Automation in Construction 81 (2017) 25–33

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Enhanced discrete particle swarm optimization path planning for UAV vision-based surface inspection

Manh Duong Phung[a], Cong Hoang Quach[a], Tran Hiep Dinh[b], Quang Ha[b,*]

[a] Vietnam National University, 144 Xuan Thuy, Cau Giay, Hanoi, Vietnam
[b] University of Technology Sydney, 15 Broadway, Ultimo, NSW 2007, Australia

ARTICLE INFO

Keywords:
Path planning
Infrastructure monitoring
Bridge inspection
Vision-based inspection
Particle swarm optimization
Unmanned aerial vehicle

ABSTRACT

In built infrastructure monitoring, an efficient path planning algorithm is essential for robotic inspection of large surfaces using computer vision. In this work, we first formulate the inspection path planning problem as an extended travelling salesman problem (TSP) in which both the coverage and obstacle avoidance were taken into account. An enhanced discrete particle swarm optimization (DPSO) algorithm is then proposed to solve the TSP, with performance improvement by using deterministic initialization, random mutation, and edge exchange. Finally, we take advantage of parallel computing to implement the DPSO in a GPU-based framework so that the computation time can be significantly reduced while keeping the hardware requirement unchanged. To show the effectiveness of the proposed algorithm, experimental results are included for datasets obtained from UAV inspection of an office building and a bridge.

1. Introduction

For robotics inspection of built infrastructure, computer vision can be used to detect most surface deficiencies such as cracking, spalling, rusting, distortion, misalignment, and excessive movements. Over the last decade, much research effort has been devoted to this theme with computer vision becoming an important component of modern Structural Health Monitoring (SHM) systems for built infrastructure such as rust detection of steel bridges [1], crack detection of concrete bridges [2–4], or bridge condition assessment [5]. In this regard, it is promising to integrate a computer vision system into mobile inspection robots, such as unmanned aerial vehicles (UAVs) [6,7] or ubiquitous robots [8,9], especially when dealing with large and hardly accessible structures like tunnels [10]. For this purpose, an efficient inspection path planning (IPP) algorithm is therefore of crucial importance.

In vision-based inspection path planning, it is required to find a trajectory that is informative enough to collect data from different views of a given structure so that the inspection robot can carry out the data acquisition of the region of interest. Depending on size of the inspecting region, the trajectory can be planned for multiple robots to coordinately conduct the data collection [11]. To be visibly processed at a later time, the data collected are often from a sensor of the time-of-flight (optical, sonar or radar) or passive optical (CCD camera) type. Since the computational time for IPP rapidly increases with the area of the region of interest, an IPP algorithm should meet the following criteria:

(i) capability of viewing/covering every surface of the region of interest via at least one planned viewpoint of the inspection sensor,
(ii) obstacle avoidance for the robot,
(iii) generation of an “optimal” path under available conditions, and
(iv) effectiveness in terms of processing time (for online re-planning and large structure inspection).

Studies on IPP, in general, can be categorised into three groups, namely cell decomposition, sub-problem separation, and other methods. In cell decomposition, the target space is decomposed in sub-regions called cells. The cell shape can be trapezoidal, square, cubic, or customised depending on critical points of Morse functions, often with a uniform size [12–14]. An exhaustive path connecting each cell is then computed for the coverage, using typically a heuristic algorithm such as wavefront [15] or spiral spanning tree [16]. Methods based on cell decomposition yield good results in terms of coverage and obstacle avoidance. As the path generated, however, may not be optimal, it is worth seeking a better and more feasible alternative. In this context, the IPP separation approach tackling the non-deterministic polynomial time (NP)-hard problems can be divided into two, the art gallery problem that finds the smallest set of viewpoints to cover the whole gallery, and the travelling salesman problem (TSP) that finds the shortest path to visit a set of given cities [17–22]. Each problem can

* Corresponding author.
E-mail addresses: duongpm@vnu.edu.vn (M.D. Phung), hoangqc@vnu.edu.vn (C.H. Quach), tranhiep.dinh@uts.edu.au (T.H. Dinh), quang.ha@uts.edu.au (Q. Ha).

http://dx.doi.org/10.1016/j.autcon.2017.04.013
Received 12 September 2016; Received in revised form 4 March 2017; Accepted 27 April 2017

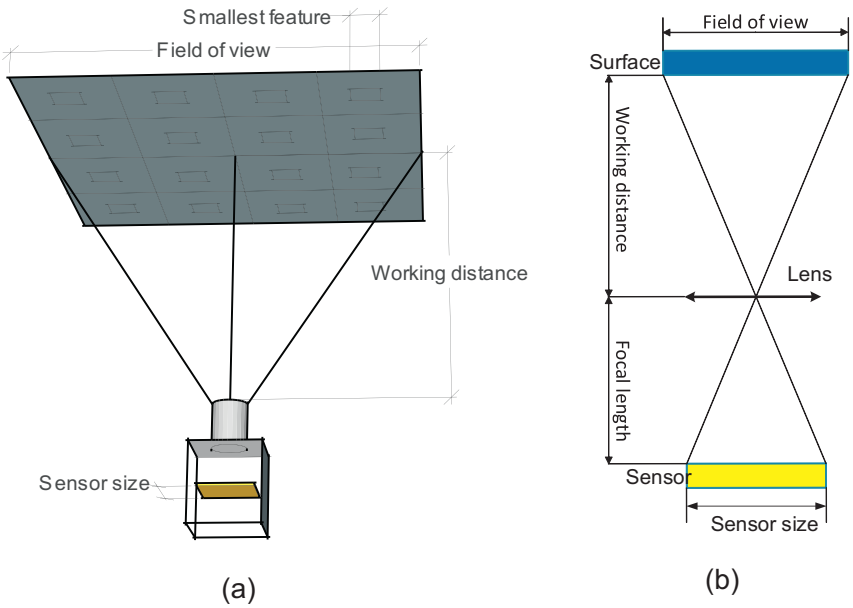

Fig. 1. Camera for inspection: (a) Camera setup in the field; (b) Relation between parameters of the camera and the field.

be solved separately using known methods such as the randomised, incremental algorithm for the art gallery problem [23,24] and the chained Lin-Kernighan heuristics for the TSP [25]. Other approaches have focused on sampling the configuration space [26], using submodular objective function [27], or employing genetic algorithms [28] but they often require constraining the robot to certain dynamic models or end with near-optimal solutions. The requirements remain not only a shorter path but also collision-free.

