A Proposed Architecture for a Heterogeneous Unmanned Aerial Vehicles System

Ahmed Barnawi, Abdullah Al-Barakati, Asif Khan, Fuad Bajaber, and Omar Alhubaiti
King Abdulaziz University, Faculty of Computing and Information Technology, Jeddah- Saudi Arabia
Email: {ambarnawi, aaalbarakati, aikhan, fbajaber, oalhubaiti}@kau.edu.sa

***Abstract*—The Multiple Autonomous Unmanned Vehicle Experimental Testbed (MAUVET) is a platform designed by our research group as an open architecture platform with open communication standards and modular software core functions. This paper describes some aspects of the software architecture for MAUVET, focusing on API and GUI interfaces. The software architecture proposed in this paper is an abstraction that hides complexity from application developers who uses the API to interact with UAVs. The testbed framework that uses a modular architecture, is meant to be easily extendable, as it employs software engineering principles such as scalability and reusability. We also present some analysis of our graphical user interface (GUI) that controls UAVs' missions. We also show how some UAVs' functions were tested in some scenarios using UAV emulator.**

***Index Terms*—embedded systems, multilayered architecture, UAV controller, unmanned aerial vehicles, test-bed.**

I. Introduction

Unmanned Aerial Vehicles (UAVs) have drawn a lot of attention in recent years. With advancements in wireless communications and digital electronics, design and development of low-cost, low power, multifunctional sensor nodes and autonomous vehicles, has become possible. Nowadays, such devices are small, smart, and can communicate with wires or wirelessly. Communication can occur either within short or long distances, with sensors of diverse types that needs low energy [1]. Such capabilities of those sensor devices, which include sensing, data processing, and communicating, enable us to design sensor networks based on collaborative effort of many nodes. Since their processing capacity is increasing over the years, nowadays several types of low-cost multifunction sensors exist. In some cases, a situation needs many sensors to sense the environment or take measurements from surroundings. Therefore, we can create a fully adaptive and reconfigurable network of independent agents, which would include heterogeneous agents and other devices.

The work in [2] discussed and highlights the issue of agent oriented software design to inspire a need for careful methodological design of such a system. In this work, we intend to lay the groundwork for the outlined system by designing a real-world testbed with different agents equipped with communication and sensing devices to test and develop a search scenarios application. A set of autonomous robots will be able to form coalitions to perform basic tasks.

In this paper, we report on the work in progress about our developed robotic testbed 'MAUVET' [3], designed to use heterogeneous robots performing coordinated tasks as shown in Fig 1. We display the advantages of our testbed by developing a 'search and find' application. In this application, we show different system components where we assign robotic agents to take part in the search process. Those agents interact with system operator via GUI interface, a part of Base Station, where operator can design, manage, and check search process.

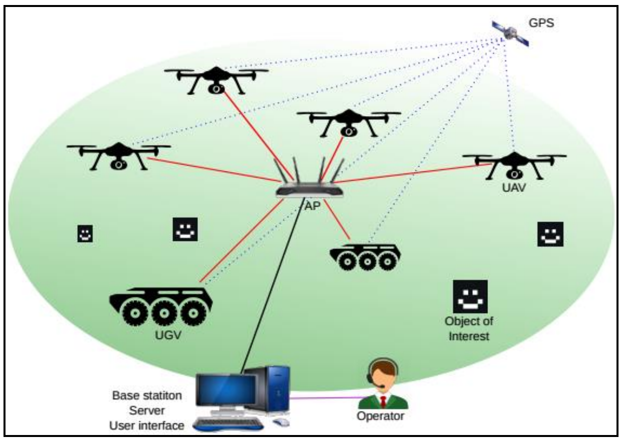


Figure 1. Configuration of experimental area

The testbed framework is of a modular architecture, is easily extendable, and it features design principles like scalability and reusability. This design abstracts software components at a high level, and allows for incorporation and control of diverse embedded devices. These components communicate through provided interfaces. We designed the testbed to be highly cohesive with minimal coupling to enable reusing components as needed and ensuring system heterogeneity.

In this paper, we focus on the architectural design of the server interface. This interface connects the operator to the system and robotic agents via a set of in-house developed API's. The structure of this paper is as follows: Section II analyzes related work. Section III gives an overview of system components. Section IV presents base station reference software architecture. Section V describes API and functions of the testbed. Section VI

Manuscript received March 2, 2018; revised June 22, 2018.

discusses API validation test strategy. Last section is conclusions.

II. Related Work

The main novelty of our project lies in the cooperative integration of a considerable number of entities in tight cooperation within one single networked system with central control. The system controls this network [3] to plan 'search and find' missions dynamically using heterogeneous teams in uncertain environments. We will further investigate such planning and coalition activities in the current project, since the link between the two projects is clear.

References [5]-[8] proposed several generic GUIs suitable for a range of UAV testing and control systems during takeoff, landing and other similar functionalities from base station.

