drones

MDPI

Article

Development of Multi-Quadrotor Simulator Based on Real-Time Hypervisor Systems

Muhammad Faris Fathoni [1], Seonah Lee [2,*], Yoonsoo Kim [3], Ki-Il Kim [4] and Kyong Hoon Kim [5,*]

[1] Department of Informatics, Gyeongsang National University, Jinju 52828, Korea; mfarisfathoni@gnu.ac.kr
[2] Department of AI Convergence Engineering (Graduate) and Aerospace and Software Engineering (Undergraduate), Gyeongsang National University, Jinju 52828, Korea
[3] Graduate School of Mechanical and Aerospace Engineering, Gyeongsang National University, Jinju 52828, Korea; yoonsoo@gnu.ac.kr
[4] Department of Computer Science and Engineering, Chungnam National University, Daejeon 34134, Korea; kikim@cnu.ac.kr
[5] School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Korea
* Correspondence: saleese@gnu.ac.kr (S.L.); kyong.kim@knu.ac.kr (K.H.K.)

Abstract: Today, simulator technology has been widely used as an important part of quadrotor development such as validation and testing. A good quadrotor simulator can simulate the quadrotor system as closely as possible to the real one. Therefore, in case of multi-quadrotor simulator, the simulator should not only can simulate a multi-quadrotor system, but also every quadrotor should be able to leverage their own resources. To solve this issues, in this paper, we present a hypervisor-based multi-quadrotor simulator. We used RT-Xen as hypervisor, a real-time Xen hypervisor. To ensure every quadrotor runs in real-time manner, we implemented quadrotor simulator in Litmus-RT which is a real-time extension of Linux. In this paper, we conducted some testing and performance evaluation for particular cases on our multi-quadrotor simulator: step-input responses, computation time, and response times. Based on the performance evaluation, our hypervisor-based multi-quadrotor simulator environment is proven to meet the real-time requirements. The results show that three important tasks in quadrotor system: Stability Controllability Augmented System (SCAS), Equation of Motion (EOM), and waypoint following task, are finished before their deadlines; in fact, 20 ms, 10 ms, and 40 ms before the deadlines for SCAS, EOM, and waypoint following, respectively.

Keywords: real-time simulator; multi-quadrotor simulator; hypervisor; waypoint following

Citation: Fathoni, M.F.; Lee, S.; Kim, Y.; Kim, K.-I.; Kim, K.H. Development of Multi-Quadrotor Simulator Based on Real-Time Hypervisor Systems. *Drones* **2021**, *5*, 59. https://doi.org/10.3390/drones5030059

Academic Editor: Diego González-Aguilera

Received: 4 May 2021
Accepted: 2 July 2021
Published: 8 July 2021

1. Introduction

Today, quadrotor UAVs play a major role in many aspect of human life. Quadrotors have been used ranging from surveillance [1], photography [2,3], agriculture [4], to quadrotor racing [5]. On the top of that, some implementations such as surveillance, is using more than one quadrotor (multi-quadrotor) in order to get more benefits. The advantage of quadrotor compared to other type of UAV is hovering ability. Furthermore, due to excellent maneuvering capability, quadrotors are able to do a mission with any kind of environment.

In the case of multi-quadrotor system, some circumstances such as flight formation and coordination, need to be computed carefully to avoid a collision between quadrotors. The environmental conditions and situations may also affect the stability of the quadrotor, either as a group or as a single quadrotor. The quadrotor may face some obstacles, different elevations, or even complex fight route that makes the quadrotor fly abruptly that may also affect the quadrotor stabilization. Besides that, every quadrotor in the group must be ensured in a stable state so it does not affect the stability of other quadrotor in the group. Therefore, the control performance of quadrotor is very important to ensure the stabilization of quadrotor.

On the scheduling point of view, quadrotor system can be abstracted as periodic task models. Each task at least has two parameters: period and execution time. This periodic

task model first was introduced by Liu and Layland [6] in which every task has two parameters: periods and execution time. In our typical quadrotor system, each quadrotor has four tasks: Equation of Motion (EOM) task, Stability Controllability Augmented System (SCAS) task, Waypoint following task and Network task. This task specification is responsible for specific function in quadrotor which will be explained in Section 3. Therefore, quad-rotor system needs a middleware or operating system to handle task scheduling.

In robotics community, Robot Operating System (ROS) plays an important role for processing tasks of robot [7]. ROS is a robotics middleware that provides a framework for developing robot software. However, although ROS is also popular in quadrotor simulation, ROS is not real-time. Generally, quadrotor is a hard-real-time system, meaning that all tasks should meet their deadline because any deadline miss could bring an accident to the quadrotor indirectly. Therefore, RTOS or real-time middleware is recommended as middleware of quadrotor system. There are many RTOS and real-time middleware available, either closed or open-source, for example: Linux, FreeRTOS, Real Time Application Interface (RTAI), VxWorks, Linux Testbed for Multiprocessor Scheduling in Real-Time (Litmus-RT), etc. With these RTOSs, tasks are guaranteed not to miss deadlines.

On the other hand, simulator technology is widely used as an important part of quadrotor development. Quadrotor simulator also useful for validation and testing. Engineers and researchers also use simulator to simulate quadrotor missions and its interactions with environments before they conduct a real-life experiment. Accidents can occur when something unexpected happens to the system, such as bugs, calculation errors, and design errors which can be fatal for quadrotor in real life implementation. These circumstances are even more complex in multi-quadrotor system. Hence, multi-quadrotor simulator platform is needed to reduce losses caused by a physical accident during flight tests. The main contributions of this paper includes:

- Providing a real-time simulator for multi-quadrotor;
- Developing a real-time-hypervisor based quadrotor simulator which makes each quadrotor can have resources independently to other quadrotors;
- Implementing a waypoint following simulation of multi-quadrotor using this platform which proven our multi-quadrotor simulator environment is feasible to be used as a testbed for any multi-quadrotor experiment; and
- Providing a performance evaluation and analysis, by recording a step-input responses, computation time, and response times of tasks.

In this paper, we develop a real-time multi-quadrotor simulator environment in hypervisor. In our simulator environment, each quadrotor simulator is run on a separate virtual machine (called domain). This separation allows each quadrotor to leverage its own resources (processor core) as if it were in real quadrotor. Hypervisor used in our simulator environment is RT-Xen, an open-source virtualization platform with real-time performance guarantees [8] (explained in Section 3.3). And for each domain, to guarantee a real-time performance, we installed Litmus-RT [9,10] which is a real-time extension of Linux (explained in Section 3.3).

2. Related Works

There are some researchers who have developed their own quadrotor simulator for their own particular purpose. MATLAB-Simulink is one of the most popular tools for developing an UAV simulator, even though it is hardly to simulate in real-time manner. In [11], a MATLAB-based quadrotor simulator has been developed to check the validity of the dynamic model and control algorithms of their in-house quadrotor. This paper [12] also uses MATLAB-Simulink for modeling and control algorithms. In [13], a quadrotor simulator platform and environment model has been implemented using MATLAB-Simulink with various sensors.

There are some widely known quadrotor simulator, either developed by quadrotor company for their own type of quadrotor or open-source simulator, for example: DJI Flight Simulator [14] and RotorS. DJI Flight Simulator is designed for enterprise users of

DJI drones. RotorS is a Gazebo-based UAV simulator which also provides some specific multi-quadrotor models [15]. Gazebo is an open-source robot simulator which is also integrated with Open Dynamics Engine (ODE) as a physics engine, and OpenGL as an API for graphics rendering. RotorS developed by the Autonomous Systems Lab of ETH Zurich. But, both Gazebo and RotorS do not provide enough support for a quadrotor simulator environment.

The implementation of quadrotor simulator can be even more complex when it comes to multi-quadrotor system, since there must be some interaction between these quadrotors. A real-time multi-UAV simulator in [16] was implemented in hardware-in-the-loop manner by using fixed-wing UAV as a model. But, this paper only provides a distributed architecture of multi-UAV simulator. Moreover, other focus of this research is CommLibX, a communication framework between simulation modules through virtual channels.

Other research, for example in [17], developed a 'near-real-time' simulation environment for multi-UAV called MUSE (Multiple UAV Simulation Environment). In this simulation environment, each Simulink-based UAV runs on each individual PC connected through UDP network to the Ground Control Station (GCS). However, to do a simulation using this environment is very costly, especially when more UAVs are needed, because it needs one PC for one simulator.

The Drone Watch and Rescue (DWR) in [18] was developed using web development technologies in client-server manner. Clients can run this simulator through any HTML5 web browser by sending a request to the server. The server which is responsible for the core of simulator would run quadrotor simulator after receiving any client's request. But, in this simulator, the client is only receives and shows on the client's screen the status of simulation, not runs the quadrotor simulator. Therefore, when running a multi-UAV simulator, all processes of all UAVs are executed in server.

