Network-based Hardware-in-the-loop Simulation for Quadcopter

By

Zihao Zhan

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

August 10, 2018

Nashville, Tennessee

Approved:

Gabor Karsai, Ph.D

Richard Alan Peters, Ph.D

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Gabor Karsai, for the patient guidance and support in this project. When I had problems during the research, Dr. Karsai always provided a lot of useful advice, helped solve the problems and gave me the right direction. Dr. Karsai also helped a lot in completing my thesis by offering many valuable opinions. This project would be much harder to complete without his help.

I would also like to thank Dr. Richard Alan Peters, who was always willing to help me when I went for him. Dr. Peters has been very helpful during my master's program as my academic advisor. His encouragement is always important to me and his review of also helped improve my thesis a lot.

Finally, I would like to thank my friends who assisted in or offered good ideas for my project. Their support really means a lot to me.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Cyber-Physical System

Cyber-physical system (CPS) is a system where computation and physical world are integrated over communication interfaces.[1] Sensors measure properties of physical world and convert them to signals accessible to computing devices. Software deployed in the computing device performs some computations according to the input signals and outputs signals to change states of physical world. Actuators receive control signals and do the corresponding action, causing changes in physical world. The changes are then measured completing the cyber-physical loop. Sensors and actuators serve as interface between physical world and computing device. Fig.1.1 is a typical representation of cyber-physical system. It can be divided into three layers: physical layer, platform layer and communication layer[2].

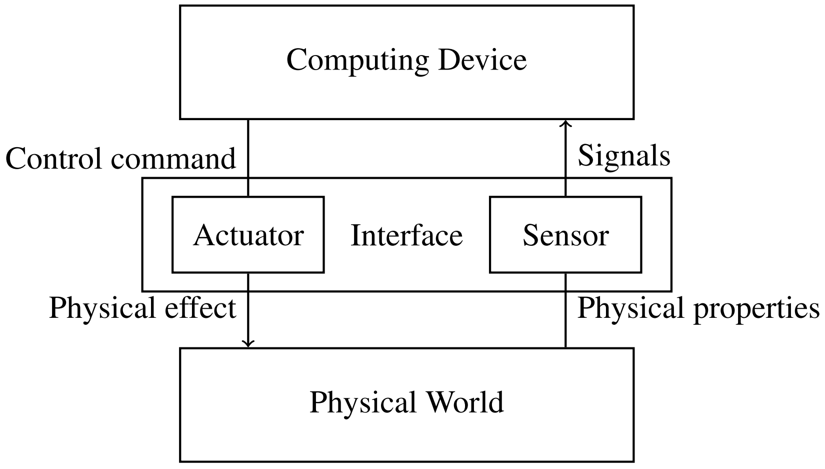


Figure 1.1: Cyber-physical system

A simple CPS example is a home temperature control system. Temperature outside the room may change frequently, a home temperature control system can keep temperature inside room in a steady range which makes people feel comfortable. Thermometers installed in house keeps monitoring the temperature in a house, when room temperature is too low, the controller turns on the heater and the temperature increases. When the temperature is too high, the heater will be turned off and temperature will decrease again. So the room temperature will be constrained in a certain range. There are many other examples of cyber-physical system such as self-driving car, unmanned aerial vehicle and traffic control. Cyber physical systems are widely used to improve people's life quality, promote industrial development and guarantee safety in many areas. So increasing number of research are done on cyber-physical systems nowadays.

Cyber physical systems are sometimes applied in areas where there are strict safety requirement like self-driving car or traffic control. A cyber-physical system is expected to satisfy the requirements of reliability and predictability[1]. It means the system should perform the expected operations, process the correct data and satisfy the specified requirements all the time[3]. Besides, a good cyber-physical system is also expected to be adaptable and cost-effective. It means the system should be adaptive to changed environments or requirements and cost fewer resources. [4]. However, these requirements may conflict each other. In real design of cyber-physical system, the goal is to make the system efficient and adaptable as much as possible under the constraint of reliability and predictability requirements.

1.1.1 Embedded System

An embedded system is a computer system that's designed for a specific purpose of mechanical and electrical devices and embedded in the device[5]. Usually it is the microprocessor-based hardware that performs computation tasks in a cyber-physical system. Thus computation in embedded system is required to be in real-time. It means when an embedded system receives a sensor signal, it should do the processing quickly and guarantee that the respond is sent within specific time constraints.

Embedded systems are usually dedicated to specific applications, so they can be optimized so that the embedded system can cost much less resources than general-purpose computers like personal computers. For example, by considering specific cases, the code size and run time for the embedded software is smaller, less energy will be consumed, a lighter system can satisfy all requirements and the developing cost would also be lower as a result.

Because embedded system is very sensitive to time, it is always an important part to do timing analysis during the development related to embedded system. It usually includes analysis of execution time and communication time.

1.1.2 Model Based Design

Moldel-based design (MBD) is a widely used technique for developing embedded softwares. It can describe complex systems designing problems in a mathematical and graphical way[6]. Model-based design typically consists of following steps:[7][8]

1. **Model physical plant.** Observe and analyze the physical plants, derive the dynamic equations. According to observed features and derived equations, build a mathematical model, which is a simplified representation of the real physical plant.

2. **Design controller algorithm.** Analyze the problem, list parameters and variables of plant model. Specify requirements with these parameters and variables. Design a suitable control algorithm for this plant to satisfy the requirements.

3. **Select a model of computation.** A model of computation defines the semantics for simulation of the system. Models define by a formal model of computation is more analyzable.

4. **Simulation.** Depending on the characteristic of simulated system, select a suitable desktop simulation. Run the plant and controller in simulation and observe their behavior.

5. **Test,verify and validate.** Deploy the controller to a selected hardware. Translate

requirements into formal specifications for verification and validation. Verify these specifications during testing.

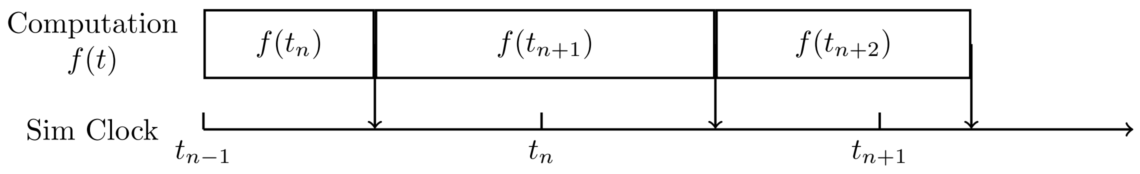
Model-based design has many advantages. A model can describe the design problem in a more clear and straightforward way, which makes the design more understandable. Using MBD approach, problems in design can be discovered earlier. Fixing bugs in the model is much easier and cheaper than on real device, thus the development cost is reduced and the reliability is increased. MDB can provide identical, clearly described models, so multiple engineers can do the developing and analysis concurrently, which greatly reduce the development cycle. What's more, repeatable experiment results can be provided by MDB. Since crashing in simulation won't cause serious effect, MDB approach can also be used for some dangerous testing, which can help develop products with high safety requirement[9][8].