Daniel *et al*. [5] proposed a UAV based solution for the surveillance system. The software offered full features for dynamic allocation of tasks for UAVs via GUI. A realistic 3D technique showed all the real-time operations information. Ground station software helped operators manage tasks of multiple UAVs efficiently. A field experiment was performed with a setup having two quadrotors with visual cameras to test the solution in a real environment.

Alberto *et al*. [6] developed a Ground Control Station (GCS) for control and navigation of multiple UAVs. The System had two core modules, one handles UAV mission planning, and the other controls flight. The system also implemented features for obstacle avoidance and formation flying. This solution used NASA's World Wind API functionality to show UAV path planning and formation setting directly on a 3D map interface. The authors used an in-house simulation environment to test the proposed method and to evaluate and measure the functionalities and performance of the GCS. They have integrated the famous Flight Simulator software 'X-Plane' with GCS. The simulator generated multiple real-time flight trajectories like a real situation. Thus, the application 'X-Plane' is very helpful in generating visuals of the simulated flight before a mission.

Paparazzi [7] is an open source UAV Ground Control Station (GCS) project. It has gained some attention among researchers. The main key feature is the real-time visualization, monitoring and control of an unmanned aerial vehicle. This software package supports different brand types of UAVs. The main shortcoming of this application is that it is not designed to support cooperative control of multiple UAV agents.

Doran *et al*. [8] proposed a Human Computer Interaction (HCI) based GUI interface for base station operations and UAV interaction. The proposed GUI can give crucial details of UAVs for judgment, decision-making and tactical understanding of the use of UAVs. The authors conducted HCI usability tests to measure performance. Tests also covered time needed to use the interface by participant and time needed to carry out a given task. Researchers considered logs and errors recorded during the task and user satisfaction level. Researchers redesigned and retested alternatives to achieve improved performance based on above test data [8].

Work in [9] presented a ground control station based on Robot Operating System (ROS). Design goals include using as much open components as possible, handling heterogeneous UxVs, developing a GUI monitoring and controlling part, and using automatic flight path generation.

Reference [10] discussed shortcomings of ROS, as in the Dronemap project. The authors focused on offloading processes to cloud rather than depending on onboard processing thus enhancing computational performance and connectivity between users and robotic agents.

ROS is a robotics middleware that includes a collection of software frameworks for robot software development [11]. Researchers of [12] listed several ROS limitations. They found few points that limited scalability of multiple robotic systems due to bandwidth and synchronization considerations in ROS design. Thus, for more system control and design flexibility, we choose ROS in this paper to develop our API from scratch based on in-house developed components.

The literature shows –among other things- that we need to analyze UAVs data in real time during the mission, and we need hardware abstraction to reduce complexity. This brings us to propose a flexible architecture that helps application developers concentrate on application logic rather than details of low-level complexity of UAV operation. A well-developed GUI controls UAVs' missions and functionalities. We integrate all mission information a single GUI screen.

III. MAUVET Software Analysis

We have conducted software analysis as part of the system development. In this section, we go through system components then we sketch system's use-case based operational requirements.

A. *System Software Components*

Fig. 2 shows detailed architecture of communication between software modules. The Software architecture of system consists of several modules, some of them run on robots while others run on Base Station (BS).

BS communication server is an application running on BS and its purpose is to:

- Set up and keep communication with robots,
- Gather data from connected robots,
- Offer data and commands to robots, and
- Offer data to user applications (including user interface).

Drone Control software of each drone communicates with communication server on BS through the low-level interface. Communication server collects received data, presents it to modules on BS and allows controlling drones from BS. Specifically, the server gives a current position, altitude and heading for each UAV, detected ground targets and their positions as well as telemetry information: flight mode, current command execution, onboard resources (battery, Wi-Fi signal level, etc.)

among other information. We have built a dedicated API in C++ using socket communication interface to offer these functionalities.

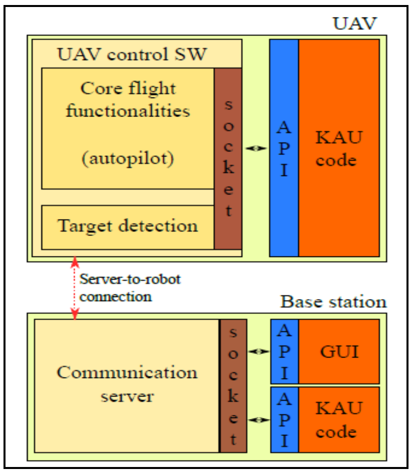


Figure 2. System communication schematics.

B. *Use-case Modeling*

Use-case Modeling is an essential activity of software development and it usually begins before system design process. It helps in visualizing system functionalities at high-level from a user's perspective. A use-case scenario reflects a unique functionality of the system. Use-Case Modeling is a stage of design that exists between system requirements analysis and design phases [13].

