Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

CAVIS: a Control software Architecture for cooperative multi-unmanned aerial VehIcle-manipulator Systems

G. Antonelli [*] **K. Baizid** [*] **F. Caccavale** [**] **G. Giglio** [**]
F. Pierri [**]

[*] *Dipartimento di Ingegneria Elettrica e dell'Informazione, Università di Cassino e del Lazio Meridionale, Via Di Biasio 43, 03043 Cassino (FR), Italy (e-mail:{antonelli, baizid}@unicas.it)*
[**] *Scuola di Ingegneria, Università degli Studi della Basilicata, Via dell'Ateneo Lucano, 10, 85100 Potenza, Italy (e-mail: {fabrizio.caccavale, francesco.pierri, gerardo.giglio}@unibas.it)*

Abstract: In this paper a Control software Architecture for Cooperative multiple unmanned aerial VehIcle-manipulator Systems (CAVIS) is presented. The core of the architecture is a set of software components, communicating each other through a set of defined messages. To handle multiple control objectives simultaneously, a library of elementary behaviors is defined; then, multiple elementary behaviors are combined, in a given priority order, into tasks (compound behaviors); to this aim the Null-Space-based Behavioral (NSB) approach has been adopted. An application example, involving a cooperative transportation of a bar by two aerial vehicle-manipulator systems, is developed to assess the performance of the proposed architecture.

Keywords: Control software architectures, Unmanned aerial vehicle-manipulator systems, Multi-robot systems, Cooperation, Coordination.

1. INTRODUCTION

In the last decades Multi-Robot Systems (MRSs) have played an important role in many robotics applications. Several efforts have been spent on the design and the development of software/hardware platforms that support MRS in cooperative tasks. The main objective was to build Control Software Architecture (CSA) that handle the complexity and the heterogeneity of such systems (Gancet et al., 2005; Ortiz et al., 2011). Consequently, some research works suggested many features that must be considered at the design phase, such as: Modularity, Integration and Reusability (Oreback and Christensen, 2003).

Recently, many middleware/framework aimed to support software development appeared, such as URBI (Baillie, 2004), Miro (Utz et al., 2002), Marie (C. Cote et al., 2002) and MRDS [1]. Moreover, some of these frameworks (which are open source for public contributions, having a distributed architecture and an easy *integration environment*) became more popular such as Player/Stage (Brian P. Gerkey, 2003) and ROS (Quigley Morgan and Gerkey, 2009). In addition, a modular framework, named OROCOS, has been proposed in Bruyninckx (2001). It supports developing applications in C/C++ programming language under three OS (Linux, Win32 and Mac); it provides real-time toolkit and several libraries such as Bayesian filtering, kinematics and dynamics. Also, the Mobile-R (Nestinger and Cheng, 2011), which provides known packages such as ANNs (Artificial Neural Networks) and GAs (Genetic Algorithms). One of the most recent platforms is ViRAT (Virtual Reality for Advanced Tele-operation) that combines users skills and robots' capabilities in the control loop; it is proposed in Khelifa and H. (2011). The platform is developed under Linux and it supports heterogeneous MRS (*rovers*, *humanoid*, etc.) and it provides an easy environment for scenario creations and many libraries (developed under C/C++) for cooperation and coordination. This huge number of middleware/framework systems (RoSta [2]) helped to create a great trend of applications (Dias et al., 2006; Oreback and Christensen, 2003).

Although the large number of CSA for MRS, only few of them treat the multi-UAV system such as Ortiz et al. (2011), Shi and Yang (2008) and Gancet et al. (2005). However, their works may not covers the wide range of these systems. Moreover, the adaptation of such existing CSA is very challenging, due to the specification imposed by each robotic system (i.e., additional software blocks are, always, needed). To this end, the consideration of the robotic system specifications during the design of the CSA becomes an indispensable issue, especially when considering systems such as Unmanned Aerial Vehicle Manipulator Systems (UAVMSs) where particular attentions must be given to issues such as safety and physical cooperation.

