2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Asynchronous Decentralized Prioritized Planning for Coordination in Multi-Robot System

Michal Čáp, Peter Novák, Martin Selecký, Jan Faigl, Jiří Vokřínek

Abstract— **In this paper, the multi-robot motion coordination planning problem is addressed. Although a centralized prioritized planning strategy can be used to solve the problem, we rather consider a decentralized variant, which is a more suitable for a robotic system of cooperating unmanned aerial vehicles (UAVs) due to communication limitations, privacy concerns, and a better exploitation of computational resources distributed among the individual robots. However, the existing decentralized prioritized planning algorithm contains synchronization points that all the agents must be able to pass synchronously, which is impractical and inefficient for a real-world deployment of the robotic systems. Therefore, we introduce a new asynchronous decentralized prioritized planning algorithm and show that the method can converge faster than both the synchronous decentralized and centralized algorithms. Further, we demonstrate the applicability of the proposed method as a coordination mechanism within a complex mission planning for a real robotic system consisting of several autonomous UAVs.**

I. Introduction

When a team of mobile robots operates in a given workspace, one of the fundamental problems is to prevent collisions among the individual team members. The problem can be addressed by the motion coordination planning that is able to provide mutually collision-free trajectories. Although the problem is straightforward to formulate as a planning problem in the Cartesian product of the state spaces of individual robots, such a problem is difficult to solve because the size of such a state space grows exponentially with the number of robots. The problem is even more challenging, if the prior knowledge about the situation, robots' goals and operational environment is only an initial estimation, and thus new updates have to be considered and fast replanning has to be performed whenever new update is available in order to prevent a collision. These additional requirements arise from a practical deployment of a multi-robot system for patrolling and surveillance missions [12], which is a subject of our research.

Regarding the practical difficulties, more pragmatical methods have been proposed to provide a fast response even though the global completeness has been sacrificed in favor of reduced computational complexity. Examples of such methods are decentralized reactive planners such as the DRCA [9] or ORCA [16]. However, due to their reactive nature only local collision-free motions can be guaranteed; hence, it may happen that a reactive technique will lead robots to a deadlock.

Multi-robot motion planners take into consideration the goal of each robot and plan trajectories that are globally conflict-free. If the robots execute the resulting joint plan precisely (or within some given tolerance), it is guaranteed that the robots will reach their goals without any collision. The multi-robot planners are typically based either on the coupled heuristic search in the joint state space of all robots or on decoupled planning. The former class of the methods is able to find optimal solutions [14], [15], but the computational complexity does not scale well with the increasing number of conflicting robots.

On the other hand, decoupled approaches can be fast enough for real-time applications. Such methods are incomplete similarly to the reactive methods, however, if a decoupled planner finds a solution, it represents a globally conflict-free plan that guarantees each robot will reach its goal if the robots will follow the planned trajectories. If a decoupled method is able to compute solutions sufficiently fast, then it can be used in a reactive manner to provide fast response to sudden changes in a form of globally conflict-free trajectories.

A widely used decoupled scheme for the multi-robot motion planning that has been shown to be effective in practice [4] is prioritized planning [3]. Recently, Velagapudi et al. presented a decentralized version of prioritized planning technique for teams of mobile robots [19], which is able to utilize the distributed computational resources of each robot and thus decreases the time tot find a solution.

However, the formulation of the decentralized algorithm is based on the assumption that the robots have a "distributed synchronization mechanism allowing them to wait for all team mates to reach a certain point in algorithm execution" [19]. Unfortunately, such mechanisms (known as distributed termination detection algorithms) are rather non-trivial to implement [10]. With unreliable communication channels, the synchronization is in general impossible to achieve, as was shown in the famous Two Generals' problem. Moreover, since the algorithm proceeds in globally-synchronized rounds, faster-computing robots have to wait at the end of each round for the longest-computing robot and thus the distributed computational power is not used efficiently. Besides, the synchronized algorithm was evaluated only in a simulated 2-d grid, and therefore, it is not clear if the algorithm is applicable to real-world robotic systems.

In this paper, we address the aforementioned issues of the synchronized decentralized prioritized planning (SDPP) and we propose a novel algorithm coined asynchronous decentralized prioritized planning (ADPP), which removes the need for the synchronization between the robots. The benefit of ADPP is its simplicity and the ability to exploit the available distributed computational power in a more efficient way than the SDPP and thus to provide a solution of the

978-1-4673-6358-7/13/$31.00 ©2013 IEEE

conflict situation in a shorter time. The results presented in Section V show that the proposed asynchronous algorithm is able to provide solutions faster than its synchronized variant. Moreover, we experimentally verified the ADPP's behavior in a practical deployment of a multi-UAV robotic system with real communication constraints, where the proposed approach exposes ability to deal with unreliable communication.