In this paper, the IPP problem is addressed by first formulating it as an extended TSP. The enhanced discrete particle swarm optimization (DPSO) is then employed to solve the IPP. Finally, parallel computing based on graphical processing units (GPU) is deployed to obtain the real-time performance. The contributions of our approach are three folds: (i) By formulating the IPP as an extended TSP, both the coverage and obstacle avoidance are simultaneously taken into account. In addition, constraints related to the kinematic and dynamic models of the robot are separated from the DPSO solution so that this solution can be applied to a broad range of robots. (ii) Three techniques including deterministic initialization, random mutation, and edge exchange have been proposed to improve the accuracy of DPSO. (iii) Parallel computation has been implemented to significantly improve the time performance of DPSO. By utilising GPU, the parallel implementation does not add additional requirements to the hardware, i.e. the developed software can run on popular laptop computers.

The rest of this paper is structured as follows. Section 2 introduces the steps to formulate the IPP as an extended TSP. Section 3 presents the proposed DPSO and its deployment for solving the IPP. Section 4 provides experimental results. Finally, a conclusion is drawn to end our paper.

2. Problem formulation

Our ultimate goal is to design a path planning system for an UAV used for inspecting planar surfaces of largely built structures like buildings or bridges. The sensor used for the inspection is a CCD camera attached to a controllable gimbal. We suppose that the 3D model of the structure and the environment are known prior to planning, for example, by using laser scanners. Here, the IPP objective is to find the shortest path for the UAV's navigation and taking photos of the target surfaces so that the images captured can be later processed to detect potential defects or damages. We first consider the IPP as an extended TSP and then solve it using the developed DPSO. This section presents the computation of viewpoint selection and point-to-point pathfinding, which are fundamental to formulate the extended TSP problem.

2.1. Viewpoint selection

The viewpoint selection involves finding a set of camera configurations that together cover the whole surfaces of interest. Let P be a finite set of geometric primitives p_i comprising the surfaces to be covered. Each geometric primitive p_i corresponds to a surface patch within the field of view of the camera. Let C be the configuration space such that every feasible configuration $c_j \in C$ maps to a subset of P. Each configuration c_i corresponds to a position (x_i,y_i,z_i) and an orientation $(\varphi_i,\theta_i,\psi_i)$ of the camera. Given a finite set of configurations C, the viewpoint selection problem on one hand calls generally for the minimum number of configurations c_i such that all elements $p_i \in P$ are covered. On the other hand, from image sticking and defect detection, the following requirements are added to the system: (i) image capturing moment is when the camera is perpendicular to the inspected surface, (ii) sufficiently high resolution to distinguish the smallest feature, s_f, and (iii) overlapping of images to a percentage o_p specified by the sticking algorithm.

It turns out that those requirements simplify our selection problem. The perpendicular requirement confines the camera orientation to the normal of the inspected surface. The resolution requirement suggests the computation of the field of view of the camera as:

$$a_{fov} = \frac{1}{2} r_c s_f, \tag{1}$$

where r_c is the camera resolution (see Fig. 1). Taken the overlapping percentage into account, the geometric primitive p_i is then:

$$p_i = (1 - o_p) a_{fov}. \tag{2}$$

The working distance from the camera to the surface can also be computed as:

$$d_k = \frac{a_{fov} f}{s_s}, \tag{3}$$

where f and s_s are respectively the focal length and sensor size of the camera. From Eqs. (2) and (3), it is possible to determine configurations c_i to cover the set of primitives P, as illustrated in Fig. 2. Specifically, for

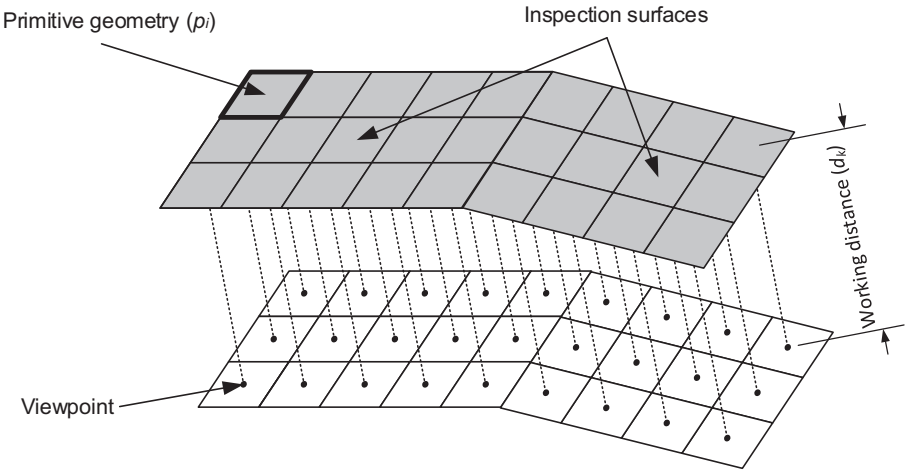

Fig. 2. Generation of inspection viewpoints.

each surface $P_k \subset P$, a grid with the cell size of p_i is first established to cover P_k. A working surface P_k^*, parallel to P_k and distant d_k from P_k, is then created. Projecting the center of each cell of P_k to P_k^* gives the position component of viewpoint c_i. The normal of P_k defines the orientation component of c_i, which is supposed to be fully controlled by the inspecting UAV so that it can be omitted in our computation.

2.2. Point-to-point pathfinding

Given the viewpoints, the shortest, obstacle-free path between every pair of them need be found to form a graph for later processing. Without loss of generality, different motion planning approaches such as roadmap, decoupling, potential field and mathematical programming can be used here depending on the UAV model and dynamic constraints [29,30]. In this work, the hierarchical decoupled approach is employed in which open- and closed-loop controllers operating at a variety of rate are linked together from top to bottom [29,31,32]. Since the majority of UAVs currently in production often already equipped with an inner-loop tracking controller and a waypoint following system, this approach can be simplified to a discrete search that produces a set of waypoints connecting two viewpoints while avoiding obstacles. For this, the workspace is first divided into a grid of voxels. Each voxel has the free or occupied status corresponding to the presence or absence of an object in that voxel. In order to consider the UAV as a particle moving without collision between voxels, all the free voxels in a sphere of a radius equal to the largest dimension of the UAV are marked as occupied. Thus, the A* algorithm [33] can be used to find the shortest path between viewpoints. In each step, the cost to move from one voxel to another surrounding neighbour is computed as:

$$L(\alpha, \beta, \gamma) = a_1\alpha^2 + a_2\beta^2 + a_3\gamma^2, \tag{4}$$

where coordinates $\alpha,\beta,\gamma \in \{-1,0,1\}$ indicate the position of neighbour, and coefficients a_1, a_2 and a_3 assign a particular weight to each direction. The total cost to move from a voxel p to the viewpoint g at step n is given by:

$$f(p) = \sum_{k=1}^{n} L_k + \left\| p - g \right\|^2, \tag{5}$$

where L_k is the motion cost at step k.

