 80, yet leads to the highest 100% goal reachability. From the performance results, we validate our CFPP mechanism's capability of successfully guiding 100% of the sensors to their goal positions without collisions, while incurring little computation latency and moderate energy consumption.

[1]Here we define that deadlocks among two or more robots (mobile sensors) occur if these robots block each other in a way such that any or all of them is/are unable to continue along its/their trajectory (traveling path) without causing a collision.

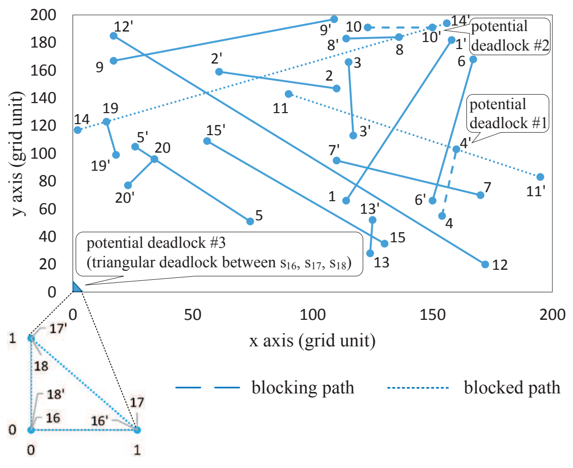


Fig. 5. Random path configuration of 20 sensors in a monitored 200×200 area with potential deadlocks #1, #2, and #3.

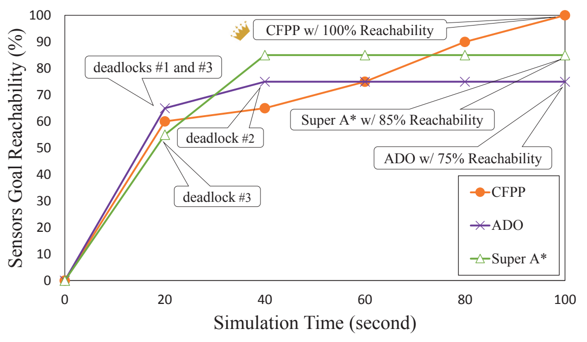


Fig. 6. Sensors goal reachability accomplished by our CFPP, ADO, and Super A* path-planning strategies with the existence of potential deadlocks #1 , #2 , and #3 in a monitored 200×200 area.

We have implemented a proof-of-concept prototype based on moving robots (LEGO MINDSTORMS NXT 9797 [3]) carrying embedded system boards (Tibbo EM1206EV [4]) to further corroborate our CFPP protocol feasibility in a real-life environment. A brief demonstration video on our experiments is available in [2].

V. Conclusion

In this paper, we devise a collision-free path planning (CFPP) mechanism to schedule moving paths for mobile sensors deployment problem. When sensors move around to self-deploy, the CFPP algorithm comes into play by systematically classifying colliding cases and employing batched movements. Our proposed CFPP guarantees sensors goal reachability and successfully guides all sensors to their destinations without causing collisions.

References

[1] Axis Communication M30 series. http://tw.axis.com/download/ds_m30series_51497_tw_1304.pdf/.
[2] CFPP demonstration video. http://bunlab.twbbs.org/filezone/index.php?dir=implement_video/thesis_video/.
[3] LEGO MINDSTORMS NXT 9797. http://www.lego.com/.
[4] TIBBO Technology. http://tibbo.com/.
[5] P. Bhattacharya and M. L. Gavrilova. "Roadmap-based Path Planning Using the Voronoi Diagram for Clearance-based Shortest Path". *IEEE Robotics and Automation Magazine*, 15(2):58–66, June 2008.
[6] S. Biagioni and K. W. Bridges. "The Application of Remote Sensor Technology to Assist the Recovery of Rare and Endangered Species". *Int'l Journal of High Performance Computing Applications*, 16(3):315–324, 2002.
[7] C. Cai, C. Yang, Q. Zhu, and Y. Liang. "Collision Avoidance in Multi-Robot Systems". *In Proc. IEEE Int'l Conference on Mechatronics and Automation*, pages 2795–2800, August 2007.
[8] M. Jager and B. Nebel. "Decentralized Collision Avoidance, Deadlock Detection, and Deadlock Resolution for Multiple Mobile Robots". *In Proc. IEEE/RSJ Int'l Conference on Intelligent Robots and Systems*, pages 1213–1219, November 2001.
[9] V. Kunchev, L. Jain, V. Ivancevic, and A. Finn. "Path Planning and Obstacle Avoidance for Autonomous Mobile Robots: A Review". *In Proc. Int'l Conference on Knowledge-based Intelligent Information and Engineering Systems*, pages 537–544, October 2006.
[10] T.-Y. Lin, H. A. Santoso, W.-T. Liu, and H.-T. Liu. "An Enhanced Sensor Deployment Scheme for Automated Smart Environments". *In Proc. IEEE Int'l Conference on Advanced Networks and Telecommunications Systems*, pages 1–6, December 2013.
[11] T.-Y. Lin, H. A. Santoso, and K.-R. Wu. "Global Sensor Deployment and Local Coverage-aware Recovery Schemes for Smart Environments". *IEEE Transactions on Mobile Computing*, accepted to appear, 2015.
[12] F. Liu and A. Narayanan. "Real Time Replanning Based on A* for Collision Avoidance in Multi-Robot Systems". *In Proc. Int'l Conference on Ubiquitous Robots and Ambient Intelligence*, pages 473–479, November 2011.
[13] B. Mahajan and P. Marbate. "Literature Review on Path Planning in Dynamic Environment". *International Journal of Computer Science and Network*, 2(1):115–118, February 2013.
[14] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. "Wireless Sensor Networks for Habitat Monitoring". *In Proc. Int'l Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88–97, September 2002.
[15] A. N. Nazif, A. Davoodi, and P. Pasquier. "Multi-agent Area Coverage Using a Single Query Roadmap: A Swarm Intelligence Approach". *Advances in Practical Multi-Agent Systems*, 325(1):95–112, September 2011.
[16] M. T. Rantanen. "A Connectivity-based Method for Enhancing Sampling in Probabilistic Roadmap Planner". *Journal of Intelligent Robotic Systems*, 64(2):161–178, November 2011.
[17] M. T. Rantanen and M. Juhola. "A Configuration Deactivation Algorithm for Boosting Probabilistic Roadmap Planning of Robots". *International Journal of Automation and Computing*, 9(2):155–164, September 2011.
[18] E. Rimon and D. E. Koditschek. "Exact Robot Navigation Using Artificial Potential Functions". *IEEE Transactions on Robotics and Automation*, 8(5):501–518, August 1992.
[19] P. Song. "A Potential Field Based Approach to Multirobot Manipulation General Robotics". *In Proc. IEEE Int'l Conference on Robotics and Automation*, pages 1217–1222, May 2002.
[20] Y. Wang and G. S. Chirikjian. "A New Potential Field Method for Robot Path Planning". *In Proc. IEEE Int'l Conference on Robotics and Automation*, pages 977–982, April 2000.