OpenUAV [7] offers a quadrotor simulator environment by leveraging Containers as a Service (CaaS) that makes users carry out simulation on the cloud. The core component of quadrotor simulator of OpenUAV was developed using Robot Operating System (ROS), Gazebo, and Docker (CaaS platform). But in this simulator, each quadrotor only uses containers, it does not own resource independently.

3. Proposed Model

3.1. Quadrotor Simulator Configuration

A typical configuration of quadrotor simulator environment contains Ground Control Station (GCS) and quadrotor. GCS is a node (usually PC) that is responsible for mission control of quadrotor and the monitor of the entire system. The user can implement any mission control related plan such as flight mode (autonomous or manual flight), formation flight, trajectory plan, mission plan (depending on the additional features of quadrotors), and many more. Sometimes, in GCS, the pilot/user can also control the quadrotor manually like flying an aircraft. Other than that, the user can monitor some data of quadrotor in GCS. The quadrotor sends particular data periodically to the GCS through a network such as status, position, real-time visual of quadrotor point-of-view (POV). In simulator, based on the data received from quadrotors, user can add a 2D/3D visualization of quadrotor. Figure 1 shows a configuration of our multi-quadrotor simulator environment which consists of three PCs: GCS and two groups of quadrotors (group 1 and group 2). Each group consists of four quadrotors.

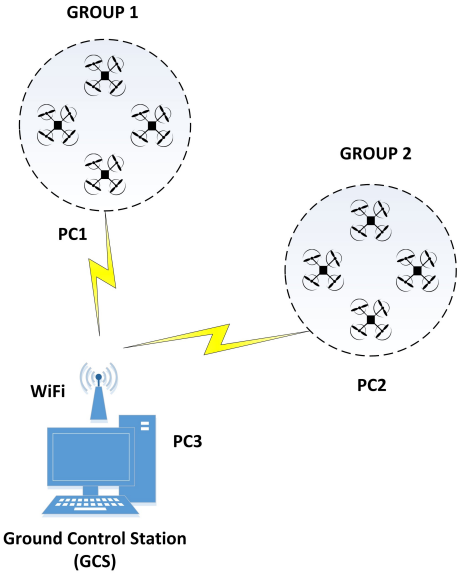


Figure 1. Configuration of multi-quadrotor simulator.

3.2. Quadrotor Model

In general, the quadrotor simulator system can be divided into two parts: EOM and SCAS. QRSim in Figure 2 shows our quadrotor model which consists of quadrotor plant (EOM), SCAS and Waypoint Folowing (and Formation). EOM (blue box) is a quadrotor plant that defines a non-linear dynamic of quadrotor. In real-life experiment, EOM is equal to the quadrotor itself. Control part is a computer (sometimes an embedded computer) that computes all control related algorithm. Control can be divided into two parts: the inner loop (local control) and the outer loop (group control). In our quadrotor simulator, local control handles a SCAS for quadrotor stabilization, UDP communication and health monitor. Meanwhile group control handles mission tasks, such as flight formation and waypoint following. Table 1 shows a list of variables that involved in this research.

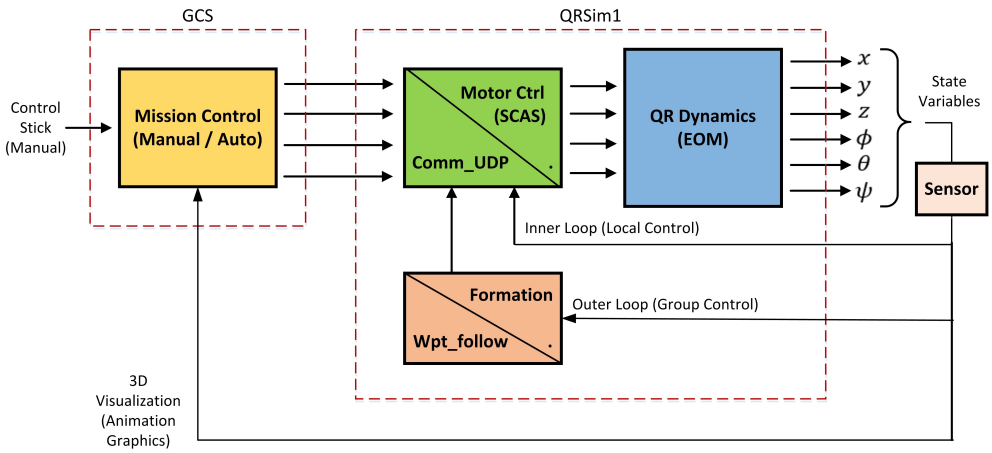
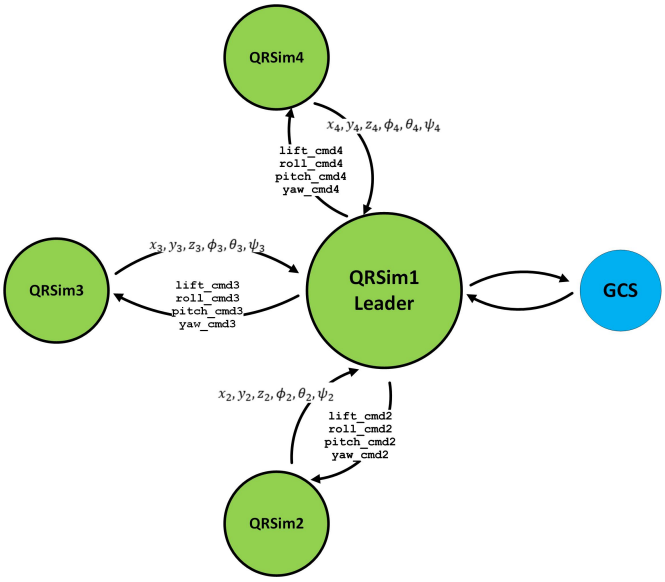


Figure 2. Quadrotor model.

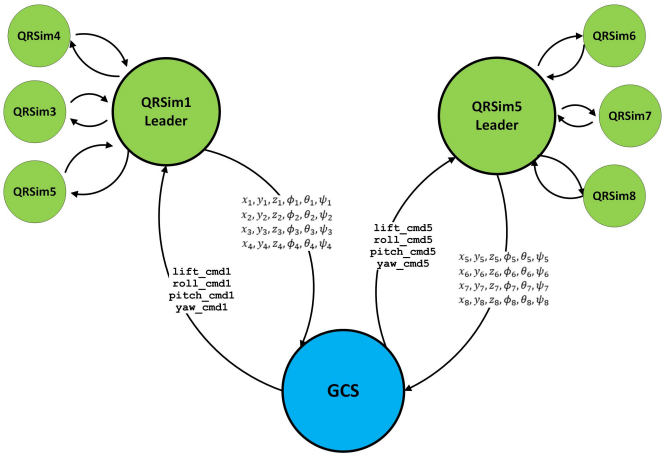
Periodically, each quadrotor sends their state variables to the GCS for monitoring and visualization on GCS. Each quadrotor group has one quadrotor that act as a leader of the group. The quadrotor followers send their state variables to the leader, then the leader passes these variables along with its own variables to the GCS, as seen in Figure 3. In exchange of these state variables, some commands (`lift_cmd, pitch_cmd, roll_cmd, yaw_cmd`), are sent from the GCS to every leader of the group. These commands are from direct user manual input or pre-defined waypoint following plan (detailed in Section 3.5). For the quadrotor followers, the commands are received from their quadrotor leader.

Table 1. The list of variables.

Symbols	Meaning
x	forward position (m)
y	side position (m)
z	vertical position (m)
u	forward velocity (m/s)
v	side velocity (m/s)
w	vertical velocity (m/s)
θ	pitch angle (deg)
ϕ	roll angle (deg)
ψ	yaw angle (deg)
p	pitch velocity (deg/sec)
q	roll velocity (deg/sec)
r	yaw velocity (deg/sec)
δ_l	lift control input (deg)
δ_r	lateral control input (deg)
δ_p	longitudinal control input (deg)
δ_y	directional (yaw) control input (deg)



(**a**) Quadrotor Follower-Quadrotor Leader



(**b**) Quadrotor Leader-GCS

Figure 3. Data-Flow Diagram (DFD) of multi-quadrotor simulator.

By using Newton's law of motion (translational acceleration = force/mass, and rotational acceleration = moment/moment of inertia), EOM of quadrotor can be written as a non-linear first order differential equation as follows [19].

