2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Distributed Real Time Control of Multiple UAVs in Adversarial Environment: Algorithm and Flight Testing Results

Mohamed Abdelkader, Yimeng Lu, Hassan Jaleel, and Jeff S. Shamma

Abstract— **We present a complete design and implementation of a system that consists of multiple quadrotors playing capture the flag game. Our main contributions in this work include a custom built quadrotor platform, an efficient implementation of a distributed trajectory planning algorithm, and a WiFi based communication infrastructure. In our design, we equip the quadrotors with autopilot modules for low level control. Moreover, we install low power computing modules for implementing the distributed trajectory planning algorithm online. Furthermore, we develop a communication infrastructure to enable coordination among the quadrotors, which is required for computing a suboptimal control action in real time. The interactions among all the hardware and software components are managed at a higher level by Robot Operating System (ROS). To test the performance of the system, we select a motivating setup of capture the flag game, which is an adversarial game played between two teams of agents called attack and defense. The system is initially simulated in the Gazebo robot simulator with software in the loop. Finally, the complete system is tested and the flight testing results are presented.**

SUPPLEMENTARY MATERIAL

A demonstration video of this work is available at: `https://youtu.be/z7IfTLO_Fyg`

Also, an open-source code that is used in the simulation and hardware testings is available at `https://github.com/mzahana/dlp`

I. INTRODUCTION

Unmanned aerial vehicles (UAV)s are now becoming ubiquitous with applications ranging from package delivery, precision agriculture, disaster assessment, surveillance, and infrastructure monitoring (see e.g., [1], [2], [3], and [4] and the references therein). An active area of research that is receiving significant research attention is the real-time distributed control of systems with multiple UAVs. Although distributed path planning has been an active area of research (see e.g., [5], [6], [7], [8], and [9]), the performance constraints because of limited battery power and limited onboard sensing, processing, and communication capabilities often prohibit us from directly employing existing techniques. The complexity of these problems is further intensified by the complex dynamics of the mobile platforms like quadrotors.

M. Abdelkader, Y. Lu, H. Jaleel and J. S. Shamma are with the King Abdullah University of Science and Technology (KAUST), Thuwal 23955–6900, Saudi Arabia. M. Abdelkader is with Physical Sciences and Engineering Division (PSE). Y. Lu, H. Jaleel and J. S. Shamma are with Computer, Electrical and Mathematical Sciences and Engineering Division (CEMSE). Email: mohamed.abdkader@kaust.edu.sa, yimeng.lu@kaust.edu.sa, hassan.jaleel@kaust.edu.sa, jeff.shamma@kaust.edu.sa. Research supported by funding from KAUST.

For a system consisting of quadrotors, which have higher order complex dynamics, are battery powered, and communicate over noisy communication channels in dynamic and uncertain environment, the control action has to be computed online to handle the uncertainties in the system dynamics and environment. In such systems, the priority in designing a control system shifts from computing an optimal control action with high computational complexity to computing a feasible control action that can be computed efficiently in real time and can guarantee a minimum required level of performance. In [10], we presented a framework that was based on this approach. In this work we verify the performance of the framework in [10] by implementing a complete multiagent system.

We present an end-to-end design and implementation of a multiagent system for playing capture the flag game. This game as presented in [9] and [11] is an adversarial game between two teams, defenders and attackers. Our system consists of four quadrotors, three of which are defenders and one is an attacker. We formulate the distributed planning problem from the perspective of the defense team. Any strategy of the defense team has to depend on the actions and strategy of the attacker. Since defenders cannot know attacker's strategy in advance, they assume a model of attacker's strategy. Then, using the assumed model, the defenders coordinate with each other to compute their actions by following the framework of model predictive control (MPC) as was presented in [10].

We first simulated the complete system in the robotic simulator environment Gazebo. In the Gazebo simulation, we used a realistic quadrotor model in which the lower level control of a quadrotor was handled by the Pixhawk PX4 autopilot software. The same auto pilot software was later used for the low level control of our actual quadrotors. To compute the actions of defenders, we implemented the LP based distributed real-time path planning algorithm from [10] with an assumed model for attacker's strategy.

