The C3UV Testbed for Collaborative Control and Information Acquisition Using UAVs

Eloi Pereira[1] and Karl Hedrick[2] and Raja Sengupta[1]

Abstract— **In this paper we introduce the Center for Collaborative Control of Unmanned Vehicles (C3UV) testbed for collaborative information acquisition.**

The C3UV testbed has been used for demonstrating a wide range of information-oriented applications executed by collaborative teams of Unmanned Aerial Vehicles (UAVs).

This paper presents the C3UV testbed from an architectural stand-point. The testbed includes a *estimation and control* architecture and a *software* architecture. The estimation and control architecture is a set of components that can be composed to perform specific missions. The software architecture supports the execution of estimation and control components and implements the Collaborative Sensing Language - a language for high-level specification of mission-level controllers for mobile sensor networks with ad-hoc resource pool and dynamic network topology.

We show the use of different layers of the architecture using examples from our field experiments and demonstrations. Our heterogeneous teams of UAVs perform several types of missions such as environmental monitoring, pedestrian search and tracking, and river mapping.

I. INTRODUCTION

The C3UV testbed is designed to organize and control networks of UAVs for information acquisition. Its first focus was on collaborative control of Unmanned Aerial Vehicles (UAVs) [1]. In 2004 the testbed was used for demonstrating 4 UAVs operated by a single operator [2]. The prevailing practice at the time was to have 3 operators control 1 UAV. Our approach is based on having the operator focusing on declaring high-level tasks to be done by graphical interfaces. These interfaces reflect the state of the system back to the operator at a level of abstraction matched to the cognitive capacity of a single individual, and entrusting both *tasks-to-platform* allocation and tasks execution to *autonomy*.

In a second phase the testbed shifted focus toward collaborative information acquisition. It has been demonstrating vision-based search and target tracking of structures such as rivers, roads and aqueducts [3], [4], search and tracking of moving targets such as convoys and pedestrians [3], [5], [6], obstacle-avoidance [7], and distributed collaborative sensing [2]. Our approach to collaborative sensing is based on the premise that information acquisition and control are tightly coupled. The common sense is to think of *estimation* as a mean to achieve *control*. We think of *control* as a mean to achieve more effective and efficient estimation of features of the world. An example of this reasoning can be found in [8] where a path-following algorithm is designed to enhance image quality by effectively controlling the camera footprint.

As the capabilities of the system increased, it became mandatory to have a well-structured software architecture (see [9] for details on the low-level software infrastructure) with clear computation interfaces and abstractions that leverage rapid development of software for control, interactive behaviours, and sensing. This led us to the design of the Collaborative Sensing Language (CSL) [10], a Domain-Specific Language (DSL) for implementing mission-level controllers that entails the necessary reflection of physical components while still being loosely-coupled with the underlying system in order to cope with network latency and scalability. CSL is provided both as a graphical language (using a Graphical User Interface (GUI)) and as an Application Programming Interface (API) implemented using webservices.

The remainder of the paper goes as follows. Section II presents the control and estimation architecture. This section also provides examples of the C3UV tests and demonstrations for each architecture layer. Section III provides a brief explanation of the CSL syntax and semantics and an overview of the software architecture including a formal verification component. We finish with conclusions and an overview of the problems that we are currently addressing.

II. CONTROL AND ESTIMATION ARCHITECTURE

The control and estimation architecture follows a platform-based design [11] where a set of atomic components form a platform that can be instantiated into a specific application. The architecture is divided in four different control and estimation layers, and the physical platform. Control components (depicted as blue boxes) are layered together with estimation components (depicted as red boxes). The reasoning used is to layer together components that tend to be tightly coupled (e.g. a target tracking controller is at the same layer as a target recognition algorithm).

A. Physical Platform

The physical platform is a set of vehicles, sensors, actuators, and communication infrastructure. C3UV has a fleet of UAVs that includes Zephyr and Zagi flying wings, Sig Rascal 110s, and Bat IVs. The Zephyr (Figure 2(b)) and Zagi flying wings are small hand-launched UAVs with electric engines and equipped with Ardupilot autopilots.