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \frac{dx(t)}{dt} = f(\mathbf{x}(t), \mathbf{u}(t); t) \\ \mathbf{x}(t) &= [x \quad y \quad z \quad u \quad v \quad w \quad \phi \quad \theta \quad \psi \quad p \quad q \quad r] \\ \mathbf{u}(t) &= [\delta_l \quad \delta_r \quad \delta_p \quad \delta_y] \end{aligned} \tag{1}$$

where, $x(t)$ is a state variable vector of quadrotor, $u(t)$ is a control input vector, and t is a continuous time variable (sec). As mentioned above (and shown in Figure 2), these state variables are the output of EOM process. So, in general the entire quadrotor simulation process can be summarized into a block diagram in Figure 4 below.

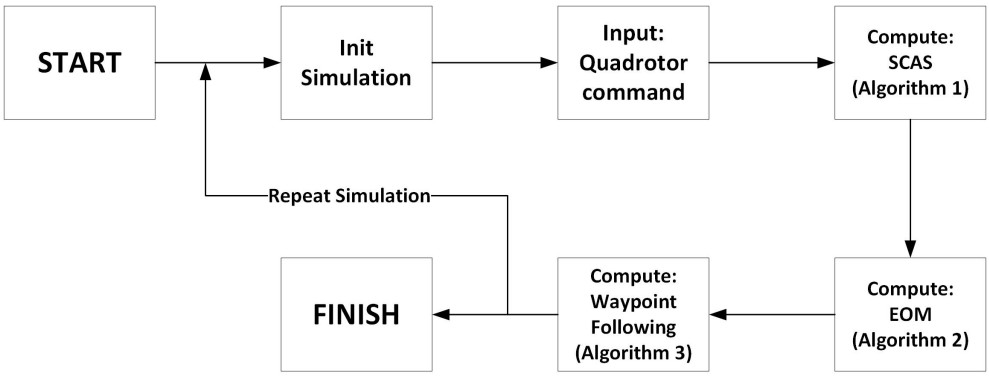


Figure 4. Block diagram of quadrotor simulation process.

Algorithm 1 shows pseudocode process of Stability Controllability Augmentation System (SCAS). To compute quadrotor states, numerical integration computations are needed [20]. In our quadrotor simulator, Euler integration method is used for integration computation which is the simplest integration method [21]. Algorithm 2 shows the pseudocode process of EOM model for nonlinear dynamics of quadrotor simulation.

Besides SCAS and EOM, our quadrotor simulator environment also has a waypoint following function handled by a group control task. Algorithm 3 shows pseudocode process of waypoint following algorithm.

Algorithm 1: SCAS.

```
/* time:  time variable (s)                                          */
/* dt:  time period (s)                                              */
/* lift_cmd, pitch_cmd, roll_cmd, yaw_cmd:  command input form GCS
   */
/* O1, O2, O3, O4:  rotor speeds (rpm)                               */
Initialize: Read the parameters of SCAS
while true do
    Receive command input from GCS: lift_cmd, pitch_cmd, roll_cmd, yaw_cmd
    Compute scas_lift_cmd, proportional integral control (PI)
    Compute scas_roll_cmd, proportional integral control (PI)
    Compute scas_pitch_cmd, proportional integral control (PI)
    Compute scas_yaw_cmd, proportional integral control (PI)
    Compute rotor speeds (rpm): O1, O2, O3, O4
    Update time: time = time + dt
```

Algorithm 2: EOM Model.

```
/* time: time variable (s)                                          */
/* dt: sampling time (s)                                            */
/* x, y, z: forward, side, and vertical positions (m)               */
/* φ, θ, ψ: roll, pitch and yaw angle (rad)                         */
/* X, Y, Z: forces (N)                                              */
/* L, M, N: moments (Nm)                                            */
/* u, v, w: translational velocities (m/s)                          */
/* p, q, r: rotational velocities (rad/s)                           */
/* u̇, v̇, ẇ: translational acceleration (m/s^2)                      */
/* ṗ, q̇, ṙ: rotational acceleration (rad/s^2)                       */
/* O1, O2, O3, O4: rotor speeds (rpm)                               */
/* f1, f2, f3, f4: rotor forces (N)                                 */
/* m, g, Ix, Iy, Izx, b, c, d, l: parameters of quadrotor system    */
Initialize: Read the parameters of quadrotor system
Read the initial state of quadrotor (x, y, z, u, v, w, φ, θ, ψ, p, q, r)
while true do
    Read/get rotor speeds: O1, O2, O3, O4
    Compute rotor forces: f1, f2, f3, f4
    Compute quadrotor forces: X, Y, Z Compute quadrotor moments: L, M, N
    Compute quadrotor translational acceleration (Newton's law, forces/mass):
      u̇, v̇, ẇ
    Compute quadrotor rotational acceleration (Newton's law, moments/inertia):
      ṗ, q̇, ṙ
    Compute quadrotor velocity (Euler integration of acceleration): u, v, w, p, q, r
    Compute quadrotor position (Euler integration of velocity): x, y, z
    Update time: time = time + dt
```

3.3. Tools

3.3.1. RT-Xen Hypervisor

Real-Time Xen (RT-Xen) is an open-source virtualization platform for systems integration with real-time performance guarantees. RT-Xen is an extension of Xen, an open-source Virtual Machine Monitor (VMM) with real-time performance requirements [22]. Unlike a mainstream Type-2 hypervisor such as VM Player, Virtual Box Xen, which runs on conventional OS, Xen is a Type-1 hypervisor that runs directly on hardware. RT-Xen bridges the gap between hierarchical real-time scheduling theory and virtualization technologies for real-time computing on virtualized environment [8]. In our quadrotor simulator, we use RT-Xen 2.2 based on Xen 4.6. Figure 5 shows Xen scheduling architecture.

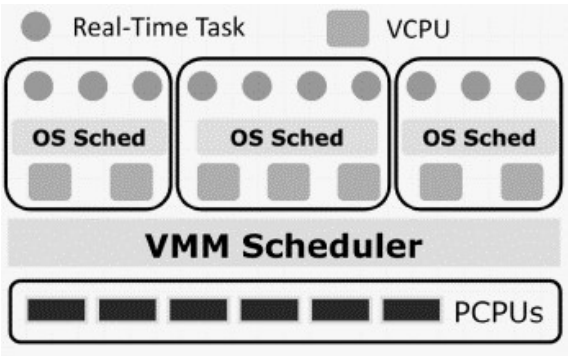


Figure 5. Xen scheduling architecture [23].

3.3.2. Litmus-RT

Litmus-RT is an extension of Linux kernel which focuses on real-time scheduling and synchronization [24]. The latest version of Litmus-RT is based on Linux 4.9.30. Litmus-RT provides abstractions and interfaces within kernel that allows to change the active scheduling policy in runtime. Other than that, for implementation purpose, Litmus-RT also provides additional system calls for real-time tasks [9,10]. In our quadrotor simulator experiment, we use one of program skeleton provided by Litmus-RT, base_mt_task.c, for setting up a multi-threaded real-time task [25]. Meanwhile, the scheduling policy used in our quadrotor simulator experiment is GSN-EDF which is Global Earliest-Deadline-First (EDF) scheduling. Detailed explanation of quadrotor simulator implementation using Litmus-RT is already published in our previous paper [19].

Algorithm 3: Waypoint Following.

```
/* cwpt: current waypoint                                          */
/* nwpt: number of waypoint                                        */
/* lift_cmd, pitch_cmd, roll_cmd, yaw_cmd: command input form GCS  */
/* waypoint: the waypoint position (coordinates) data              */
/* distance: the distance of current position to the next waypoint
   */
/* alt_uav: the current altitude of quadrotor                      */
/* alt_dmd: the demanded altitude of the waypoint                  */
/* hdg_uav: the current heading of quadrotor                       */
/* hdg_dmd: the demanded heading of the waypoint                   */
/* dsmall: the small value of distance                             */
Initialize: Read the waypoint
while cwpt < nwpt do
    Compute the distance
    Compute alt_uav = get_alt_uav()
    Compute alt_dmd = get_alt_dmd()
    if alt_uav ≠ alt_dmd then
        Update lift_cmd
    Compute hdg_uav = get_hdg_uav()
    Compute hdg_dmd = get_hdg_dmd()
    if hdg_uav ≠ hdg_dmd then
        Update yaw_cmd
    Moving forward: Update pitch_cmd
    if hdg_uav ≠ hdg_dmd then
        Update yaw_cmd
    if distance < dsmall then
        cwpt+ = 1
        pitch_cmd = 0
```

3.3.3. Feather-Trace

Feather-Trace is an open-source light-weight event tracing tools for RTOS on Intel processors [26]. There are some advantages using feather-trace as a tracing tool: multiprocessor-safe, low overhead, and wait-free. Multiprocessor-safe means tracing activities on one processor never affect other processors. Feather-trace can be downloaded from the website [27] or can be obtained as a companion software of Litmus-RT [24].