2.3. Modelling the IPP as a TSP

For given viewpoints and paths between them, a graph can be built to model the IPP as an extended TSP. We define each viewpoint as a node, i, and the path between two viewpoints as an edge, e_{ij}. The length, l_{ij}, of edge e_{ij} is the cost to travel from node i to node j determined by Eq. (5). If the path between node i and node j is blocked due to obstacles, a virtual path between them is defined and a very large cost is assigned for the path. Denoting the set of all nodes by V and the set of all edges by E, we restrict motion of the UAV to the graph $G = (V,E)$. The IPP task is now to find a tour, with a minimum cost, that visits each node (viewpoint) exactly once, including the way back to the initial node. Let T be the set of these nodes.

By associating a binary variable

$$\lambda_{ij} = \begin{cases} 1 & \text{if edge } e_{ij} \in E \text{ is in tour} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

with each edge in the graph, the IPP is then formulated as follows:

$$\min \sum_{e_{ij} \in E} l_{ij}\lambda_{ij} \tag{7}$$

$$\text{subject to} \sum_{j \in V,\ i \neq j} \lambda_{ij} = 2 \quad \forall\, i \in V \tag{8}$$

$$\sum_{i,j \in T,\ i \neq j} \lambda_{ij} \leq |T| - 1 \quad \forall\, T \subset V,\ T \neq \varnothing \tag{9}$$

$$\lambda_{ij} \in \{0, 1\}, \tag{10}$$

where $|T|$ is the number of nodes in the tour. The objective function in Eq. (7) defines the shortest tour. The constraint in Eq. (8) implies each node in the graph has exactly one incoming edge and one outgoing edge, i.e., the tour passes through each node once, while condition (9) ensures no sub-tours, i.e., the tour returns to the original node after visiting all other nodes.

3. Enhanced discrete particle swarm optimization for inspection path planning

Particle swarm optimization (PSO), inspired by social behavior of bird flocking or fish schooling, is a population-based stochastic technique designed for solving optimization problems [34]. In PSO, a finite set of particles is generated, each particle seeks the global optimum by moving and evolving through generations. Initially, each particle is assigned to a random position and velocity. It then moves by updating its best previous position, P_k, and the best position of the swarm, G_k. Let x_k and v_k be respectively the position and velocity of a particle at generation k. The position and velocity of that particle in the next generation is given by:

$$v_{k+1} \leftarrow w.\ v_k + \varphi_1 r_1.\ (P_k - x_k) + \varphi_2 r_2.\ (G_k - x_k) \tag{11}$$

$$x_{k+1} \leftarrow x_k + v_{k+1}, \tag{12}$$

where w is the inertial coefficient, φ_1 is the cognitive coefficient, φ_2 is

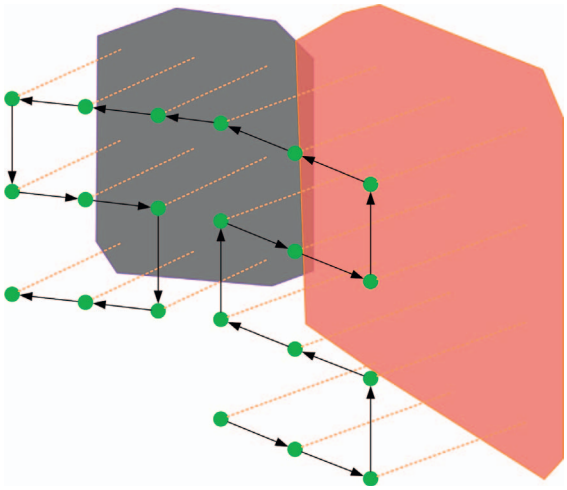

Fig. 3. Initialization of particle using back-and-forth path.

the social coefficient, and r_1, r_2 are random samples of a uniform distribution in the range [0,1]. Eqs. (11) and (12) imply that the motion of a given particle is a compromise between three possible choices including following its own way, moving toward its best previous position, or toward the swarm's best position. The ratio between choices is determined by the coefficients w, φ_1, and φ_2.

3.1. DPSO approach to the IPP

Since the IPP defined in Eqs. (7)–(10) is a discrete optimization problem, enhanced algorithms for discrete particle optimization (DPSO) will be developed for our problem, motivated by Clerc [35]. For this, let us begin with an outline of our approach to solve the IPP problem using DPSO with improvements in initialization, mutation, edge exchange and parallel implementation.

First, let define the position of particles as sequences of $N + 1$ nodes, all distinct, except that the last node must be equal to the first one:

$$x = (n_1, n_2, \ldots, n_N, n_{N+1}),\ n_i \in V,\ n_1 = n_{N+1}, \tag{13}$$

where N is the number of nodes, $N = |V|$. Since each sequence is a feasible tour satisfying Eqs. (8) and (9), to minimise the objective function (7) according to Eqs. (11) and (12), we need to define the velocity and numerical operators for the particles' motion.

From Eq. (12), it can be seen that a new position of a particle can be evolved from the position of its current generation via the velocity operator, considered here as a list of node transpositions:

$$v = ((n_{i,1}, n_{j,1}),(n_{i,2}, n_{j,2}), \ldots, (n_{i,\|v\|}, n_{j,\|v\|})), \tag{14}$$

where $n_i, n_j \in V$ and $\|v\|$ is the length of the transposition list.

In DPSO, particle velocities and positions are updated by using the following operations:

- The *addition* between a position x and a velocity v is found by applying the first transposition of v to x, then the second one to the result, etc. For example, with $x = (1,4,2,3,5,1)$ and $v = ((1,2),(2,3))$, by applying the first transposition of v to x and keeping in mind the equality between the first and last nodes, we obtain (2, 4, 1, 3, 5, 2). Then applying the second transposition of v to that result gives (3, 4, 1, 2, 5, 3), which is the final result of $x + v$.
- The *subtraction* between a position x_2 and a position x_1 is defined as the velocity v, i.e., $x_2 - x_1 = v$, such that by applying v to x_1 we obtain back x_2.
- The *addition* between a velocity v_1 and a velocity v_2 is defined as a new velocity, $v_1 \oplus v_2 = v$, which contains the transpositions of v_1 followed by the transpositions of v_2.
- The *multiplication* between a real coefficient c with a velocity v is a new velocity, $c.v$, defined as follows:
  - For $c = 0, c.v = \emptyset$.
  - For $0 < c \leq 1, c.v = ((n_{i,1},n_{j,1}), (n_{i,2},n_{j,2}), \ldots, (n_{i,c\|v\|},n_{j,c\|v\|}))$.
  - For $c < 0$ and $c > 1$, we omit these cases since they do not occur in our DPSO.