In addition to these challenges, tasks management and missions planning must be considered in the design phase

[⋆] This work was supported by the European Community 7th Framework Program under grant o.287617 (IP project ARCAS - Aerial Robotics Cooperative Assembly system)
[⋆⋆]Authors are in alphabetical order
[1] http://www.microsoft.com/robotics/

[2] http://wiki.robot-standards.org/

of the software architecture. Recently, several research works adopted concepts such as hierarchical decomposition of missions planning (Keith et al., 2009). The mission is decomposed into a set of elementary behaviors (i.e., generic functionalities in term of software), thus in the case of complex scenarios, it is very useful to combine many of them together, e.g., by resorting to the Null-Space based Behavioral (NSB) concept (Antonelli et al., 2010).

To handle these challenges we developed a Control software Architecture for cooperative multi unmanned aerial VehIcle manipulator Systems (CAVIS) to support researches dealing with cooperative UAVMSs. Moreover, to deal with CSA design complexity we take benefit from the concept of *decomposition of system into components*, which is reported as a very critical issue for modern software architecture design (Nesnas et al., 2003). The CSA schema is developed through exploitable generic behaviors that are instantly referred to the current state of each UAVMS. The task planning concept is intended to support cooperative UAVMS to address complex scenarios based on the NSB approach (Antonelli et al., 2010). The software is composed of several components that, together, support some important features such as: flexibility, modularity and reusability. An application example, involving a cooperative bar transportation scenario by two UAVMS, is developed to assess the proposed CSA performance.

2. CAVIS: A GENERAL DESCRIPTION

In this Section, the proposed Control software Architecture for Cooperative multi unmanned Aerial VehIcle manipulator Systems (CAVIS) is presented. Due to the complexity of the control of multi-UAVMSs, the CSA needs to be developed by taking into account several requirements

(1) support multiple UAVMSs and their corresponding physical constraints, for example actuator limits;
(2) support vehicles' heterogeneity in both attached manipulator and sensor equipments;
(3) be compatible with a cooperative case study;
(4) include a library of suitable functionalities.

2.1 Main components

With a bottom-up perspective, the core concepts needs to decompose the overall control problem are listed below:

- **Elementary behavior**: at the kinematic level, it is the *atomic* functionality to be controlled;
- **Task**: is a set of arranged elementary behaviors in priority order (*Compound behavior*).
- **Action**: is a *conceptual* layer, that includes several tasks.
- **Mission**: is a set of ordered actions that are assigned to given UAVMS.
- **Scenario**: is the higher-level description of the control problem, it must include at least one mission.

Figure 1 represents the hierarchy among the above defined concepts.

The proposed architecture can be seen as a chain of circulating information among different software components. To this end, the architecture working strategy is defined by a Finite State Machine (FSM), that responds to well

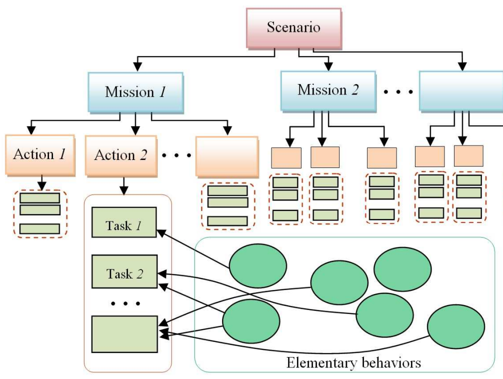


Fig. 1. Hierarchical decomposition of Scenario.

defined requests referred to each UAVMS state. Figure 2 shows the main software components of the proposed architecture:

- **Perception**: it provides relevant measures of the UAVMS and the environment (not described in this paper);
- **Planner**: its role is to generate the mission plan offline for a given scenario (not described in this paper);
- **Tasks Manager** (TM): it manages the whole mission and contains two main components:
  (1) **Coordination and Synchronization** (CS): its main role is the time scheduling of the assigned actions; more precisely, it synchronizes actions having time dependency or time delay.
  (2) **Supervisor** (SP): it selects the suitable task to be executed, based on the actual states of the robotic system.
- **Control Interface** (CI): its role is to implement the kinematic control to achieve the assigned task. Moreover, it assigns motion for each vehicle of the team in case of coordinated control, through the **Motion Splitter** (MS).