Fig. 3 shows basic functionalities of our testbed. The scenario consists of several consecutive steps, which happen in the following sequence (Considering that system hardware initialization has happened beforehand),

1) Operator feeds system parameters of mission via GUI.
2) Software running on BS calculates a search plan and assigns search areas and trajectories to UAVs based on their types and capabilities.
3) UAVs 1, 2 and 3 (supposing we appointed those three UAVs for this mission) start searching their assigned areas until either they find object(s) of interest or search finishes or user stops it.
4) (Optional) UAV 4 gives actual images from camera and operator may control it to check specific sectors. Based on camera images, operator may instruct system to recalculate search plan and system then instructs UAVs to follow the new plan.
5) In case a UAV leaves the area for any reason, BS recalculates search plan and instructs UAVs to follow through.
6) Mission ends when either UAVs find all objects of interest or search completes.
7) During mission, UAVs work in harmony as system ensures prompt control of UAVs and transports traffic exchanged between various components based on programmed logic.
8) Control traffic is data sent to control behavior of UAVs throughout the mission while transport traffic is data feed sent from UAVs to BS, i.e. images feed, sensor information, GPS coordinates, target information, …, etc.

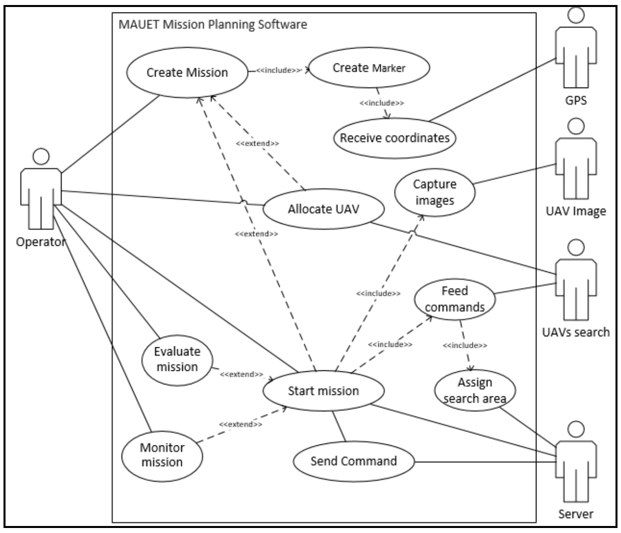


Figure 3. Use-Case diagram of high-level functionalities and actors of the system.

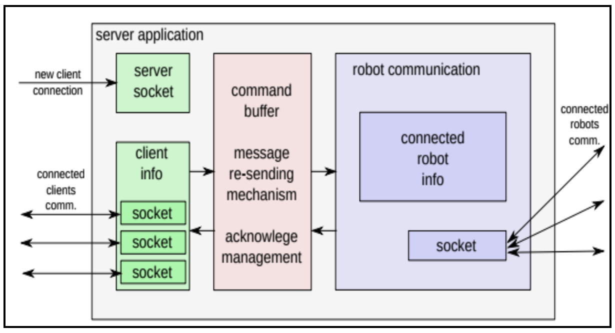


Figure 4. Base station architecture overview.

IV. Base Station Software Architecture

The basic purpose of the 'MUAVET' system server is to offer a central node connecting user applications with control software running on onboard PCs of robots/UAVs. Fig. 4 shows basic block architecture of the server. Left side connection is TCP based and connects server to user interface. Right-hand interface is UDP based and connects server to UAVs. Server manages passage of messages from client applications to UAVs and vice versa to guarantee safe delivery of UAV commands and other data and to optimize throughput of communication network.

Fig. 6 shows an abstract view of GUI architecture, focusing on the design pattern. We have developed GUI of application based on Model View Controller (MVC) design pattern as Fig. 5 shows. The primary reason of using MVC [14] architecture is its ability to enable low coupling in design by separating application's front-end logic from backend. By keeping back-end code into 'Model' and front-end code into 'View' and 'Controller' we can achieve low coupling. MVC architecture also eases separation of input from output; 'Controller' handles Input, while 'View' handles output.

GUI allows operators to upload a flight plan, watch mission data in real time like UAV's battery, airspeed … etc. In addition, it is capable of commanding UAVs, updating waypoints, aborting flights, allowing a manual override, analysis, and results generation.