3.2. Augmentations to the DPSO

In order to speed up the convergence and avoid being stuck in the local minimum, we propose to enhance optimization performance of the DPSO as follows.

3.2.1. Deterministic initialization

The swarm in DPSO, having no prior knowledge of the searching space, is initialized with its particles at random positions. This initialization works well for a relatively small search space.

For large structure, the searching result depends, to a great extent, on the initial positions of the particles. Therefore, in order to increase the probability of reaching the global optimum, we propose to exploit features of viewpoints to generate several seeding particles to facilitate the evolution of the swarm in the search space. In our application, viewpoints are generated based on a grid decomposition. Consequently, a back-and-forth tour would generate a near-optimal path, as shown in Fig. 3, if no obstacles occur. From this observation, positions are deterministically assigned for several particles during the initialization process.

3.2.2. Random mutation

Similar to other evolutionary optimization techniques such as the genetic algorithm or ant colony system, the PSO performs both exploration and exploitation of the search space. Initially, particles are far from each other so they explore different regions in the search space. After evolving through generations, the swarm converges and starts to make more exploitation. At this stage, distances between particles will gradually reduce to the size termed “swarm collapse” [34], whereby many particles will become almost identical.

In order to avoid the collapse situation and keep the balance between exploration and exploitation, random mutations for particles are employed. After every i generations, identical particles are filtered. The remaining are then sorted according to their cost values. Finally, only one-third of the smallest particles are kept for the next generation. All others are disturbed, each in different and randomly-chosen dimensions.

3.2.3. Edge exchange

The enhancement is based on the geometric feature for which crossing edges can be exchanged to result in a shorter tour. Here, as 3D cross checking may be difficult, a complete search similarly to the 2-opt algorithm is employed to compare each valid combination of the swapping mechanism for edges [36]. In this search, every possible exchange of edges is evaluated and the one with the most improvement is chosen. Fig. 4 illustrates the case when an edge exchange between (2, 6) and (3, 7) to shorten the tour. Since this augmentation is computational demanding, it should be used only when the random mutation does not make any difference.

3.2.4. Parallel implementation on GPU

Owing to the rapidly increased performance with thousands of cores, a graphics processing unit (GPU) can outperform the traditional CPUs for problems that are suitable for processed by SIMD (single instruction multiple data). As our optimization algorithms are also a SIMD-based, we can take this advantage to implement in parallel the proposed DPSO in GPUs to reduce computation time.

The diagram and pseudo code for parallel implementation are shown in Figs. 5 and 6 respectively. After initialization, parameters of a particle such as the velocity, position, and fitness are computed in

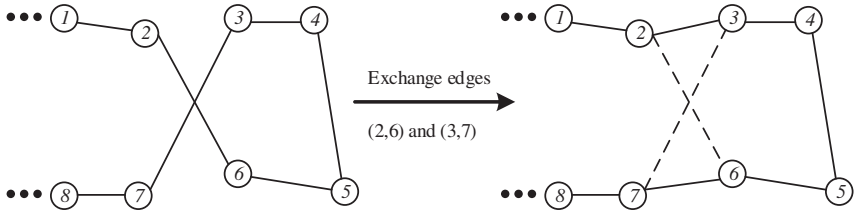

Fig. 4. DPSO augmentation using edge exchange.

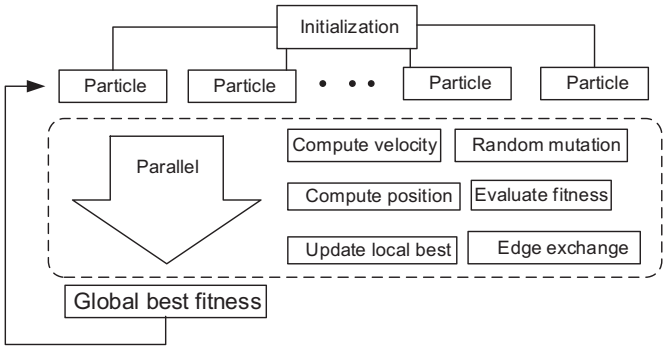

Fig. 5. Parallel implementation of the DPSO on GPU.

parallel, each particle in a different thread. At the end of each generation, the results are saved to the global memory to update these particle parameters and then a new parallel computation round starts.

In UAVs, parallel programs can be implemented in recent onboard computers having good GPU capability and low power consumption such as Jetson TK1 with 192 CUDA Cores (Kepler GPU), 5 W [37]. The board can be configured as either the main or supplemental board communicated with other components via standard communications protocols like MAVLink. However, if the battery power is highly limited as in some micro UAVs, an alternative solution is to stream the sensory data to the ground control station (GCS) and utilise the GPU of a laptop to conduct the path planning. The result is then uploaded to the UAV via GCS for planning/re-planning and navigation.

```
   /* Host:                                                              */
 1 Load w, φ1, φ2, swarm_size to global memory;
 2 Load graph of shortest paths and travelling costs to global memory;
 3 Copy initalized particles from CPU to global memory;   /* see Fig.7 line 1-8 */
 4 Set threads_per_block = swarm_size;
 5 Call kernels to evolve each particle in a seperate thread;
   /* Device:                                                            */
 6 Kernel move_particle(*particles){
 7   Get *particle corresponding to thread_id;
 8   Update position and fitness of particle;          /* see Fig.7 line 10-17 */
 9   Update global_best to global memory and synchronize threads;
10 }
11 Kernel random_mutation(*particles){
12   Sort particles using thrust library;
13   Randomize 2/3 worst particles;
14   Update global_best to global memory and synchronize threads;
15 }
16 Kernel edge_exchange(*graph){
17 foreach i < (number_of_nodes − 2) do
18 |   foreach i < j < (number_of_nodes − 2) do
19 |   |   Swap nodes(i, j) and evaluate fitness;
20 |   end
21 end
22   Update global_best to global memory and synchronize threads;
23 }
```