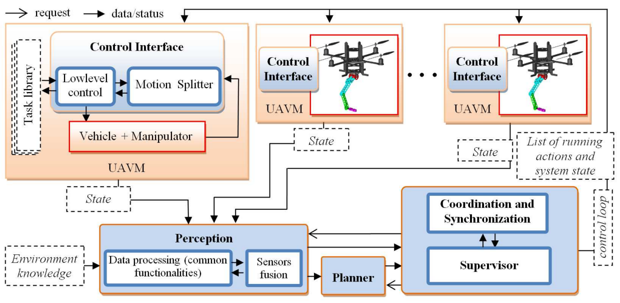


Fig. 2. Main software components of CAVIS architecture.

3. DECOMPOSITION OF THE CONTROL PROBLEM

In this section, the core of the control problem (i.e., the elementary behaviors, and their compositions named tasks), and details of the main components architecture are given.

3.1 Elementary behaviors

An elementary behavior assigned to a UAVMS can be analytically described by a task variable, $\boldsymbol{\sigma} \in \mathbb{R}^m$, that can be written as a function of the UAVMS configuration, $\boldsymbol{\zeta} = \left[\boldsymbol{p}_V{}^{\mathrm{T}}\boldsymbol{\phi}_V{}^{\mathrm{T}}\boldsymbol{q}^{\mathrm{T}}\right]^{\mathrm{T}}$, given by the vehicle position ($\boldsymbol{p}_V \in \mathbb{R}^3$) and orientation (expressed via a triple of Euler angles $\boldsymbol{\phi}_V \in \mathbb{R}^3$), and the manipulator joint positions ($\boldsymbol{q} \in \mathbb{R}^{n_M}$). The configuration-dependent task function, $\boldsymbol{\sigma}(\boldsymbol{\zeta})$, represents the relationship between the task variable and $\boldsymbol{\zeta}$, while the Jacobian matrix of the task, $\boldsymbol{J}_\sigma \in \mathbb{R}^{m\times(6+n_M)}$, can be defined via the differential relationship as follow:

$$\dot{\boldsymbol{\sigma}} = \frac{\partial\boldsymbol{\sigma}(\boldsymbol{\zeta})}{\partial\boldsymbol{\zeta}}\dot{\boldsymbol{\zeta}} = \boldsymbol{J}_\sigma(\boldsymbol{\zeta})\dot{\boldsymbol{\zeta}}\,. \tag{1}$$

In order to reach a desired value, $\boldsymbol{\sigma}_d$ of the task function, a reference value $\boldsymbol{\zeta}_r$ of the UAVMS configuration vector must be computed. To this aim, an inverse kinematics algorithm can be adopted (Siciliano et al., 2009), i.e.,

$$\dot{\boldsymbol{\zeta}}_r = \boldsymbol{J}_\sigma^\dagger(\dot{\boldsymbol{\sigma}}_d + \boldsymbol{\Lambda}\tilde{\boldsymbol{\sigma}}), \tag{2}$$

where $\boldsymbol{J}_\sigma^\dagger = \boldsymbol{J}_\sigma^{\mathrm{T}}\left(\boldsymbol{J}_\sigma\boldsymbol{J}_\sigma^{\mathrm{T}}\right)^{-1}$ is a right pseudo-inverse of $\boldsymbol{J}_\sigma$, $\boldsymbol{\Lambda}$ is a gain matrix and $\widetilde{\boldsymbol{\sigma}} = \boldsymbol{\sigma}_d - \boldsymbol{\sigma}$ is the task error.

Elementary behaviors are defined at four different levels: vehicle, manipulator, end-effector and cooperative UAVMSs. The first three levels are related to each agent, while the fourth one involves two or more UAVMSs.

A library of elementary behaviors has been implemented in CAVIS, it is worth remarking that this is not an exhaustive overview, since new elementary behaviors can be defined by the user based on the mission needs.