Research supported in part by ONR-BAA N00014-O7-C-0225: "Berkeley Robust Aerobot Information Network", National Science Foundation (CNS1136141), and Fundação para a Ciência e Tecnologia (SFRH/BD/43596/2008).

[1]Systems Engineering, Department of Civil and Environmental Engineering, UC Berkeley. Emails: eloi@berkeley.edu, sengupta@ce.berkeley.edu

[2]Department of Mechanical Engineering, UC Berkeley. Email: khedrick@me.berkeley.edu

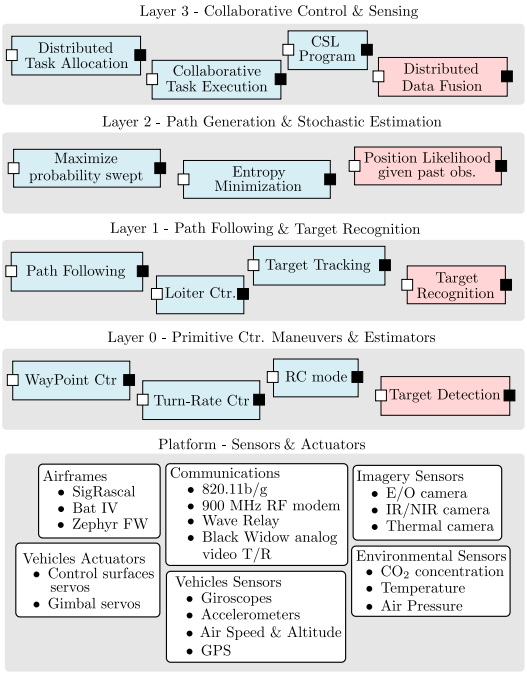


Fig. 1. Estimation and control architecture.

(a) Bat IV and SigRascal UAVs (b) Zephyr UAV

Fig. 2. Platforms.

The Rascals (bottom left of Figure 2(a)) are a 9 ft wingspan, 1.5h endurance platform used for testing and demonstrations. We have also executed heterogeneous demonstrations with a UAV network including Rascals and the Bat IV. The Bat IV (bottom right of Figure 2(a)) is a larger (12 ft wingspan) 8 hour endurance platform with on-board generators to provide power for a payload (up to 25 pounds of payload).

Both the SigRascals and the BAT UAVs are equipped with Piccolo II autopilots from Cloud Cap Technologies for low-level control (waypoint and turn rate control) and 1.8 GHz Pentium PC-104s for high-level control (distributed task allocation, and collaborative manoeuvres) and sensory data processing (computer vision processing). All air-to-air and air-to-ground communication required for autonomy is performed over IEEE 802.11 radios. The Piccolo 900 MHz RF modem is used for safety purposes, such as Remote-Control (RC) operation of the UAVs until they achieve safe altitude separation, or for emergency UAV recovery on failure of an autonomy experiment. UAVs are equipped with a wide variety of sensors specific for each mission. Sensors range from a variety of imagery sensors (low-cost Electro-Optic (EO) cameras, Infra-Red (IR) and near IR cameras, and thermal cameras), sensors for environmental monitoring such as CO_2 concentration sensors, temperature sensors, and air-pressure sensors.

B. Primitive Controllers and Estimators

Layer 0, named "primitive controllers & estimators", deals with low-level control dynamics of the vehicles (e.g. follow a sequence of turn-rate commands or autonomously go from one GPS waypoint to another), and target detection. Here target detection refers to the classification of the pixels in each image in a video stream. For example, when autonomously searching for a river, an image pixel is classified as "water" (target) or "non-water" [12] (Figure 4(a) shows an on-board image of a river where yellow pixels are classified as water). For autonomous pedestrain detection and tracking pixels would be classified as "human" or "non-human" [6]. This layer also includes low-level sensor data acquisition and logging components. Layer 0 includes waypoint controllers, turn-rate controllers, and remote-control (RC). These features are provided by Commercial Off-The-Shelf (COTS) autopilots such as Piccolo II from Cloud Cap Technologies (CCT) or the open-source autopilot Ardupilot.