In this paper, we used two tools from feather-trace: `st-trace-schedule` and `st-jobs-stat`. The `st-trace-schedule` tool is used for tracing all scheduling decisions during simulation. The result of this tracing tool is recorded in BIN file. Furthermore, to obtain a job statistics and tracing result from BIN file in readable file format, `st-jobs-stat` tool is used.

3.4. System Architecture

For experiment purposes, we design and implement a hypervisor-based quadrotor simulator. In our quadrotor simulator environment, we used two PCs with RT-Xen hypervisor installed on each PC. Four domains are mounted on each PC which each domain is installed with Linux based Litmus-RT. Each domain runs one quadrotor simulator, so each PC represents a group of four quadrotor simulators. Then, both PCs are connected to the GCS's PC. Figure 6 shows an architecture of hypervisor-based quadrotor simulator.

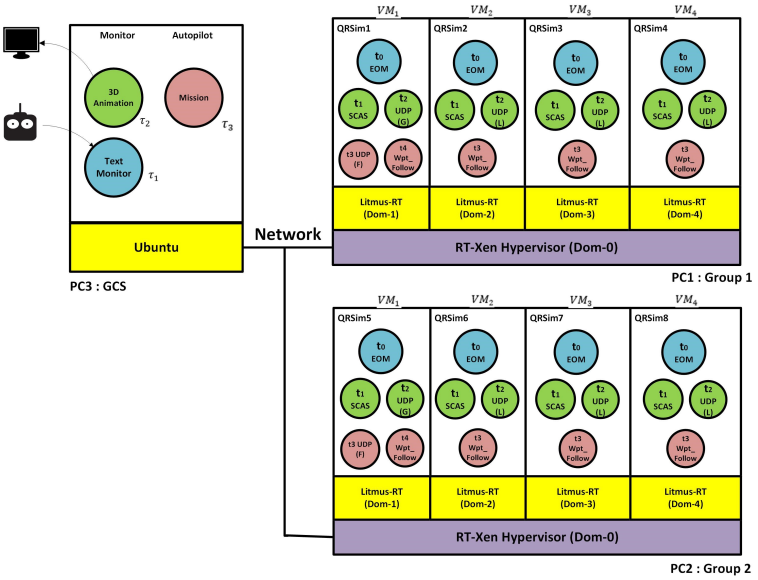


Figure 6. Architecture of Hypervisor-based Quadrotor Simulator.

3.5. Task Configurations

There are five tasks in every quadrotor simulator: EOM, SCAS, UDP, Formation, and Waypoint following. These tasks are computed periodically with various rate configurations. These rates determine how many times the task is computed in one period of time. The higher the rate, the more precise the computation will be. In this paper, we assume the deadline of tasks are equal to the periods. Table 2 shows task configurations of quadrotors.

Table 2. Task configurations.

	QR1	QR2	QR3	QR4	QR5	QR6	QR7	QR8
EOM	100 Hz	100 Hz	100 Hz	100 Hz	100 Hz	100 Hz	100 Hz	100 Hz
SCAS	50 Hz	50 Hz	50 Hz	50 Hz	50 Hz	50 Hz	50 Hz	50 Hz
Wpt_Follow	25 Hz	25 Hz	25 Hz	25 Hz	25 Hz	25 Hz	25 Hz	25 Hz
GCS_UDP	25 Hz	-	-	-	25 Hz	-	-	-
QR1_UDP	-	25 Hz	25 Hz	25 Hz	-	-	-	-
QR2_UDP	25 Hz	-	-	-	-	-	-	-
QR3_UDP	25 Hz	-	-	-	-	-	-	-
QR4_UDP	25 Hz	-	-	-	-	-	-	-
QR5_UDP	-	-	-	-	-	25 Hz	25 Hz	25 Hz
QR6_UDP	-	-	-	-	25 Hz	-	-	-
QR7_UDP	-	-	-	-	25 Hz	-	-	-
QR8_UDP	-	-	-		25 Hz	-	-	-

As shown in the DFD (Figure 3) before, since all quadrotor followers (QR2, QR3, QR4, QR6, QR7 and QR8) are not connected to each other directly, they only make a connection with their group leader (QR1 or QR5). Therefore, all connections between quadrotor followers and GCS, or between groups, are through the quadrotor leaders.

4. Experiments and Performance Evaluation

4.1. System Setup

Although there is no specific requirement for PC used in our quadrotor simulator environment except for storage which needs at least 22 GB for each domain (linux requirement), for experiment purpose in this research, we used two relatively similar PC specifications in terms of number of cores and memory size (as shown in Table 3). In our experiment, number of cores determines the maximum number of quadrotors involved in flight simulation, since each VCPU represents one quadrotor.

Table 3. Hardware specification of PC1 and PC2.

	PC1	**PC2**
No. of Cores	4	4
Memory	4 GB	4 GB
Disk	229.1 GB	297.9 GB

Table 4 shows the virtual machine (domain) specifications and configurations in PC1.

Table 4. The virtual machine specifications and configurations in PC1.

Virtual Machine	**Dom-1 VM**	**Dom-2 VM**	**Dom-3 VM**	**Dom-4 VM**
UAV No.	QRSim1	QRSim2	QRSim3	QRSim4
Virtual CPU No.	1	1	1	1
Virtual RAM (KB)	1024	1024	1024	1024
Virtual HD (GB)	22	22	22	22
RTOS	Litmus-RT	Litmus-RT	Litmus-RT	Litmus-RT
Sched. Policy	Global EDF	Global EDF	Global EDF	Global EDF
UDP IP Address	10.0.0.11	10.0.0.12	10.0.0.13	10.0.0.14
Input Port No.	5001	5002	5003	5004
Output Port No.	4001	4002	4003	4004

Table 5 shows the virtual machine (domain) specification and configurations in PC2.

Table 5. The virtual machine specifications and configurations in PC2.

Virtual Machine	**Dom-1 VM**	**Dom-2 VM**	**Dom-3 VM**	**Dom-4 VM**
UAV No.	QRSim5	QRSim6	QRSim7	QRSim8
Virtual CPU No.	1	1	1	1
Virtual RAM (KB)	1024	1024	1024	1024
Virtual HD (GB)	22	22	22	22
RTOS	Litmus-RT	Litmus-RT	Litmus-RT	Litmus-RT
Sched. Policy	Global EDF	Global EDF	Global EDF	Global EDF
UDP IP Address	10.0.0.15	10.0.0.16	10.0.0.17	10.0.0.18
Input Port No.	5005	5006	5007	5008
Output Port No.	4005	4006	4007	4008

For experiment purposes, the quadrotor model used in our experiment is a simple generic model, and not a specific type/brands of quadrotor model. Table 6 below shows the specifications of our generic quadrotor model. These quadrotor parameters are initialized in EOM process, as seen in Algorithm 2.

Table 6. Generic quadrotor model specifications.

Parameters	Values	Units
Mass (m)	1.0	kg
Thrust coefficient (b)	1.0	Ns^2
Drag coefficient (d)	10.0	Nms^2
x axis inertia (I_x)	1.0	kgm^2
y axis inertia (I_y)	1.0	kgm^2
z axis inertia (I_z)	1.0	kgm^2
z-x axis inertia (I_{zx})	1.0	kgm^2
Arm length (l)	1.25	m

4.2. Implementations

Our multi-quadrotor simulator environment is implemented using C programming language. As mentioned in Figure 6, each PC runs four domains of quadrotors and one domain for Domain-0. Domain-0 is responsible for controlling all domains (guest OSs) and also has a driver for direct access to the hardware [8].

On GCS PC, GUI is divided into two windows: variable monitor and 3D Animations. The Animations is implemented using OpenGL library. For simulation experiment, user can fly either a single quadrotor or multi-quadrotor. In multi-quadrotor scenario, all quadrotors will automatically fly in formation to avoid colliding with each other. Figure 7 shows the UI of GCS.

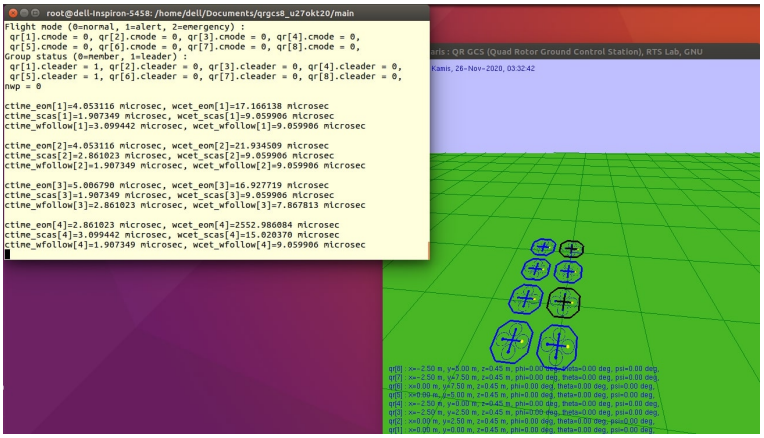


Figure 7. UI of GCS.