- Vehicle Position (VP). It is aimed at controlling the position of the vehicle $\boldsymbol{\sigma}_{VP} = \boldsymbol{p}_V$.
- Vehicle Yaw (VY). It is aimed at controlling the yaw angle of the vehicle ($\boldsymbol{\sigma}_{VY} = \boldsymbol{\psi}_V$).
- Vehicle Obstacle Avoidance (VOA). It allows the vehicle to avoid obstacles in its workspace. Let $\boldsymbol{p}_{ob} \in \mathbb{R}^3$ denote the position of the obstacle, the task function can be defined as $\sigma_{VOA} = \|\boldsymbol{p}_V - \boldsymbol{p}_{ob}\|^2$.
- Mechanical Joint Limit (MJL). Its aim is to maintain a certain distance between the actual joint positions and the corresponding limits (Siciliano et al., 2009).
- Robot Manipulability (RM). An appropriate manipulability index (see e.g., Siciliano et al. (2009)) can be chosen as a task function to avoid singularities.
- Robot Nominal Configuration (RNC). In some cases it is required to keep the robotic arm close to a specific configuration (e.g., arm folded on itself during take-off or landing). Its task function is defined as $\boldsymbol{\sigma}_{RNC} = \boldsymbol{q}$.
- End-Effector Position (EEP). It is aimed at controlling the position of the end-effector, whose expression in terms of $\boldsymbol{\zeta}$ can be obtained by the kinematic model of the UAVMS (Arleo et al., 2013).
- End-Effector Orientation (EEO). It is aimed at controlling the orientation of the end-effector. Again, the end-effector orientation in terms of $\boldsymbol{\zeta}$ is given by the kinematic model of the UAVMS (Arleo et al., 2013).
- End-Effector Configuration (EEC). It is obtained by merging the behaviors EEP and EEO: its goal is to control the end-effector pose.
- Object Configuration (OC). It is aimed at imposing a desired motion trajectory of an object grasped by multiple UAVMSs.
- Object Obstacle Avoidance (OOA). In the presence of an unexpected obstacle during a cooperative task, the UAVMSs holding the object should avoid the obstacle, while keeping the grasp geometry. The task function is similar to σ_{VOA}.

3.2 Tasks

In case of the degrees of freedom (DOFs) required for a given task execution are lower than the DOFs of the system, the system is kinematically redundant and the redundant DOFs can be exploited to fulfill secondary tasks (e.g., by resorting to a task-priority approach, such as the NSB control (Antonelli et al., 2010)).

The overall system velocity is obtained by properly merging the velocity vectors computed for each behavior as if it is acting alone; the velocity contribution of a lower-priority behavior is projected onto the null space of the higher-priority behaviors, so as to remove those velocities components that would conflict with it. The overall system velocity is computed according to the following

$$\dot{\boldsymbol{\zeta}}_r = \dot{\boldsymbol{\zeta}}_1 + \sum_{k=2}^{N_t}\boldsymbol{N}_{1,k-1}\dot{\boldsymbol{\zeta}}_k, \tag{3}$$

where the subscript k denotes the task priority, N_t is the number of behaviors to be fulfilled and

$$\boldsymbol{N}_{1,k} \;=\; \left(\boldsymbol{I} - \boldsymbol{J}_{1,k}^\dagger\boldsymbol{J}_{1,k}\right),$$

is a projector onto the null space of the augmented Jacobian, $\boldsymbol{J}_{1,k}$, defined as

$$\boldsymbol{J}_{1,k} = \left[\boldsymbol{J}_1{}^{\mathrm{T}}\boldsymbol{J}_2{}^{\mathrm{T}}\dots\boldsymbol{J}_k{}^{\mathrm{T}}\right]^{\mathrm{T}}. \tag{4}$$

Many elementary behaviors can be combined together into one task function: therefore they can be arranged in a certain priority order to attain a complex behaviors. The combination of many elementary behaviors is called *task* (compound behavior), where, one task can include one or more elementary behaviors. Furthermore two tasks might differ only for the priority order used to arrange their elementary behaviors.

Priority order among elementary behaviors in a task depends on the objective of the task or on other practical consideration such as safety (e.g., obstacle avoidance behaviors might have a higher priority). Behaviors having low-priority can be achieved only if they compatible with those having higher-priority. Hence, not all elementary behaviors can be combined with each others, for example the RM conflicts RNC, given that both of them assign specific joint positions to the manipulator.

3.3 Actions

An *action* groups together several tasks that are logically related to the action itself, where a given task can belong to different actions. Moreover, they are used to raise the problem description level. Actions implemented in this CSA are detailed bellow:

- moveV: its role is to control the motion of the vehicle; it contains tasks related to vehicle trajectory tracking (in case we want to control only the vehicle without manipulator) and the vehicle obstacle avoidance as well as tasks for manipulator reconfiguration.

