J Ambient Intell Human Comput (2015) 6:773–782
DOI 10.1007/s12652-015-0282-y

ORIGINAL RESEARCH

Parallel and distributed computing for UAVs trajectory planning

Domenico Pascarella[1] · **Salvatore Venticinque**[2] · **Rocco Aversa**[2] · **Massimiliano Mattei**[2] · **Luciano Blasi**[2]

Received: 1 December 2014 / Accepted: 28 April 2015 / Published online: 10 May 2015

Abstract The problem of generating optimal flight trajectories for an unmanned aerial vehicle in the presence of no-fly zones is computationally expensive. It is usually solved offline, at least for those parts which cannot satisfy real time constraints. An example is the core paths graph algorithm, which discretizes the operational flight scenario with a finite dimensional grid of positions-directions pairs. A weighted and oriented graph is then defined, wherein the nodes are the earlier mentioned grid points and the arcs represent minimum length trajectories compliant with obstacle avoidance constraints. This paper investigates the exploitation of two parallel programming techniques to reduce the lead time of the core paths graph algorithm. The former employs some parallelization techniques for multi-core and/or multi-processor platforms. The latter is targeted to a distributed fleet of unmanned aerial vehicles. Here the statement of the problem and preliminary development are discussed. A two-dimensional scenario is analysed by way of example to show the applicability and the effectiveness of the approaches.

✉ Domenico Pascarella
d.pascarella@cira.it

Salvatore Venticinque
salvatore.venticinque@unina2.it

Rocco Aversa
rocco.aversa@unina2.it

Massimiliano Mattei
massimiliano.mattei@unina2.it

Luciano Blasi
luciano.blasi@unina2.it

1 Soft Computing Laboratory, CIRA (Italian Aerospace Research Centre), Capua, Italy

2 Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italy

Keywords UAV · Trajectory planning · Core paths graph · Parallel computing

1 Introduction

Unmanned aerial vehicles (UAVs) have being received ever increasing attention in the scientific literature. Recent advances in their technology have allowed the emergence of a wide new range of applications, such as military operations (Schumacher et al. 2007), disaster management (Quaritsch et al. 2010), agricultural surveillance (Israel 2011), urban terrain surveillance (Gross et al. 2006), etc.

Without the need of an on-board pilot, an aerial vehicle can be designed to accomplish the so-called D-cube (Dull, Dangerous, Dirty) missions (Ingham 2008). Dull operations are too monotonous or require excessive endurance for human occupants. Dirty operations are hazardous and could pose a health risk to a human crew. Dangerous operations could result in the loss of life for the on-board pilot.

Nevertheless, as outlined in Pascarella et al. (2013), nowadays UAVs are mostly Remotely Piloted Vehicles (RPVs), since their operations are remotely performed by large teams of human operators. Ground operators must be endowed with the proper expertise and this represents a significant cost factor. Furthermore, dull tasks could relieve the remote pilots if they were performed autonomously.

This consideration pushes the necessity to extend aerial platforms capabilities related to autonomy. Data link constraints are another important motivation for autonomy in unmanned systems. Bandwidth is a limited resource and communications with the ground station could be not available in emergency situations. On-board autonomous subsystems could also provide faster reactions to external context changes due to their direct access to sensor data.

Generating optimal flight trajectories that are compliant with the problem constraints is a common problem in autonomous UAVs (Mattei and Blasi 2010). Constraints derive from the mission scenario (such as the avoidance of no-fly zones) and from the vehicle dynamics (e.g., the turn radius, the climb and descent angles, etc.). These algorithms pose a not negligible computational cost which, while not affecting offline applications, has a remarkable impact on possible online applications.

The online trajectory optimization involves an online processing of the flight trajectories to allow for a time-variant mission scenario, such as the occurrence of new obstacles. The problem is further complicated because real time replanning to account any change into the mission objectives and scenarios is a mandatory requirement in these kinds of applications and implies the invariable completion of the reaction to an external stimulus within an hard deadline.

Thus, the reference problem for an online application is a real time planning and replanning problem, that is a dynamic optimization problem. As a consequence, a strategy for an improvement of the temporal behavior is essential for an effective implementation.

A promising method for the speed-up of the trajectory planning process is the parallelization of the computation, since current processors have more cores per micro-chip instead of single higher clock rates. Moreover, such an algorithm may be fitted in with future developments related to collaborative networks of UAVs. This is a suitable solution for complex planning in a distributed environment (i.e., partially controllable and not observable by a single entity).

As stated in Shima and Rasmussen (2009), a loose collection of vehicles that have some objectives in common is a collaborative team. If the vehicles are working together to achieve a common objective, then they are a cooperative team. The main motivation for team cooperation stems from the possible synergy, as the group performance is expected to exceed the sum of the performance of the single UAVs. Such cooperation entails: global information, because each UAV can share its sensor information with the rest of the group via a communication network; resource management, because a cooperative decision algorithm allows efficient allocation of the overall group resources over the multiple targets; robustness, because the team can reconfigure its distributed architecture in order to minimize the performance degradation if a failure occurs.

Here we investigate the parallelization of an algorithm that converts the trajectory optimization problem into a minimum cost path search in a weighted and oriented graph, called core paths graph (CPG) in Mattei and Blasi (2010).

The CPG defines a discrete set of admissible connection paths in the space domain. The weights of the CPG arcs are obtained solving convex quadratic programming optimization problems. The particular case of piecewise polynomial trajectories minimizing flight paths length is fully developed in Mattei and Blasi (2010). The CPG process can also exploit a distributed implementation that is endorsed on a fleet of cooperative UAVs. In detail, we are interested in broadening the basic CPG algorithm discussed in Mattei and Blasi (2010) in order to reduce its lead time for online operations.