1.2 Real-time Simulation

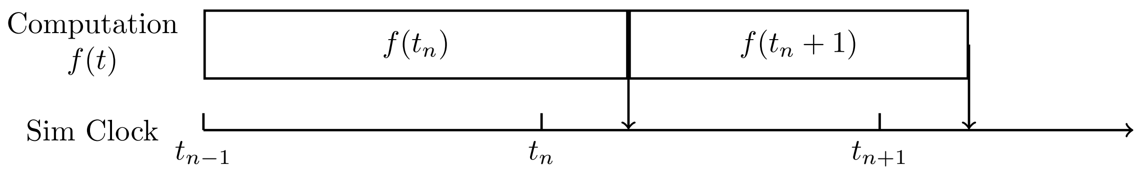
Simulation is a technique that imitates the behavior and performance of a system through mathematical computations. It's widely used for designing and analyzing electrical or mechanical systems.

In a simulation, states of a simulated system are updated successively through solving dynamic equations. Depending on whether the time steps change, simulations can be categorized into fixed time-step simulation and variable time-step simulation. Real-time simulation is typically a fixed time-step simulation designed so the simulated plant will update internal variables and outputs within the same length of time as its physical counterpart would.

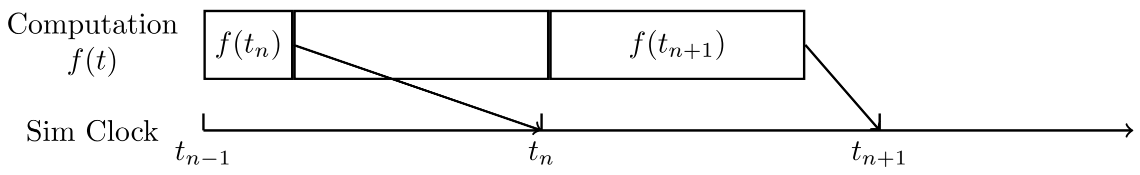
Actual time needed for complete computation in one step doesn't have to be the same as the logical time-step. Sometimes it is shorter and sometimes it is longer. In Fig.1.2a computation takes shorter time than a time-step and in Fig.1.2b computation takes longer time. They are all offline simulations. The computation results become available as soon



(a) Offline Simulation: Faster than real-time



(b) Offline Simulation: Slower than real-time



(c) Real-time simulation

Figure 1.2: Real-time simulation and other simulations

as they finish their computation, so the exact moment is irrelevant to simulation time-step.

In a real-time simulation in Fig.1.2c, the simulator update its internal variables and outputs in a given time-step. When the actual computation is finished before next time-step, simulator doesn't proceed to next computation. Instead, it waits until next time step's clock tick. If the computation time exceeds the given time-step, the simulator is erroneous and this situation is called as overrun.

In this way, the real-time simulator accurately runs the simulation in the same speed as its physical counterparts. Thus it is expected to have the same performance as the real physical part. Execution of real-time simulation in one time-step consists of following 4 steps:

1. Read inputs and write outputs

2. Solve equations for the model

3. Exchange results between different nodes in simulation

4. Wait until the next round's execution starts

1.2.1 Sotware-in-the-loop Simulation

Software-in-the-loop (SIL) simulation is a technique to run the whole system, including plant model and controller, in the software environment. In SIL simulation, both controller and plant are implemented in the same environment with same time-steps. The simulation can be completely isolated from physical world, so the simulation time doesn't have to be same as time in real world . Faster simulation allows more test to be run in a short time periods and shorten the developing time. Slower simulation allows more computation to be done in an execution, so more complicated systems can be simulated.

1.2.2 Hardware-in-the-loop Simulation

Contrast to SIL simulation, Hardware-in-the-loop (HIL) simulation is a kind of real-time simulation where the plant model is run by real-time simulator and the controller is implemented in a physical embedded system. These two part interact with each other through I/O interfaces. Fig.1.3 shows a typical example of HIL simulation.

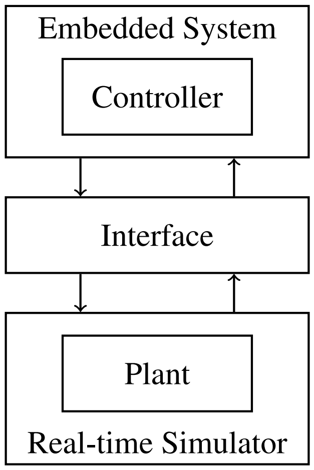


Figure 1.3: Example of Hardware-in-the-loop simulation

HIL simulation is a cost effective technique used for system-level test of the design of embedded systems.[10]. In SIL simulation, plants and controllers are usually simplified due to limited resources. Therefore it can't simulate the real hardware environment of an embedded system. But HIL simulation runs the controller in a real embedded system in physical world, so the system is tested in a more realistic environment. Using HIL simulations, engineers are able to perform optimization and debugging on the hardware level. HIL simulation can also be used to do perform tests that would be dangerous or expensive in the real world so that a more reliable final products can be developed with a relatively low cost.

An HIL simulation has to sufficiently represent system in order to provide useful results. The real-time requirements mentioned in Section1.2 need to be satisfied. So the simulation computer and embedded system should support real-time operation and high data transfer rate. The selection of I/O interface is also important. Interfaces may influence data integrity or latency in simulation and result in different results.

Multiple methods can be used to analyze an HIL simulation's performance. We can compare simulated results with actual experiment results to see how similar the performances are. What's more, a timing analysis can be done to measure the latency and its effect in simulation.

1.3 Packet-Switched Network

Packet switching is an approach to transmit data that is grouped into packets through a network with links and switches. A packet consists of a header and a payload. A header contains information for controlling or directing the packet, e.g. address, protocol. The payload contains data to be transmitted. A packet needs travel through packet switches and links from its source to its destination and it causes network delay.

In packet-switched networks, the delay consists of 4 parts, processing delay, queuing delay, transmission delay and propagation delay. Processing delay is the time used for processing the packet, e.g. routing. Transmission is the time between the start and finishing of a transmission, it can be represented by L/R where L is packet length and R is transmission rate. Propagation delay is the time a packet spend on physical medium (e.g. fiber optics). The queuing delay is the time a packet waits in buffer.[11]. Generally, processing delay, transmission delay and propagation delay are on the order of microseconds to milliseconds. Queuing delay can be as small as order of milliseconds but it could be very long if many packets wait in a queue.

1.4 Quadcopter

Quadcopter is an aerial vehicle driven by four rotors installed at four ends of a cross frame. Two rotors at the two ends of a same arm rotate clockwise and the other tow rotate counterclockwise. Through changing four rotors' speed, a user can control the quadcopter to rise, fall and fly toward certain direction. Quadcopter is one of the simplest aircrafts which can be described by simple dynamic equations because of its symmetry[12]. Despite its simple structure, a quadcopter can perform many complex actions. The good performance and low cost make quadcopters a popular consumer and industrial product. They have been widely used for photography, express delivery, environmental monitoring. And the simplicity of quadcopter also makes it an ideal object for research.

1.5 Thesis Objectives

The goal of this thesis is to apply hardware-in-the-loop simulation technique to the design of a quadcopter control system. The quadcopter model is built and simulated in Simulink, the controller is deployed in BeagleBone Black (BBB) and they interact with each other through network. To be more specific, user datagram protocol (UDP) is used for communication.