- moveA: it is dedicated to the control of the manipulator, it contains tasks such as end-effector position and/or orientation, manipulability optimization and mechanical joint limit.
- graspO: its role is to deliver the motion to the whole system to hold a precise end-effector configuration during the grasping/releasing objects. Basically it includes the moveA action with an additional flag (open/close) to the gripper.
- moveO: it controls the motion of the object, in case of cooperation, the object desired motion is converted in desired motion of the end-effectors of the involved UAVMSs, therefore this action can be converted to a moveA action for each UAVMS.
- moveC: this action is aimed at controlling the motion of embedded sensors such as camera. In case the sensor is mounted on the end-effector, this action becomes equivalent to moveA, while it becomes equivalent to moveV if the sensor is mounted on the vehicle.

In addition, two more actions are implemented to turn on/off the UAVMS and involve specific system motion such as initialization, their names are onV and offV.

From the software point of view, actions are implemented through functions that outputs the desired velocities for the whole system. While its input is composed by the Desired physical State (DS) and the Current physical State (CS) and the task ID.

3.4 Mission

It is a set of actions that are executed one by one and assigned for a known UAVMS. For example, a mission that involves one UAVMS to transport an object from one location to another, could be composed by four ordered actions: moveV (get close to the object to be transported), moveA (reach adequate pre-grasping configuration), graspO (grasping), moveO (object's transportation).

During the mission execution, actions are sorted by their achievement order and eventually are linked to each other by time constraints, e.g., move the vehicle close to the object (action 1) before grasping (action 2). In case of cooperation, actions belonging to different missions, also, can be linked to each other by time constraint. Each action is defined into a mission by its interval of time in which it is executed. This interval is defined by $[t_I, t_F]$, with respect to the instant when the mission is begin. Figure 3 shows two missions assigned for two UAVMSs, where, the action 3 of the second UAVMS has time dependency to the second action of the first UAVMS.

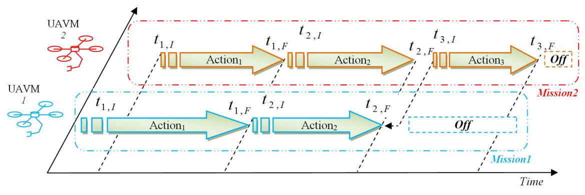


Fig. 3. Missions of two UAVMSs with different actions.

4. TASK MANAGER

The Task Manager (TM) is in charge of the execution process of the generated action plan. During the execution, the original plan can be subjected to some changes depending on the current status of the whole system. The main components of the TM are given below:

- **Supervisor (SP)**: According to the mission needs the SP can switch between running tasks, based on a set of defined metrics.
  (i) the action tolerance, the action is considered achieved when the value of the error $\boldsymbol{e}$, is less than a defined threshold, $\boldsymbol{e}$ is given by the difference between the desired final value of the task function and the actual one;
  (ii) the safety distance from obstacles, d;
  (iii) the distance from mechanical joint limits of the manipulator: $\boldsymbol{q} \in]\underline{\boldsymbol{q}}, \overline{\boldsymbol{q}}[$;
  (iv) the time tolerance Δt to the action final time t_F.
- **Coordination and Synchronization (CS)**: the role of this component is to perform time scheduling of the action plan; it is involved when cooperative UAVMSs require to synchronize their actions. This is achieved by managing the t_I of the actions that have time dependency to the running ones.

Figure 4 shows a flowchart of the Supervisor, where, a logical hierarchy among task tolerance, safety and mechanical joint limits guides the specific task to be activated at the next sampling time based on a corresponding flag that is suitably set. The tolerance to the time delay may drive the overall mission to failure, when, the action time is attaining the allowed delay Δt.