An example of a layer 0 application is an environmental sensing mission performed with a Zephyr UAV in RC mode equipped with a K30 CO_2 concentration sensor from SenseAir and an Ardupilot autopilot. The data is acquired and logged on-board using the arduino board of the autopilot. Figure 3 presents three dimensional sampling data. The

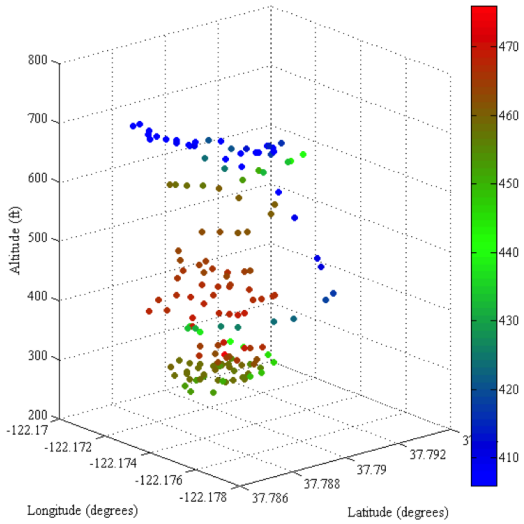


Fig. 3. CO_2 concentration flight data, February 2012, Oakland, CA.

motivation for this mission is to investigate the use of UAVs as mobile sensors for estimating CO_2 Eddy covariance flux using concentration sensors. Small UAVs are being equipped with low-cost CO_2 concentration sensors and gust probes and use mobility in order to get three dimensional samples that are going to be used for estimating the CO_2 fluxes [13].

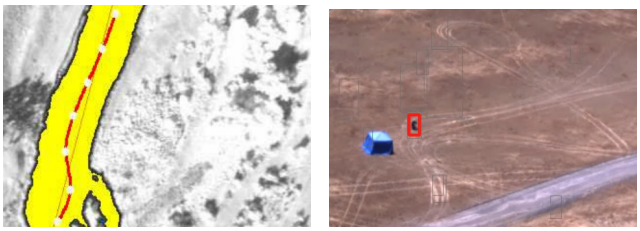
C. Path Following and Target Recognition

Layer 1, named "path following & target recognition", includes path following controllers (e.g. following linear

structures such as rivers, roads, and line coast), static and mobile target trackers (e.g. tracking pedestrians), and target recognition (e.g. river recognition, pedestrian recognition, etc.). More precisely, target recognition is image segmentation. At this layer, each image in a video stream would be segmented to declare the target in the image as a whole, or not. The target would also be geo-localized. For example, in autonomous river search the image would be segmented to mark the river component, as in figure 4(a) (yellow area is recognized as "river"). This is achieved by clustering the classified pixels.

In 2008 C3UV designed and tested a path following controller. The path following controller is composed by a spatial sliding mode controller designed to follow a desired aircraft path which places the camera-footprint on the desired locations, and a kinodynamic controller that directly track a camera-footprint path[8]. UAV's yaw-by-rolling and the kinodynamic controller takes this coupling into account to position the sensor footprint on the target at the expense of positioning the UAV off the target. Controlling the camera footprint makes this strategy suitable to estimate and track linear structures such as rivers or roads.

Layer 1 includes vision-based target recognition components which are used together with path following or target tracking controllers. In 2007 we investigated the use of a UAV to autonomously searching a given area for a structure such as river or coastal line, identify the structure if present and map the coordinates of the structure based on imagery collected from an on-board sensor. Figure 4(a) presents flight data from the on-board estimation of a river center line at Camp Roberts, CA. The estimated center line (depicted in red in the figure 4(a)) was used to close the navigation control loop [12].



(a) Onboard processed river image. Camp Roberts, CA.

(b) Onboard processed pedestrian image. Camp Roberts, CA.

Fig. 4. Target recognition examples.