Fig. 6. Pseudo code for parallel computation of DPSO on GPU.

```
   /* ------------------------ Computation on CPU ------------------------  */
   /* Initialization:                                                       */
 1 Set swarm parameter w, φ1, φ2, swarm_size;
 2 foreach particle in swarm do
 3 |     Initialize particle's position with 10% specific and 90% random;
 4 |     Compute fitness value of each particle;
 5 |     Set local_best value of each particle to itself;
 6 |     Set velocity of each particle to zero;
 7 end
 8 Set global_best to the best fit particle;
   /* ------------------------ Computation on GPU ------------------------  */
   /* Evolutions:                                                           */
 9 repeat
10 |   foreach particle in swarm do
11 |   |   Compute new velocity;                          /* using Eq.11 */
12 |   |   Compute new position;                          /* using Eq.12 */
13 |   |   Update fitness of new position;
14 |   |   if new fitness < local_best then
15 |   |   |   local_best = new fitness;
16 |   |   end
17 |   end
18 |   if current_generation reaches collapsed_cycle then   /* Random mutation */
19 |   |   Sort all particles by fitness;
20 |   |   Randomize 2/3 worst particles;
21 |   end
22 |   Find the particle with the best fitness and update global_best;
23 |   if global_best not improved then                     /* Edge exchange */
24 |   |   foreach particle in swarm do
25 |   |   |   Swap each pair of nodes and evaluate fitness;
26 |   |   |   Choose the swap with best fit;
27 |   |   end
28 |   end
29 until max_generation not reached and
30       global_best not remaining unchanged for a pre-specified number of generations;
```

Fig. 7. Pseudo code of the enhanced DPSO algorithm.

3.3. Enhanced DPSO pseudo code

For vision-based inspection, to take into account obstacle avoidance of the UAV, a selected combination of random and deterministic initialization for each particle in the swarm is performed on a CPU while its evolutions, including computation of updated particles' velocity and position, random mutation and edge exchange, are implemented in parallel on a GPU.

By making use of all advantages of the enhanced DPSO algorithm, the pseudo code for our proposed algorithm incorporating the above-mentioned augmentations is shown in Fig. 7.

4. Experimental results

Experiments have been carried out on two real datasets recorded by laser scanners mounted on a UAV for inspection of an office building and a concrete bridge. The first dataset represents a floor of the building with a size of 25 m × 12 m × 8 m. The second dataset represents a part of the bridge including piers and surfaces with a size of 22 m × 10 m × 4.5 m. Figs. 8a and 9a show the datasets in point cloud representation. In order to apply the IPP algorithm to the datasets, planar surfaces and boundaries need to extracted from them. For this task, we have developed a software for automatic interpretation of unordered point cloud data described in details in [38]. The software uses the Random sample consensus (RANSAC) algorithm combined with data obtained from an inertial measurement unit (IMU) to detect planar surfaces. The convex hull algorithm is then employed to determine their boundaries. The remaining point cloud is clusterized to obstacle objects by finding the nearest neighbour in a 3D Kd-tree structure. Through the software, users are able to select the surfaces they want to inspect, as shown in Figs. 8b and 9b, respectively.

In all experiments, coefficients $w = 1$, $\varphi_1 = 0.4$, $\varphi_2 = 0.4$ are chosen for the DPSO. The number of particles is set to 100. The random mutation is executed in every three generations and the edge exchange

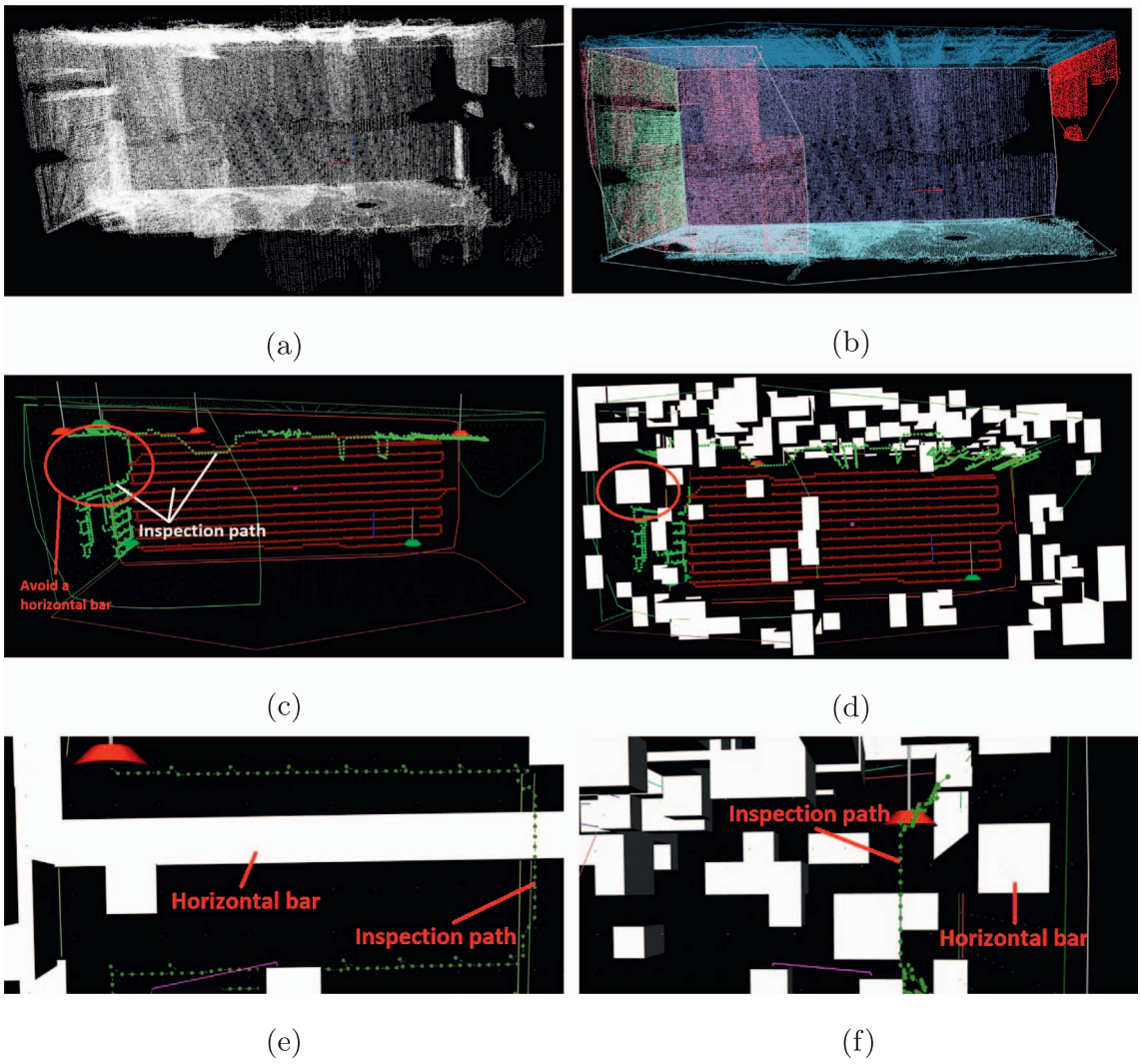

Fig. 8. Experiment with the dataset recording one floor of an office building: (a) Raw data recorded by laser scanners ; (b) Detected planar surfaces and their boundaries; (c) Path planning to inspect the surfaces of ceiling, left wall, and back wall; (d) Inspection path with the appearance of obstacles; (e) Part of inspection path avoiding an obstacle (front view); (f) Part of inspection path avoiding an obstacle (side view);.

is carried out if the random mutation does not improve the result. The parallel implementation is developed based on the CUDA platform. The programs, including both serial and parallel versions, are executed in a laptop computer with Core™i7 CPU and GeForce@ GTX 960M GPU.