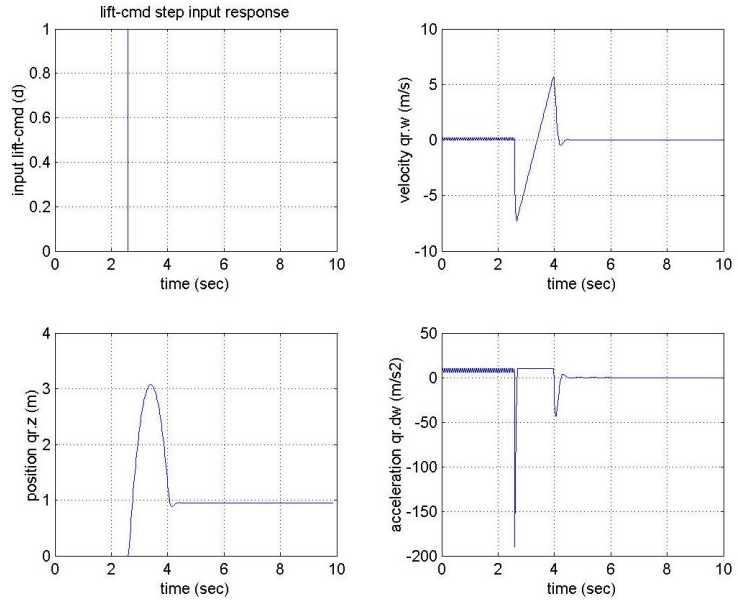
In our quadrotor simulator environment, there are two flight mode options: manual flight and waypoint following. Manual flight is selected if the user wants to fly the quadrotor manually using keyboard input or joystick. Meanwhile, waypoint following flight simulates the quadrotor that follows a particular trajectories/waypoints. However, in current implementation of multi-quadrotor simulation (both manual and waypoint following flight), a group of quadrotors always flies in formation.

For waypoint following mission purpose, coordinates can be inputted either using text file or arguments input in UI (terminal) by user. These waypoint input methods can be done in GCS. There are at least three variables needed to be inputted for waypoint following which is defined as a coordinate: x (position in x-axis), y (position in y-axis), and z (altitude in z-axis).

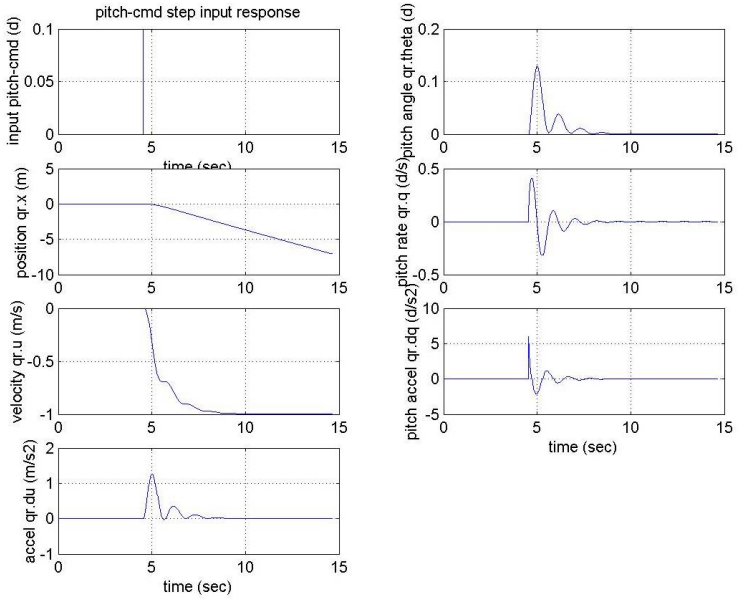
For a formation flight of multi-quadrotor, every quadrotor will get waypoint data that has been added by 2.5 m from the inputted coordinates, so every quadrotor will creates a distance of 2.5 m radius accordingly. Therefore, GCS will sends a command based on these waypoint coordinates to every quadrotor leader of group. This formation flight strategy is important because it makes sure all quadrotors do not collide with each other, or too close to other quadrotors.

4.3. Step Input Response

Step input response is applied for knowing how each quadrotor system responds to a sudden input. This kind of experiment is important because extreme deviation from a steady state may have crucial effects on the quadrotor systems. Therefore, step input response can gives information on the quadrotor stability. There are four kinds of step input that have been applied: lift command, roll command, pitch command, and yaw command. This similar step input response experiments on quadrotor simulator have been published also in our previous paper [19]. The result of step input responses can be seen in Figure 8 below.

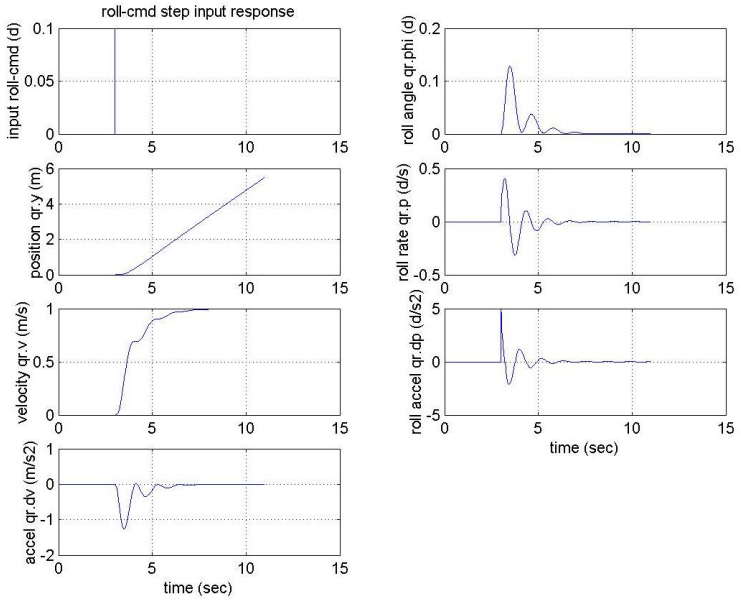


(**a**) lift_cmd

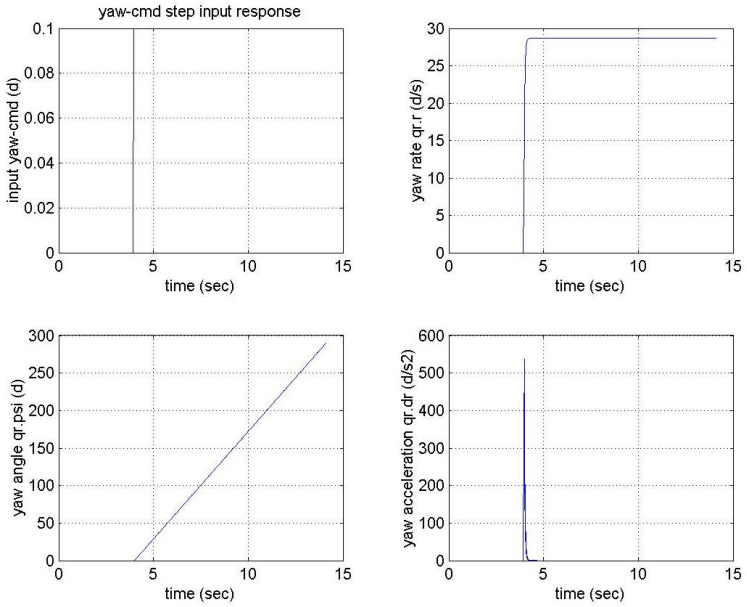


(**b**) pitch_cmd

Figure 8. *Cont.*



(**c**) roll_cmd



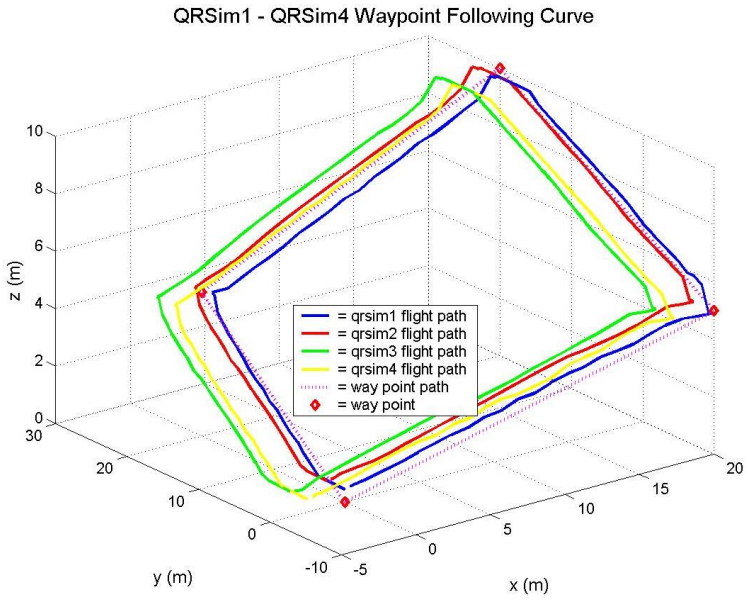
(**d**) yaw_cmd

Figure 8. Step–Input Response.