First a software quadcopter model is built and a controller is designed for this model. After finishing software-in-the-loop simulation, the controller is deployed onto BBB and perform HIL simulation. The performance of HIL simulation using network for communication is what this thesis focuses on. Main goals of this thesis are:

1. Building quadcopter model and controller

2. Performing HIL simulation for quadcopter system

3. Measuring and analyzing latency and its effect in HIL simulation

4. Modeling the latency, doing timing analysis for network-based HIL simulation

Chapter 2

Related Work

2.1 Implementation and Analysis of HIL Simulation

2.1.1 Real-time Simulators

The earliest simulators are analog simulators developed for military purposes. Analog simulators used electronic DC amplifiers to perform varieties of computations such as integration, multiplication, division, etc. By interconnecting these computation units, a system obeying given differential equations can be built. So analog simulators are usually very large, hard to build and maintain and can only be used to simulate specific systems[13]. This is later replaced by digital simulation due to the development of microprocessor and digital signal processing (DSP) technologies[14]. RTDS developed by RTDS Technologies Inc is known as the first commercialized real-time digital simulator [15]. Both analog and digital parts were used to perform real-time simulation for power systems. Today it is one of the most widely used real-time simulator for industry, education and research. The first fully digital real-time simulator is ARENE introduced in 1996 [16]. It can simulate high frequency phenomenon in standard multipurpose parallel computers. Later, more general-purpose processor-based simulators are introduced by OPAL-RT Technologies Inc and dSpace using MATLAB/Simulink as the main model tool for simulation [17][18].

Simulink is a graphical programming environment integrated with MATLAB and developed by MathWorks for model-based design and multidomain simulation.[19] It provides block libraries and solvers for different purposes of simulation. The graphical environment and integration with MATLAB simplify the modeling and analysis process. Simulink also supports code generation so that C code can be automatically generated and deployed onto embedded system. All these properties make Simulink a powerful and user-friendly software for real-time simulation in many areas.

2.1.2 HIL Simulation Analysis

Hardware-in-the-loop simulation is used to evaluate and test the design of an embedded system. It's a necessary step during product development to increase the reliability of a system. In HIL simulation, multiple factors can influence the performance and produce erroneous results. Uncertainty in HIL simulation can be attributed to two sources, one is the difference between simulated model and real physical plant, the other is caused by the interface between two parts.

To evaluate HIL simulation's performance, Bacic proposed two standards, ”transparency” and ”robustness of prediction” to be used in [20]. ”Transparency” means the interface between two parts introduce no difference between simulation and real system. ”Robustness of prediction” defines how accurate the real response is predicted in the simulation. These two standards are measured by comparing the boundary conditions. The goal of an HIL simulation is to maximize transparency and minimize error of prediction.

Ren et al. studied how different interfaces in power system HIL simulation can affect stability and accuracy in [21]. In [22] Krenn et al. tested HIL simulated robotics operations and found that higher sampling rate and smaller delay can increase simulation stability. Ayasun et al. evaluate the stability of a simulation-stimulation interface whose parameters are sampling rate and delay time in [23], finding that time delay is the main concern for stability of power hardware-in-the-loop simulation.

Reference [24] analyzed the time delay effect in HIL simulation for PMSM drive system using root-locus method. Results show that latency greatly influences inaccuracy and instability. And a method to compensate the latency is proposed by use of an inverse relationship.

Later part of this thesis will focus on the delay time caused by network communication, relation between delay and sampling rate and the corresponding influence on simulation performance.

2.2 Networked Control

A networked control system is a closed-loop control system where the feedback is provided through a network[25]. The interface used for HIL simulation in this thesis is the network, thus it can be seen as a networked control system and analyzed in a similar way. Network could cause following problems in HIL:[25][26]

1. Network delay will be introduced to the control loop.
2. Packet loss will occur and influence the simulation.
3. No universal clocks are available to synchronize controller in physical system and model plant in real-time simulation.

As stated in section2.1.2, delay in HIL simulation will cause instability and inaccuracy. Packet loss and asynchronization also introduce uncertainty to simulation. To improve the reliability of simulation, delay need to be decreased or compensated. Yu et al. studied how packet dropout and delay time could affect the stability of a networked control system in [27]. The results shows that packet dropout won't cause instability. On the contrary, dropping old packets and accepting new packets can guarantee shorter delay time and the stability can be improved as a consequence.

Due to network delay's great impact on network-based HIL simulation, in this thesis the delay will be analyzed and measured. There are many ways to do timing analysis. RTSSim [28] is a simulation framework that can perform simulation-based analysis for timing. In reference [29], Bohlin et al. proposed a Hill Climbing with Random Restart (RRHC) algorithm for doing best-effort response-time analysis.

2.3 Quadcopter Model and Control

2.3.1 Model Description

Luukkonen provided a mathematical model for quadcopter in[30]. A quadcopter with 6 degrees of freedom(DOF) is defined in Fig.2.1.

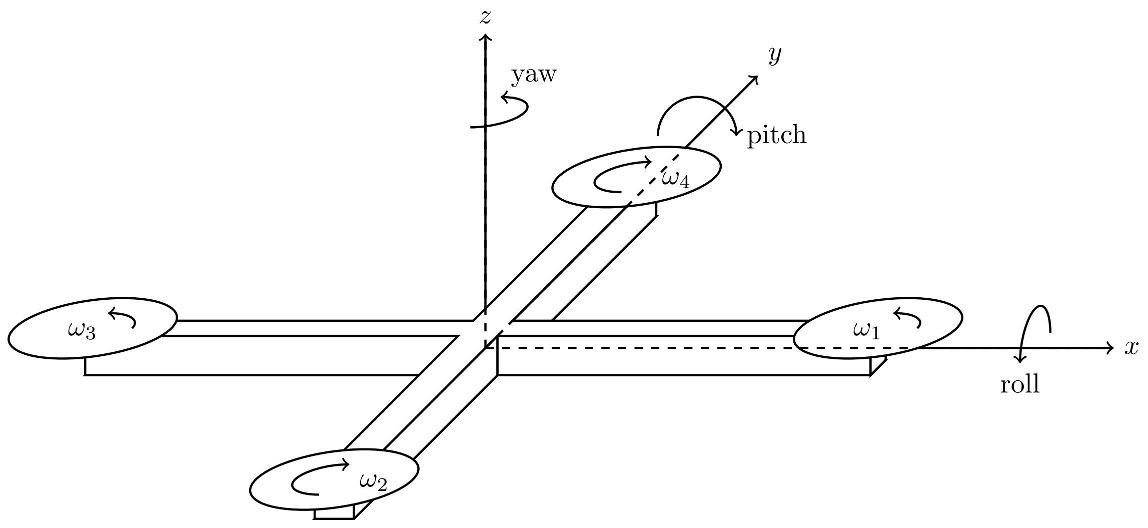


Figure 2.1: Quadcopter Model

In inertial frame linear position is described by x, y, z axes as ξ and its angular position is described by Euler angle η. In body frame, its translational velocity is V_B and the angular velocity is ν.

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \quad V_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}, \quad \nu = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.1}$$

Roll angle ϕ is the rotation around axis x. Pitch angle θ is the rotation around axis y. Yaw angle ψ is the rotation around axis z.