4.1. Path generation and DPSO convergence

Figs. 8c and 9c show the paths generated to inspect three selected surfaces of each dataset. Figs. 8d and 9d show the paths in the appearance of obstacles. It can be seen that the back-and-forth pattern is dominant in those paths, except essential changes when having obstacles or switching between surfaces. Fig. 8e and f presents the front and side views of a zoom-in part of the inspection path showing that obstacles were avoided. Fig. 9e and f shows similar results for the bridge dataset.

Fig. 10 shows the graphs of the fitness value as an objective function of the generation number for the two inspection cases of a building and a bridge. In each graph, the fitness represents the cost to traverse the inspection path. From the dataset of the office building, the DPSO by solving the extended TSP improves 22.2% of the travelling cost and converges within 60 generations. For the second dataset of the bridge, those numbers are 37.9% and 80, respectively. The difference is accounted for by the variation in size of the inspection surfaces and the structural complexity of the environments. That is to say in terms of algorithms, care should be given when considering parameters for the exploration (number of particles) and exploitation (number of generations).

4.2. Effect of the augmentations on the DPSO

Table 1 presents the effect of augmentations on the performance improvement over DPSO in percentage by applying our enhanced algorithm. Here, with the dataset obtained from building inspection, the deterministic initialization significantly improves the processing time by 2.8 times and slightly improves the travelling cost by 1.4%. Notably, the computational efficiency in terms of fast convergence actually comes from the improvement of evolving generations of the swarm by means of initialization. On the other hand, it is not surprised that the edge exchange introduces some enhancement on the travelling cost as it uses brute force transpositions. Likewise, the parallel implementation introduces the most significant impact on the computation time thanks to the parallel processing capability taking advantage of the SIMD feature of the DPSO.

To show consistency in the effectiveness of the proposed approach, we compare our enhanced DPSO algorithm not only with the conventional DPSO but also with an ant colony system (ACS), where the ACS is

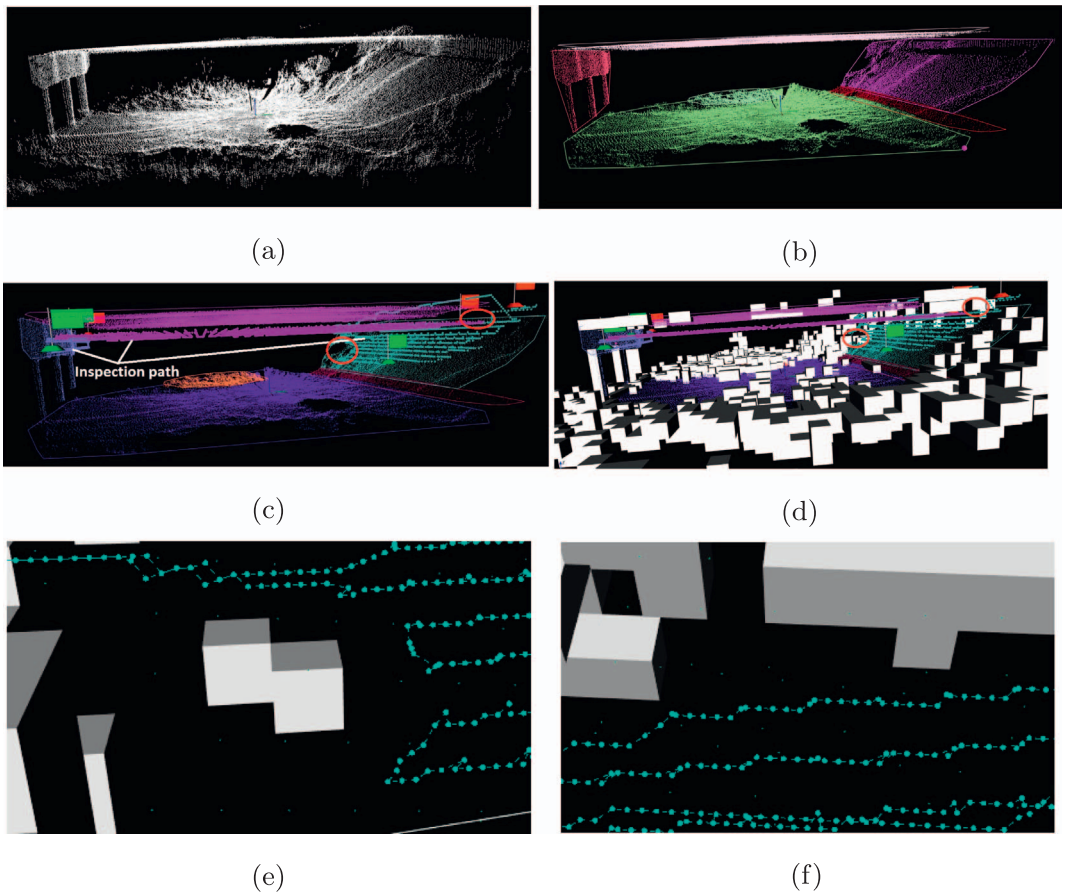

Fig. 9. Experiment with the dataset recording a part of a bridge: (a) Raw data recorded by laser scanners ; (b) Detected planar surfaces and their boundaries; (c) Path planning to inspect the piers, top surface, and slope surface; (d) Inspection path with the appearance of obstacles; (e) Part of inspection path avoiding obstacles at surface bottom; (f) Part of inspection path avoiding obstacles at surface top.

implemented as in [39]. In the comparison, each algorithm was executed over 15 trials. Table 2 shows the results expressed in the average value and the standard deviation of the processing time and the travelling cost. Compared with the ACS algorithm, our enhanced DPSO for the bridge inspection dataset has shown on average an improvement of 15% in the travelling cost and 87 times in the computation time. Owing to a significant improvement in processing time, the enhanced DPSO can be applied for real-time automated inspection.