During the Tactical Network Topology (TNT) exercises in 2009 and 2010, C3UV tested a vision in-the-loop pedestrian tracking system [6]. The experiment consisted of two aircraft, a Sig Rascal with a side-mounted camera attached to a two-axis gimbal and a Bat 4 with a front-mounted camera that was also attached to a two-axis gimbal. The main goal of the demonstration was to track a pedestrian as it left a tent (see Figure 4(b)). A UAV was tasked to monitor the tent. As soon as a pedestrian was detected leaving the tent, the information was transmitted to the second UAV that started the tracking mission while the other UAV continued surveillance of the tent. For further details on the vision processing algorithm see [5].

D. Path Generation and Stochastic Estimation

As stated before the C3UV architecture is designed for Information Acquisition. This intelligence is resident in Layer 2. The estimation component at this layer produces a likelihood on the position of a static or moving target conditioned on all past observations (e.g. if the target is not detected under the current footprint, then the likelihood map is decreased upon the area of the footprint). The control component generates a path computed to minimize the uncertainty in this likelihood function (more precisely, minimize entropy [2], [14]). This is illustrated for a static target in figure 5(a) (contour red lines denote high probability while the blue ones denote low probability). The initial likelihood function was a spacial Gaussian function that got updated during execution regarding the position and footprint of the UAV. In that particular experiment [15] the UAV was a Scan Eagle from Insitu searching for an orange kayak representing a life raft.

(a) Probability Density Function of target estimation (orange kayak).

(b) Two UAVs collaboratively detecting a target. Camp Roberts, CA.

Fig. 5. Stochastic estimation examples.

This technique is also used for pedestrian tracking (figure 4(b)). In this case the target is moving and the likelihood function is computed on-board the aircraft as well as the trajectory generation [6].

In a prior experiment we showed how two UAVs could collaborate in a target search [2], [14]. In this configuration, the likelihood function of the position of the target is conditioned jointly on the observations of both aircraft. The collaborative aspects of this control strategy are known as Distributed Data Fusion (DDF) and are included in layer 3.

E. Collaborative Control and Sensing

Layer 3 deals with collaborative control (e.g. several UAVs patrolling collaboratively a line) and collaborative sensing (e.g. UAVs sharing information to improve target information).

In 2007 C3UV tested at Camp Roberts a real-time algorithm for combined probabilistic search and track of a stationary target using multiple UAVs. The searching algorithm was implemented using DDF. Vision-based sensing was employed, using fixed downward-looking cameras. In this test, two UAVs simultaneously searched for a target. While doing so, UAVs kept a distributed likelihood map of the position of the target given past observations. Each UAV updated its likelihood map upon new observations (see the

single UAV case in subsection II-D), broadcasted its own version of the likelihood map, and fused its information with the maps from its neighbours. This procedure forms a distributed map, disseminating the information amongst all UAVs and thus providing means for more efficient and robust searching and tracking strategies. Figure 5(b) shows the flight trajectories of two UAVs searching for a target (a red tarp in the runway). The blue UAV found the target. The red UAV visits the target upon the information received by the blue UAV.

The information structure for distributed task allocation resides in the Mission State Estimate (MSE) [16]. MSE is wrapped on a message and is broadcast periodically by each UAV. The MSE broadcasted by a UAV is a function of the knowledge acquired by itself and the MSEs it receives from others. A UAV has write authority over its own state, and the state of the task it is executing. As for the others, it re-writes the most recent information it has received (according to GPS timestamps). The commander of the system (a.k.a. publisher) writes new tasks into its own MSE and removes tasks that are either complete or aborted. A new UAV joins the system simply by starting to broadcast MSEs. The MSE information floods. Thus information propagates in partial connected networks and in intermittently disconnected networks.

UAVs pick tasks by applying local decision functions to its MSE. When a UAV is idle it searches over the demanding tasks in its current MSE. A simple decision function is to pick the closest demanding task (in case the task a spacial characterization such as visit a point or patrol an area). We have implemented more sophisticated algorithms based on solving travelling salesman problems and multi-vehicle travelling salesman problems [17].