The rotational matrix from body frame to inertial frame is:

$$R = \begin{bmatrix} cos\psi cos\phi & cos\psi sin\theta sin\phi - sin\psi cos\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ sin\psi cos\phi & sin\psi sin\theta sin\phi + cos\psi cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi \\ -sin\theta & cos\theta sin\phi & cos\theta cos\phi \end{bmatrix} \tag{2.2}$$

Translation matrix W_η is defined as:

$$\nu = W_\eta \dot{\eta}, \quad W_\eta = \begin{bmatrix} 1 & 0 & -sin\theta \\ 0 & cos\phi & cos\theta sin\phi \\ 0 & -sin\phi & cos\theta cos\phi \end{bmatrix} \tag{2.3}$$

Assume the quadcopter is symmetric with mass m, it is inertial matrix is:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.4}$$

Thrust T^B and torque τ_B in body frame can be represented by four rotors' angular velocity:

$$T^B = \begin{bmatrix} 0 \\ 0 \\ k\sum_{i=1}^{4} \omega_i^2 \end{bmatrix}, \tau_B = \begin{bmatrix} lk(-\omega_2^2 + \omega_4^2) \\ lk(-\omega_1^2 + \omega_3^2) \\ \sum_{i=1}^{4} \tau_{M_i} \end{bmatrix}, \tau_{M_i} = b\omega_i^2 + I_M \dot{\omega}_i \tag{2.5}$$

Where k is lift constant, b is drag constant, I_M is the inertial moment of rotor and l is distance between two arms.

The dynamic of a quadcopter is described by following equations:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \begin{bmatrix} cos\psi sin\theta cos\phi + sin\psi sin\phi \\ sin\psi sin\theta cos\phi - cos\psi sin\phi \\ cos\theta cos\phi \end{bmatrix} \tag{2.6}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} (I_{yy} - I_{zz})qr/I_{xx} \\ (I_{zz} - I_{xx})pr/I_{yy} \\ (I_{xx} - I_{yy})pq/I_{zz} \end{bmatrix} - I_r \begin{bmatrix} q/I_{xx} \\ -p/I_{yy} \\ 0 \end{bmatrix} (\omega_1 - \omega_2 + \omega_3 - \omega_4) + \begin{bmatrix} \tau_\phi/I_{xx} \\ \tau_\theta/I_{yy} \\ \tau_\psi/I_{zz} \end{bmatrix} \tag{2.7}$$

$$\ddot{\eta} = \begin{bmatrix} 0 & \dot{\phi} cos\phi T_\theta + \dot{\theta} sin\phi/cos^2\theta & -\dot{\phi} sin\phi cos\theta + \dot{\theta} cos\phi/cos^2\theta \\ 0 & -\dot{\phi} sin\phi & -\dot{\phi} cos\phi \\ 0 & \dot{\phi} cos\phi/cos\theta + \dot{\phi} sin\phi T\theta/cos\theta & -\dot{\phi} sin\phi/cos\theta + \dot{\theta} cos\phi T\theta/cos\theta \end{bmatrix} \tag{2.8}$$

In this thesis, the quadcopter model used for test is built based on equations 2.6-2.8.

2.3.2 Controller Design

Many algorithms are used for controlling quadcopters. Proportional-integral-derivative (PID) controller[31] is a simple but effective for feedback control loop. PID control can be expressed by Eq.2.9. K_p, K_i, K_d are coefficients for proportional, integral and derivative terms respectively. Output $u(t)$ is the desired control value, input $e(t)$ is the error between the reference value and measured process value.

$$u(t) = K_\text{p} e(t) + K_\text{i} \int_0^t e(t')dt' + K_\text{d} \frac{de(t)}{dt} \tag{2.9}$$

Back-stepping[32] is a method used for stabilizing non-linear dynamic systems. Using back-stepping approach one can solve the stabilization problem of a large system through solving the problem for subsystems.

H_∞[33] is another technique used in the controller for stabilizing the system. A system can be represnted by the following formula:

$$\begin{bmatrix} z \\ v \end{bmatrix} = \mathbf{P}(s) \begin{bmatrix} w \\ u \end{bmatrix} = \begin{bmatrix} P_{11}(s) & P_{12}(s) \\ P_{11}(s) & P_{12}(s) \end{bmatrix} \begin{bmatrix} w \\ u \end{bmatrix}, \quad u = \mathbf{K}(s)v \tag{2.10}$$

Where z is the error signal, v is measured variable, w is exogenous input and u is manipulated variable.

Dependency of z on w can be expressed as

$$z = F_l(\mathbf{P},\mathbf{K})w, \quad F_l(\mathbf{P},\mathbf{K}) = P_{11} + P_{12}\mathbf{K}(I - P_{22}\mathbf{K})^{-1}P_{21} \tag{2.11}$$

The objective of H$_\infty$ controller is to find the controller that minimize $F_l(P,K)$ according to the H$_\infty$ norm. Kalman filter[34] is a technique used to estimate quadcopter's states. It can estimate a variable based on measured results an predicted results considering their uncertainty. In this thesis, the quadcopter controller contains a Kalman filter and uses simple PID controllers to control the quadcopter model.

Chapter 3

Method

3.1 Quadcopter System Description

In this thesis, the quadcopter is defined by dynamics described in section 2.3.1. On the basis of the simple quadcopter model, an inertial measurement unit (IMU) is added to it. The quadcopter model output four sensor signals, altitude and yaw are values obtained directly from model, acceleration and angular velocity come from IMU, they are biased noisy signals, with which roll and pitch can be computed. Fig.3.1 shows the the whole quadcopter system.

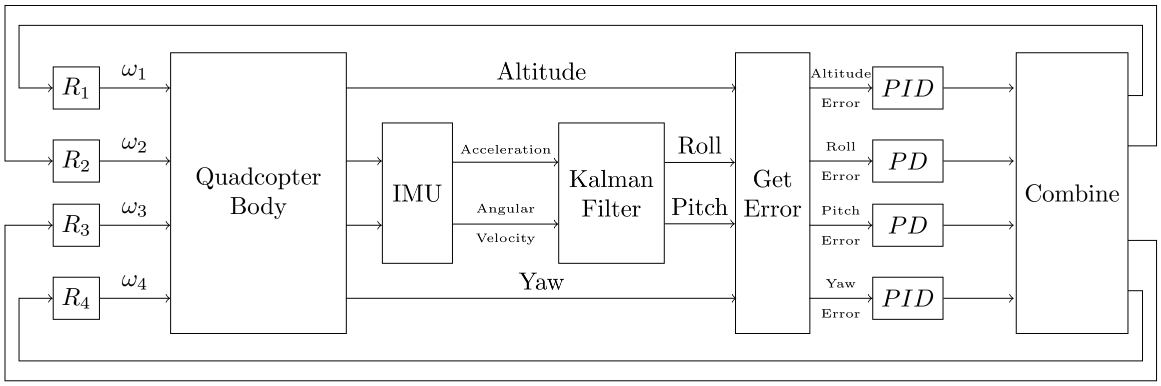


Figure 3.1: Quadcopter system model

In the controller, a Kalman filter is used to process acceleration and angular velocity obtained from IMU and estimate the roll and pitch. Then roll and pitch signals together with altitude and yaw are compared with reference signals and "Get Error" blocks compute the errors of each signal. These errors are input of PID controllers. 2 PID controllers are used to control altitude and yaw, 2 PD controllers are used to control roll and pitch. After the control commands are computed for altitude, roll, pitch and yaw, commands for controlling four rotors should be computed by combining commands for these variables.