For practical implementation, we custom-built all the quadrotors to satisfy our requirements. We equipped each quadrotor with a low-power computing module, which was an ODROID XU4 embedded Linux computer. Moreover, we installed WiFi modules on each quadrotor so that they can coordinate with each other. Thus, by equipping the quadrotors with low-power computing and communication modules, we enabled them to efficiently solve a distributed optimization problem online. For onboard path planning, we implemented the algorithm from [10] in C++, and the algorithm was executed on the ODROID XU4 embedded computer. The interactions among all the hardware and software components in the system were managed at a higher level by the Robot Operating System (ROS) [12].

To evaluate the performance of our system, we simulated and rigorously tested the actual system under various operating conditions. An important aspect of our flight test results is human in the loop. In the distributed optimization algorithm, the defenders used an assumed model for attacker's strategy to compute their control actions. However, to create an actual scenario, we setup a joystick to enable a human operator to control the attacker quadrotor in both the Gazebo environment and the practical setup. Thus, with a human operating the attacker, we can claim with confidence that our flight test results are a true reflection of the actual performance of our system and our online distributed optimization algorithm.

II. PROBLEM FORMULATION

A. System Description

The example setup selected to verify the performance of the hardware platforms and the the proposed algorithms is capture the flag game as presented in [13]. In this section we present a detailed description of the system.

- *Battlefield*: The battlefield for the game is a grid, which is defined as a collection of sectors $\mathcal{S} = \{s_1, \cdots, s_{n_s}\}$, where n_s is the total number of sectors. The defense zone is a collection of base sectors, $\mathcal{S}_{\text{base}} \subset \mathcal{S}$, that should be defended, and reference sectors, $\mathcal{S}_{\text{ref}} \subset \mathcal{S}$, that should be occupied by defenders to provide a protective perimeter around base sectors.
- *Teams*: The set of players $\mathcal{A} = \{a_1, a_2, \ldots, a_{n_p}\}$ is divided into two teams, defender team (*d*) and attacker team (*e*). Each team is represented by a set

$$\mathcal{A}^r = \{a_1^r, \cdots, a_{n_r}^r\},$$

where $r \in \{d, e\}$ is the type of the team, a_i^r is the i^{th} player of type r, and n_r is the total number of agents in the respective team r such that $n_d + n_e = n_p$.

- *States*: The state of each sector s_i is $x_{s_i} \in \mathbb{Z}_+$, which is the number of its occupants, where $\mathbb{Z}_+$ is the set of non-negative integers. The state of an agent $a_i \in \mathcal{A}$ is

$$x_{a_i} = [x_{s_1,a_i} \ldots x_{s_{n_s},a_i}]^T \in \mathcal{B}^{n_s},$$

where $\mathcal{B} \triangleq \{0, 1\}$ and

$$x_{s_j,a_i} = \begin{cases} 1 & a_i \text{ occupies } s_j \\ 0 & \text{otherwise.} \end{cases}$$

The state of the grid with respect to all the agents is

$$\mathbf{x} = \sum_{a_i \in \mathcal{A}} x_{a_i},$$

and the state of the grid with respect to a team $r \in \{d, e\}$ is

$$\mathbf{x}^r = \sum_{a_i^r \in \mathcal{A}_r} x_{a_i^r}.$$

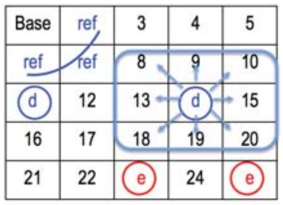


Fig. 1. Representation of an example battlefield. The defense zone is $\mathcal{S}_{\text{base}} = \{s_1\}$ labeled 'Base' and $\mathcal{S}_{\text{ref}} = \{s_2, s_6, s_7\}$ labeled 'ref'. There are two defenders that are located at sectors s_{11} and s_{14}. Neighborhood of the defender at sector s_{14} is represented by the arrows pointing to sectors $\{s_8, s_9, s_{10}, s_{13}, s_{15}, s_{18}, s_{19}, s_{20}\}$. Attackers are labeled as 'e' and are located at sectors s_{23} and s_{25}.