The rest of the paper is organized as follows: The problem is defined in the next section while an overview of prioritized planning approaches is presented in Section III. The main contribution of the paper, the ADPP algorithm, is described in Section IV and its evaluation results together with a comparison with the previous approaches is presented in Section V. A report on the practical deployment is dedicated to Section VI. Finally, concluding remarks are in Section VII.

II. Problem

Consider n robots $r_1, \ldots, r_n$ operating in a subregion $\mathcal{W}$ of 2-d or 3-d Euclidean space. Each robot r_i is characterized by its starting and goal positions $s_i \in \mathcal{W}$, $g_i \in \mathcal{W}$, respectively. The task is to find a set of space-time trajectories $P = \{p_1, \ldots, p_n\}$, such that $p_i : \mathbb{R}_{\geq 0} \to \mathcal{W}$ is a mapping from time to positions in $\mathcal{W}$, $p_i(0) = s_i$, $p_i(t_i) = g_i$ and the trajectories are mutually collision free, i.e., $\forall i, j : i \neq j \Rightarrow \neg C(p_i, p_j)$, where $C(p_i, p_j)$ denotes a space-time mutual collision relation between p_i and p_j. Informally, two trajectories collide (are in a conflict) when the bodies of two robots touch, or intersect at some time t_{col}.

We assume that each robot is equipped with a wireless communication device allowing to broadcast messages. Each of these messages will be eventually received by every other team member. Further, we assume that the communication channel preserves the ordering of messages that were sent in. However, in a practical verification of the proposed method using a real-robotic system, the message delivery is not guaranteed. Moreover the number of robots can change during the mission.

III. Prioritized Planning

Prioritized planning [3], [17], [1] represents a pragmatic solution of the addressed problem of the multi-robot motion planning, which is known to be PSPACE-hard [5]. Although prioritized planning is in general incomplete, it is fast even for large multi-robot teams in complex environments. In prioritized planning, each robot is assigned a unique priority and the algorithm proceeds sequentially from the highest priority robot to the lowest priority one. The robots consider the higher priority robots moving along their planned trajectories as moving obstacles, which they have to avoid. When the algorithm finishes, each robot is assigned a trajectory not colliding with either higher priority robots, since the robot avoided a collision with those, nor with lower priority robots that avoided a conflict with the given trajectory themselves.

In the case a single robot is unable to find a trajectory that is conflict-free with higher-priority robots, the overall algorithm fails as it does not perform backtracking. Clearly,

Algorithm 1 Centralized Prioritized Planning (CPP)

Ensure: After the algorithm finishes, $Traj_i$ contains the final computed trajectory for the robot with priority i. If the robot cannot find a trajectory not colliding with higher priority robots, $Traj_i$ stores $\emptyset$.

```
1: procedure CPP(⟨s_1, g_1⟩, ..., ⟨s_n, g_n⟩)
2:     Avoids ← ∅
3:     for i ← 1 ... n do
4:         Traj_i ← BEST-TRAJ_i(s_i, g_i, Avoids)
5:         Avoids ← Avoids ∪ {Traj_i}
6:     end for
7: end procedure

8: function BEST-TRAJ_i(s, g, avoids)
9:     return the best trajectory from s to g not conflict-
10:            ing with trajectories in avoids.
11:            Otherwise return ∅.
12: end function
```

such a scheme is sensitive to initial prioritization of the robots in the team; however, there are simple heuristics for choosing an efficient ordering [17]. The main benefit of the prioritized plannning strategy is a fast runtime in relatively uncluttered environments typically encountered in multi-robot applications, especially if we consider domains such as operations of cooperating UAVs.

A straightforward implentation of the prioritized strategy is a centralized algorithm. A brief overview of the centralized algorithm we considered for evaluation is presented in the next section. The proposed ADPP algorithm is based on the decentralized synchronized algorithm [19], and therefore, it is described in more detail in Section III-B.

A. Centralized Algorithm

A collision-free operation of a multi-robot team can be ensured by forcing all robots to communicate their objectives to a centralized planner, which centrally computes a solution and informs the robots about the trajectory they have to follow in order to maintain the conflict-free operation. As a baseline for evaluation of decentralized algorithms, we considered cooperative *A** [13] (see Algorithm 1), a centralized algorithm based on the prioritized planning scheme (CPP).

We assume that BEST-TRAJ$_i(s, g, avoids)$ is an application specific routine that returns an optimal trajectory for robot i in space $\mathcal{W}$ that avoids robots that follow the respective trajectories in $Avoids_i$. Such a routine can be implemented by any motion planning technique for planning with moving obstacles (e.g., [18] or [11]).

B. Decentralized, but Synchronized Algorithm

A decentralized multi-robot planning algorithm utilizing the prioritized planning scheme has been presented in [19] and it is denoted as the *synchronized decentralized prioritized planning* (SDPP) algorithm here. Algorithm 2 lists the pseudocode of the SDPP, slightly adapted for exposition purposes.