The ”Combine” block can be expressed by following equation:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_{\text{Altitude}} \\ C_{\text{Roll}} \\ C_{\text{Pitch}} \\ C_{\text{Yaw}} \end{bmatrix} \tag{3.1}$$

ω_i represents the ith rotor’s speed, $C_{\text{Altitude}}, C_{\text{Roll}} C_{\text{Pitch}} C_{\text{Yaw}}$ represent the commands for altitude, roll, pitch and yaw respectively. So the controller processed sensor signals from quadcopter and output commands for four rotors. Four rotors in the quadcopter model will then influence the quadcopter body’s states, sensor signals will change too.

The model is built in Simulink using Simscape library. The model is run by Simulink real-time desktop to perform HIL simulation. The controller is designed in Simulink and deployed to BeagleBone Black using code generator. UDP is used for communication between them. A round counter is used in the model, which is increased by 1 at each round and sent out with sensor signals. In controller, the round number is output with the control commands. In this way, when a set of control commands is received, it can be matched with the sensor signals sent out. The difference between their timestamps is the total delay in this system.

3.2 Assumption of Simulation Delay

One of the main objectives of this thesis is to analyze delay time in HIL simulation. First we are going to analyze the whole system and make some assumptions of where the delays come from. Then a model of the delays will be built based on the assumption. The simulation result will be compared with the actual measured results to verify this assumption. In this section, the assumption of delay in HIL simulation is described.

In an HIL simulation, two types of actions can cost time. The first type is execution time of either real-time simulator or in controller, the other is network delay. As stated in section 1.3, network delay in packet-switching system consists of four parts: processing delay, queuing delay, transmission delay and propagation delay. In this case, the processing delay can be counted as part of execution time of model and controller. Transmission delay and propagation delay together can be called delivery time. What remains is queuing delay. In this simulation, we are interested in the time elapsed between a packet is sent out from the real-time simulated plant and the corresponding packet returns to plant. This time is called round trip time. Thus round trip time T_r, execution time T_e, deliver time T_d and queuing delay D_q satisfy following equation:

$$T_r = T_e + T_d + D_q \tag{3.2}$$

The simulated model and controller in BBB perform 3 operations, input, processing and output. Assume input and output operations cost no time but they can change the buffer state in PC and BBB. Now let's see the whole procedure of how is a sensor signal sent out and how the corresponding control command comes back.

1. First, a packet is sent out from simulated model, it takes some time to be delivered to BBB and this packet enters the buffer in BBB.

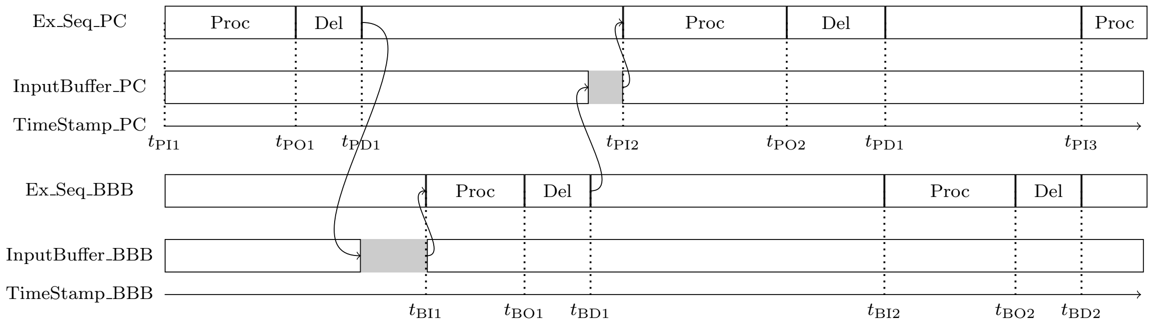
2. If all packets arrived before this one have been fetched, in the next round's execution of controller, this packet will be fetched and the data will be processed by controller. Otherwise this packet has to wait in the buffer.

3. After the processing finishes, a packet contains control command is sent out by BBB and arrives at buffer in PC after some delivery time.

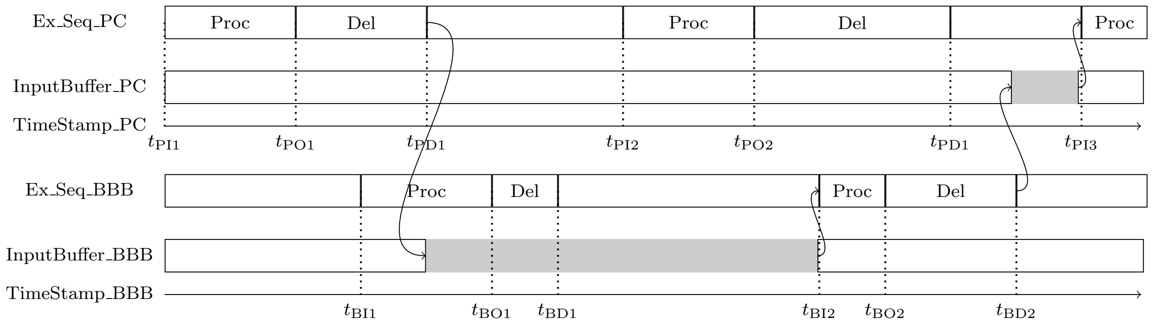
4. Similarly, this packet waits in buffer until all packets arrive before it are fetched. Finally, this packet is fetched and the command is received by actuators in the model.

The total time elapsed in above steps is the round trip time T_r. In the following part, we will see all possibilities the round trip time could be under above assumptions.

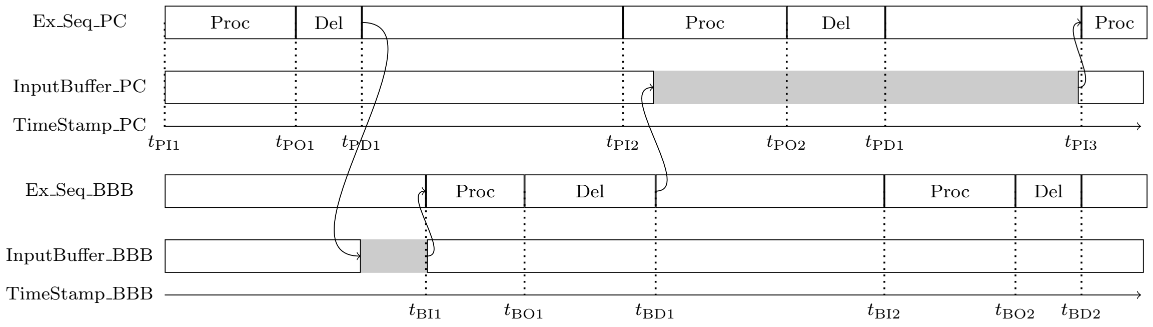
If all buffers are cleared at the beginning, we can draw three possible execution sequences in Fig.3.2, from which we track a packet from when it is sent out from plant to the moment it returns to the plant (Note: We don't consider the output buffer because packets are almost fetched immediately after entering output buffer).