This paper is an extended version of Pascarella et al. (2014). Compared with the previous work, here we provide a detailed state-of-the-art survey, an in-depth analysis of the test results and some further considerations about the distributed variation of the CPG algorithm.

2 Related work

During the past years, a lot of work has been carried out on the trajectory optimization for many kinds of vehicles. The variational formulation is probably the most natural and rigorous one for this class of problems. However, the possibility to solve complex problems with variational methods is very poor.

Many papers deal with numerical direct and indirect methods based on the solution of a non linear programming (NLP) problem (Betts 1998; Hargraves and Paris 1987). Unfortunately NLP turns out to be quite burdensome for many practical applications.

By means of some approximations, feasible trajectories can be generated following a purely geometrical approach based on topological techniques creating a sequence of way points. This sequence can derive from probabilistic or potential methods (Hwang and Ahuja 1992; Cen et al. 2007). A purely geometrical approach has been proposed in Asseo (1998) for flight trajectories generation using circular arcs combined with straight connection paths. The A* search algorithm is applied in Yang and Zhao (2004) to generate trajectories over a four-dimensional discretized search space.

An interesting technique, taking into account also flight dynamics, is based on the so called "motion primitives" (Dever et al. 2006; Frazzoli et al. 2002), where flight path is defined through a series of trim conditions and manoeuvres. Definition of smooth trajectories based on Dubins curves compliant with curvature as well as operational constraints are presented in Anderson et al. (2005) and in Chitsaz and LaValle (2007). Different approaches based on mixed integer logical programming (MILP) with a Receding Horizon philosophy have also been proposed (Schouwenaars et al. 2004; Borrelli et al. 2006).

Non-deterministic methods also have received much attention showing their effectiveness and robustness in a wide range of engineering problems. An example of a space plane re-entry trajectory optimization can be found in Yokoyama and Suzuki (2005), whereas hybrid soft computing and evolutionary techniques were used in Vaidyanathan et al. (2001) to compute optimum flight path for UAVs under several aerodynamic constraints.

A real-time free flight path optimization based on improved genetic algorithms is reported in Hu et al. (2004). A numerical potential field combined with a genetic optimizer has been applied for mobile robot path planning in Lei et al. (2006). Again in the field of evolutionary techniques, an example of path planning based on a particle swarm optimizer (PSO) can be found in Raja and Pugazhenhi (2009).

In Li et al. (2006) a mixed fuzzy-PSO technique has been implemented and applied to identify the optimum route over a planning area represented by a mesh of equal square cells. In Saska et al. (2006) smooth trajectories are described through cubic splines whose parameters are optimized by a PSO-based procedure. A further example of PSO application in the field of flight path optimization in a constrained environment can be found in Blasi et al. (2013). An example of path planning in a constrained environment based on a particle swarm optimizer (PSO) can be found in Blasi et al. (2013). Alternative approaches related to hybrid simulation are used in Ficco et al. (2014) and in Gigante et al. (2015).

A few methods for the parallelization of trajectory planning have been discussed in the scientific literature. For example, Kopřiva et al. (2012) presents a parallelization approach of the AA* algorithm. The authors in Kider et al. (2010) introduce R*GPU, a parallel extension of R*, whereas a parallel extension of probabilistic roadmaps is reported in Pan et al. (2010).

Several works are also available as regards the multi-vehicle routing problem by means of decentralized and cooperative management, such as Murray (2007) and Faied et al. (2010).

Anyway, this is the first proposal of a parallelized and a distributed extension of the CPG algorithm.

3 The mathematical problem formulation

Let us consider a two-dimensional space domain $\Delta \subseteq \mathbb{R}^2$ in which to fly, possibly non connected, with a boundary $\delta\Delta$ that is determined by the presence of no-fly zones and/or obstacles. A parametric trajectory function in Δ is denoted with

$$s(\cdot) : t \in [t_0, t_f] \rightarrow s(t) = \big(x(t), y(t), z(t)\big) \tag{1}$$

where (x, y, z) are the coordinates in a given Cartesian reference system.

We denote with $\dot{s} = \frac{ds}{dt}$ the first-order temporal derivative of the position s and with $E = (s, \dot{s})$ the vector of positions and their first derivatives.

A flight trajectory from $E_0 = (s_0, \dot{s}_0)$ to $E_f = (s_f, \dot{s}_f)$ in Δ is a continuously differentiable C^1 oriented curve θ_{E_0,E_f} in the independent variable t, connecting the point s_0 to the point s_f with assigned first derivatives (tangents) $\dot{s_0}$ and $\dot{s_f}$ and at the extremes. Thus, the flight trajectory θ_{E_0,E_f} has the following expression

$$\theta_{E_0,E_f} \triangleq \Big\{ s_{[t_0,t_f]}(\cdot) \in C^1_{[t_0,t_f]} : s(t_0) = s_0, \quad s(t_f) = s_f, \quad \dot{s}(t_0) = \dot{s}_0, \quad \dot{s}(t_f) = \dot{s}_f \Big\} \tag{2}$$

Other essential properties for a flight trajectory (e.g., Δ-compatibility, Δ-admissibility, Σ-admissibility, controllability, etc.) are thoroughly described in Mattei and Blasi (2010).

The cost of every admissible flight trajectory θ_{E_0,E_f} is a real nonnegative function $C(\cdot)$, which may be related to the path length or height, to the fuel consumption, to the risk or to other variables inherent to the flight mission.