Algorithm 2 Synchronized Decentralized Prioritized Planning (SDPP) – pseudocode for the robot i

Ensure: After the algorithm finishes, $Traj_i$ contains the final trajectory for robot i. If no solution was found, $Traj_i$ stores $\emptyset$.

```
1: procedure SDPP(s, g, priority)
2:     S_i ← s; G_i ← g; I ← priority; Avoids_i ← ∅; Traj_i ← ∅
3:     REPLAN
4:     wait for all other robots to finish the planning
5:     process messages and update Avoids_i set
6:     while not global termination detected do
7:         CHECK-CONSISTENCY
8:         wait for all other robots to finish the iteration
9:         process messages and update Avoids_i set
10:    end while
11: end procedure

12: procedure CHECK-CONSISTENCY
13:    if Traj_i collides with Avoids_i then
14:        REPLAN
15:    end if
16: end procedure

17: procedure REPLAN
18:    Traj_i ←BEST-TRAJ_i(S_i, G_i, Avoids_i)
19:    BROADCAST(I, Traj_i)
20: end procedure
```

Before the start of the algorithm, each robot is assigned a unique priority $I \in 1 \dots N$, with N being the size of the multi-robot team and $I = 1$ denoting the robot with the highest priority.

The algorithm proceeds in synchronized iterations. In each iteration, the robots recompute their trajectory if necessary and subsequently broadcast it to other robots. A robot must recompute its trajectory in the case its current trajectory is in conflict with any trajectories of the higher priority robots computed and communicated in the previous iterations.

The algorithm finishes when all the robots cease to communicate and either hold a trajectory, or they were not able to find a collision-free solution.

IV. Asynchronous Prioritized Planning

Due to its synchronous nature, the SDPP algorithm is cumbersome to implement and does not fully exploit the computational power distributed among individual robots. In every iteration, the robots that finished their trajectory planning routine sooner, or did not have to re-plan at all, sit idle while waiting for the robots with a higher workload in that iteration (or simply slower computation), even though they could theoretically resolve some of the conflicts they have among themselves in the meantime and thus speed up the overall algorithm run. An example of a situation, where the asynchronous algorithm would be beneficial is illustrated in Figure 1.

Algorithm 3 Asynchronous Decentralized Prioritized Planning (ADPP) – pseudocode for the robot i

```
1: procedure ADPP(s, g, priority)
2:     S_i ← s; G_i ← g; I ← priority; Avoids_i ← ∅; Traj_i ← ∅
3:     EXEC-ASYNC(REPLAN)
4:     wait for global termination
5: end procedure

6: message handler RECEIVE-INFORM(j, traj)
7:     if j < I then
8:         Avoids_i ← (Avoids_i \ ⟨j, _⟩) ∪ {⟨j, traj⟩}
9:         EXEC-ASYNC(CHECK-CONSISTENCY)
10:    end if
11: end message handler

12: procedure EXEC-ASYNC(Routine)
13:    terminate currently running computation (if any)
14:    execute asynchronously Routine
15: end procedure
```

To deal with such an inefficiency and to remove the need for the distributed termination detection at each iteration, we propose an asynchronous decentralized variant of the prioritized planning coined the *asynchronous decentralized prioritized planning* (ADPP) algorithm, whose pseudocode is exposed in Algorithm 3. The asynchronous algorithm replaces the concept of globally synchronized iteration (while loop in Algorithm 2) by a reactive approach in which every robot reacts merely to incoming INFORM messages. Upon receiving an INFORM message (RECEIVE-INFORM routine in Algorithm 3), the robot simply replaces the information about the trajectory of the sender robot in its $Avoids_i$ set and checks whether its current trajectory is consistent with the contents of the new $Avoids_i$ set. If the current trajectory is inconsistent the robot triggers replanning, otherwise the robot can remain idle.

Further, contrary to the SDPP, the REPLAN and CHECK-CONSISTENCY routines are executed asynchronously. If a computation is already running for an older contents of the $Avoids_i$ set, (i.e., previously invoked REPLAN or CHECK-CONSISTENCY routines) the computation is interrupted and started anew with the updated contents of the $Avoids_i$ set.

A. Correctness of the ADPP

The ADPP algorithm is correct, which means that if it terminates then the found trajectories are mutally collision free or it is reported that a feasible solution has not been found. The proof the algorithm correctness is based on showing the algorithm terminates and soundness of the found solution.

Proof (ADPP termination): The termination of the algorithm can be shown by an induction on an individual robot priority i that each robot r_i eventually stops sending the INFORM message.