Table 1
Percent improvement of the DPSO by augmentations.

Algorithm	Building dataset		Bridge dataset	
	Time (%)	Travelling cost (%)	Time (%)	Travelling cost (%)
Initialization	280	1.4	310	1.7
Random mutation	×	5.0	×	5.5
Edge exchange	×	13.8	×	15.7
Parallel on GPU	6570	×	6720	×

×: not applicable.

5. Conclusion

In this paper, we have presented an enhanced discrete particle optimization (DPSO) algorithm for solving the inspection path planning

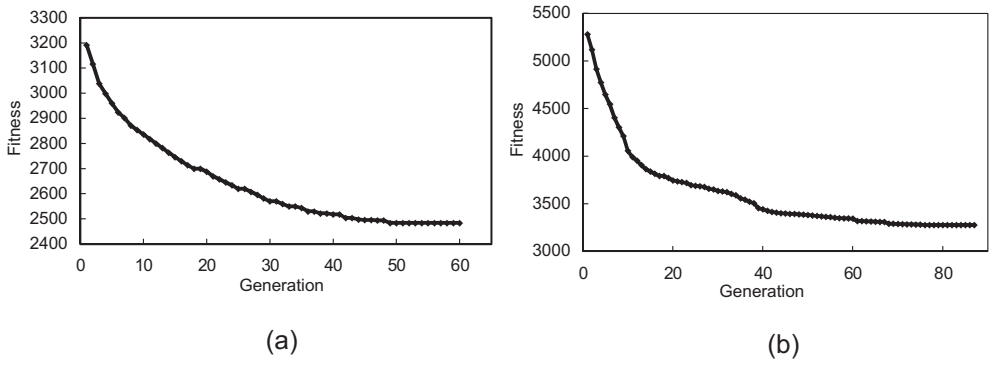

Fig. 10. DPSO convergence from datasets of: (a) Office building; (b) Concrete bridge.

Table 2
Comparison between the enhanced DPSO, DPSO and ACS algorithms.

Algorithm	Building dataset		Bridge dataset	
	Time (s)	Travelling cost	Time (s)	Travelling cost
Enhanced DPSO	32.9 ± 1.2	2490.2 ± 38.9	41.6 ± 1.5	3358.5 ± 59.7
DPSO	2253.8 ± 25.2	2998.1 ± 44.3	2928.4 ± 33.8	4130.7 ± 65.8
ACS	2560.1 ± 16.3	2763.6 ± 56.8	3617.2 ± 23.4	3862.3 ± 87.4

(IPP) problem that is formulated as an extended travelling salesman problem (TSP) considering simultaneously the coverage and obstacle avoidance. By augmenting with deterministic initialization, random mutation, edge exchange and parallel implementation on GPU, the proposed DPSO can greatly improve its performance in both time and travelling cost. The validity and effectiveness of the proposed technique are verified in successful experiments with two real-world datasets collected by UAV inspection of an office building and a concrete bridge. In a future work, the algorithm will be extended for inspection of non-planar surfaces and incorporation of online re-planning strategies to deal with inspection of built infrastructure of an irregular shape.

Acknowledgments

The first author would like to acknowledge an Endeavours Research Fellowship (ERF-PDR-142403-2015) provided by the Australian Government. This work is supported by a University of Technology Sydney Data Arena Research Exhibit Grant 2016 and the Vietnam National University Grant QG.16.29.