This task allocation strategy was used in [2] for collaboratively patrol a line segment by three UAVs. It has been used as the primary strategy for task allocation of the Collaborative Sensing Language (CSL) missions [10].

CSL programs are also layer 3 controllers. Section III provides more information about CSL.

III. Collaborative Sensing Language

In this section we present the Collaborative Sensing Language (CSL), a Domain-Specific Language (DSL) for feedback control of distributed ad-hoc mobile sensor networks (MSN) [10].

CSL is used for specifying mission-level controllers to accomplish objectives with a dynamic resource pool. CSL comprises two sub-languages: a declarative language named Mission Control Language (MCL) and an imperative language named Runtime Patching Language (RPL).

The main research efforts behind CSL-MCL is motivated by the need of specifying controllers for distributed mobile sensor networks where resources and network topology can dynamically change during execution. Network topology may change due to sensors being added or removed or because their structure of interactions change over time (e.g. local connectivity in a wireless network). MCL controllers concern with high-level mission behaviours such as task precedence or concurrent execution. MCL do not specify the allocation of tasks to platforms, coordination policies between platforms, or lower-level platform controllers. All of this is embedded in the CSL execution engine. This makes CSL a simple but yet powerful abstraction to specify missions decoupled from low-level details and the dimension of the resource pool.

The motivation for developing CLS-RPL is to provide human operators with the ability to change CSL-MCL controllers at runtime. This feature is of significant relevance for persistent operations where the mission strategy is likely to change over time due to unforeseen situations (e.g. change of operational environment, change of economical conjecture, etc.). RPL allows a user to delete, update or add missions.

Next we explain MCL and RPL into more detail.

A. Mission Control Language

CSL-MCL is a graphical language stored into an XML representation. MCL programs are the composition of missions. Missions are sets of tasks to be executed by the MSN. MCL allows missions to be composed in sequence or in parallel. The sequential composition of missions is specified using a transition which is conditioned by predicates on the state of the tasks and time premisses. State predicates can be quantified using scopes. The scope `mission` is an universal quantifier that makes the statement true if all tasks of the corresponding mission satisfy the predicate. The scope `task` is an existential quantifier that makes the statement true if at least on task of the corresponding mission satisfies the predicate. A task can be in four different states, `TODO`, `ASSIGN`, `DONE`, and `CANCELLED`. The state `TODO` denotes that the task is published but nor yet assigned, the state `ASSIGN` means that the task was assigned by a resource, the state `DONE` means that the task terminated, and `CANCELLED` means that the task got cancelled.

Figure 6 shows an MCL program with three missions and four tasks. Mission `mission1` consists of two tasks,

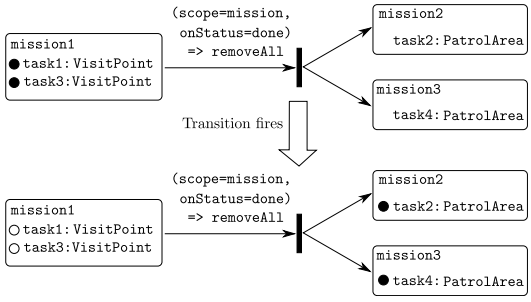


Fig. 6. Example of a CSL-MCL program.

`task1` and `task2`, both of type `VisitPoint`. The token is used as a flag to signal if a task has or has not already been used to fire a transition. This is needed since once certain conditions are satisfied by a given task, they will always be true. For example, "if `task1` is done" is always true once `task1` is completed. An empty token means that the tasks has been already used to fire the transition. The

condition on the transition has scope `mission` and predicate `onStatus=done` meaning that all tasks must have terminated for the transition to be fired. When the transition is fired, the action `removeAll` executes, removing all tokens from `mission1` tasks and creating a token to each task of `mission2` and `mission3`.