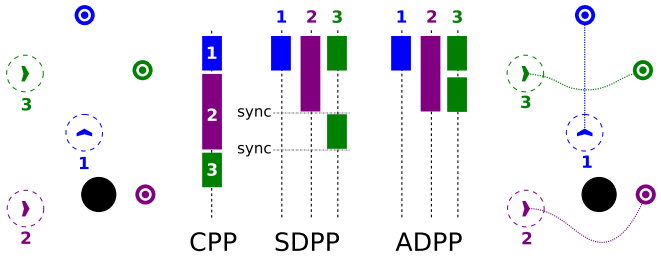


Fig. 1: Example multi-robot coordination problem. **Left:** start and target positions of the robots 1, 2 and 3. We assume that robot 2 plans longer, because it has to avoid the black obstacle. **Middle:** Sequence diagrams showing the planning process in CPP, SDPP and ADPP. **Right:** the final trajectories.

Initial step: There is no robot with the priority higher than the robot r_1; thus, the highest priority robot r_1 informs the lower priority robots only once in the first iteration of the algorithm. After that it remains silent since its trajectories will always be consistent with an empty set of trajectories.

Induction step: Let consider the following induction hypothesis: *"after the robots with priorities* $1 \dots k-1$ *stopped communicating, eventually also the robot with the priority* k *stops sending* INFORM *messages"*. Now, assume this is not the case and there is a situation such that the robot k would end up sending INFORM messages forever. However, for such a case the robot must have its mailbox continually being filled with INFORM messages; so, its RECEIVE-INFORM handler routine gets invoked infinitely many times. In a consequence, the robot would possibly need to recompute its best trajectory and subsequently inform the lower priority robot infinitely often. That implies existence of a sender for each such message and hence by necessity there must be at least one robot with a priority higher than k, which keeps sending INFORM messages forever, which contradicts the induction hypothesis. ■

As a consequence of the consecutive silencing of the robots from high to lower priorities, it is straightforward to see that in the course of the ADPP algorithm execution, each agent makes at most N iterations before it terminates.

Proof (ADPP correctness): Now, we know the algorithm terminates and thus we can show the final variables $Traj_i$ store a set of non-conflicting trajectories. Since each robot eventually sends its last INFORM message and ceases to communicate, all other robots with a lower priority eventually process all last INFORM messages from all robots with higher priority together with their final trajectories (being either a valid trajectory, or $\emptyset$). At that moment, all the tuples $\langle j, Traj_j \rangle$ for all $j > i$ are stored in the set $Avoids_i$ of the robot r_i. Subsequently the robot eventually invokes the CHECK-CONSISTENCY routine for the last time and thus either $Traj_i$ will end up unchanged, recomputed and again non-conflicting with either of $\langle j, Traj_j \rangle$ for all $j < i$, or being invalid ($\emptyset$). Finally, the robot informs all the robots with priorities lower than i and ceases to communicate. When the last robot stops communicating, all the $Traj_i$ variables are either set correctly, or the algorithm failed to find a solution for some of the participating robots. ■

As already noted, the ADPP is incomplete, which can be simply shown by the following example. Imagine the highest priority robot finds a solution, which does not allow to find a collision-free trajectory by other robots while for a different choice of priorities there can be a feasible solution for all robots.

B. Implementation Notes and Plan Execution

In this work we do not make attempts to limit the number of recipients of each message (e.g. as in [19]) because a) we assume that the robots use a wireless communication channel, in which it is cheaper to broadcast a single message than to transmit multiple single-recipient messages and b) in open systems the identities of lower-priority agents are often unknown a priori.

A practical detection if the termination condition is satisfied can be implemented without any communication overheads as follows. If the total number of the robots in the team is fixed, the following marking-based termination-detection mechanism can be used. A robot r_i marks its trajectory as *final* if it is collision-free with the current $Avoids_i$ set and the trajectory of the robot with the priority $i-1$ is marked final. This can be communicated to other robots by marking the trajectory in the INFORM message with a final flag (if $Traj_i$ has changed) or by broadcasting a short FINAL message to confirm that the current $Traj_i$ is still valid, but now also final. The initial trajectory of r_1 is trivially marked final. When a robot broadcasts its final trajectory, it can safely terminate its computation. Once the final trajectory has been generated by the lowest-priority robot r_n, the computation terminates globally; hence, if a robot receives the INFORM message with the final trajectory of the lowest-priority robot, it can deduce the global termination condition has been reached.

If the size of the team is not fixed, one can for example bound the maximum time of the BESTTRAJ planning and pronounce the computation terminated if the system is silent for more than the given time.

After the termination condition is reached and all the trajectories are collision-free (otherwise the planner can report an infeasible situation to an operator) the robots start to follow the found trajectories. Here, it is worth to mention that robots (e.g., UAVs that cannot stop their motion during flight) may follow the previous plans during the computation (we assume the first initial plan is collision-free, e.g. loitering). In such a case, one can imagine that an individual robot r_i can start execution its new collision-free plan immediately when it is found and no new updates of the $Avoids_i$ set has been received during planning. Even though the common acceptance of the global plan provides the theoretical guarantee of all robots reaching their goals, the latter approach with immediate execution of the plan provides a faster response of the particular robot to the new situation. In the results presented in this paper, we consider

the first approach in evaluation presented in Section V, while for practical deployment using real Multi-UAV robotic system we rather consider the latter approach, see Section VI.