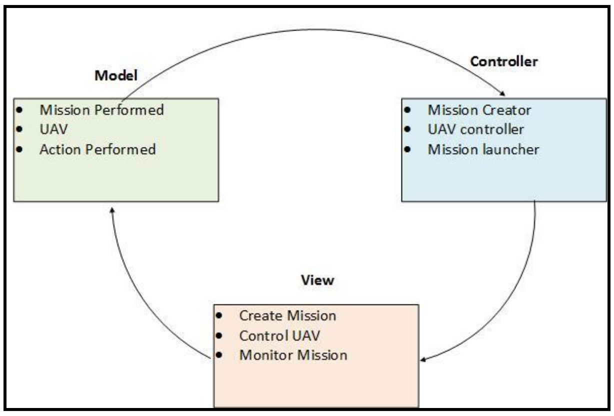


Figure 5. Shows basic MVC architecture for MAUET mission planning software

Fig. 5 above describes how each part of the 'MAUET Mission Planning Software' MVC modules interacts with other parts:

1) Operator sends a request through GUI. 'Controller' module intercepts this request.
2) Navigation logic of 'Controller' dispatches the request to the relevant application logic in 'Model'
3) 'Model' holds functions of application logic, algorithms, data access … etc., while 'Model' processes requests through related application logic, returning results to 'Controller' dispatcher.
4) 'Controller' dispatches those results then specifies which module 'View' would present to client.
5) 'View' gives result to client application.

The platform composes of several layers as shown in Fig. 6. Each layer is a set of services in the application. Upper layers see bottom layers as a set of services. We apply abstraction using APIs to reduce complexity of the system. Developer does not need to know internal details of the system, as they are abstract from developer's perspective. Developer uses API to interact with low-level system components of the architecture by mapping needed functionality of application using upper layers onto specific components. Here API is an abstraction of all system components and is the application interface of the architecture.

There is a central server offering central point of communication and connecting all entities, namely user applications (including user interface) and robots. Connection between user applications and server uses TCP socket interface. Connection between server and UAVs uses a datagram-based UDP protocol.

The system offers functionalities of UAV Control software to other modules through two interfaces: A low-level interface using a standard TCP/IP socket and GUI application should not typically use it. The socket interface offers an open and portable way of connecting server and user applications written in any arbitrary programming language.

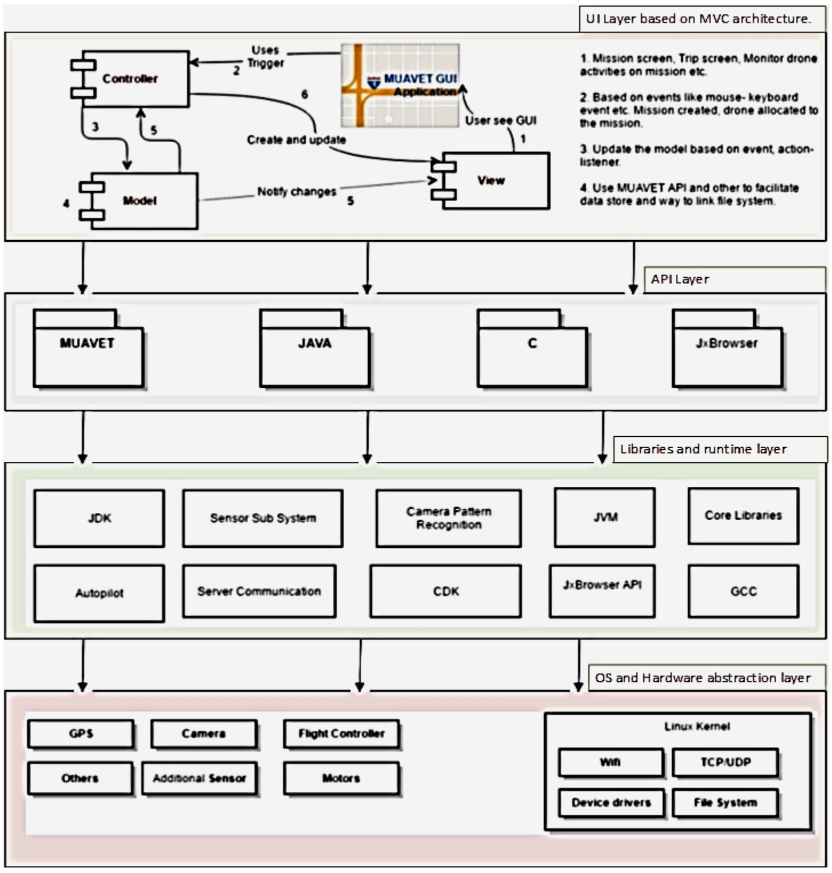

UI Layer based on MVC architecture.
Controller
Uses Trigger
2
MUAVET GUI Application
6
User see GUI
1
3
5
Create and update
View
4
Model
Notify changes
5
1. Mission screen, Trip screen, Monitor drone activities on mission etc.
2. Based on events like mouse- keyboard event etc. Mission created, drone allocated to the mission.
3. Update the model based on event, action-listener.
4. Use MUAVET API and other to facilitate data store and way to link file system.
API Layer
MUAVET
JAVA
C
JxBrowser
Libraries and runtime layer
JDK
Sensor Sub System
Camera Pattern Recognition
JVM
Core Libraries
Autopilot
Server Communication
CDK
JxBrowser API
GCC
OS and Hardware abstraction layer
GPS
Camera
Flight Controller
Others
Additional Sensor
Motors
Linux Kernel
Wifi
TCP/UDP
Device drivers
File System

Figure 6. Layer architecture for MAUET mission planning software

We have built a C++ application interface (API) above the socket interface in form of C++ functions encapsulated by the 'MUAVET' Interface class. GUI application code can use UAV control functionalities simply by calling function from this API. Both interfaces are available on the server as well as on UAVs' onboard computers. It is up to user to decide the architecture of the top-level system (e.g. centralized or distributed). The only major difference between server and onboard API is that onboard applications cannot send commands to other UAVs.