The search of the optimal flight trajectory connecting E_0 to E_f consists in finding a flight trajectory $\theta^*_{E_0,E_f}$ that is compliant with the desired properties and is a solution of the minimization problem

$$\theta^*_{E_0,E_f} = \underset{\theta_{E_0,E_f} \in \Theta^{\Delta,\Sigma}}{\operatorname{argmin}} \; C\big(\theta_{E_0,E_f}\big) \tag{3}$$

The set of all the Σ-admissible optimal flight trajectories connecting every Δ-compatible E with each other is called the optimal Δ-connection set.

The problem (3) and the search of the optimal Δ-connection set are complex tasks because the searching space $\Theta^{\Delta,\Sigma}_{E_0,E_f}$ is infinite dimensional, the cost function can be nonlinear and non-convex and the constraints are generally nonlinear and non-convex.

Some assumptions and approximations are introduced in order to convert the optimization problem (3) into a minimum cost path search over a graph. A Δ-core paths graph (ΔCPG) is a discrete approximation of the optimal Δ-connection set, wherein the nodes are suitable Δ-compatible E vectors and the arcs are Σ-admissible optimal flight trajectories connecting the nodes. We denote with N_{CPG} the set of CPG nodes and with Υ_{CPG} the set of CPG arcs. The weight $W_{i,j}$ of the arc connecting the i-th node with the j-th node represents the cost of the optimal flight trajectory from E_i to E_j.

The authors in Mattei and Blasi (2010) prove that the infinite dimensional problem (3) to calculate the CPG weights $W_{i,j}$ can be converted into the following finite dimensional problem

$$W_{i,j} = \min_{\delta \in \Omega} \left(\theta^f_{E_i,E_j}(\delta) \right), \quad \forall E_i, E_j \in \mathrm{N}_{CPG} \tag{4}$$

wherein Ω is the set of parameters δ ensuring Σ-admissible flight trajectories.

Once a CPG has been built and the weights $W_{i,j}$ have been calculated, the optimal flight trajectory between two nodes of the graph can be determined adopting a search algorithm for a minimum cost path in a graph. In Mattei and Blasi (2010) polynomial functions are used for the flight trajectory parameterization. The trajectory $\theta^f_{E_i,E_j}(\delta)$ can be expressed as

$$\begin{aligned} \theta^f_{E_i,E_j}(\delta) \triangleq \{ y(x) = \delta_1 x^{p-1} + \cdots + \delta_{p-1} x + \delta_p : \\ x \in [x_0, x_f], \; \delta \in \Omega \subseteq \mathbb{R}^p, \\ y(x_0) = x_0, \; y(x_f) = y_f, \\ \dot{y}(x_0) = \dot{y}_0, \; \dot{y}(x_f) = \dot{y}_f \} \end{aligned} \tag{5}$$

The equalities constraints in (5) and the obstacle avoidance requirements can be converted in a matrix notation, as described in Mattei and Blasi (2010). Moreover, if we assume that the cost of a trajectory is proportional to its length, the cost function can be represented as a quadratic function of δ and the optimization problem (4) can be converted into

$$\begin{aligned} W_{i,j} = \min_{\delta \in \Omega \subseteq \mathbb{R}^p} \; & \delta^T Q \delta \\ \text{s.t.} \; & \\ & A_I \delta \leq b_I \\ & A_E \delta \leq b_E \end{aligned} \tag{6}$$

The problem (6) is a convex quadratic programming problem and can be efficiently solved using quadratic programming algorithms. The matrices Q, A_I, A_E and the vectors b_I, b_E are accurately described in Mattei and Blasi (2010).

4 CPG algorithm

The CPG computation includes first an offline processing phase, which entails the setting of the CPG topology and the evaluation of the weights of the arcs. The topology is built by choosing the set N_{CPG} of CPG nodes. A criterion can be a uniform grid of points and directions with increased density in the proximity of no-fly zones and obstacles. Once a cost function $C(\cdot)$ has been selected, the weights $W_{i,j}$ of the arcs can be determined by solving the problem (6) if we adopt polynomial trajectories.

The verification of the desired properties for $W_{i,j}$ is another important step. This verification is carried out into the following three different phases:

1. evaluation of the properties that can be directly verified on the relations between E_i and E_j (such as distance limitations among the extreme points s_i and s_j): if these properties are not satisfied, there is not an arc between E_i and E_j;
2. evaluation of the solvability of the problem (6): if this has no admissible solutions for (E_i, E_j), then there is not an arc between E_i and E_j;
3. evaluation of a posteriori checks on the optimal trajectory between E_i and E_j: if the solution of the problem (6) does not respect other Σ-properties (not verified at point 1, such as obstacles avoidance), then the optimal trajectory for (E_i, E_j) is not feasible and there is not an arc between E_i and E_j.

The pseudo-code for the CPG algorithm is composed by Algorithm 1 and Algorithm 2. The parametric structure *scenario* contains the relevant environment data (i.e., the space domain, no-fly zones, obstacles, etc.), $\mathrm{N}_{CPG_{points}}$ and $\mathrm{N}_{CPG_{directions}}$ are respectively the set of points and the set of directions of the CPG nodes and A_{CPG} is the adjacency matrix of the CPG. The check functions perform the properties verification.