The readers interested in deeper analysis of the algorithm are referred to the results of our preliminary work [2].

V. Evaluation

The expected benefit of the proposed ADPP algorithm has been evaluated in a comparision with the previous approaches CPP, SDPP on a series of randomly generated problem instances. The particular problem instances follow a common structure based on a 20 m x 20 m large 2-d environment populated by n disc-shaped robots having the radius 0.4 m. The robots plan their trajectories using an A* planner operating on a spatio-temporal 4-connected grid graph, where the heuristic is the time needed to travel the Euclidean distance from the current node to the destination node at the maximum speed. The robots can move on the edges of the graph with the constant speed of 1 m/s or they can wait for 0.5 s on any of the vertices in the graph. The "wait" action can be used repeatedly.

The evaluated runtime characteristics of the algorithms are the wall-clock runtime, communication complexity (the number of broadcasted messages) and solution quality. The characteristics have been measured using the Intel Core 2 Duo CPU running at 2.1 Ghz. In the case of decentralized algorithms, a simulated concurrent execution has been considered using a discrete-event simulation. The arriving time of each message is used to measure execution time of the particular planning instances to determine the real computational requirements as the planning is executed on n independent computers assuming perfect communication with no latency. The simulator of concurrent execution was implemented using Alite multi-agent simulation toolkit. The complete source code of the experimental environment (including the simulator) and the video recordings of the experiments are available at `http://agents.fel.cvut.cz/~cap/adpp/`.

In the evaluated problems, the number of robots varies from 30 to 100. The start position for each robot in the scenario was selected randomly from a uniform distribution (see Figure 2), the goal position was generated randomly in the distance from the start position that was taken from the interval $(5, 10)$. Further, we asserted that no two robots share the start node and no two robots share the destination node. For each problem with a particular number of robots 10 different random trials was performed and average values of the measured characteristics are presented here.

The wall-clock runtime represents the real-world time a particular algorithm would need to find a solution. The wall-clock time for the CPP is equal to its CPU-time and can be measured directly. The average wall-clock runtime of the decentralized algorithms on random scenarios with n robots was obtained by running an n concurrent processes simulation of the algorithm execution. The average required wall-clock runtime are depicted in Figure 3a. The results show that the decentralized algorithms requires shorter runtime than the centralized solver. Further, the proposed ADPP provides significantly better runtime performance than the SDPP, especially in dense problem instances with many conflicting robots.

Regarding the computational complexity, we can consider the CPP algorithm used in a distributed system as follows. All the robots have to communicate their objectives to the central solver. Then, the central solver finds a solution and informs each robot about its new path. Thus, the number of required messages can be computed analytically and the communication complexity of the CPP is $2n$. The required number of exchanged messages is ploted in Figure 3b. It is shown that for sparse problems, the decentralized algorithms require fewer messages than the centralized approach. The communication complexity of the decentralized algorithms starts exceeding the centralized approach for more than 60 robots.

The quality of the found solutions is measured as a cumulative time spent by robots navigating their trajectories and defined as

$$cost(P) = \frac{dur(P) - dur(P')}{dur(P')}, \tag{1}$$

where $dur(P) = \sum_{i=1}^{n} t_i^{dest}$, t_i^{dest} denotes the shortest time instant at which the robot r_i reaches its destination $dest_i$ and P' is the set of the best trajectories for each robot if collisions are ignored.

Results of the measured average costs are shown in Figure 3c. The decentralized algorithms return a slightly worse solution (on average) than the CPP algorithm. The reason is the replanning condition used by the decentralized algorithms. The condition states that a robot should replan its trajectory only if the trajectory is inconsistent with its view of the situation. Thus, the robot may receive an updated trajectory from a higher-priority robot that allows an improvement in its current trajectory, but since the trajectory may be still consistent, the robot will not exploit such an opportunity for an improvement.

Finally, Figure 3d shows the failure rates of the individual algorithms as a function of the number of robots in a scenario. We can see that the incompleteness of the prioritized planning (i.e., there is at least one robot that failed to find a consistent trajectory) starts exhibiting itself only in relatively dense scenarios.