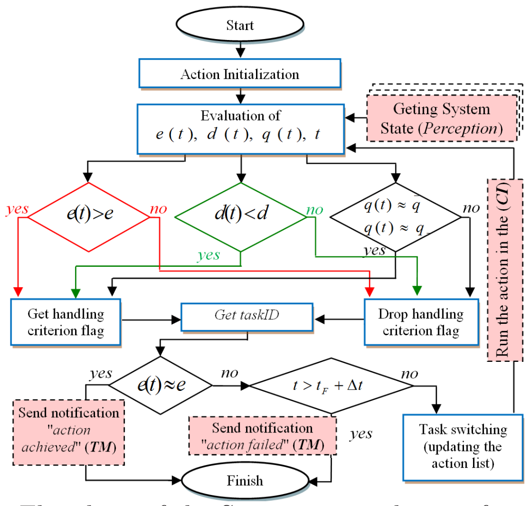

Start
Action Initialization
Evaluation of $e(t)$, $d(t)$, $q(t)$, t
Geting System State (Perception)
yes
$e(t)>e$
no
$d(t)<d$
no
$q(t) \approx q$
$q(t) \approx q$
no
yes
yes
Run the action in the (CI)
Get handling criterion flag
Get taskID
Drop handling criterion flag
yes
$e(t) \approx e$
no
$t > t_F + \Delta t$
no
Send notification "action achieved" (TM)
Send notification "action failed" (TM)
yes
Task switching (updating the action list)
Finish

Fig. 4. Flowchart of the Supervisor in charge of switching among tasks. The ≈ symbol is used to simplify the notation, it embeds the presence of proper thresholds.

5. WORKING PROCESS: A GENERAL OVERVIEW

Communication among different components is guaranteed by a set of messages that transfer requests and/or knowledge from one component to another. This knowledge also includes notifications about tasks achievement. For instance, the TM provides an actions list to the Control Interfaces (CIs) located on each UAVMS, where each action contains a set of tasks. Each action is described by: name, task identity (referes to the selected task by the SP), initial time, final time, DS and CS. The flexibility of the action planning (i.e, actions are composed by several tasks) is used by the SP to select the proper task that each UAVMS must perform, according to the actual status of the robotic system, the mission and the environment. Continuity of the control law must be preserved during the switching phases. Actions running on different UAVMSs are synchronized, based on the updated information coming from the Perception to make the tasks plan feasible.

Information flow out from the TM are composed by a set of ordered actions allocated to certain UAVMSs. Each action includes many information related to its start and ending time, tasks that it construct, etc.. Beside the action list, the CI receives the current system's state, which is necessary for low-level control and the MS, in the case of physical cooperation. To this end, the CI generates commands to actuators of the corresponding vehicle as well as the attached manipulator. Regarding CI components, they not only encapsulate the control functionalities required by the active action, but also are intended to generate the corresponding UAVMSs state.

6. EXAMPLE OF APPLICATION

The proposed architecture has been tested in simulation by involving two quadrotors equipped with a 5 DOFs robotic arm, with 5 revolute joints (figure 2). The simulation model of the UAVMS has been developed under Matlab/SimMechanics©. The objective is to perform a complex cooperative scenario where the UAVMSs approach an object (bar), grasp and move it along a desired trajectory. The assigned missions for each UAVMS can be described via a set of actions, including different tasks to be selected by the SP (figure 5):

- onV: bring-up the UAVMS.
- moveV: UAVMS take off; it includes the task *VP*.
- moveV: moves the vehicle toward the object and, at the same time, set the arm to a particular configuration; it includes the task *VP+RNC*,
- moveA: moves the arm to reach the best configuration for grasping and the end-effector in the grasp position; it includes the task *EEC+RM*,
- graspO: performs the grasp;
- moveO: moves the object along an assigned desired trajectory; it involves two behaviors: *OC* and *OOA+OC* (where *OOA* has the higher priority),
- graspO: releases the object,
- moveV: moves the UAVMS toward the base station; it includes the task *VP+RNC*,
- moveV: UAVMS landing; it includes *VP*.
- offV: shutdown the UAVMS.

The desired trajectory is considered as a straight line. During the transportation phase the system motion is challenged by an unexpected obstacle: when the distance between the obstacle and the object is below a certain safety value, the SP switches from *OC* to *OOA+OC*; then, switch back to *OC* when the obstacle is overcame.

Figure 6 shows some snapshots of the mission. In detail, figure 6(a) represents the initial configuration with the two

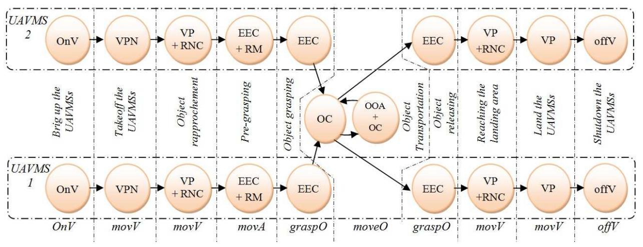


Fig. 5. FSM of the supervisor.