B. Runtime Patching Language

Run-time patching language (RPL) is intended to patch the MSN controller dynamically. Thus the MSN controller can be changed during execution. This is done by patching an already operating MCL program at run time. An external controller or operator, can dispatch an RPL program to the CSL Execution Engine to reduce, grow or otherwise alter the MSN controller at run-time. RPL has three commands: add, delete, and update. An RPL program is a sequence of such commands. The imperatives can add, delete, or modify the syntactic elements in MCL, i.e. mission definitions, transitions, input arcs, and output arcs.

C. CSL Architecture

Figure 7 presents the overall CSL architecture.

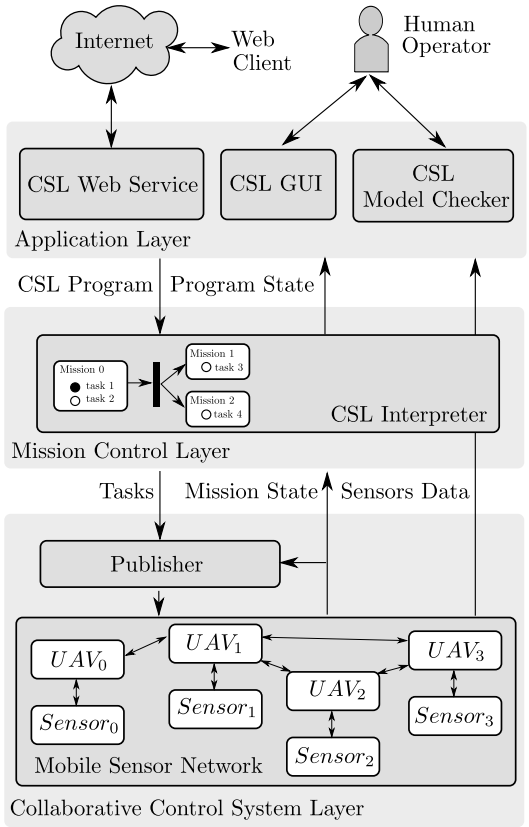


Fig. 7. CSL Architecture.

1) Application Layer: The application layer is composed by a Graphical User Interface (GUI), a CSL WebService, and a CSL model checker.

The GUI allows a human operator to graphically specify CSL-MCL programs, follow the execution of the current CSL-MCL program as well as the state of the Mobile Sensor Network, and patch new CSL-MCL programs during execution using the CSL-RPL.

The CSL WebService (CSL-WS) defines a web-based Application Programming Interface (API) for the CCS using CSL as the interface language. CSL-WS is implemented over the Simple Object Access Protocol (SOAP). We foreseen in a near future shifting this implementation towards a Representational State Transfer (REST) compliant architecture, removing the need for SOAP and Web Services Description Language (WSDL).

The CSL Model Checker (CSL-MC) is a tool for formally verify if programs written in CSL satisfy a specification given in Computational Tree Logic (CTL). CTL allows us to write specifications such as "it is always the case that there exists a computation path where all tasks got executed." In CTL this is expressed as AE$(DONE(task_0) \wedge \ldots \wedge DONE(task_n))$ where $n-1$ is the number of tasks.

Our approach to formally analyse CSL programs is to provide a translation to Generalized Stochastic Petri Nets (GSPN) [18]. By translating CSL to a well-known model of computation we facilitate the use of available tools for such models. Petri Nets [19] was the natural choice since CSL has similar structure and semantics. The translated CSL program is composed with a model of the execution environment, i.e. the platform that executes the CSL program (also expressed using Petri Nets). We currently model the platform at the level of the allocation of tasks to resources. We implemented two types of platform models that uses two different allocation policies: arbitrary allocation (AA) and First-Come First-Served (FCFS) allocation.

The translation is implemented by a compiler from CSL to SMART - a software package that integrates high-level logical (functional) and timing/stochastic (nonfunctional) modelling formalisms including GSPN [20]. The compiler is implemented using ANTLR[1] [21]. The compiler returns a file that can be executed into the SMART model checker. For more details of the `csl2smart` compiler see [22].

2) Mission Control Layer: The mission control layer entails the CSL interpreter. It receives new CSL programs and executes them. The output is a stream of tasks that is passed to the publisher. The mission control layer also provides the state of the program and the state of the mission to the application layer.