- *Actions*: Actions are described by transfer of agents between sectors in $\mathcal{S}$. For an agent a_j, we define an input with respect to sector s_i as

$$u_{s_i \to s_k, a_j} = \begin{cases} 1 & x_{s_i,a_j} = 1 \text{ and } a_j \text{ transfers to } s_k \\ 0 & \text{otherwise.} \end{cases}$$

An augmented input vector for agent a_j is

$$\mathbf{u}_{s_i,a_j} = [u_{s_i \to s_1,a_j} \ldots u_{s_i \to s_{i-1},a_j} u_{s_i \to s_{i+1},a_j} \ldots u_{s_i \to s_{n_s},a_j}]^T.$$

If agent a_j occupies sector s_i, then $\mathbf{u}_{s_i,a_j} \in \mathcal{B}^{(n_s-1)}$ represents its transfer to some other sector in the grid. The complete input vector of agent a_j with respect to all the sectors in the grid is

$$\mathbf{u}_{a_j} = [\mathbf{u}_{s_1,a_j}^T \ \mathbf{u}_{s_2,a_j}^T \ldots \mathbf{u}_{s_{n_s},a_j}^T]^T \in \mathcal{B}^{n_u}.$$

where $n_u = n_s(n_s - 1)$. Similar to states, the input vectors for defenders and attackers are

$$\mathbf{u}^d = \sum_{a_i^d \in \mathcal{A}^d} \mathbf{u}_{a_i^d} \qquad \mathbf{u}^e = \sum_{a_i^e \in \mathcal{A}^e} \mathbf{u}_{a_i^e}$$

B. Dynamical Model and Constraints

The evolution of the state of s_i with respect to agent a_j is modeled by the following linear model.

$$x_{s_i,a_j}^+ = x_{s_i,a_j} + \sum_{s_k \in \mathcal{N}(s_i)} u_{s_k \to s_i,a_j} - \sum_{s_k \in \mathcal{N}(s_i)} u_{s_i \to s_k,a_j} \tag{1}$$

where x_{s_i,a_j}^+ is the updated state of sector s_i with respect to agent a_j. The first and the second terms in (1) represent the flow of a_j into and out of sector s_i, respectively, with respect to the neighborhood set of s_i defined by $\mathcal{N}(s_i)$. The neighborhood set of a sector s_i is the set of all sectors that surrounds it (see Fig. 1). The state of a sector is constrained to be non-negative (only positive resources).

$$x_{s_i,a_j} \in \mathcal{B} \triangleq \{0, 1\},$$
$$0 \le \sum_{s_k \in \mathcal{N}(s_i)} u_{s_i \to s_k,a_j} \le x_{s_i,a_j} \tag{2}$$

for all $a_j \in \mathcal{A}$.

The dynamical model (1) and the constraints (2) for the defender team can be written in a compact form as

$$\begin{cases} (\mathbf{x}^d)^+ = \mathbf{x}^d + \mathbf{B}_{\text{in}}\mathbf{u}^d - \mathbf{B}_{\text{out}}\mathbf{u}^d = \mathbf{x}^d + \mathbf{B}\mathbf{u}^d \\ \mathbf{0} \leq \mathbf{B}_{\text{out}}\mathbf{u}^d \leq \mathbf{x}^d \\ \mathbf{x}^d \in \mathcal{B}^{n_s},\ \mathbf{u}^d \in \mathcal{B}^{n_u} \end{cases} \tag{3}$$

for appropriate $\mathbf{B}_{\text{in}}$ and $\mathbf{B}_{\text{out}}$. We refer the reader to [10] and [9] for further details on the problem setup.

C. Problem Setup

In this game, the objective is to find optimal control inputs for defenders, $(\mathbf{u}^*)^d[t]$ for all $t \in \{0, 1, \ldots, T_{\text{game}}\}$, such that interventions of the attackers into the base, $\mathcal{S}_{\text{base}}$, are minimized. To achieve this objective, we formulate the cost function at time instant t as a linear function of the state $\mathbf{x}[t]$

$$J_t(\mathbf{x}[t]) = (\alpha \mathbf{x}^e[t] + \beta \mathbf{x}_{\text{ref}}[t])^T\, \mathbf{x}^d[t] \tag{4}$$