UAVMSs on the ground, figure 6(b) shows the UAVMSs in hovering, after the takeoff phase, figure 6(c) shows the approach to the object (the zoom on the bottom left corner shows a detail of the end-effectors and the object), figure 6(d) shows the pre-grasp arm reconfiguration, figure 6(e) shows the object motion along the planned trajectory and figure 6(f) shows the object obstacle avoidance.

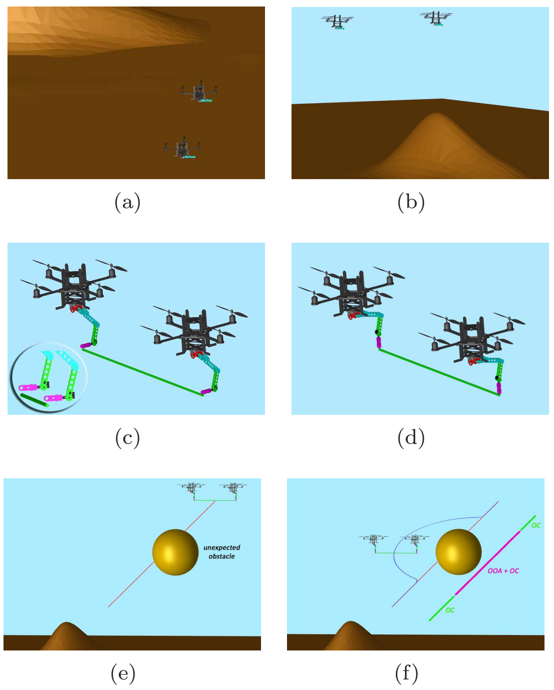
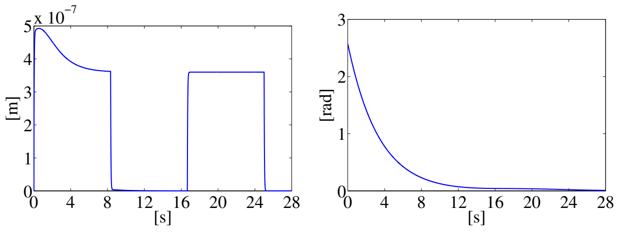


Fig. 6. Snapshots of the mission of bar transporting.

Figures 7–9 show the task errors for the scenario. For the sake of brevity we report errors only of one UAVMS, since similar results have been obtained for the second UAVMS. Figure 7 shows the task errors during the action moveV: in detail, figure 7(a) reports the error norm of the vehicle position (primary behavior), while figure 7(b) shows the norm of the joint position error with respect to the desired configuration (secondary behavior). Figure 8 shows the performance obtained during the action moveA: in figure 8(a) the norm of position and orientation errors of the end-effector are reported (i.e., the error of the behavior *EEC*), while figure 8(b) reports the manipulability index of the robotic arm, $w(\boldsymbol{q})$, (Siciliano et al., 2009)

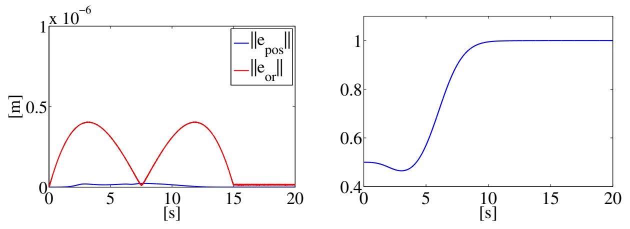
$$w(\boldsymbol{q}) = \sqrt{\det(\boldsymbol{J}_a(\boldsymbol{q})\boldsymbol{J}_a^{\mathrm{T}}(\boldsymbol{q}))},$$

where $\boldsymbol{J}_a$ is the Jacobian matrix of the arm, normalized to its maximum allowed value, w_{max}. Finally, figure 9 shows the performance obtained during the object motion (moveO action). In detail the actual path followed by the object is reported in figure 9(a): it can be noticed that, during the obstacle avoidance phase, the object does not track the planned path, which is engaged again after overcoming the obstacle; figure 9(b) shows the distance between the object and the obstacle, it is noted that the safety distance of 1.5 m is always guaranteed.