3) Collaborative Control System Layer: The collaborative control system (CCS) is composed by a task publisher and a Mobile Sensor Network (MSN).

The publisher is provided with all tasks information from the CCS. It monitors this information along with RPL changes to appropriately add, update, and delete tasks to the set of tasks being executed by the CCS. The Publisher has a well-defined operational loop. The loop begins with checking for RPL commands changing the mission controller. Secondly, it checks to see if any transition can be fired. If one can be fired, it is, and old tasks/tokens are removed and new tasks/tokens are added. Finally, the publisher checks

[1]Acronym for "Another Tool for Language Recognition".

for updates of the CSL state (task state and resource state) written by the CCS.

The Collaborative Control System accepts a list of tasks (Task State) and distributively allocates individual tasks to UAVs. The assigning costs are continuously recomputed and redistributed among the network using a distributed composition protocol. This helps adapt to changes in network topology. When UAVs enter, they begin to bid on tasks. When UAVs exit, they stop bidding on tasks, and their tasks are reallocated. If a UAV suddenly stops responding, a timeout based on cost of the task being previously performed is enforced. Once an individual UAV selects a task, it utilizes its on-board low-level controllers to physically execute the task. For further details on CCS see [23].

The MSN is a network of autonomous mobile sensors that can: execute a set of task primitives (e.g. visit a point and take a picture, patrol an area and stream video, etc.). MSN assumes an ad-hoc resource pool, i.e. resources can come and go. Moreover, the communication topology might change over time.

CSL has been thoroughly tested in Hardware-In-Loop (HIL) simulations and flight tests. It was demonstrated in the Fall 2007, Spring 2008 TNT exercise at Camp Roberts, CA, and Fall 2008 Large Tactical Sensor Networks (LTSN) Demonstration in Quantico, VA.

IV. CONCLUSION

We introduce the C3UV testbed for collaborative control and information acquisition. The testbed includes a control and estimation architecture that can be instantiated into different applications such as path following, target search and tracking, collaborative task execution, and distributed task allocation. Control and estimation components are layered together according to their degree of interdependency.

The C3UV testbed also provides an abstraction for the implementation of mission-level controllers. This abstraction is formalized and implemented as a DSL, called the Collaborative Sensing Language. We provide a short overview of the syntax and semantics of CSL as well as the architecture of the execution engine and means for formal verification.

We are currently working on maritime applications, more concretely we on the development of vision-based search and tracking algorithms that can be used for tracking maritime targets such as boats, or marine mammals. We are also working on new DSLs for controlling mobile robotic systems with dynamic structure based on the BigActor model [24].

ACKNOWLEDGMENT

The authors would like to thank the C3UV team.

REFERENCES

[1] E. Frew, X. Xiao, S. Spry, T. McGee, Z. Kim, J. Tisdale, R. Sengupta, and J. Hedrick, "Flight demonstrations of self-directed collaborative navigation of small unmanned aircraft," in *AIAA 3rd "Unmanned Unlimited" Conference*, 2004, pp. 1–14.

[2] A. Ryan, J. Tisdale, M. Godwin, D. Coatta, D. Nguyen, S. Spry, R. Sengupta, and J. Hedrick, "Decentralized control of unmanned aerial vehicle collaborative sensing missions," in *American Control Conference, 2007. ACC'07*. Ieee, 2007, pp. 4672–4677.

[3] E. Frew, T. McGee, Z. Kim, X. Xiao, S. Jackson, M. Morimoto, S. Rathinam, J. Padial, and R. Sengupta, "Vision-based road-following using a small autonomous aircraft," in *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, vol. 5. IEEE, 2004, pp. 3006–3015.

[4] S. Rathinam, Z. Kim, A. Soghikian, and R. Sengupta, "Vision based following of locally linear structures using an unmanned aerial vehicle," in *44th IEEE Conference on Decision and Control and European Control Conference. CDC-ECC'05*. IEEE, 2005.