This objective function captures two main performance aspects at a particular time step t. The term $\mathbf{x}^e[t]^T\mathbf{x}^d[t]$ is maximized when there is maximum overlap between the locations of defenders and attackers. Therefore, for $\alpha < 0$, the first term captures pursuit behavior by the defenders. The higher the magnitude of α, the more aggressive the defenders will be in pursuing the attackers. An attacker is captured if its sector is occupied by a defender as well. However, to avoid collisions for testing purposes, we will assume that an attacker is captured if a defender comes within some specified distance. Similarly, $\mathbf{x}_{\text{ref}}[t]^T\mathbf{x}^d[t]$ increases as the defenders remain in the reference sectors. Thus, the second term captures surveillance behavior when $\beta < 0$, and higher magnitude of β forces the defenders to remain in $\mathcal{S}_{\text{ref}}$ and protect $\mathcal{S}_{\text{base}}$ from there. The coefficients α and β are selected to assign the desired weights to each of the behavior and they are normalised so that $|\alpha|, |\beta| \in [0, 1]$ and $|\alpha| + |\beta| = 1$.

D. Local Optimization Problem

Each agent solves a local linear program problem (5) that has objective function (4) and constraints (3) over a prediction horizon T_p. In each optimization run, an agent needs to include only its neighbors' current positions either through communication or sensing. Each agent can avoid static predefined obstacles locations by including a single linear constraint $\mathbf{X}^T_{\text{obstacles}}\mathbf{X}^{d_i} = 0$. The problem solution includes an estimate of the agents next transitions as well as its neighbors over T_p. It only executes the first control input at $t = 0$ as in a receding horizon sense. For a more rigorous analysis on the proposed approach, the reader is referred to the previous work in [10] and [9].

$$\begin{aligned} \min\ & [\alpha \mathbf{X}_e + \beta \mathbf{X}_{\text{ref}}]^T\ \mathbf{X}^{d_i} \\ \text{s.t.}\quad & \mathbf{X}^{d_i} - \mathbf{T}_u\ \mathbf{U}^{d_i} = \mathbf{T}_{x_0}\ \mathbf{x}^{d_i}[0] \\ & \mathbf{T}_{u,c}\ \mathbf{U}^{d_i} \leq \mathbf{T}_{x_0,c}\ \mathbf{x}^{d_i}[0] \\ & \mathbf{0} \leq \mathbf{X}^{d_i} \leq \mathbf{1} \\ & \mathbf{0} \leq \mathbf{U}^{d_i} \leq \mathbf{1} \\ & \mathbf{X}^{\mathbf{T}}_{\text{obstacles}}\mathbf{X}^{d_i} = 0 \\ & \mathbf{X}^{T}_{\text{collision}}\mathbf{X}^{d_i} = 0 \end{aligned} \tag{5}$$

In contrast to the centralized problem, an extra equality constraint is added, which is the last constraint in (5) to guarantee collision-free one-step ahead transitions of defenders. $\mathbf{X}_{\text{collision}}$ encodes the sectors that should be avoided in the transitions.

III. SIMULATION SETUP

In this section, we present our simulation setup that validates the proposed algorithm in simulated real-time scenarios. We aim to provide simulations that are highly realistic and as close as possible to actual hardware implementation. The following subsections describe the simulation setup.

A. Simulation Components

We use quadrotors as our testing platforms as they provide high maneuverability even in constrained test spaces. Each quadrotor is controlled by two levels of controllers. A low-level controller of attitude, velocity, and position stabilization is handled by the open-source PX4 autopilot firmware[1]. A high-level controller executes the proposed algorithm and provides position setpoints to the low-level controller.

Our simulation setup includes four components as follows.

- *Distributed linear program algorithm*: An implementation of the proposed algorithm as a C++ class. The algorithm's class provides convenience functions to easily setup problem's parameters and inputs, executes the proposed LP algorithm. The algorithm uses the open-source linear program C++ library, `GLPK` [2]. Also, a ROS C++ node is available to interface between the algorithm class and ROS environment.
- *PX4*: Open-source autopilot software, which provides low-level control to the quadrotor's states including orientation, velocity, and position. The same software is used in both simulation and hardware testings. The PX4 firmware used in the simulations is called software-in-the-loop, SITL. The reader is referred to the PX4 developer guide[3] on how to setup the SITL simulation for multiple UAVs.
- *Gazebo*: Robotic simulator which provides realistic simulation of a robot's dynamics and simulated sensors. Also, it can simulate a more realistic environment by including a wind and magnetic field profiles.
- *ROS*: Robot Operating System, which provides a convenient interface between all simulation components and reduces (or eliminates) required changes to transfer the implementation to actual hardware setup.