(a) Execution of controller and simulator are separated



(b) Execution of controller and simulator overlap with each other



(c) Execution of controller and simulator overlap with each other

Figure 3.2: Possible Execution Sequence In PC and BBB

In Fig.3.2, Ex_Seq_PC and Ex_Seq_BBB represent the execution of plant and controller respectively. InputBuffer_PC and InputBuffer_BBB are used to record whether the tracked packet is in input buffer of PC or BBB (Black block means the packet is in buffer). We use time stamps with subscripts in the form of (XYZ) to mark the time. X=P means the time stamp is about PC, X=B means the time stamp is about to BBB. Y= I, O, D represent input, output and end of delivery respectively. Z represents the relative round of execution. The intervals between two executions are same, which equal to sample time e.g. $t_{\mathrm{PI3}} - t_{\mathrm{PI2}} = t_{\mathrm{PI2}} - t_{\mathrm{PI1}}$. But the processing time and delivery time may change e.g. $t_{\mathrm{PO2}} - t_{\mathrm{PI2}} \neq t_{\mathrm{PO1}} - t_{\mathrm{PI1}}$.

If all buffers are empty at the beginning, from Fig.3.2 we can see that the roundtrip time can be 1 or 2 times of the sample time depending on the relative execution time and delivery time of two processes in controller and simulator. The shortest possible delay is one sample time.

The situation when all buffers are empty is discussed above. But actually it is not possible because it is hard to start the plant and controller at the exact same time. Whatever starts first, it will fill the other's input buffer. Next let's see the situations where buffers are not empty. Considering similar situations as Fig.3.2, the difference is at time t_{PI1}, 1 packets are stored in BBB buffer.

There could be two situations depending on the buffer size. Let L be the maximum number of packets the buffer can store.

1. $L > 1$, then this packet will enter input buffer, waiting to be fetched and processed. In this case, Fig.3.2a, Fig.3.2b and Fig.3.2c will have similar performances at first round. At time t_{BI1}, previous packet will be fetched and processed. This packet will be processed at time t_{BI2}. Roundtrip time depends on the value of $t_{\mathrm{BD2}} - t_{\mathrm{PD2}}$.

2. $L = 1$, then for situation in Fig.3.2b, the previous packet will be fetched, new packet will enter buffer and the performance will be same. However, for situation in Fig.3.2a and Fig.3.2c, where $t_{\mathrm{BI1}} > t_{\mathrm{PD1}}$, this packet will be dropped. In the next round, it will become a situation with empty buffer, same as what was discussed before.

   (a) $t_{\mathrm{BD2}} - t_{\mathrm{PD2}} < 0$

It means the packet is delivered to PC before its third round execution starts. Then the packet will be processed in third round, then the roundtrip time is two sample time.

(b) $t_{\text{BD2}} - t_{\text{PD2}} > 0$

It means the packet is delivered to PC after its third round execution starts. Then simulation in the PC will use data obtained in last round. This packet will be processed in forth round. Then the roundtrip time is three sample time.

We now expand this problem to the situation where there are n packets in BBB's buffer at time t_{PI1}. It is similar the situation where $n = 1$ and also depends on whether there is packet dropping. If the packet it not dropped, then roundtrip time is at least $(n+1)(Sampletime)$. If $t_{\text{PIn}} + 2 < t_{\text{BDn}} + 1$, the roundtrip time is $(n+2)(Sampletime)$. If this packet is dropped, then consider next packet's roundtrip time in the situation where $(n-1)$ packets are in buffer.

Above relations between roundtrip time T_r and packet number n in buffer can be summarized by following equation:

$$T_r(n) = \begin{cases} 1 \times T_s & \text{if } n = 0 \text{ and } t_{\text{PD1}} < t_{\text{BI1}} \text{ and } t_{\text{BD1}} < t_{\text{PI2}} \\ 2 \times T_s & \text{if } n = 0 \text{ and otherwise} \\ (n+1) \times T_s & \text{if } n > 0 \text{ and } n < L \text{ and } t_{\text{BDn+1}} < t_{\text{PIn+2}} \\ (n+2) \times T_s & \text{if } n > 0 \text{ and } n < L \text{ and } t_{\text{BDn+1}} > t_{\text{PIn+2}} \\ T_r(n-1) & n = L \end{cases} \tag{3.3}$$

T_s is sample time. Above equation is correct when consider only input buffer in BBB and assume that every time a packet is delivered from BBB to PC, it enters an empty buffer in PC.

From Eq.3.3 we can see that normally the roundtrip time is $(n+1)T_s$, when a process try to access data in an empty buffer, the total roundtrip time will be increased by 1. When a packet tries to enter a full buffer, the packet will be dropped and roundtrip time will be decreased by 1 starting from next round. Now we can consider a general case, assume at time $t_{\mathrm{PI}1}$, simulation in PC starts execution. n_1 packets are in PC's input buffer, whose size is L_1. n_2 packets are in BBB's input buffer, whose size is L_2. During the whole trip, n_d packets are dropped. Processes try to read data from empty buffer n_e times. Then the round trip time can be written as:

$$T_r(n_1, n_1, n_d, n_e) = (n_1 + n_2 + n_d - n_e + 1) \times T_s \tag{3.4}$$

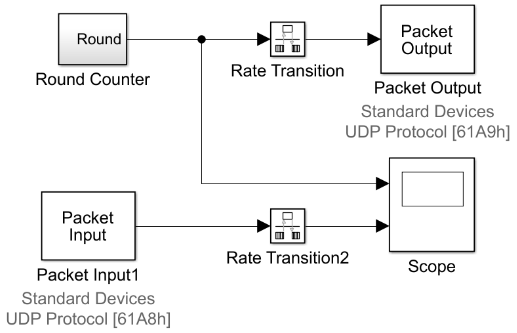
To verify above assumptions, a model will be built based on above assumptions to simulate the delays. And delay from a real HIL simulation will be measured. By comparing simulated results and actual results, the assumption can be verified.

In section 3.5.1 we will measure the execution time of xth round's executions, which is $T_{ex} = t_{BOx} - t_{BIx}$. In section 3.5.2 we will measure the round trip time T_r and delivery time T_d. With round trip time, execution time and delivery time known, queuing delay can be calculated. In section 3.4 we build a model according to Fig.3.2 and compare the simulated latency with the actual measured latency.

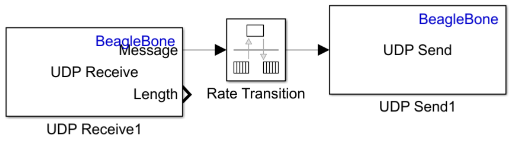
3.3 HIL Simulation Test

Fig.3.3 is a simple system for HIL Simulation. In Fig.3.3a during every execution, a round counter will count the rounds of simulation, and every output packet is marked by a unique integer. In Fig.3.3b, this packet will be processed by the controller, and the controller will send a packet back to plant model with the same data.

According to section 3.2, buffer in both PC and BBB will influence roundtrip time. To analyze buffer's effect, the buffer size in BBB is 1 and buffer size in PC is large enough. So we can observe the roundtrip time affected by packet dropping and reading from the empty buffer.