Algorithm 1: CPG_algorithm (*scenario*)

```
begin
    $(\mathrm{N}_{CPG_{points}}, \mathrm{N}_{CPG_{directions}}) \longleftarrow$ set_CPG_topology($scenario$)
    $A_{CPG} \longleftarrow$ CPG_arcs_computing$(\mathrm{N}_{CPG_{points}}, \mathrm{N}_{CPG_{directions}})$
end
```

Algorithm 2: CPG_arcs_computing $(\mathrm{N}_{CPG_{points}}, \mathrm{N}_{CPG_{directions}})$

```
begin
    for $i \longleftarrow 1$ to size$(\mathrm{N}_{CPG_{points}})$ do
        for $j \longleftarrow 1$ to size$(\mathrm{N}_{CPG_{directions}})$ do
            $end\_points \longleftarrow$ feasibility_points_check$(i, j)$
            for $k \longleftarrow 1$ to size$(end\_points)$ do
                $end\_directions \longleftarrow$
                feasibility_directions_check$(end\_points)$
                for $l \longleftarrow 1$ to size$(end\_directions)$ do
                    $trajectory \longleftarrow$
                    quadratic_optimization$(i, j, k, l)$
                    if $trajectory \neq$ null then
                        $admissibility \longleftarrow$
                        post-feasibility_check$(trajectory, scenario)$

                        if $admissibility =$ true then
                            $set\_arc(i, j, k, l,$ cost$(trajectory))$
                        end
                    end
                end
            end
        end
    end
end
```

The lead time of the CPG algorithm depends on the number of nodes $|\mathrm{N}_{CPG}| = \sum_{i=0}^{|\mathrm{N}_{CPG_{points}}|} |\mathrm{N}_{CPG_{directions}}|_i$, wherein $|\mathrm{N}_{CPG_{directions}}|_i$ is the number of directions for the i-th point.

A theoretical estimation of the lead time should take into account the computational complexity of the `CPG_arcs_computing` function. If we suppose that the number of directions is the same for every point, the computational complexity is

$$O\left(|\mathrm{N}_{CPG_{directions}}|^2 \cdot |\mathrm{N}_{CPG_{points}}| \cdot \left(|\mathrm{N}_{CPG_{points}}| - 1\right) \cdot T_{QP}\right) \quad (7)$$

wherein T_{QP} is the computational complexity of the quadratic programming solver. For instance, primal-dual interior point methods for convex quadratic programming usually exhibit polynomial-time complexity (Monteiro et al. 1990).

The online adaptation of the CPG algorithm solicits an enlargement of the CPG against any change in the scenario. The enlargement is in way of including the vertexes that are related to new no-fly zones or obstacles and the current vehicle position. As a consequence of the update of the CPG topology, `CPG_arcs_computing` function is performed in online fashion starting from the new nodes.

5 Parallel CPG algorithm

The CPG algorithm is computationally heavy due to the size of the graph in a real scenario and some expedients are convenient in order to make the proposed approach less time-consuming for an online adaptation of the algorithm. A proper choice of the discrete set of CPG nodes, of the Σ-properties and of the cost function can already reduce the required time for the CPG construction.

Anyway, the algorithm can benefit from the use of multi-core and/or multi-processor platforms since the CPG construction can be formulated as a parallel process. It does not present loop-carried dependencies, namely one iteration of the loop does not depend upon the results of other iterations of the loop and the loop can be executed in any order, even in parallel. There exists a strict one-to-one relation between the iteration order (i, j, k, l) and the evaluated arc, therefore a CPG parallel process does not imply any potential race condition. Indeed, the arc weights computations do not depend on the sequence or timing of the parallel threads.

An OpenMP implementation of a parallel CPG algorithm is addressed here. OpenMP is an application program interface (API), jointly defined by a group of major computer hardware and vendors (OpenMP 2013). It provides a portable and scalable model for developers of shared memory and multi-threaded parallel applications. The API supports C/C++ and Fortran on several architectures. The adopted programming language for the CPG algorithm is C, wherein the OpenMP consists of a set of compiler pragmas that control how the program parallelism works.

The application begins with a single thread, that is the master thread. As the program executes, the application may encounter parallel regions in which the master thread creates a thread team. Each thread member of the team executes within the parallel region. The end of the parallel region marks implicitly a barrier synchronization for all the threads, that will wait here until the last thread completes. Afterwards, the thread team is removed and the master thread continues the sequential execution.

The OpenMP for directive splits a for-loop into a parallel region so that every thread gets different sections of the loop and it executes its own sections in parallel. In the case of the CPG algorithm, one or more of the for-loops can be parallelized amongst a team of threads. Each thread can handle some iterations of the loop, i.e., some arcs of the graph. As a consequence of the one-to-one relations between iterations and arcs, the CPG data structure does not have to be protected from concurrent accesses and the overhead due to OpenMP parallelism management is minimum.

Here we report the parallel implementation of the second-level for-loop of the CPG algorithm by way of example. This loop scans all the directions starting from the current CPG point and proceeds with the feasibility checks and the arcs processing by executing the lower-levels for-loops.

The OOQP (Object-Oriented software for quadratic programming) package (Gertz and Wright 2001) has been used as regards the resolution of the problem (6). OOQP is a C++ package for solving convex quadratic programming problems. It employs object-oriented programming techniques for a primal-dual interior point algorithm. It contains code that can be used out of the box to solve a variety of quadratic programming problems. It can be used as a standalone off-the-shelf solver, as an embeddable solver (i.e., as an integrated code in a custom application) or as a development framework.

Within this work, we have used OOQP as an embedded solver into a C application for CPG algorithm. In the following, we briefly describe the test scenario and the test results.