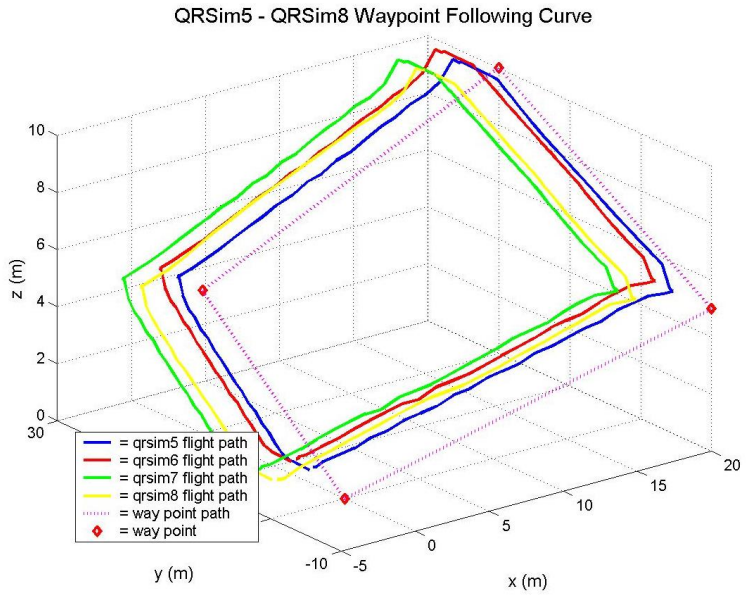
In Figure 8 above, it shows that the quadrotor system is back to the stable state several seconds after the application of a step input. Besides that, positions and angles do not come back to zero after 4 s for step-input of lift_cmd, after 7 s for step-input of pitch_cmd, after 7 s for step-input of roll_cmd, and after 6 s for step-input of yaw_cmd. This means the responses are stable and damped with small value of overshoot, settling time and steady state error.

4.4. Waypoint Following Simulation

For experiment purpose, waypoint following simulation is conducted by multi-quadrotor. The quadrotors fly from a starting point, following other designated waypoints and back to the starting point. Figure 9 shows the example of waypoints following a route by quadrotors group 1 and group 2.



(**a**) Group 1 (QRSim1, QRSim2, QRSim3, QRSim4)

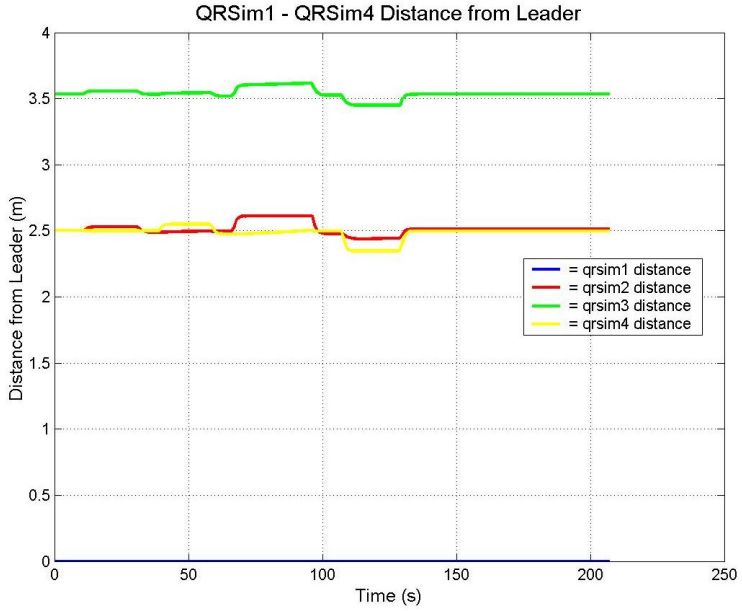


(**b**) Group 2 (QRSim5, QRSim6, QRSim7, QRSim8)

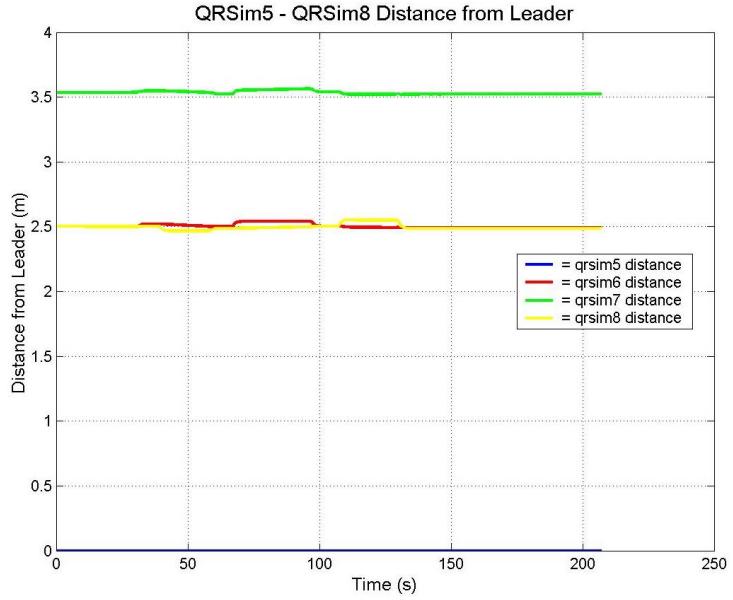
Figure 9. Waypoint following route.

The waypoint simulation has been conducted with five waypoints that were inputted using text file. If waypoints are inputted directly using arguments in terminal, user can simulate another route by inputting another set of waypoints by making the current position of quadrotor as a starting point.

Figure 10 shows the distance between quadrotor followers and their quadrotor leader. Following the initial distance between quadrotors mentioned in Section 4.2, Figure 10 also shows that the positions of quadrotor followers are never too close to the quadrotor leader during the present flight mission. Therefore, it can avoid collision between quadrotors.



(**a**) Group 1 (QRSim1-QRSim4)