(a) Model In Simulink



(b) Controller in BBB

Figure 3.3: System Used for Measuring Latency

3.4 Model of Latency

Fig.3.4 is a model built according to Fig.3.2 to simulate the latency in network-based HIL simulation. This model consists of queue blocks and delay blocks. It ca be separated into two parts. One part simulate PC's performance and the other simulates BBB's performance. And these two parts are controlled by two unrelated clocks.

In this model, we can set the execution time, delivery time, buffer size, sample rate, clock drifting for both PC and BBB parts. Precision of this simulation is controllable, the minimum time can be as small as at microsecond level. By changing these parameters to the actual values in HIL simulation, we are expected to obtain simulation results similar to actual measurements.

3.5 Latency Measurement

As mentioned before, to analyze the latency in HIL simulation, actual delays measured in HIL simulation will be compared with an assumption. In sections 3.2-3.4 the assumption is described. In this section, the methods of measuring delay in different part of HIL simulation will be described.

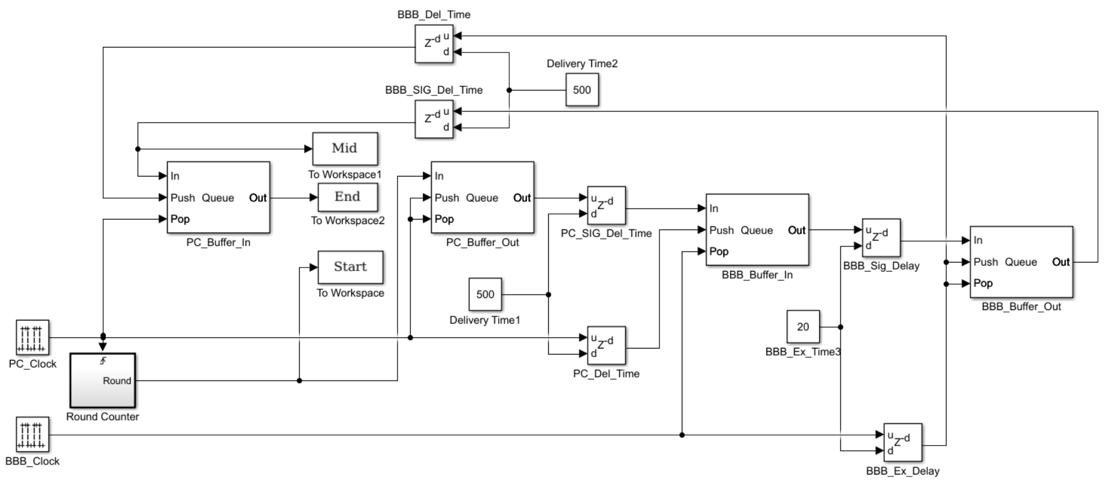


Figure 3.4: Model for simulating latency

3.5.1 Execution Time of Controller

In this case, we are interested in the execution time of controllers, i.e. $T_e = t_{\mathrm{BO1}} - t_{\mathrm{BI1}}$ in Fig.3.2a. To measure the execution time, we must use some instructions that can cause measurable effects at the beginning and end of execution. To be more specific, GPIO pins in BBB are utilized to generate these measurable effects. At the beginning of execution, a pulse is generated in a GPIO pin at time t_{p1}. In the end of execution, the other pulse is generated at time t_{p2}. These two pulses triggered at $t_{\mathrm{BI1}} = t_{\mathrm{p1}}, t_{\mathrm{BO1}} = t_{\mathrm{p2}}$ can be observed using oscilloscope. With the observations, the execution time is the interval between two pulses, which can be computed by formula $T_e = t_{p2} - t_{p1}$.

3.5.2 Latency Caused by Network

Network latency is $(t_{BC1} - t_{PO1}) + (t_{PC2} - t_{BO1})$ in Fig.3.2a or $(t_{BC2} - t_{PO1}) + (t_{PC3} - t_{BO2})$ in Fig.3.2b. So it is the round trip time minus by execution time. As stated before, in this thesis network latency is separated into delivery time T_d and queuing delay D_q.

Use Fig.3.5 to describe the structure of this system. At first, a packet with sensor signals are sent out from plant, after some time t_1, it arrives at BBB and waits in buffer. When the controller starts execution, it takes data from the buffer and do some computation, store the computation result in output buffer and output it in the end of execution. Then after some time t_2, the packet sent by controller arrives at buffer in PC and wait there until next execution of the simulation. $D_{nq} = t_1 + t_2$ is the delivery time we want to measure. t_1 and t_2 can be represented by time stamps in Fig.3.2. E.g. $t_1 = t_{PD1} - t_{PO1}, t_2 = t_{BD1} - t_{BO1}$ in Fig.3.2a.

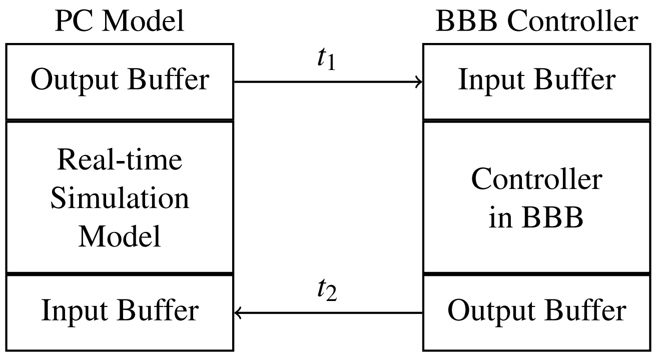


Figure 3.5: Communication between Simulink model and BBB controller

Using packet capture software like Wireshark we can capture packets in PC or BBB. Using Fig.3.2a as an example, if we capture packet in PC, we can compute the time $\Delta t_P = t_{BD1} - t_{PO1}$. If we capture packet in BBB, we can compute the time $\Delta t_B = t_{BO1} - t_{PD1}$. Then we can compute the delivery time using following equation:

$$T_d = t_1 + t_2 = t_{PD1} - t_{PO1} + t_{BD1} - t_{BO1} = (t_{BD1} - t_{PO1}) - (t_{BO1} - t_{PD1}) = \Delta t_P - \Delta t_B \quad (3.5)$$

When a packet returns, we can obtain the round trip time T_r. And if we have known the round trip time T_r, the execution time T_e and the delivery time T_d, queuing delay D_q can be computedd by equation:

$$D_q = T_r - T_e - T_d \quad (3.6)$$

So far, actual roundtrip time, execution time, delivery time and queuing delay has been measured. These measurements are compared with the simulation results of model in section 3.4

3.5.3 Control Effect

Delay in different part of HIL simulation is analyzed in previous sections. In this section, latency's effect on control of quadcopter will be evaluated. The quadcopter model in section 3.1 is used, it can be described by a graph in Fig.3.1.

First, the model runs in SIL simulation, plant and controller both run in Simulink using real-time simulation. We manually add latency to this system, observe the output until the system reaches instability. Then we run the same model using HIL simulation, plant runs in Simulink and controller runs in BBB, they interact with each other through UDP. Finally, we measure the latency in HIL simulation, observe the control performance and find out its relation with latency.

In this case, PID controllers are used in the quadcopter controller, we evaluate the control performance through step response. Reference values of altitude, pitch and yaw are constant and we observe the step response of roll.