5.1 Test scenario and test results

The considered test scenario is a two-dimensional example with two circular no-fly zones in a rectangular space region Δ (Fig. 1), so we suppose that the aerial vehicle moves only in a plane (i.e., at a constant height). We also assume that the CPG nodes are the combination of the 115 points

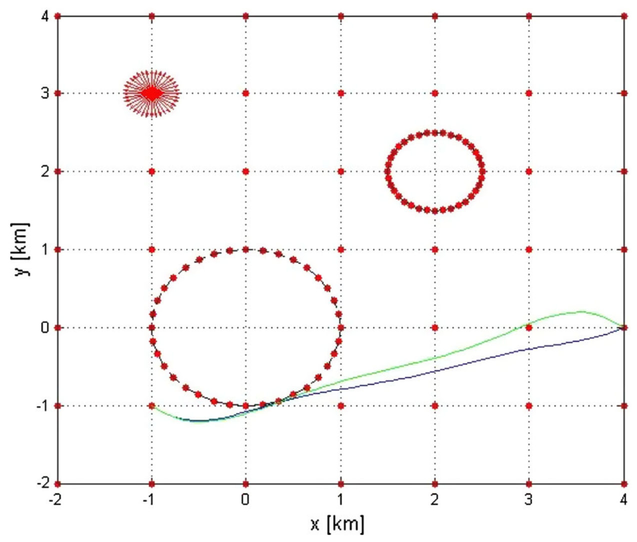

Fig. 1 Two-dimensional test scenario

marked with circles in Fig. 1 and of a discrete set of 36 equally spaced directions identified with the star of arrows centred in the point $(-1, 3)$ in such a figure.

This choice produces a set of 4140 nodes and a number Υ_{CPG} of arcs that is equal to $|\mathrm{N}_{CPG}| \cdot (\mathrm{N}_{CPG} - 1)$ in the case of a complete graph without auto-connections. Nevertheless, some nodes are not Δ-compatible and further reduction of the arcs is obtained by applying the following Σ properties: a distance limitation of 5 km among the connected points and a difference limitation of 70° among the extreme directions. Finally, we consider 8th-order polynomial trajectories to connect nodes, namely $p = 9$. Constraints on the radius of curvature of the vehicle have not been considered.

Figure 1 shows two sample trajectories through respectively a green and a blue line. They both start from the point $(-1, -1)$ with a direction of $-40°$ and arrive at the point (4, 0) (the former with a direction of $-40°$, the latter with a direction of 30°).

A general purpose workstation with Intel Core i5- 2520 M has been used as testbed. It is a processor with a clock speed of 2.50 GHz and two physical cores. It supports hyper-threading technology, so every physical core consists of two logical cores, which share the execution resources, but have their own architectural state. From the operating system's point of view, logical cores are handled like physical cores, therefore the Intel Core i5-2520 M processor exhibits 4 cores. The authors in Russ and Stütz (2012) and in Meier et al. (2011) point out some UAVs applications which have a on-board computing power that is comparable to our test platform's computing power.

Table 1 reports the runtime performances for the proposed parallel CPG implementation. In detail, we have

Table 1 Measures in seconds of the wall-clock time of the parallelized for-loop iteration

#Threads	Iterations				
	#1	#2	#3	#4	#5
1	161.08	338.93	326.99	320.57	291.30
2	92.29	257.95	235.57	212.04	186.29
3	87.11	199.26	180.81	164.03	151.54
4	81.46	161.53	153.33	150.36	134.47
5	80.33	159.25	151.88	147.36	131.67
6	73.81	158.11	145.40	136.71	124.01
7	74.50	165.33	144.40	134.98	127.84
8	73.04	150.03	147.45	146.36	129.62

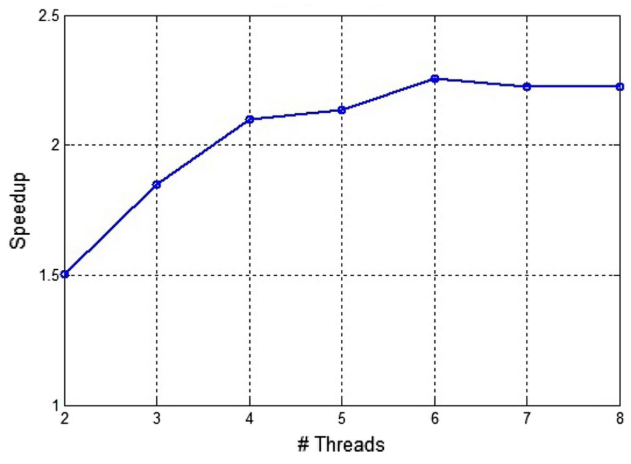

Fig. 2 Average speedup of the parallel CPG implementation

measured the wall-clock times of the first 5 iterations of the parallelized for-loop (the second-level for-loop) of the CPG algorithm. Measurements have been accomplished by means of `omp_get_wtime` function. Each measure in Table 1 is the average of 10 runs. A single optimal trajectory is computed in about 0.3 s. Best results have been obtained with 8 threads.

Figures 2 and 3 respectively show the speedup and the efficiency of the parallel CPG implementation. Speedup and efficiency are defined by the following formulas

$$Speedup = \frac{T_{sequential}}{T_{parallel}} \tag{8}$$

$$Efficiency = \frac{Speedup}{N_{cores}} \tag{9}$$

wherein $T_{sequential}$ is the wall-clock time of the sequential for-loop iteration, $T_{parallel}$ is the wall-clock time of the parallelized for-loop iteration, $N_{threads}$ is the number of threads in the team and N_{cores} is the number of (logical)

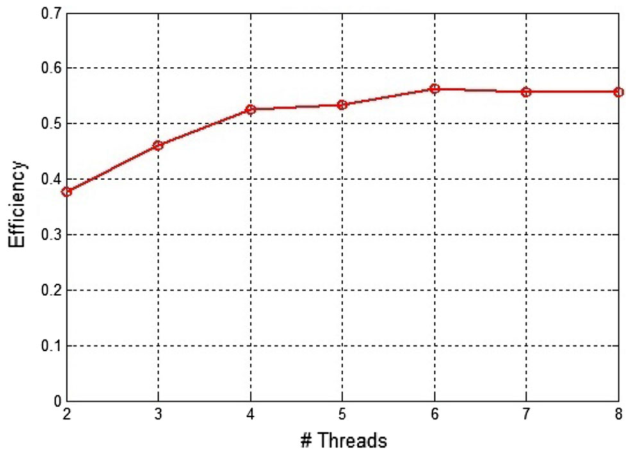

Fig. 3 Average efficiency of the parallel CPG implementation

processor cores (4 for the Intel Core i5-2520M). $T_{sequential}$ and $T_{parallel}$ have been evaluated by summing the measured wall-clock times of each iteration fixed the thread number.