[5] Z. W. Kim and R. Sengupta, "Target detection and position likelihood using an aerial image sensor," in *ICRA*, 2008, pp. 59–64.

[6] J. Garvey, B. Kehoe, B. Basso, M. Godwin, J. Wood, J. Love, S. Liu, Z. Kim, S. Jackson, Y. Fallah *et al.*, "An autonomous unmanned aerial vehicle system for sensing and tracking," in *AIAA Infotech@Aerospace Conference*, 2011.

[7] T. McGee, R. Sengupta, and K. Hedrick, "Obstacle detection for small autonomous aircraft using sky segmentation," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 4679–4684.

[8] S. Jackson, J. Tisdale, M. Kamgarpour, B. Basso, and J. Hedrick, "Tracking controllers for small uavs with wind disturbances: Theory and flight results," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 2008, pp. 564–569.

[9] B. Basso, B. Kehoe, and J. Hedrick, "A multi-level modularized system architecture for mobile robotics," in *Proceedings Dynamic Systems and Control Conference*, 2010.

[10] J. Love, J. Jariyasunant, E. Pereira, M. Zennaro, K. Hedrick, C. Kirsch, and R. Sengupta, "Csl: A language to specify and re-specify mobile sensor network behaviors," in *Proc. of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, 2009.

[11] K. Keutzer, S. Malik, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 12, pp. 1523–1543, 2000.

[12] S. Rathinam, P. Almeida, Z. Kim, S. Jackson, A. Tinka, W. Grossman, and R. Sengupta, "Autonomous searching and tracking of a river using an uav," in *American Control Conference*. IEEE, 2007.

[13] H. Chen, R. Hansen, J. Huang, E. Pereira, R. Swick, D. Vizzini, R. Sengupta, C. Kirsch, F. Landolt, M. Lippautz *et al.*, "Cloud computing on wings: Applications to air quality," *Proc. American Astronautical Society Guidance and Control Conference*, 2012.

[14] J. Tisdale, Z. Kim, and J. Hedrick, "Autonomous uav path planning and estimation," *Robotics & Automation Magazine, IEEE*, 2009.

[15] R. Sengupta, J. Connors, B. Kehoe, Z. Kim, T. Kuhn, and J. Wood, "Autonomous search and rescue with scaneagle," Technical report, prepared for Evergreen Unmanned Systems and Shell International Exploration and Production Inc, Tech. Rep., 2010.

[16] M. Godwin, S. Spry, and J. Hedrick, "Distributed collaboration with limited communication using mission state estimates," in *American Control Conference, 2006*. IEEE, 2006, pp. 7–pp.

[17] S. Rathinam and R. Sengupta, "Algorithms for routing problems involving uavs," *Innovations in Intelligent Machines-1*, 2007.

[18] E. Pereira, A. Pinto, R. Sengupta, and A. Sangiovanni-Vincentelli, "Formalization and analysis of csl using generalized stochastic petri nets," UC Berkeley, Tech. Rep., 2010.

[19] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[20] G. Ciardo, R. L. Jones, III, A. S. Miner, and R. I. Siminiceanu, "Logic and stochastic modeling with smart," *Perform. Eval.*, vol. 63, no. 6, Jun. 2006.

[21] T. Parr and R. Quong, "Antlr: A predicated-ll (k) parser generator," *Software: Practice and Experience*, vol. 25, no. 7, pp. 789–810, 1995.

[22] E. Pereira, A. Pinto, R. Sengupta, and R. Bodik, "Framework for formal analysis of csl programs using gspn model," UC Berkeley, Tech. Rep., 2011.

[23] J. Tisdale, A. Ryan, M. Zennaro, X. Xiao, D. Caveney, S. Rathinam, J. Hedrick, and R. Sengupta, "The software architecture of the berkeley uav platform," in *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*. IEEE, 2006, pp. 1420–1425.

[24] E. Pereira, C. Kirsch, R. Sengupta, and J. Borges de Sousa, "Bigactors - a model for structure-aware computation," in *4th International Conference on Cyber-Physical Systems*. ACM/IEEE, April 2013.