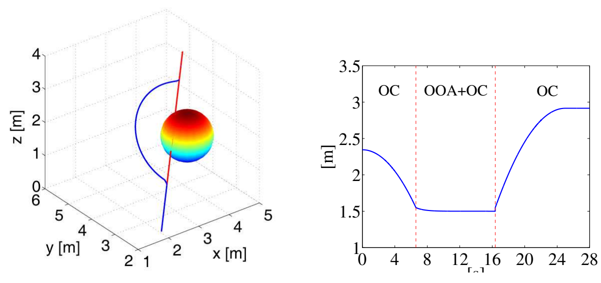
(a) vehicle position error (b) joints position error

Fig. 7. Task errors during the action moveV.



(a) end-effector position and orientation error norm (b) normalized manipulability

Fig. 8. Performance during the action moveA.



(a) planned (red) and actual (blue) object trajectory (b) distance between the object and the obstacle

Fig. 9. Object transportation performance.

7. CONCLUSION

In this paper, a new control software architecture is developed, aimed at driving missions to be performed by cooperative Unmanned Aerial Vehicles Manipulator Systems (UAVMSs). The main objective of the software is to support a large range of possible cooperative scenarios and missions. The architecture is designed around a set of software components that handle the current states of the involved UAVMSs and provide basic functionalities. The architecture is based on the decomposition of the overall control problem in simpler *atomic* control problems. Next steps will integrate the developed software within ROS environment in order to port it to users hand.

REFERENCES

Antonelli, G., Arrichiello, F., and Chiaverini, S. (2010). The NSB control: a behavior-based approach for multi-robot systems. *Paladyn Journal of Behavioral Robotics*, 1, 48–56.

Arleo, G., Caccavale, F., Muscio, G., and Pierri, F. (2013). Control of quadrotor aerial vehicles equipped with a robotic arm. In *21st Mediterranean Conference on Control Automation (MED), 2013*, 1174–1180.

Baillie, J.C. (2004). Towards a universal robotic body interface. In *EEE/RAS International Conference on Humanoid Robots*, volume 1, 33–51.

Brian P. Gerkey, Richard T. Vaughan, A.H. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *International Conference on Robotics and Automation.*

Bruyninckx, H. (2001). In *Open robot control software: the OROCOS project*, volume 3, 2523–2528.

C. Cote, Y. Brosseau, D.L., Raievsky, C., and Michaud, F. (2002). Robotic software integration using MARIE. *International Journal of Advanced Robotics*, 3(1), 55–60.

Dias, M., Zlot, R., Kalra, N., and Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7), 1257–1270.

Gancet, J., Hattenberger, G., Alami, R., and Lacroix, S. (2005). Task planning and control for a multi-UAV system: architecture and algorithms. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, 1017–1022.

Keith, F., Mansard, N., Miossec, S., and Kheddar, A. (2009). Optimization of tasks warping and scheduling for smooth sequencing of robotic actions. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 1609–1614.

Khelifa, B. Ryad, C. and H., T. (2011). Virat: An advanced multi-robots platform. In *Industrial Conference on Electronics and Applications (ICIEA)*, 564–569.

Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., and Estlin, T. (2003). Claraty and challenges of developing interoperable robotic software. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, 2428–2435 vol.3.

Nestinger, S. and Cheng, H. (2011). Mobile-r: A reconfigurable cooperative control platform for rapid deployment of multi-robot systems. In *Robotics and Automation, 2011 IEEE International Conference on*, 52–57.

Oreback, A. and Christensen, H.I. (2003). Evaluation of architectures for mobile robotics. *Autonomous Robots*, 14(1), 33–49.

Ortiz, A., Bonnin-Pascual, F., Garcia-Fidalgo, E., and Beltran, J. (2011). A control software architecture for autonomous unmanned vehicles inspired in generic components. In *19th Mediterranean Conference on Control Automation*, 1217–1222. doi:10.1109/MED.2011.5983136.

Quigley Morgan, C.K. and Gerkey (2009). ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software.*

Shi, G. and Yang, S. (2008). Intelligent control of UAV with neuron-fuzzy approach under hierarchical architecture. In *7th World Congress on Intelligent Control and Automation*, 5238–5243.

Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2009). *Robotics: modelling, planning and control.* Springer Verlag.

Utz, H., Sablatnog, S., Enderle, S., and Kraetzschmar, G. (2002). Miro middleware for mobile robot applications. *IEEE Tran. on Robotics and Autom.*, 18(4), 493–497.