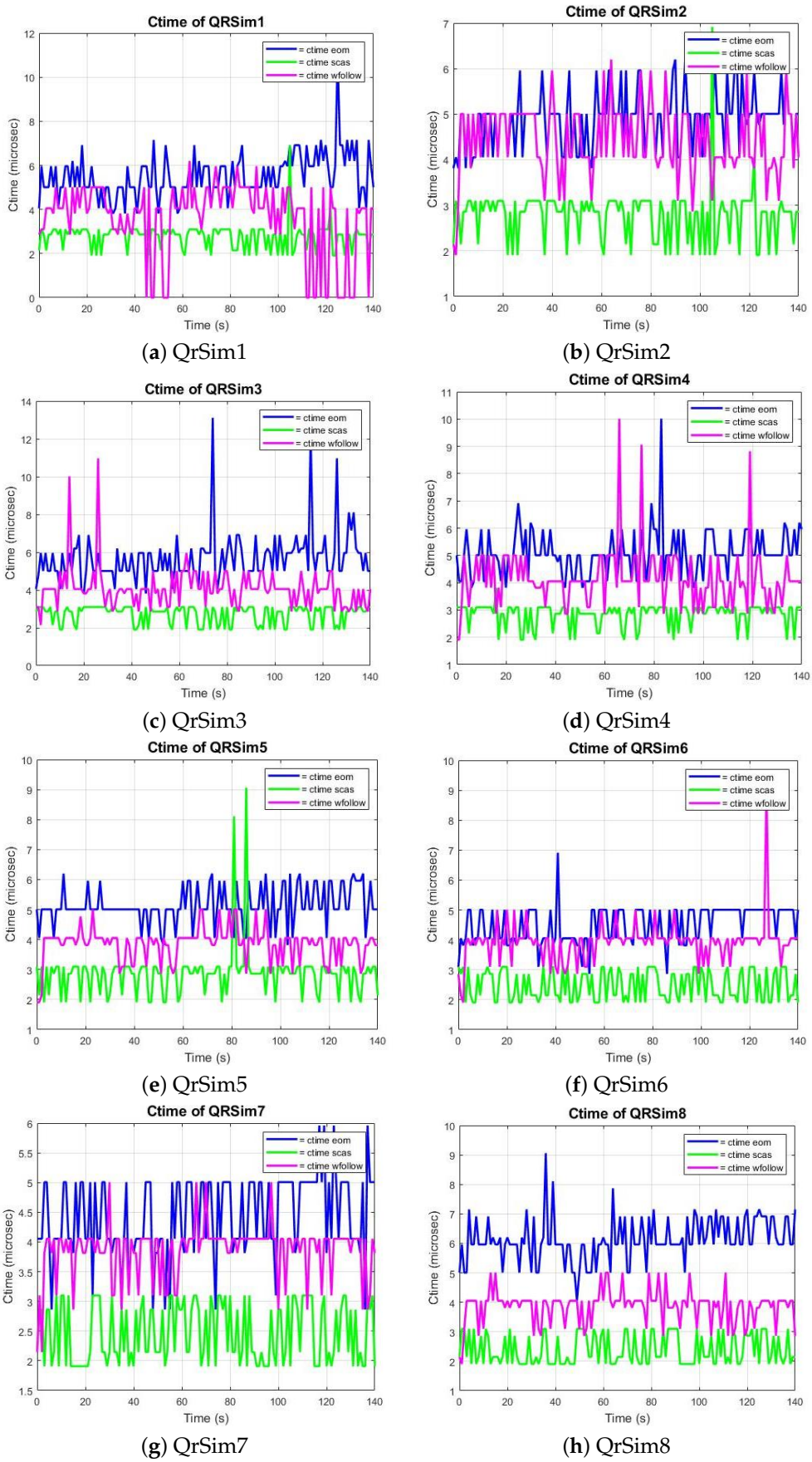


(**b**) Group 2 (QRSim5-QRSim8)

Figure 10. Distance of quadrotor followers from quadrotor leader.

4.5. Computation Time

Computation time is a time it takes to compute a particular task/thread. In our quadrotor simulator, the computation time is recorded by getting the time right before particular task (or thread) is called and after it is finished (right before and after each process of Algorithms 1–3 is called). In this paper, three tasks are recorded: EOM task, SCAS task, and Waypoint Following task, which are the three most important tasks in our quadrotor simulator. Figure 11 below shows the computation time of EOM, SCAS, and waypoint following tasks for every quadrotor during the flight mission shown in Figure 9.



(**a**) QrSim1 (**b**) QrSim2

(**c**) QrSim3 (**d**) QrSim4

(**e**) QrSim5 (**f**) QrSim6

(**g**) QrSim7 (**h**) QrSim8

Figure 11. Computation time of QrSim.

Table 7 shows the average and the maximum of computation times of EOM task, SCAS task and waypoint following task. This table also shows that the computation time of each quadrotor meets the real-time requirements (see the requirement in Table 2).

Table 7. The virtual machine specifications and configurations in PC2.

Quadrotors (QrSim)	Average Computation Time (ms)			Maximum Computation Time (ms)		
	EOM	SCAS	Wayp. Follow.	EOM	SCAS	Wayp. Follow.
QrSim1	0.0055107	0.0027917	0.0036287	0.0109673	0.0069141	0.0061989
QrSim2	0.0049138	0.0027917	0.0044234	0.0061989	0.0069141	0.0061989
QrSim3	0.0056747	0.0027291	0.0040734	0.0131130	0.0030994	0.0109673
QrSim4	0.0050829	0.0028441	0.0041343	0.0100136	0.0038147	0.0100136
QrSim5	0.0051370	0.0028644	0.0038536	0.0061989	0.0090599	0.0050068
QrSim6	0.0045114	0.0024806	0.0039195	0.0069141	0.0030994	0.0090599
QrSim7	0.0044420	0.0024687	0.0038519	0.0059605	0.0030994	0.0050068
QrSim8	0.0061329	0.0023943	0.0038654	0.0090599	0.0030994	0.0050068

4.6. Response Time

The system is considered real-time if the execution of task is completed before the deadline. In general, there are two types of real-time system: soft real-time system, and hard real-time system. Soft real-time system tolerates the deadline miss of task computation. In contrast, hard real-time system does not tolerate any deadline miss. The distinction between these two terms is based on the consequence of deadline miss to the system. Particularly, in hard real-time system, the system must hit every deadline. Any deadline miss occurring in the hard real-time system could be catastrophic. For example, if deadline miss occurs on important tasks in quadrotor system, the quadrotor may crash.

The easiest way to identify whether the system hit or miss the deadline is to check the response time of the task. Response time of task is the time between the task is released and the task is finished. Therefore, with assumptions that the deadline is equal to the period, the system is guaranteed not to miss the deadline as long as the response time is no longer than the period.

In our quadrotor simulator, the response time is recorded using one of feather-trace tools: `st-trace-schedule`. This tool traces all events during simulation, from start to finish. Then, `st-job-stats` tool produces a CSV file of task statistics of tracing result. Table 8 shows the average response time, maximum response time and the appearance of deadline miss on each quadrotor, obtained by feather-trace tools.

Table 8. Response times and the appearance of deadline miss on quadrotor simulators

	Average Response Times (ms)	Maximum Response Times (ms)	Deadline Miss?
QrSim1	0.11657189	7.196699	0
QrSim2	0.12546295	3.921376	0
QrSim3	0.11671483	3.382988	0
QrSim4	0.13197191	3.377784	0
QrSim5	0.12608457	7.499158	0
QrSim6	0.1210677	11.069232	0
QrSim7	0.12375752	6.508314	0
QrSim8	0.1187417	6.888683	0

Based on the task configurations shown in Table 2 before, since it assumes the deadline of tasks are equal to their periods, which are 100 Hz (10 ms) for EOM task, 50 Hz (20 ms) for SCAS task, and 25 Hz (40 ms) for waypoint following task. Therefore, according to Table 7, all tasks hit the shortest period (10 ms). Specifically for QrSim6, the 11.069232 ms of maximum response times is owned by one of QrSim6's tasks which has a period of 40 ms.

5. Conclusions

5.1. Discussions

It is noted that the main purpose of developing a quadrotor simulator is for validation and testing in quadrotor development. The quadrotor simulator also can be used as a testbed for quadrotor related research before real-life experiment. Following these purposes, there are some advantages of our quadrotor simulator environment in this research as follows.

- The proposed model is based on real-time hypervisor, so that it can be used for simulation in heterogeneous computing (multi-platform UAV).
- Our quadrotor simulator is real-time guaranteed because the tasks implemented in our quadrotor simulator are a real-time tasks that run in Litmus-RT.
- Our quadrotor simulator environment also provides a manual flight mode using any type control input such as joystick, keyboard, etc. So it can be used for drone pilot training.
- Our quadrotor simulator also can be further developed to Software-In-The-Loop-Simulation (SILS) and Hardware-In-The-Loop-Simulation (HILS) configuration.

Along with some advantages, our quadrotor simulator environment in this research also has some disadvantages.

- SCAS used in our quadrotor simualtor is Proportional Integratioj (PI) control.
- The quadrotor data used for our quadrotor simulator is generic quadrotor, not a data from real quadrotor.

5.2. Concluding Remark

In this paper, we have presented our multi-quadrotor simulator in hypervisor environment. Our multi-quadrotor simulator offers a new approach to develop a multi-quadrotor simulator that almost resembles the real-life quadrotor by leveraging hypervisor technologies (RT-Xen). By using RT-Xen, we have made a multi-quadrotor simulator environment in which four quadrotors can run independently in one PC with their own computing resources, without compromising real-time principles. To ensure all tasks in every quadrotor simulator (domain) run in real-time, we have developed a quadrotor simulator using Litmus-RT, a real-time extension of Linux. In this paper, we have also developed multi-quadrotor simulation features, waypoint following and flight formation, which prove that our multi-quadrotor simulator environment is suitable to be used as a testbed for any kind of multi-quadrotor applications.

In this paper we have conducted step-input response tests which show there is no problem with stabilization of inner-loop control. In addition, we also have recorded the computation time of three important tasks, which shows there is no problem with the performance of the quadrotor. Furthermore, we also have traced and recorded the response time, which shows that three important tasks in quadrotor system: Stability Controllability Augmented System (SCAS), Equation of Motion (EOM), and waypoint following task, are finished before their deadlines; in fact, 20 ms, 10 ms, and 40 ms before the deadlines for SCAS, EOM, and waypoint following, respectively.