The speedup and the efficiency (Figs. 2, 3) initially grow because the number of threads is lower than the number of cores. They settle in the region in which $N_{threads}$ is slightly larger than N_{cores} because all the available resources have been employed for the parallelization and the overhead effects are significant.

6 Distributed CPG algorithm

A fleet of collaborative UAVs can be arranged as a distributed system, that is a system in which the individual UAVs are located on a network and coordinate their actions by passing messages. In this case, the problem (6) can be distributed amongst the team of vehicles. Indeed, a distributed CPG algorithm would reduce the global processing time by taking advantage of the computing powers of the single vehicle platforms. Moreover, a distributed approach may be compliant with the key requirements of robustness, scalability and flexibility for the high-level control of a fleet of UAVs. For example, a centralized planner represents a single point of failure and its complexity quickly grows with the problem dimension, accordingly to the expression (7). For this reason, we focus on the use of a distributed approach in which the individual vehicles can work on a partition of the CPG.

However, an effective protocol for the fleet allocation shall be set up and the coordination may require the vehicles to exchange large quantities of information about the environment and the optimal trajectories. Constraining the communication may be unavoidable, although it potentially limits the situational awareness of the single UAVs and their cooperative capacities. Hence, a compromise between a global situational awareness and acceptable performances shall be achieved.

Here we assume that the fleet is homogeneous, so the problem (6) has the same formulation and the same parameters for all the UAVs members. The pseudo-code for the distributed strategy is composed by Algorithm 2 and Algorithm 3.

In Algorithm 3, N_v is the number of vehicles in the team, $\mathrm{N}^{(i)}_{CPG_{points}}$ is the set of CPG points that are assigned to the i-th vehicle and $A^{(i)}_{CPG}$ is the adjacency matrix that is processed by the i-th vehicle. This strategy distributes the processing of a starting point and then merges the single adjacency matrices into A_{CPG}.

The robustness of our solution in real scenarios would be affected by an UAV failure if it occurs during the CPG processing. In this case, the starting points that were assigned to the failed UAV shall be redistributed to the remaining alive UAVs.

Algorithm 3: `distributed_CPG_algorithm` (*scenario*)

```
begin
    $\left(\mathrm{N}_{CPG_{points}}, \mathrm{N}_{CPG_{directions}}\right) \longleftarrow$ set_CPG_topology (scenario)
    $\left(\mathrm{N}^{(i)}_{CPG_{points}}\right) \longleftarrow$
    assign_CPG_points_to_vehicles $\left(\mathrm{N}_v, \mathrm{N}_{CPG_{points}}\right)$
    for $i \longleftarrow 1$ to $\mathrm{N}_v$ do
        $A^{(i)}_{CPG} \longleftarrow$ CPG_arcs_computing $\left(\mathrm{N}^{(i)}_{CPG_{points}}, \mathrm{N}_{CPG_{directions}}\right)$
    end
    $A_{CPG} \longleftarrow$ CPG_matrices_merge $\left(A^{(i)}_{CPG}\right)$
end
```

Two policies have been considered for the allocation of CPG points (Fig. 4): a geographic criterion and a fair criterion. The former assigns a CPG point to the closest UAV, which should be the most engaged vehicle by the processing of such a starting point. The latter randomly assigns CPG points in equal parts in order to supply a uniform distribution of the processing workload.

Besides, the distributed strategy has been simulated on a single platform by means of a variation of the parallel CPG algorithm. In more detail, a parallel OpenMP implementation of the first-level for-loop of `CPG_arcs_computing` function has been used in order to reproduce the distribution amongst the vehicles. Indeed, this loop scans all CPG points and processes the optimal flight trajectories starting from the current CPG point. Hence, a parallelization of the first-order for-loop of `CPG_arcs_computing` function is equivalent to the distribution of CPG points processing to some degree. The outcome is a double-level parallel CPG process.

Table 2 reports the test results with four vehicles. The first row concerns the sequential criterion. The test scenario and the testbed are the same of Sect. 5.1. The vehicles are located