Communication server collects received data, gives them to modules on BS and allows control of UAVs from BS. Specifically, Server gives a current position, altitude and heading for each UAV, detected ground targets and their positions as well as telemetry information: flight mode, current command execution, onboard resources (battery, Wi-Fi signal level … etc.). Server gives data in an asynchronous Way. Each robot may receive control commands from applications running on either the onboard computer or the Base Station. Due to this architecture, a conflict between onboard and BS commands may arise and needs management. A similar conflict might occur between multiple applications running on BS or OBC.

To ensure priority to control commands coming from BS, a local control setting (software switch) user can set from BS is there. If enabled, it allows control of a drone from the onboard application. If disabled, autopilot ignores local commands. This enables operator at BS to take over UAV control in case of onboard software problem.

Control of UAV by user application happens on a relatively high level in terms of GPS waypoints UAV should travel through in mission. Autopilot takes care of low-level control of UAV to follow designated trajectory with minimal error. User might set maximal allowed velocity, while autopilot may need to slow UAV down near waypoints to decrease position errors.

Flight data logging is available both on robot's onboard computer and BS. There should not be a significant difference between server and onboard logs if all robots' data is available on BS. On-board log is useful for offline debugging, e.g. in situations when communication with the Base Station is lost. It is also possible to log all camera-captured images onboard, while UAV only sends some of the images to BS due to wireless traffic limitation.

The sequence diagram shown in Fig. 7 illustrates the sequence of events that occur from mission initialization until UAVs carry out their assigned tasks.

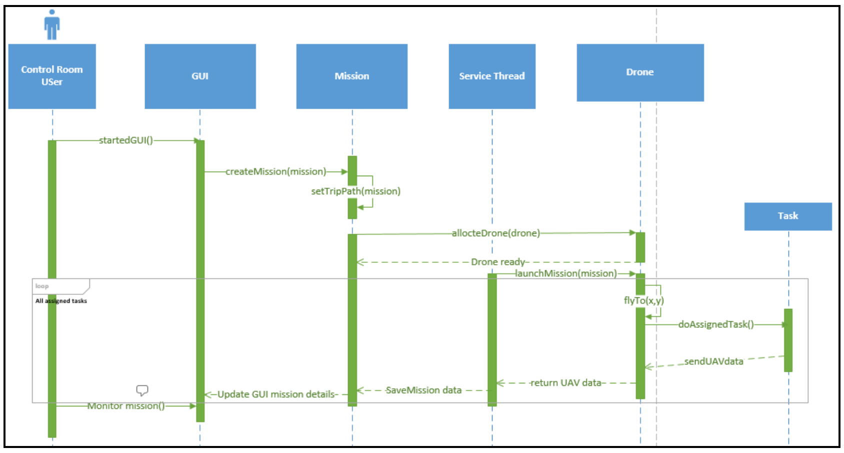


Figure 7. Sequence of events when UAVs carry out assigned tasks

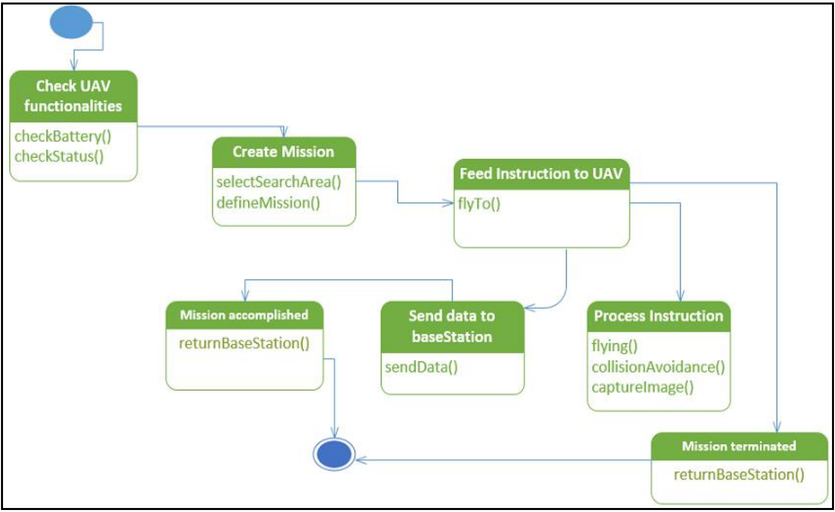


Figure 8. State diagram of events when UAVs executes assigned tasks

Fig. 8, shown below, illustrates high-level view of UAV's state diagram when executing assigned tasks. Showing system's states change sequence based on events occurring from start of initialization of a trip until UAVs carry out assigned tasks and return to designated home location.