In general, multi-quadrotor simulator is a testbed for quadrotor research and development. Therefore, there are some future work possibilities and focuses based on this paper:

- Further study and research on flight control in group UAV configuration.
- Further study and research on the distance between the different drones of different groups and its relation with aerodynamic interference.
- Using another specific types/brands quadrotor model, especially for Urban Air Mobility implementations.
- Other multi-quadrotor missions and control researches area : collision avoidance, autonomous flight, multi-quadrotor reconnaissance mission, AI-based control.

- A new task model for real-time multi-quadrotor simulator, such as mixed-critical real-time systems and multi-rate task model.
- A new scheduling model or scheduling algorithm for hierarchical real-time systems.

Author Contributions: Conceptualization, M.F.F., Y.K., and K.H.K.; software, M.F.F.; validation, K.-I.K. and S.L.; writing original draft preparation, M.F.F.; writing review and editing, Y.K. and K.H.K.; supervision, S.L.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported partly by the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grant No. NRF-2018R1D1A3A03000717), and by the Human Resources Development of the Korea Institute of Energy Technology Evaluation and Planning (KETEP) grant funded by the Ministry of Trade, Industry and Energy (No.20194030202430).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yang, H.C.; AbouSleiman, R.; Sababha, B.; Gjioni, E.; Korff, D.; Rawashdeh, O. Implementation of an Autonomous Surveillance Quadrotor System. In Proceedings of the AIAA Infotech@Aerospace Conference, Seattle, DC, USA, 6–9 April 2009.
2. Remondino, F.; Barazzetti, L.; Nex, F.; Scaioni, M.; Sarazzi, D. UAV Photogrammetry for Mapping and 3D Modeling-Current Status and Future Perspectives. In Proceedings of the ISPRS—International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Strasbourg, France, 2–6 September 2011; pp. 25–31.
3. Flores, D.A.; Saito, C.; Paredes, J.A.; Trujillano, F. Aerial photography for 3D reconstruction in the Peruvian Highlands through a fixed-wing UAV system. In Proceedings of the 2017 IEEE International Conference on Mechatronics (ICM), Churchill, Australia, 13–15 February 2017; pp. 388–392.
4. Radoglou-Grammatikis, P.; Sarigiannidis, P.; Lagkas, T.; Moscholios, I. A compilation of UAV applications for precision agriculture. *Comput. Netw.* **2020**, *172*, 107148. [CrossRef]
5. Jung, S.; Lee, H.; Hwang, S.; Shim, D.H. Real Time Embedded System Framework for Autonomous Drone Racing using Deep Learning Techniques. In Proceedings of the 2018 AIAA Information Systems-AIAA Infotech @ Aerospace, Kissimmee, FL, USA, 8–12 January 2018.
6. Liu, C.L.; Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* **1973**, *20*, 46–61. [CrossRef]
7. Schmittle, M.; Lukina, A.; Vacek, L.; Das, J.; Buskirk, C.P.; Rees, S.; Sztipanovits, J.; Grosu, R.; Kumar, V. OpenUAV: A UAV Testbed for the CPS and Robotics Community. In Proceedings of the 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), Porto, Portugal, 11–13 April 2018; pp. 130–139.
8. Xi, S.; Wilson, J.; Lu, C.; Gill, C. RT-Xen: Towards real-time hypervisor scheduling in Xen. In Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT), Taipei, Taiwan, 9–14 October 2011; pp. 39–48.
9. Calrino, J.M.; Leontyev, H.; Block, A.; Devi, U.C.; Anderson, J.H. LITMUS-RT: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers. In Proceedings of the 2006 27th IEEE International Real-Time Systems Symposium (RTSS'06), Rio de Janeiro, Brazil, 5–8 December 2006; pp. 111–126.
10. Brandenburg, B.B. Scheduling and Locking in Multiprocessor Real-Time Operating Systems. Ph.D. Thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2011.
11. Fern, H.C.T.E.; De Silva, A.T.A.; De Zoysa, M.D.C.; Dilshan, K.A.D.C.; Munasinghe, S.R. Modelling, simulation and implementation of a quadrotor UAV. In Proceedings of the 2013 IEEE 8th International Conference on Industrial and Information Systems, Peradeniya, Sri Lanka, 17–20 December 2013; pp. 207–212.
12. Akcakoca, M.; Atici, B.M.; Gever, B.; Oguz, S.; Demirezen, U.; Demir, M.; Saldiran, E.; Yuksek, B.; Koyuncu, E.; Yeniceri, R.; et al. A Simulation-Based Development and Verification Architecture for Micro UAV Teams and Swarms. In Proceedings of the AIAA Scitech 2019 Forum, San Diego, CA, USA, 7–11 January 2019.
13. Mutter, F.; Gareis, S.; Schätz, B.; Bayha, A.; Grüneis, F.; Kanis, M.; Koss, D. Model-Driven In-the-Loop Validation: Simulation-Based Testing of UAV Software Using Virtual Environments. In Proceedings of the 2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, Las Vegas, NV, USA, 27–29 April 2011; pp. 269–275.
14. DJI Flight Simualtor. Available online: https://www.dji.com/simulator (accessed on 20 October 2020).
15. Furrer, F.; Burri, M.; Achtelik, M.; Siegwart, R. RotorS—A Modular Gazebo MAV Simulator Framework. In *Robot Operating System (ROS): The Complete Reference*; Koubaa, A., Ed.; Springer International Publishing: Cham, Switzerland, 2016; Volume 1, pp. 595–625.
16. Goktogan, A.H.; Nettleton, E.; Ridley, M.; Sukkarieh, S. Real time Multi-UAV Simulator. In Proceedings of the 2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422), Taipei, Taiwan, 14–19 September 2003; Volume 2, pp. 2720–2726.

17. Kim, D.-M.; Kim, D.; Kim, J.; Kim, N.; Suk, J. Development of near-real-time simulation environment for multiple UAVs. In Proceedings of the 2007 International Conference on Control, Automation and Systems, Seoul, Korea, 17–20 October 2007; pp. 817–820.
18. Rodríguez-Fernández, V.; Menéndez, H.D.; Camacho, D. Design and development of a lightweight multi-UAV simulator. In Proceedings of the 2015 IEEE 2nd International Conference on Cybernetics (CYBCONF), Gdynia, Poland, 24–26 June 2015; pp. 255–260.
19. Fathoni, M.F.; Jo, Y.-I.; Kim, K.H. Prototype Development of the Real-Time Quadrotor UAV Simulation in Litmus-RT. In Proceedings of the Methods and Applications for Modeling and Simulation of Complex Systems, Communications in Computer and Information Science, Singapore, 30 October–1 November 2019; Tan, G., Lehmann, A., Teo, Y., Cai, W., Eds.; Springer: Singapore, 2019; Volume 1094, pp. 260–266.
20. Stevens, B.L.; Lewis, F.L.; Johnson, E.N. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*, 3rd ed.; Wiley-Blackwell: Hoboken, NJ, USA, 2015; pp. 2–3.
21. Fathoni, M.F.; Wuryandari, A.I. Comparison between Euler, Heun, Runge-Kutta and Adams-Bashforth-Moulton integration methods in the particle dynamic simulation. In Proceedings of the 2015 4th International Conference on Interactive Digital Media (ICIDM), Bandung, Indonesia, 1–5 December 2015; pp. 1–7.
22. RT-Xen: Real-Time Virtualization Based on Xen. Available online: https://sites.google.com/site/realtimexen/ (accessed on 7 July 2021).
23. Xi, S.; Xu, M.; Lu, C.; Phan, L.T.X.; Gill, C.; Sokolsky, O.; Lee, I. Real-time multi-core virtual machine scheduling in Xen. In Proceedings of the 2014 International Conference on Embedded Software (EMSOFT), Jaypee Greens, India, 12–17 October 2014; pp. 1–10.
24. Litmus-RT. Available online: http://www.litmus-rt.org/ (accessed on 7 July 2021).
25. LITMUS-RT User-Space Library: Liblitmus. Available online: https://github.com/LITMUS-RT/liblitmus (accessed on 7 July 2021).
26. Brandenburg, B.B.; Anderson, J.H. Feather-Trace: A Light-Weight Event Tracing Toolkit. In Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2007), Pisa, Italy, 8 July 2007; pp. 20–27. Available online: http://www.cse.unsw.edu.au/~kevine/ospert/program/ospert07.pdf (accessed on 7 July 2021).
27. Feather-Trace: A Light-Weight Event Tracing Toolkit. Available online: https://wwwx.cs.unc.edu/~bbb/feathertrace/ (accessed on 7 July 2021).