The proposed simulation setup is depicted in Fig. 2

The simulation setup runs on an i7 computer which runs Ubuntu 14.04, ROS Indigo, and Gazebo 7.

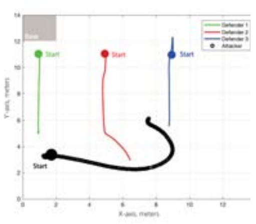
B. Simulation Results

In this subsection, we present simulation results of some test cases using the proposed simulation setup.

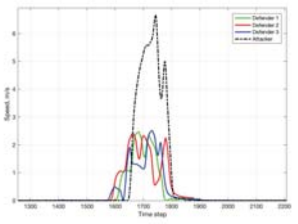
[1]`px4.io`

[2]`https://www.gnu.org/software/glpk/`

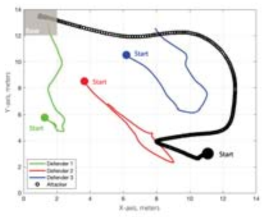
[3]`https://dev.px4.io/en/simulation/multi-vehicle-simulation.html`



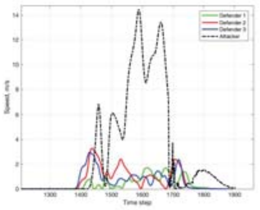
(a) Position trajectories of simulation case 1



(b) Velocity trajectories of simulation case 1



(c) Position trajectories of simulation case 2



(d) Velocity trajectories of simulation case 2

Fig. 3. (a) and (b) shows position and velocity trajectories of all quadrotors in simulated scenario where attacker was captured at position $(7.8, 6.1)$. Figures (c) and (d) show position and velocity trajectories for case 2.

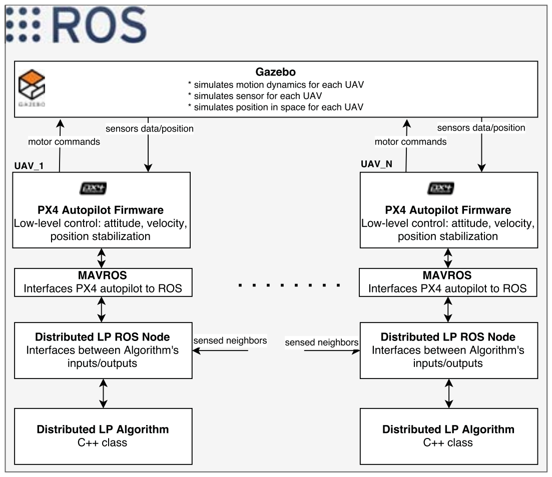

ROS
Gazebo
* simulates motion dynamics for each UAV
* simulates sensor for each UAV
* simulates position in space for each UAV
motor commands
sensors data/position
motor commands
sensors data/position
UAV_1
UAV_N
PX4 Autopilot Firmware
Low-level control: attitude, velocity, position stabilization
PX4 Autopilot Firmware
Low-level control: attitude, velocity, position stabilization
MAVROS
Interfaces PX4 autopilot to ROS
MAVROS
Interfaces PX4 autopilot to ROS
Distributed LP ROS Node
Interfaces between Algorithm's inputs/outputs
sensed neighbors
sensed neighbors
Distributed LP ROS Node
Interfaces between Algorithm's inputs/outputs
Distributed LP Algorithm
C++ class
Distributed LP Algorithm
C++ class

Fig. 2. Architecture of simulation setup

Case 1: In this case the problem setup includes 3 quadrotors in the defenders' team and one quadrotor in the attacker's team. The UAVs battle inside a $2D$ square grid of 7×7 sectors, and each sector is 2×2 meters. The strategy coefficients were set to $\alpha = -0.99$ and $\beta = -0.01$ for all defenders to encourage attacking behavior. To make the battle more interesting, the attacking quadrotor is controlled by a human operator through a joystick, which controls the quadrotor's lateral velocities. An interesting note is that, the human controlled quadrotor is allowed to exhibit any behavior although the defenders assume a specific predictive attacking model. This helps to evaluate the distributed defending strategies against non-modeled attacking behavior.