VI. Deployment

In this section, we present results of the ADPP algorithm deployment in our multi-UAV system, which demonstrate a practical applicability of the proposed algorithm in real world scenarios. The considered multi-UAV system is being developed under a long-running research effort in our center [12], which is concerned with development of multi-agent control algorithms for teams of cooperating UAVs to autonomously carry out tasks such as patrolling, tracking or area surveillance in tactical missions. In fact, the main

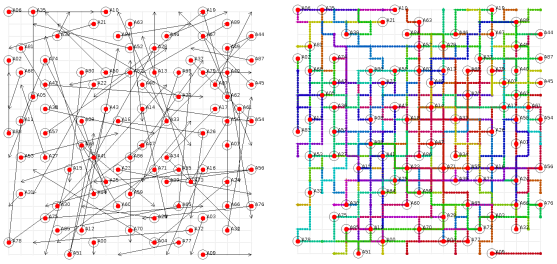
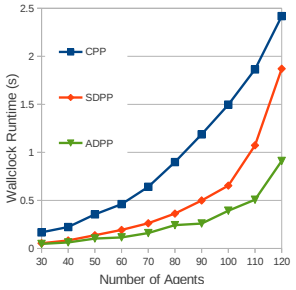
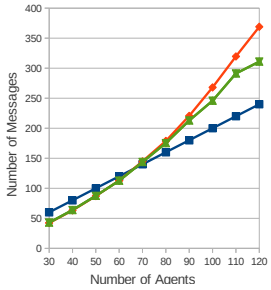


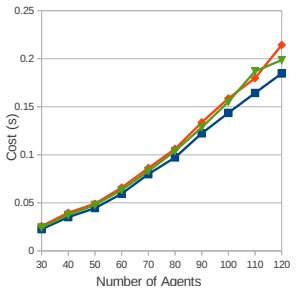
Fig. 2: One instance of a random scenario with 90 robots. The start and goal positions of each robot are depicted on the left, the final solution found is on the right.



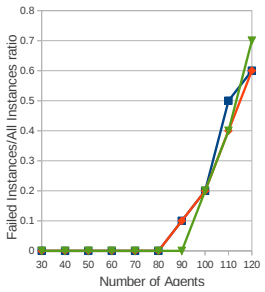
(a) Average wall-clock runtime



(b) Average messages broadcasted



(c) Average cost (prolongation of trajectories)



(d) Failed instances ratio

Fig. 3: Results from the random scenario

motivation behind the development of the proposed ADPP algorithm was to develop a conflict resolution mechanism that can be deployed to real UAVs, run on the embedded onboard computational resources and be able to deal with real-world communication constraints. The application context and a brief description of the hardware equipment is presented in Section VI-A. Although the core of the proposed ADPP algorithm is identical for desktop as well as on-board computers, there are particular issues that have to be addressed for its integration and deployment in field experimenting. These issues and proposed solutions are disccused in Section VI-B. Finally, practical achievements in the so-called superconflicit scenarios are presented in Section VI-C.

A. Multi-UAV Robotic TestBed

Our test bed consists of two UAVs based on the Unicorn airframe equipped with the Kestrel Autopilot from Lockheed Martin (see Figure 4). In addition, the platform is equipped with the Gumstix Overo EarthStorm embedded computer for an on-board computation and Xbee 2.4GHz RF module to enable direct UAV-to-UAV messaging. Our algorithms are primarily targeted to large UAV teams. However, due to the limited number of physical UAVs we have at our disposal, a mixed-reality approach [6] is used to scale up. Specifically, we use two real airborn UAVs in a field (see Figure 5), while the rest of the team is simulated. A description of the integrated system, which allows a single human operator to use a high-level interface to task a team of autonomous UAVs and related results of field experiments cas be found in [12].

To address the flying capabilities of the employed UAVs, especially a relatively high inaccuracy in the plan execution due to unstable wind, we model the UAV (for the purposes of the trajectory planning and conflict resolution) as a relatively large cylindric safe zone around each UAV. Specificially, we set the radius to roughly 100 m and the half-height of the cylinder to 10 m. The parameters were chosen empirically to reflect the accuracy of the autopilot to follow the given trajectory in difficult wind conditions.

An individual UAV can communicate using the radio link having the shared capacity 25 kBps under ideal conditions. However, the serial link that connects the XBee modem to the on-board computer limits the maximum achievable broadcast throughput to 5kBps per UAV. To keep the latency low and to use the bandwidth efficiently, the data are transferred in a raw form and thus the data delivery is not guaranteed.

B. Closed-Loop ADPP

Four main issues have to be addressed to apply a conflict resolution algorithm (as the proposed ADPP) to a real-world settings of a complex mission control. Firstly, the whole team or individual team members may be re-tasked by an operator and thus the goals of any of the UAVs may change at any time during the mission execution. Secondly, the trajectories generated by the ADPP are often executed imprecisely by the UAVs, especially under unstable wind conditions. Thirdly, the used radio communication channel does not guarantee message delivery and thus some messages may get lost. Fourthly, during the mission execution, some UAVs may be abruptly removed from the robotic team, while some new ones can be added.