V. Testbed API and Functions

Our Application Programming Interface allows sending commands to UAVs and retrieving data from UAVs. There is a significant difference between sending commands to a UAV and receiving data from it. API typically sends a UAV command once and there is a mechanism to ensure correct and prompt delivery of commands to UAVs. Otherwise, API notifies calling application that UAV did not receive the command. On the other hand, API refreshes data from UAVs with every new measurement and sends it to registered receivers periodically.

Communication server buffers commands and will repeatedly send them to UAVs until server receives acknowledgement from said UAVs, or until user sends another command canceling current one. Server notifies users about command acknowledgements returned by UAVs, as shown in Fig. 9.

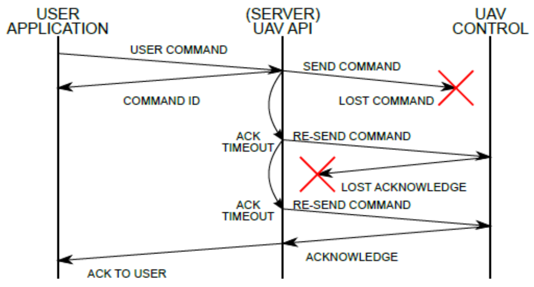


Figure 9. UAV command retransmission and acknowledgement.

TABLE I: CORE API FUNCTIONS

Function	Description	Parameters
Arm	User must arm UAVs before doing any other operation, e.g. before starting motors or taking off. This is a safety feature to prevent accidental activation of the UAV.	No parameters
Disarm	Disarm command disables UAV until re-armed again.	No parameters
Takeoff	A UAV on the ground starts its motors and takes off, hovering until some other it receives another command.	No parameters
Land on position	Description: UAV flies to target GPS position, descends to ground level and turns off its motors.	2D position latitude, longitude
Land	UAV descends and lands on current position.	No parameters
Hold Position	UAV interrupts any processed command and stops mid-air. The current trajectory plan is discarded, if any.	No parameters
FlyTo	UAV will fly to the given 3D GPS position as fast as possible while satisfying preset maximal velocity limit. The requested (maximal) velocity may be set using SetFlyVelocity command.	3D position longitude, latitude, altitude

To distinguish acknowledgments to different simultaneous commands, API gives a unique identifier (id) to every sent command and matches acknowledgement's id to this id when received. API treats acknowledgements in a manner like other data coming from UAVs, so it does not acknowledge nor re-send them. Table I discusses some of API's core functions.

VI. API Validation Test Strategy

We did the testing of various modules of this application in this stage using simulation to reduce the risk of mishap or malfunction to the actual UAVs during flight missions.

A. *Testing Environment*

The system we used for this round consists of:

- Debian based Linux operating system [15]: We selected Debian for high package compatibility and abundance of support material.
- UAV simulator: A program written in C consisting of UAV movement simulation and a sender/receiver module. This program does not simulate UAV failure due to any internal or external conditions (i.e. simulates ideal UAV operation scenario).
- API modules: Written in Java programming language, this part is like the API software that we will be using on the actual implementation (i.e. with real hardware) albeit with limited command support. Table I, shows command support at the time of testing.

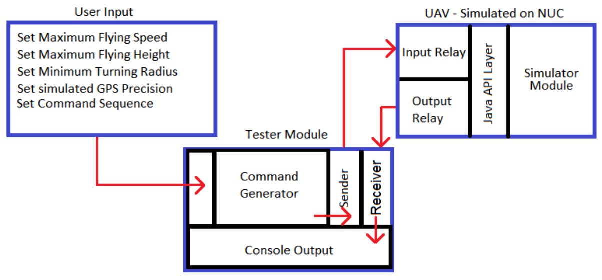


Figure 10. Testing system model.

B. *Testing Goal and Strategy*

Since no failures are simulated, focus of this test was to confirm software correctness and exact implementation of the UAV operation modes state diagram (shown in Fig. 11) (especially the Java API part) and readiness for deployment. We consider results satisfactory if no software exceptions arise or any design oversights happen.

We performed black box testing for this series of testing runs. Although we can have access to the innards of the 'MUAVET' Java API and the drone's simulator-software (developed in-house), we have decided to test the system from the point of view of an external user, for when the test proves successful we can conclude that the inner workings are in order.

The state diagram shown in Fig. 11 summarizes the high view of different paths testing can take during different runs.