Fig. 3(a) shows the UAV trajectories during the battle. As we can see, defenders show distributed attacking behavior in order to block the human controlled attacker from entering the base zone, before it is eventually captured at position $(7.8, 6.1)$. Although defenders assume that the attacker would follow a path the minimizes distance to the base which the human did not actually commit to, the were still able to be around the attackers at all times. There are situations where the attacker was able to reach the base by having much more velocity compared to attackers (around 5 times more) which is trivial in this case, but the figure are omitted due to limited space.

Case 2: : In this case, the human controlled attacker is given much more speed than defenders. Defenders still exhibit attacking behaviors by using the same strategy coefficients $\alpha = -0.99$ and $\beta = -0.01$. In this case the attacker operator was able to drag defenders towards his end of the field and utilize the higher speed authority to escape through the right side of the area, and eventually reaching the base. Figures 3(c) and 3(d) show the corresponding position and velocity trajectories during the simulation trail of case 2. The velocity of the human controlled attacker reached up to 6 times the velocity of defenders which increased the ability of the attacker to reach the base.

The previous two simulation cases show that defenders are able to execute distributed strategies to capture the attacker when the capabilities, e.g. maximum velocities, are comparable. Also, although the human operator has full view of the game situation and did not follow the attacking behavior assumed by defenders, it was challenging to reach the base with comparable velocity.

IV. HARDWARE SETUP & TESTING

As mentioned before, the main focus of this work is to have a real-time distributed algorithm that can be implemented in low-cost and resource-constrained setups. In multi-rotor UAVs, power consumption and weight are major

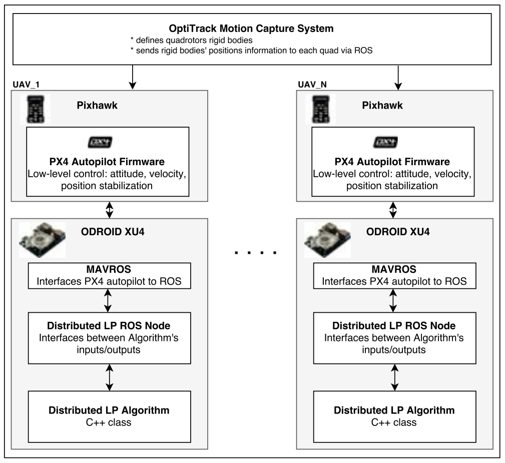

OptiTrack Motion Capture System
* defines quadrotors rigid bodies
* sends rigid bodies' positions information to each quad via ROS
UAV_1
UAV_N
Pixhawk
PX4 Autopilot Firmware
Low-level control: attitude, velocity, position stabilization
ODROID XU4
MAVROS
Interfaces PX4 autopilot to ROS
Distributed LP ROS Node
Interfaces between Algorithm's inputs/outputs
Distributed LP Algorithm
C++ class
. . . .

Fig. 4. Architecture of hardware setup

constraints in their design, which limits their endurance. Therefore, it is always preferred to reduce payload and power consumption as much as possible. In that regard, we opt to use low-power and low-weight affordable components that allow us to achieve the computational needs and real-time execution for the proposed approach.

In this section, we present our hardware setup and evaluate the performance of the proposed distributed algorithm in real-time. Although it is more realistic to perform experiments in a real outdoor environment, we start by testing it in a controlled indoor environment, in order to ensure better debugging and discover/fix possible malfunctioning.

A. Hardware Components

The main hardware components are summarized as follows:

- Indoor flying arena equipped with OptiTrack motion capture system.
- Quadrotors: 25cm wide
- Pixhawk: Open-source autopilot
- Odroid XU4: Onboard embedded Linux computer
- Communications: A WiFi communication setup is used in order to communicate mocap position estimates to quadrotors flight controllers.