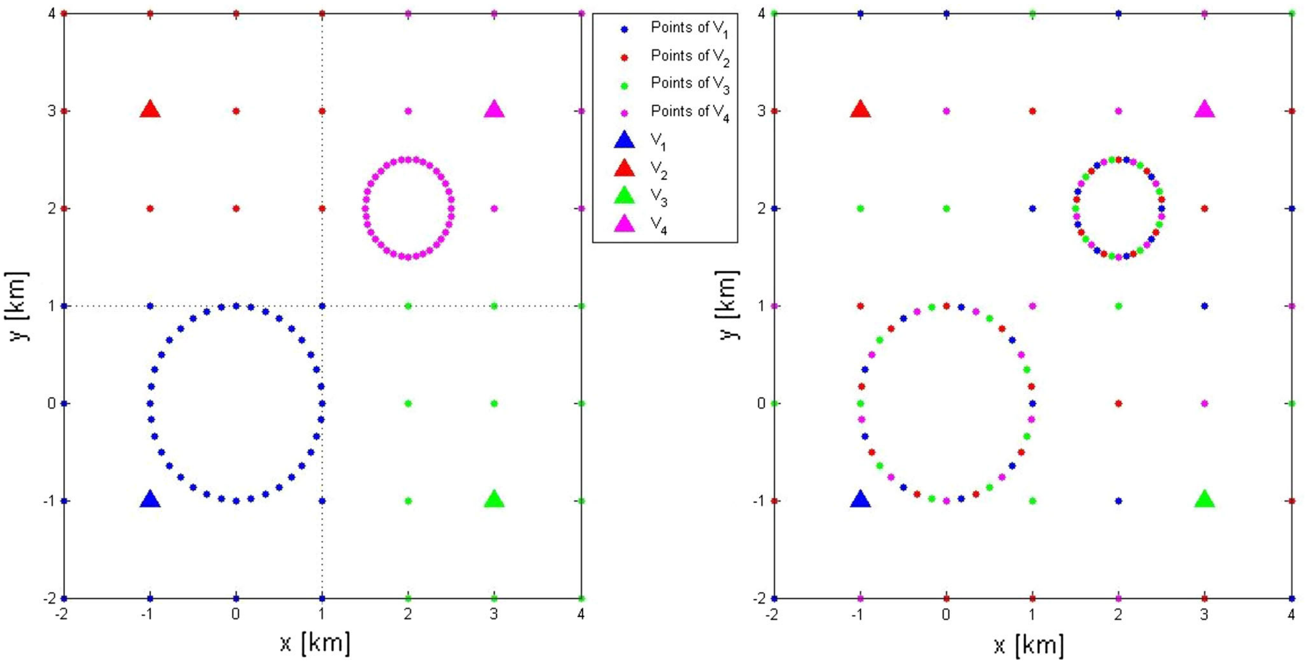

Fig. 4 Geographic criterion (*left*) and fair criterion (*right*) for the allocation of CPG points amongst the team of vehicles (V_1, V_2, V_3, V_4)

Table 2 Measures in seconds of the wall-clock time of the distributed CPG implementation

Criterion	Vehicles				Global
	#1	#2	#3	#4	
None	29708.98				29708.98
Geographic	18269.71	2410.31	1945.41	15492.43	18269.78
Fair	13276.62	13144.87	12591.35	13017.08	13276.71

Table 3 Speedup and efficiency of the distributed CPG implementation

Criterion	Speedup	Efficiency
Geographic	1.63	0.41
Fair	2.24	0.56

at the coordinates $(-1, -1)$, $(-1, 3)$, $(3, -1)$ and $(3, 3)$. The measurement of the global wall-clock time goes from the setting of the topology to the merging of adjacency matrices. Table 3 shows the speedup and the efficiency of the proposed implementation of the distributed CPG algorithm.

The results in Table 2 highlight non-uniform lead times of the vehicles because their workload are imbalanced, even for the fair criterion. More balanced criteria are difficult to arrange since the CPG algorithm is irregular: its distribution of work and data cannot be characterized a priori because these quantities are input-dependent (e.g., they are related to the feasibility of the arcs). Thus, the irregularity of the CPG algorithm inhibits an equal distribution of work over vehicles.

7 Conclusion

Parallel and distributed extensions of a CPG algorithm have been proposed in order to improve the lead time for an online CPG application. The parallelization, even by high level primitives such as OpenMP and over off-the-shelf hardware, shows improving performance figures. However, absolute turnaround does not allow for real time exploitation in dynamically changing scenarios yet, at least without dedicated hardware like massive parallel architecture.

For this reason, future work aims at exploring the effectiveness of such kind of test-beds like GPU. On the other hand, the feasibility of a heuristic approach which could provide a subset of solutions, but in real time, will be considered. The design of a protocol for an effective CPG implementation on a fleet of UAVs is an additional objective to take into account the communication overhead in a distributed environment.

References

Anderson E, Beard R, McLain T (2005) Real-time dynamic trajectory smoothing for unmanned air vehicles. IEEE Trans Control Syst Technol 13(3):471–477

Asseo SJ (1998) In-flight replanning of penetration routes to avoid threats zones of circular shapes. In: Proceedings of Aerospace and Electronics Conference (NAECON 1998), IEEE, pp 383–391

Betts JT (1998) Survey of numerical methods for trajectory optimization. J Guid Control Dyn 21:193–207

Blasi L, Barbato S, Mattei M (2013) A particle swarm approach for flight path optimization in a constrained environment. Aerosp Sci Technol 26(1):128–137

Borrelli F, Subramanian D, Raghunathan AU, Biegler LT (2006) MILP and NLP techniques for centralized trajectory planning of multiple unmanned air vehicles. In: Proceedings of 2006 American Control Conference, IEEE

Cen Y, Wang L, Zhang H (2007) Real-time obstacle avoidance strategy for mobile robot based on improved coordinating potential field with genetic algorithm. In: Proceedings of the 16th IEEE International Conference on Control Applications, IEEE, pp 415–419

Chitsaz H, LaValle SM (2007) Time-optimal paths for a Dubins airplane. In: Proceedings of the 46th IEEE Conference on Decision and Control (CDC), IEEE, pp 2379–2384

Dever C, Mettler B, Feron E, Popovic J, McConley M (2006) Nonlinear trajectory generation for autonomous vehicles via parameterized maneuver classes. J Guid Control Dyn 29:289–302

Faied M, Mostafa A, Girard A (2010) Vehicle routing problem instances: application to multi-UAV mission planning. In: Proceedings of AIAA Guidance, Navigation and Control Conference, AIAA

Ficco M, Avolio G, Battaglia L, Manetti V (2014) Hybrid Simulation of distributed large-scale critical infrastructures. In: Proceedings of 2014 International Conference on Intelligent Networking and Collaborative Systems (INCoS), IEEE, pp 616–621