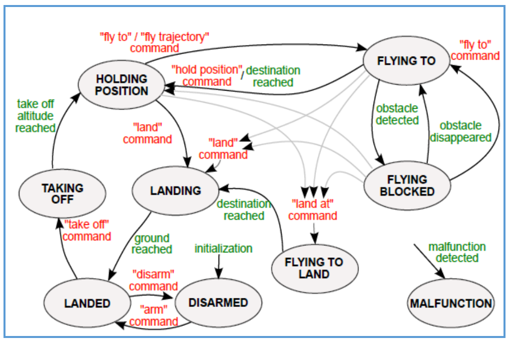


Figure 11. UAV operation modes state diagram

This is a general diagram for real operation of UAVs. For this testing scenario, both “Flying Blocked” and “Malfunction” states were not met nor were “malfunction detected,” “obstacle detected” or “obstacle disappeared” transitions occur due to the absence of failure simulation in the current version of the simulator.

Red transitions show transitions that user commands trigger, green ones show those caused by internal or external state changes. Note that not all transitions are drawn for simplicity.

We conducted testing manually, i.e. user selected the command sequence to send to tester module. This generates the proper command format for the Java API. We deduct success or failure of a testing run by seeing the output that periodically appears both in the console of the API as well as in that on UAV simulator console.

C. *Sample Scenarios*

A path from the state diagram was chosen (in testing, all paths from the diagram—excluding those passing through states or transitions we have pointed out previously—were tested thoroughly). As per the state diagram, Fig. 12, the following sequence of consecutive commands creates that path.

Takeoff → Arm → Takeoff → disarm → land → disarm.

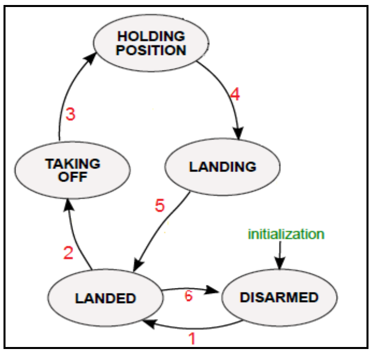


Figure 12. State diagram for a sample test with valid commands

To test the transition sequence that system follows as per defined path, we send some valid and some invalid commands to the UAV. A command is considered valid or not based on state diagram shown in Fig. 11. For example, take-off command was sent before the UAV is armed which is prohibited as shown in Fig. 13.

Another case is to send disarm command while UAV flying. For the command sequence discussed earlier, testing takes UAV through transitions 1-6 (shown in Fig. 12) in order.

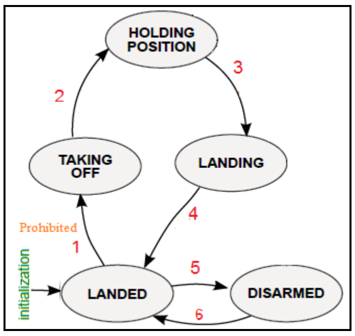
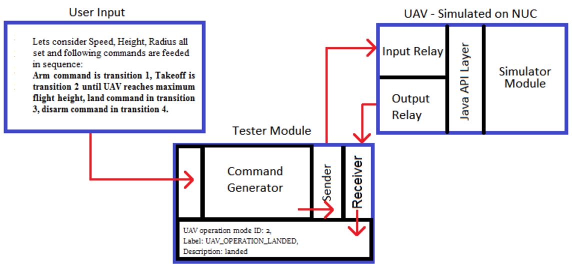


Figure 13. State diagram for sample test with invalid command



Lets consider Speed, Height, Radius all set and following commands are feeded in sequence:
Arm command is transition 1, Takeoff is transition 2 until UAV reaches maximum flight height, land command in transition 3, disarm command in transition 4.

UAV operation mode ID: 2,
Label: UAV_OPERATION_LANDED,
Description: landed

Figure 14. Testing system model sample command correct order

Arm command is transition #1; Takeoff is transition #2, until UAV reaches defined flight height (configured by user at beginning of test (see Fig. 10) and so on.

To figure out the outcome of each command, we study verbose output at the console. Sample output is shown in Fig. 14 (all commands sent in order)

In conclusion, the experiment has met its goals. No software errors or exceptions has occurred nor have we met any logical misbehavior. We judge testing as successful.

VII. Conclusions

In this paper, we presented a reference software architecture for a state of the art Heterogeneous UAVs testbed (MAUVET). These types of testbeds are becoming essential for testing applications based on UAV systems. The system specification describes the architecture and software components of the system and defines the interfaces between its components and between system and users. The testbed uses a flexible software architecture that is easy to extend and is scalable. The paper also discussed the integration between the Base Station, main server, and UAVs, and how the API offers core UAV functionalities at a high level for developers, who would then focus on experiment and application design rather than low-level complexity, which we hide from developer. Finally, a set of experimental results were presented to illustrate the flexibility, usefulness, and efficiency of the proposed architecture.

Acknowledgment

King Abdul-Aziz City for Science and Technology (KACST) has funded this project, under grant no. (AP -

35 - 162). The authors, therefore, acknowledge and thank KACST for its technical and financial support.

References

[1] S. Sendra, J. Floret, M. Garcia, and J.F. Toledo, "Power saving and energy optimization techniques for wireless sensor networks," *Journal of Communications*, vol. 6, pp. 439–459, September 2011.