It is clear that the afforementioned issues strenghten the requirements on the fast response of the mission planning, as the whole setup of the task can quickly change. Therefore, the conflict resolution also has to provide a prompt response. To address these requirements, we adapted the original ADPP to work in a closed-loop fashion. The pseudo code of the closed-loop version of the ADPP (called CLAPP) is given in Algorithm 4. In CLAPP, an execution of the trajectories is continuously monitored and a replanning is invoked if necessary. The replanning is triggered: a) if the robot is assigned a new goal G_i (e.g., by an operator); or b) if the robot has diverted from its planned trajectory. For the latter case, we use an application-specific predicate $far(Traj_i, S_i)$ that is assumed to be true if the robot's actual position

Algorithm 4 Closed-loop Asynchronous Decentralized Prioritized Planning - pseudocode for the robot i

1: S_i stores robot's current position; G_i stores robot's current goal

2: **procedure** CLAPP$_{\text{INIT}}(priority)$
3: $Avoids_i \leftarrow \emptyset$; $Traj_i \leftarrow \emptyset$; $I \leftarrow$ priority;
4: EXEC-ASYNC(REPLAN)
5: **end procedure**

6: **when** G_i has a new value **or** $far(Traj_i, S_i)$
7: EXEC-ASYNC(REPLAN)
8: **end when**

9: **when** $Traj_i$ has a new value
10: FOLLOW($Traj_i$)
11: **end when**

12: **periodically**: BROADCAST($I, Traj_i$)

Fig. 4: Unicorn fixed-wing UAV from Lockheed Martin

S_i is too far (above a given tolerance in space or time) from its current trajectory $Traj_i$ (lines 6-8). Observe that each such forced replanning triggers a new asynchronous prioritized planning process and thus it may cause a cascade of replannings for lower-priority UAVs. However, just as in the standard ADPP, each of the robots will eventually adapt a conflict-free trajectory or reports a failure to find one. From a practical point of view, it should be noted that such an extension was possible due to the asynchronous nature of the ADPP. Implementing such a dynamic mechanism with a centralized or a synchronized algorithm would be much less natural.

Because of the possible message loss, the UAVs broadcast their planned trajectory not only when it changes, but also periodically onwards.

The possible dynamic team reconfiguration does not allow to wait for a global termination of the ADPP run, and therefore, a robot starts executing its collision-free trajectory immediately when it is planned (lines 9-11). For the trajectory generation (BESTTRAJ routine) we use an any-time RRT*-based [7] spatio-temporal trajectory planner, which is restricted to provide a solution within one second.

Fig. 5: The UAV during a field experiment.

C. Superconflict experiment

The behavior of the proposed technique within a complex mission is demonstrated in a so-called "superconflict" scenario. We considered four UAVs with starting positions placed at the corners of a square and their goals being at the respective diagonal opposite corners. Hence, all the airplanes are initially in a conflict in the middle of the square, see Figure 6. In order to show the behaviour of the CLAPP technique more clearly, the BESTTRAJ trajectory planners of the individual UAVs were constrained to use only a fixed flight altitude and a fixed flight velocity.

In this experimental setup, two real UAVs (Plane1 and Plane2) are attached to a hardware-in-the-loop simulator and two others (Plane3 and Plane4) are simulated. The control algorithms are deployed and run on the Gumstix on-board computers. The hardware UAVs use the safe zone radius 110 m, while the simulated ones use 70 m. The virtual UAVs are controlled by the identical software as the hardware UAVs; however, they run within virtual machines on a desktop computer. Both real and simulated UAVs communicate via their XBee radio modules. The Kestrel autopilot of the hardware UAVs is connected to a high-fidelity flight simulator Aviones[1]. When the mission is started the UAVs execute the CLAPP algorithm to coordinate their motions. The resulting traces that were recorded during the experiment are shown in Figure 7 and can also be seen in the attached video. One can see the typical phenomena of prioritized planning – the highest-priority Plane 1 keeps its first straight-lane trajectory, while all other UAVs need to change their first trajectory to adapt.

VII. CONCLUSION

In this paper, we present a novel asynchronous decentralized algorithm for the multi-robot motion coordination problem. The algorithm removes the need for an explicit synchronization of the robots in between individual computational rounds. From a practical point of view, the newly proposed asynchronous algorithm is a more straightforward and it is easier to implement, because it does not require a distributed termination detection to synchronize the agents at