Frazzoli E, Dahleh M, Feron E (2002) Nonlinear trajectory generation for autonomous vehicles via parameterized maneuver classes. J Guid Control Dyn 25:116–129

Gertz EM, Wright SJ (2001) OOQP user guide. Technical Memorandum ANL/MCS-TM-252, Argonne National Laboratory. Mathematics and Computer Science Division

Gigante G, Gargiulo F, Ficco M (2015) A semantic driven approach for requirements verification. Intelligent distributed computing VIII, Springer International Publishing, Studies in computational intelligence 570:427–436

Gross D, Rasmussen S, Chandler P, Feitshans G (2006) Cooperative operations in urban terrain (COUNTER). In: Proceedings of Society of Photo-Optical Instrumentation Engineers Conference, vol 6249

Hargraves CR, Paris SW (1987) Direct trajectory optimization using nonlinear programming and collocation. J Guid Control Dyn 10:338–342

Hu XB, Wu SF, Jiang J (2004) On-line free-flight path optimization based on improved genetic algorithms. Eng Appl Artif Intell 17(8):897–907

Hwang YK, Ahuja N (1992) A potential field approach to path planning. IEEE Trans Robot Autom 8:23–32

Ingham LA (2008) Considerations for a roadmap for the operations of unmanned aerial vehicles (UAV) in South African airspace. PhD thesis, Universiteit Stellenbosch University

Israel M (2011) A UAV-based roe deer fawn detection system. Int Arch Photogramm Remote Sens XXXVIII:1–5

Kider JT, Henderson M, Likhachev M, Safonova A (2010) High-dimensional planning on the GPU. In: Proceedings of 2010 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 2515–2522

Kopřiva V, Šišlák D, Pěchouček M (2012) Towards parallel real-time trajectory planning. In: Advances on practical applications of agents and multi-agent systems: 10th international conference on practical applications of agents and multi-agent systems. Advances in Intelligent and Soft Computing, vol 155. Springer, Berlin, Heidelberg, pp 99–108

Lei L, Wang H, Wu Q (2006) Improved genetic algorithms based path planning of mobile robot under unknown environment. In: Proceedings of IEEE International Conference on Mechatronics and Automation, IEEE, pp 1728–1732

Li S, Sun X, Xu Y (2006) Particle swarm optimization for route planning of unmanned aerial vehicles. In: Proceedings of 2006 IEEE International Conference on Information Acquisition, IEEE, pp 1213–1218

Mattei M, Blasi L (2010) Smooth flight trajectory planning in the presence of no-fly zones and obstacles. J Guid Control Dyn 33(2):454–462

Meier L, Tanskanen P, Fraundorfer F, Pollefeys M (2011) PIXHAWK: A system for autonomous flight using onboard computer vision. In: Proceedings of 2011 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 2992–2997

Monteiro RDC, Adler I, Resende MGC (1990) A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming and its power series extension. Math Oper Res 15(2):191–214

Murray RM (2007) Resent research in cooperative control of multi-vehicle systems. J Dyn Syst Meas Control 129(5):571–583

OpenMP (2013) OpenMP application program interface, version 4.0. OpenMP Architecture Review Board

Pan J, Lauterbach C, Manocha D (2010) g-planner: real-time motion planning and global navigation using GPUs. In: Proceedings of the National Conference on Artificial Intelligence, Association for the Advancement of Artificial Intelligence (AAAI), vol 2, pp 1245–1251

Pascarella D, Venticinque S, Aversa R (2013) Agent-based design for UAV mission planning. In: Proceedings of the 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2013), IEEE, pp 76–83

Pascarella D, Venticinque S, Aversa R, Mattei M, Blasi L (2014) A parallel and a distributed implementation of the core paths graph algorithm. In: Intelligent distributed computing VIII, Springer International Publishing, Studies in Computational Intelligence 570:417–426

Quaritsch M, Kruggl K, Wischounig-Strucl D, Bhattacharya S, Shah M, Rinner B (2010) Networked UAVs as aerial sensor network for disaster management applications. Elektrotech Informationstechnik 127(3):56–63

Raja P, Pugazhenhi S (2009) Path planning for mobile robots in dynamic environments using particle swarm optimization. In: Proceedings of 2009 International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom'09), IEEE, pp 401–405

Russ M, Stütz P (2012) Airborne sensor and perception management: a conceptual approach for surveillance UAS. In: Proceedings of 2012 15th International Conference on Information Fusion (FUSION), IEEE, pp 2444–2451

Saska M, Macas M, Preucil L, Lhotska L (2006) Robot path planning using particle swarm optimization of ferguson splines. In: Proceedings of 2006 IEEE Conference on Emerging Technologies and Factory Automation (ETFA06), IEEE, pp 833–839

Schouwenaars T, Feron E, How J (2004) Receding horizon path planning with implicit safety guarantees. In: Proceedings of 2004 American Control Conference, IEEE, vol 6, pp 5576–5581

Schumacher C, Chandler PR, Patcher M, Patcher LS (2007) Optimization of air vehicles operations using mixed-integer linear programming. J Oper Res Soc 58:516–527

Shima T, Rasmussen S (2009) UAV cooperative decision and control: challenges and practical approaches. Society for Industrial and Applied Mathematics, Philadelphia

Vaidyanathan R, Hocaoglu C, Prince TS, Quinn RD (2001) Evolutionary path planning for autonomous air vehicle using multiresolution path representation. In: Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS), IEEE, vol 1, pp 69–76

Yang HI, Zhao YJ (2004) Trajectory planning for autonomous aerospace vehicles amid known obstacles and conflicts. J Guid Control Dyn 27:997–1008

Yokoyama N, Suzuki S (2005) Modified genetic algorithm for constrained trajectory optimization. J Guid Control Dyn 28:139–144