References

[1] K.-W. Liao, Y.-T. Lee, Detection of rust defects on steel bridge coatings via digital image recognition, Autom. Constr. 71 (Part 2) (2016) 294–306, http://dx.doi.org/10.1016/j.autcon.2016.08.008.
[2] C.M. Yeum, S.J. Dyke, Vision-based automated crack detection for bridge inspection, Comput. Aided Civ. Inf. Eng. 30 (10) (2015) 759–770, http://dx.doi.org/10.1111/mice.12141.
[3] G. Li, S. He, Y. Ju, K. Du, Long-distance precision inspection method for bridge cracks with image processing, Autom. Constr. 41 (2014) 83–95, http://dx.doi.org/10.1016/j.autcon.2013.10.021.
[4] R. Adhikari, O. Moselhi, A. Bagchi, Image-based retrieval of concrete crack properties for bridge inspection, Autom. Constr. 39 (2014) 180–194, http://dx.doi.org/10.1016/j.autcon.2013.06.011.
[5] R. Zaurin, R. Catbas, Integration of computer imaging and sensor data for structural health monitoring of bridges, Smart Mater. Struct. 19 (1) (2009), http://dx.doi.org/10.1088/0964-1726/19/1/015019.
[6] A.A. Woods, H.M. La, Q. Ha, A novel extended potential field controller for use on aerial robots, Proceedings of the 12Th IEEE International Conference on Automation Science and Engineering (CASE), 2016, pp. 286–291 , http://dx.doi.org/10.1109/COASE.2016.7743420.
[7] A. Ellenberg, A. Kontsos, F. Moon, I. Bartoli, Bridge deck delamination identification from unmanned aerial vehicle infrared imagery, Autom. Constr. 72 (Part 2) (2016) 155–165, http://dx.doi.org/10.1016/j.autcon.2016.08.024.
[8] T. Bock, The future of construction automation: technological disruption and the upcoming ubiquity of robotics, Autom. Constr. 59 (2015) 113–121, http://dx.doi.org/10.1016/j.autcon.2015.07.022.
[9] Y. Yu, N. Kwok, Q. Ha, Color tracking for multiple robot control using a system-on-programmable-chip, Autom. Constr. 20 (2011) 669–676, http://dx.doi.org/10.1016/j.autcon.2011.04.013.
[10] R. Montero, J. Victores, S. Martnez, A. Jardn, C. Balaguer, Past, present and future of robotic tunnel inspection, Autom. Constr. 59 (2015) 99–112, http://dx.doi.org/10.1016/j.autcon.2015.02.003.
[11] B. Zhang, W. Liu, Z. Mao, J. Liu, L. Shen, Cooperative and geometric learning algorithm (CGLA) for path planning of UAVs with limited information, Automatica 50 (3) (2014) 809–820, http://dx.doi.org/10.1016/j.automatica.2013.12.035.
[12] H. Choset, Coverage for robotics - a survey of recent results, Ann. Math. Artif. Intell. 31 (2001) 113–126, http://dx.doi.org/10.1023/A:1016639210559.
[13] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, S. Thrun (Eds.), Principles of robot motion: theory, algorithms, and implementation, The MIT Press, 2005.
[14] E. Acar, H. Choset, A. Rizzi, P. Atkar, D. Hull, Morse decompositions for coverage tasks, Int. J. Robot. Res. 21 (4) (2002) 331–344, http://dx.doi.org/10.1177/027836402320556359.
[15] V. Shivashankar, R. Jain, U. Kuter, D. Nau, Real-time planning for covering an initially-unknown spatial environment, Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, 2011, pp. 63–68 iSBN 978-1-57735-501-4.
[16] Y. Gabriely, E. Rimon, Spiral-stc: an on-line coverage algorithm of grid environments by a mobile robot, Proceedings of the IEEE International Conference in Robotics and Automation (ICRA), vol. 1, 2002, pp. 954–960 , http://dx.doi.org/10.1109/ROBOT.2002.1013479.
[17] B. Englot, F. Hover, Three-dimensional coverage planning for an underwater inspection robot, Int. J. Robot. Res. 32 (9-10) (2013) 1048–1073, http://dx.doi.org/10.1177/0278364913490046.
[18] G.A. Hollinger, B. Englot, F.S. Hover, U. Mitra, G.S. Sukhatme, Active planning for underwater inspection and the benefit of adaptivity, Int. J. Robot. Res. 32 (1) (2013) 3–18, http://dx.doi.org/10.1177/0278364912467485.
[19] P. Janousek, J. Faigl, Speeding up coverage queries in 3D multi-goal path planning, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2013, pp. 5082–5087 , http://dx.doi.org/10.1109/ICRA.2013.6631303.
[20] P. Wang, K. Gupta, R. Krishnamurti, Some complexity results for metric view planning problem with traveling cost and visibility range, IEEE Trans. Autom. Sci. Eng. 8 (3) (2011) 654–659, http://dx.doi.org/10.1109/TASE.2011.2123888.
[21] P.S. Blaer, P.K. Allen, View planning and automated data acquisition for three-dimensional modeling of complex sites, J. Field Rob. 26 (11-12) (2009) 865–891, http://dx.doi.org/10.1002/rob.20318.
[22] M. Saha, T. Roughgarden, J.-C. Latombe, G. Snchez-Ante, Planning tours of robotic arms among partitioned goals, Int. J. Robot. Res. 25 (3) (2006) 207–223, http://dx.doi.org/10.1177/0278364906061705.
[23] T. Danner, L.E. Kavraki, Randomized planning for short inspection paths, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2000, pp. 971–976 , http://dx.doi.org/10.1109/ROBOT.2000.844726.
[24] B. Englot, F. Hover, Planning complex inspection tasks using redundant roadmaps, in: H.I. Christensen, O. Khatib (Eds.), Robotics Research, Springer Tracts in Advanced Robotics, vol. 100, Springer International Publishing, 2016, pp. 327–343 , http://dx.doi.org/10.1007/978-3-319-29363-9_19.
[25] D. Applegate, W. Cook, A. Rowe, Chained Lin-Kernighan for large traveling salesman problems, INFORMS J. Comput. 15 (1) (2003) 82–92, http://dx.doi.org/10.1287/ijoc.15.1.82.15157.
[26] G. Papadopoulos, H. Kurniawat, N. Patrikalakis, Asymptotically optimal inspection planning using systems with differential constraints, Proceedings of the IEEE International Conference in Robotics and Automation (ICRA), 2013, pp. 4126–4133 , http://dx.doi.org/10.1109/ICRA.2013.6631159.
[27] G. Hollinger, B. Englot, F. Hover, U. Mitra, G. Sukhatme, Uncertainty-driven view planning for underwater inspection, Proceedings of the IEEE International Conference in Robotics and Automation (ICRA), 2012, pp. 4884–4891 , http://dx.doi.org/10.1109/ICRA.2012.6224726.
[28] P. Jimenez, B. Shirinzadeh, A. Nicholson, G. Alici, Optimal area covering using genetic algorithms, Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 2007, pp. 1–5 , http://dx.doi.org/10.1109/AIM.2007.4412480.
[29] C. Goerzen, Z. Kong, B. Mettler, A survey of motion planning algorithms from the perspective of autonomous UAV guidance, J. Intell. Robot. Syst. 57 (1) (2009) 65, http://dx.doi.org/10.1007/s10846-009-9383-1.
[30] N. Dadkhah, B. Mettler, Survey of motion planning literature in the presence of uncertainty: considerations for UAV guidance, J. Intell. Robot. Syst. 65 (1) (2012) 233–246, http://dx.doi.org/10.1007/s10846-011-9642-9.
[31] S. Scherer, S. Singh, L. Chamberlain, M. Elgersma, Flying fast and low among obstacles: methodology and experiments, Int. J. Robot. Res. 27 (5) (2008) 549–574, http://dx.doi.org/10.1177/0278364908090949.
[32] D. Jung, P. Tsiotras, On-line path generation for unmanned aerial vehicles using b-spline path templates, J. Guid. Control. Dyn. 36 (6) (2013) 1642–1653, http://dx.doi.org/10.2514/1.60780.
[33] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107, http://dx.doi.org/10.1109/TSSC.1968.300136.
[34] J. Kennedy, R. Eberhart, Y. Shi (Eds.), Swarm Intelligence, Morgan Kaufmann, 2001.
[35] M. Clerc, Discrete particle swarm optimization, illustrated by the traveling salesman problem, in: G. Onwubolu, B. Babu (Eds.), New Optimization Techniques in Engineering, Studies in Fuzziness and Soft Computing, vol. 141, Springer Berlin Heidelberg, 2004, pp. 219–239 , http://dx.doi.org/10.1007/978-3-540-39930-8_8.
[36] G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and its Variations, Springer US, 2007, http://dx.doi.org/10.1007/b101971.
[37] Y. Ukidave, D. Kaeli, U. Gupta, K. Keville, Performance of the NVIDIA Jetson TK1 in HPC, 2015 IEEE International Conference on Cluster Computing (CLUSTER), 2015, pp. 533–534 , http://dx.doi.org/10.1109/CLUSTER.2015.147.
[38] M. Phung, C. Quach, D. Chu, N. Nguyen, T. Dinh, Q. Ha, automatic interpretation of unordered point cloud data for UAV navigation in construction, Proceedings of the 14Th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2016, http://dx.doi.org/10.1109/ICARCV.2016.7838683.
[39] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Trans. Evol. Comput. 1 (1) (1997) 53–66, http://dx.doi.org/10.1109/4235.585892.