The system architecture is depicted in Fig. 4, and detailed description is as follows. We perform our indoor flight test, in the flying arena of our Robotics, Intelligent Systems & Control (RISC) lab. The arena has dimensions of $6 \times 7 \times 3$ meters. The arena is equipped with OptiTrack motion capture system[4] which acts as an indoor positioning system, as GPS receivers do not perform well indoors. The motion capture system provides position information of defined rigid bodies e.g. quadrotors. A Linux machine equipped with ROS receives the position information from motion capture system and publishes them as ROS topics which are then transmitted to each UAV, to become aware of its position in the environment as well as its neighbors.

[4]`http://optitrack.com`

Fig. 5. Quadrotor setup

Each UAV is a small quadrotor of 0.25m width, see Fig. 5. It is equipped with a Pixhawk autopilot which runs open-source PX4 firmware that is used for the attitude, velocity, and position stabilization. Pixhawk is serially connected to an on-board embedded Linux computer, ODROID XU4. ODROID XU4 runs ROS Indigo, MAVROS package (ROS interface to Pixhawk), and a ROS node that interfaces the proposed algorithm to ROS. ODROID mainly receives positions from motion capture system via WiFi, executes the proposed algorithm, and sends position setpoints to Pixhawk.

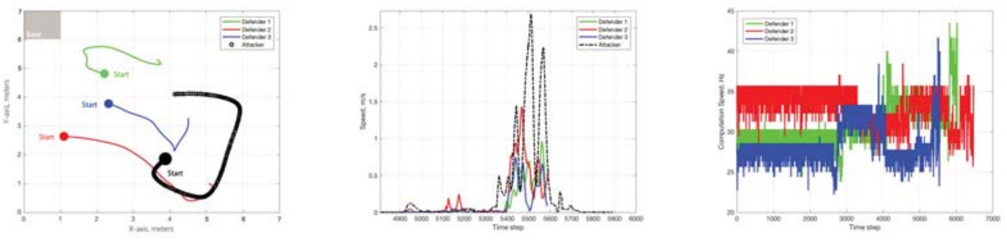
B. Hardware Testing & Results

Prior to flight test experiments, we first test the execution speed of the algorithm on ODROID XU4 for different problem setups. This allows the selection of a reasonable problem configuration that can be run in real-time on the hardware of interest. Table I shows the average execution speed of one run of the algorithm on ODROID XU4, in Hz, for different problem setups.

TABLE I
ALGORITHM EXECUTION SPEED ON ODROID XU4

Setup #	Sectors	Agents	Prediction steps (T_p)	**Freq.** (Hz)
1	10×10	3 vs. 1	3	2.3
2	10×10	3 vs. 1	2	5.5
3	10×10	3 vs. 1	1	22.7
4	7×7	3 vs. 1	3	11.0
5	7×7	3 vs. 1	2	27.8
6	7×7	3 vs. 1	1	111.0

As we can see from Table I, the proposed algorithm can be executed at relatively high rates for different problem configurations. In our experiments, we choose configuration 5. The corresponding optimization problem size in terms of number of variables and constraints is $(n_s + n_s^2)T_p = 4900$, $(5n_s + 2n_s^2)T_p = 10094$, respectively. It is important to note that the problem structure is highly sparse. The total number of non-zero element in the overall constraints matrix is of order $\mathcal{O}((2T_p + 4N^2)n_s^2)$, and the total number of all elements is of order $\mathcal{O}(2T_p^2 n_s^4)$. Where $N = \max_i |\mathcal{N}(s_i)| \ \ i \in \{1, \cdots, n_s\}$ is the maximum number of reachable sectors in 1 time step. For the chosen problem configuration, the sparsity is approximately 98%. This sparsity is taken into account in the algorithm implementation, which, in result, dramatically boosts the computation speeds on ODROID XU4 as presented in Table I.



(a) Position trajectories (b) Velocity trajectories (c) Execution frequency profile of defenders during a real trial

Fig. 6. Fig. (a) shows position trajectories of all quadrotors during a real experiment where the attacker was captured at position $(4.1, 4.1)$. Fig. (b) shows the corresponding speed profile. Fig. (d) shows the algorithm execution speed for all defenders during the game run, where the average is around 30 Hz

Hardware trials were performed using similar configuration as in the simulation test. In particular, the strategy coefficients for defenders were set to $\alpha = -0.99, \beta = -0.01$ which encourages attacking behavior. Fig. 6(a) shows the position trajectories of the all quadrotors in a real experiment, where the human controlled attacker was captured at position $(4.1, 4.1)$. The corresponding velocity profile is depicted in Fig. 6(b).