[2] O. Shehory and A. Sturm, *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, New York: Springer, 2014, p. 331.

[3] A. Barnawi and A. Al-Barakati, "Design and implantation of a search and find application on a heterogeneous robotic platform," *Journal of Engineering Technology*, vol. 6, pp. 235-239, October 2017.

[4] E. Foe, M. Kudelski, L. Gambardella, and G. A. Di Caro, "Connectivity-aware planning of search and rescue missions," in *Proc. 11th IEEE Int. Symp. on Safety, Security, and Rescue Robotics*, Linköping, Sweden, 2013, pp. 21–26.

[5] D. Perez, I. Maza, F. Caballero, D. Scarlatti, E. Casado, and A. Ollero, "A ground control station for a multi-uav surveillance system," *Journal of Intelligent and Robotic Systems*, vol. 69, pp. 119-130, January 2013

[6] A. T. Angonese and P. F. F. Rosa, "Ground control station for multiple UAVs flight simulation," in *Proc. IEEE Latin American Robotics Symposium*, Arequipa, 2013, pp. 136-141.

[7] B. Pascal, D. Antoine, G. Michel, H. Pierre-Selim, and T. Jeremy, "The paparazzi solution," in *Proc. 2nd US-European Competition and Workshop on Micro Air Vehicles*, Sandestin, 2006, pp. 1-15.

[8] D. Cavett, M. Coker, R. Jimenez, and B. Yaacoubi, "Human computer interface for control of unmanned aerial vehicles," in *Proc. Systems and Information Engineering Design Symposium*, Charlottesville, 2007, pp. 1-10.

[9] J. C. Del Arco, D. Alejo, B. C. Arrue, J. A. Cobano, G. Heredia, and A. Ollero, "Multi-UAV ground control station for gliding aircraft," in *Proc. 23th Mediterranean Conf. on Control and Automation*, Torremolinos, 2015, pp. 36-43.

[10] A. Koubaa, B. Qureshi, M. F. Sriti, Y. Javed, and E. Tovar, "A service-oriented cloud-based management system for the internet-of-drones," in *Proc. IEEE Int. Conf. on Autonomous Robot Systems and Competitions*, Coimbra, 2017, pp. 1-4.

[11] Robot Operating System (2017). [Online]. Available: http://www.ros.org/

[12] Analyzing ROS Distribution Capabilities. (2016). [Online]. Available: http://www.dcs.gla.ac.uk/research/rosie/ros-limits-2016-08-11.html

[13] M. I. Muhairat and R. E. Al-Qutaish, "An approach to derive the use case diagrams from an event table," in *Proc. 8th Int. Conf. on Software Engineering Parallel and Distributed Systems*, Wisconsin, 2009, pp. 33-38

[14] D. PaulPop and A. Altar, "Designing an MVC model for rapid web application development," in *Proc. 24th DAAAM Int. Symp. on Intelligent Manufacturing and Automation*, Zadar, 2014, pp. 1172-1179.

[15] Debian Operating System. (2018). [Online] Available: https://www.debian.org/

Ahmed Barnawi is a full professor at the Faculty of Computing and IT, King Abdul-Aziz University, Jeddah, Saudi Arabia, where he works since 2007. He received his Ph.D. at the University of Bradford, UK in 2006. His research interests include robotics cognitive radios system, next generation networks and cloud computing. He is the managing director of KAU Cloud Computing Research Group. He is also a holder of multiple patents in wireless communications.

Abdullah Al-Barakati is an Assistant Professor at King Abdul-Aziz University, Saudi Arabia. He was the head of the Information Systems Department. He received a B.Sc. (Hons) degree in Computer Science in 2004, a M.Sc. degree in Software Engineering and a Ph.D. degree in Computer Science in 2012, from the University of Sussex, UK. His research interests revolve on the use of Big Data and Web technologies.

Asif Irshad Khan is a Ph.D. holder and now works as a faculty member in the Department of Computer Science at King Abdul-Aziz University, Saudi Arabia. He has over fifteen years of experience as a professional academician and researcher. He published several research articles in leading journals and conferences. His current research interest includes Software Engineering with a focus on Software Product Line Engineering.

Fuad Bajaber received his Ph.D. from the Department of Computing, Bradford University, UK. He received his MSc in Computer Science from George Washington University, USA, and B.S. in Computer Science from King Abdul-Aziz University, Saudi Arabia. Dr. Fuad is now an assistant professor in the Department of Information Technology at King Abdul-Aziz University where his main research interests include design, analysis and measurement of wireless sensor and ad hoc networks as well as cloud computing.

Omar Alhubaiti is a research assistant at King Abdul-Aziz University, Saudi Arabia. He received his B.Sc. in Computer Science from Faculty of Computing and Information Technology at King Abdul-Aziz University in 2017. His research interests are computer networking, software engineering, and robotic systems.