[1] http://aviones.sourceforge.net/

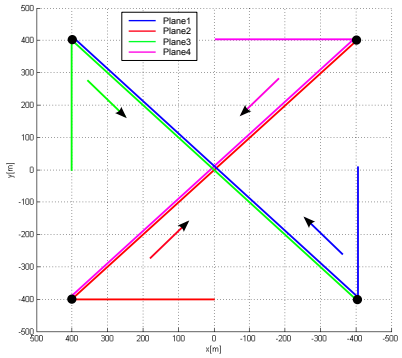


Fig. 6: Superconflict scenario – UAV missions

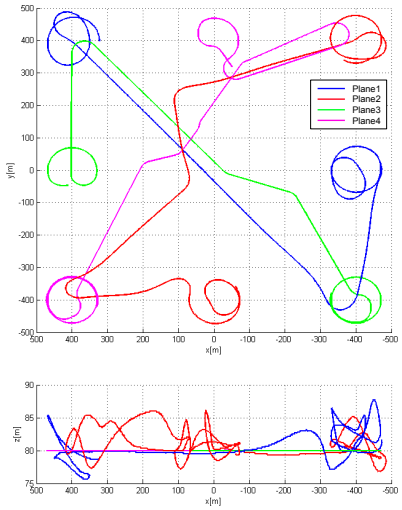


Fig. 7: Superconflict scenario – traces of the UAVs

the end of each computational round, which, in fact, is a difficult to guarantee using an unreliable communication channel. Besides, the new algorithm in practice finds solutions faster and it better exploits available computational resources in the distributed environment. The presented evaluation results of the real-time performance show that the speed up improvements of the asynchronous decentralized algorithm can be significant in comparison with the synchronous decentralized algorithm. Regarding a comparison of the centralized and decentralized approaches, the results indicate that the decentralized algorithms are advantageous as the number of the needed messages to find a solution can be similar for both cases; however, the decentralized algorithms are significantly faster.

The motivation behind the ADPP design was to develop a conflict resolving planning algorithm that can be deployed in the real-world multi-UAV system with an unreliable communication and limited on-board computational power. The closed-loop derivation of the ADPP exhibits that the proposed asynchronous coordination mechanism can be employed in a distributed robotic application with several autonomous UAVs under real-world communication constraints and meets the desired requirements. In future work, we would like to investigate incremental motion planning algorithms (e.g. D* [8]) in place of BESTTRAJ routine to avoid premature interruptions of running planners.

Acknowledgements: This work was supported by the Grant Agency of the Czech Technical University in Prague, grant no. SGS13/143/OHK3/2T/13, by the Ministry of Education, Youth and Sports of Czech Republic within the grants no. LD12044 and 7H11102 (D3CoS), and by the ARTEMIS Joint Undertaking under the number 269336 (www.d3cos.eu). The work of J. Faigl was supported by the Czech Science Foundation (GAČR) under research project No. 13-18316P.

REFERENCES

[1] M. Bennewitz, W. Burgard, and S. Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems*, 41(2):89–99, 2002.

[2] Michal Cap, Peter Novak, Jiri Vokrinek, and Michal Pechoucek. Asynchronous decentralized algorithm for space-time cooperative pathfinding. In *Spatio-Temporal Dynamics Workshop (STeDy), SFB/TR 8 Spatial Cognition Center Report, No. 030-08/2012*, 2012.

[3] Michael Erdmann and Tomas Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:1419–1424, 1987.

[4] Carlo Ferrari, Enrico Pagello, Jun Ota, and Tamio Arai. Multirobot motion coordination in space and time. *Robotics and Autonomous Systems*, 25(3-4):219 – 229, 1998.

[5] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, December 1984.

[6] Michal Jakob, Michal Pěchouček, Peter Novák, Michal Čáp, and Ondra Vaněk. Towards incremental development of human-agent-robot applications using mixed-reality testbeds. *IEEE Intelligent Systems*, 27(2):19–25, 2012.

[7] Karaman and Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, June 2011.

[8] Sven Koenig and Maxim Likhachev. D* lite. In *Proceedings of the national conference on artificial intelligence*, pages 476–483. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[9] E. Lalish. *Distributed Reactive Collision Avoidance*. BiblioBazaar, 2011.

[10] Friedemann Mattern. Algorithms for distributed termination detection. *Distributed computing*, 2(3):161–175, 1987.

[11] Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev. Anytime safe interval path planning for dynamic environments. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4708–4715. IEEE, 2012.

[12] Martin Selecký, Antonín Komenda, Michal Štolba, Tomáš Meiser, Michal Čáp, Milan Rollo, Jiří Vokřínek, and Michal Pěchouček. Deployment of multi-agent algorithms for tactical operations on uav hardware (demonstration). In *Proceedings of AAMAS 2013 (to appear)*, 2013.

[13] David Silver. Cooperative pathfinding. In *AIIDE*, pages 117–122, 2005.

[14] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.

[15] Trevor Scott Standley and Richard E. Korf. Complete algorithms for cooperative pathfinding problems. In Toby Walsh, editor, *IJCAI*, pages 668–673. IJCAI/AAAI, 2011.

[16] Jur Van Den Berg, Stephen Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. *Robotics Research*, pages 3–19, 2011.

[17] Jur van den Berg and Mark Overmars. Prioritized motion planning for multiple robots. In *IROS*, pages 430–435, 2005.

[18] Jur van den Berg and Mark Overmars. Kinodynamic motion planning on roadmaps in dynamic environments. In *IROS*, pages 4253–4258. IEEE, 2007.

[19] Prasanna Velagapudi, Katia P. Sycara, and Paul Scerri. Decentralized prioritized planning in large multirobot teams. In *IROS*, pages 4603–4609. IEEE, 2010.