Fig. 6(c) show executions frequency profiles of 3 defenders quadrotors during the a game trial. The average execution speed of the algorithm was around 30Hz. Such relatively high rate allows quadrotors to produce fast decisions, which result in smooth and rapid reactions against attackers. The main message of the hardware experiment presented here is to show the ability of the proposed algorithm reasonably well on actual hardware and in real-time using low power computing modules.

V. CONCLUSIONS

We presented a complete design and implementation of a multiagent system consisting of four quadrotors. As an example setup, the system was designed to play capture the flag with three defenders and one attacker. We equipped the custom built quadrotors with suitable onboard computing and communication devices with the capability to solve the required optimization problems onboard efficiently. We implemented an LP based trajectory planning algorithm and verified that the algorithm is suitable for computing the update commands for quadrotors in real time. To generate a realistic scenario, the attacking quadrotor was controlled by a human operator and the control actions for the defenders were generated by solving the distributed optimization problem online with coordination among the defenders. This system presented in this work can be used as a testbed for developing real-time distributed algorithms for multi-UAV systems for various applications.

As a future work, an outdoor setup would allow testing of more realistic scenarios in larger spaces, where several strategies and behaviors can be examined. In addition, it would be interesting to compare the defenders strategies against different assumed enemy models.

REFERENCES

[1] R. D'Andrea, "Guest editorial can drones deliver?" *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 3, pp. 647–648, 2014.

[2] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 46–56, 2012.

[3] E. Honkavaara, H. Saari, J. Kaivosoja, I. Pölönen, T. Hakala, P. Litkey, J. Mäkynen, and L. Pesonen, "Processing and assessment of spectrometric, stereoscopic imagery collected using a lightweight uav spectral camera for precision agriculture," *Remote Sensing*, vol. 5, no. 10, pp. 5006–5039, 2013.

[4] S. Gupte, P. I. T. Mohandas, and J. M. Conrad, "A survey of quadrotor unmanned aerial vehicles," in *Southeastcon, 2012 Proceedings of IEEE*. IEEE, 2012, pp. 1–6.

[5] M. Flint, M. Polycarpou, and E. Fernandez-Gaucherand, "Cooperative path-planning for autonomous vehicles using dynamic programming," in *Proceedings of the IFAC 15th Triennial World Congress*, 2002, pp. 1694–1699.

[6] W. B. Dunbar and R. M. Murray, "Model predictive control of coordinated multi-vehicle formations," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 4. IEEE, 2002, pp. 4631–4636.

[7] G. Ferrari-Trecate, L. Galbusera, M. P. E. Marciandi, and R. Scattolini, "Model predictive control schemes for consensus in multi-agent systems with single-and double-integrator dynamics," *IEEE Transactions on Automatic Control*, vol. 54, no. 11, pp. 2560–2572, 2009.

[8] M. A. Müller, M. Reble, and F. Allgöwer, "Cooperative control of dynamically decoupled systems via distributed model predictive control," *International Journal of Robust and Nonlinear Control*, vol. 22, no. 12, pp. 1376–1397, 2012.

[9] G. C. Chasparis and J. S. Shamma, "Lp-based multi-vehicle path planning with adversaries," *Cooperative Control of Distributed Multi-Agent Systems*, pp. 261–279, 2008.

[10] M. Abdelkader, H. Jaleel, and J. S. Shamma, "A distributed framework for real time path planning in practical multiagent systems," in *IFAC World Congress, Toulouse, France*, 2017, to appear. [Online]. Available: https://www.dropbox.com/sh/urqebc8jz6gk235/AAASpIi69Q37-KubSHf32Z8ra?dl=0

[11] R. D'Andrea and M. Babish, "The roboflag testbed," in *American Control Conference, 2003. Proceedings of the 2003*, vol. 1. IEEE, 2003, pp. 656–660.

[12] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[13] H. Huang, J. Ding, W. Zhang, and C. J. Tomlin, "Automation-assisted capture-the-flag: A differential game approach," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 3, pp. 1014–1028, May 2015.