During SIL simulation, we observe one step response in each simulation and run multiple simulations with different latency. During HIL simulation, we observe multiple step responses in one simulation and match the response with the latency.

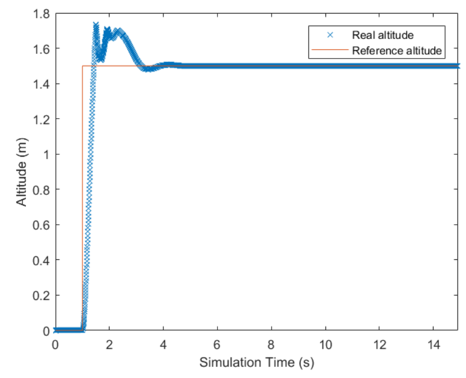
We know that network-based HIL simulation may have several problems. Besides network delay, packet drop and asynchronous clock an cause inaccuracy. Through comparing performance of HIL and SIL simulation in this case, we will find that only network delay influence the control effect. And if we can find a method to measure the delay in HIL simulation, the control performance can be predicted using data collected from SIL simulation.

Chapter 4

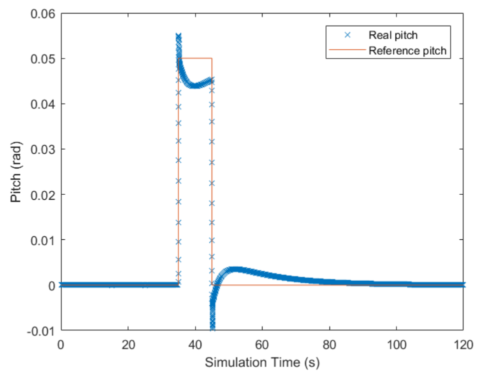
Experimental Results

4.1 Software In the Loop Simulation

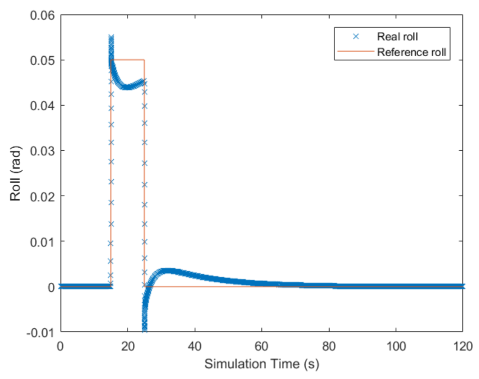
In this section, the SIL simulation is performed. Results of SIL simulation represent the best result that can be obtained. The latency in SIL simulation is constant 1 sample time (5ms). Part of step responses of altitude, roll, pitch and yaw in SIL simulation are shown in Fig. 4.1. Because roll, pitch and yaw can affect each other, we use pulse functions instead of step functions in this experiment. As a result, pitch and roll's settling time can't be measured.



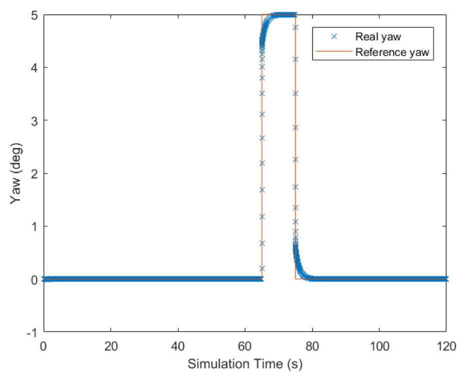
(a) Altitude



(b) Pitch

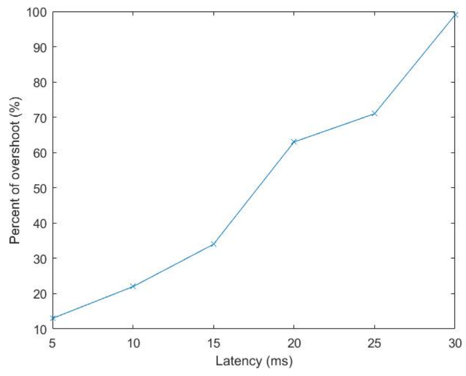


(c) Roll

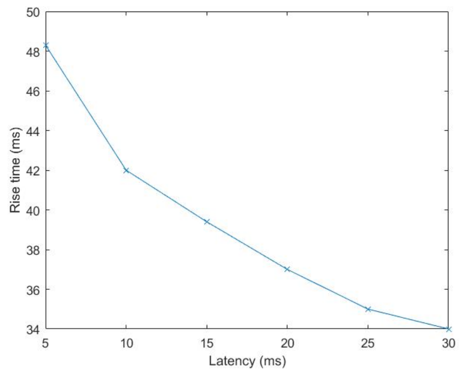


(d) Yaw

Figure 4.1: Output of quadcopter model



(a)



(b)

Figure 4.2: Latencys' effect on rise time and overshoot for controller of roll

Analyzing the results we will know:

1. Step response for altitude has long rise time, relative big overshoot, long settling time and no steady-state error.

2. For pitch and roll, the responses are very similar, the rise time is pretty short, they have some over shoot, settling time is very long

3. Yaw's response has small rise time and no overshoot. Settling time is small compared to roll and pitch.

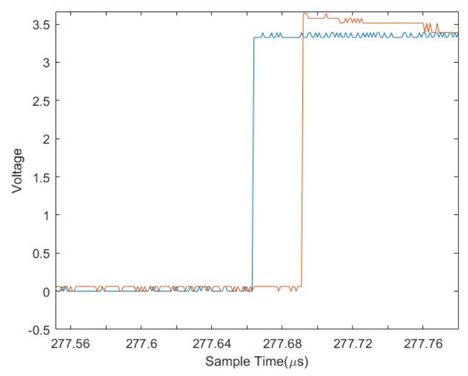
In this thesis, to measure the latency's effect, pitch or roll will be used for evaluating the control performance because these two values are most sensitive to latencies. In SIL simulation, latency can be manually set to a fixed value. Collect the data of roll's step response in SIL simulation, record the rise time and overshoot corresponding to different latency. Rise time is defined as the time required for the response to rise from 10% to 90% and overshoot is defined as the amount an output exceeds its steady-state value. The result is shown in Fig.4.2.

When latency is large, the actuators will receive delayed feedback, which is computed with delayed error signals. Since error signals keep decreasing at the beginning, the feedback will be larger if it is delayed and the controlled signals will rise faster. Therefore, in Fig.4.2a, the overshoot is larger and in Fig.4.2b the rise time is smaller as the latency increases.

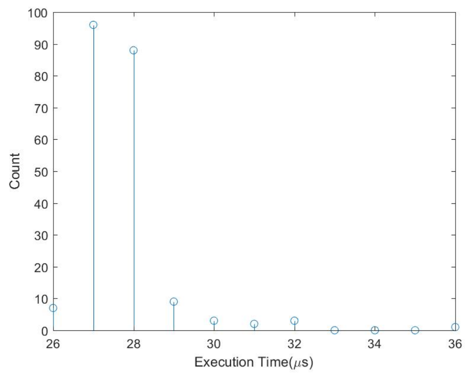
4.2 Latency Measurement for Communication Model

4.2.1 Execution Time

Execution time in BBB is measured using two pulses generated at the beginning and end of execution shown in Fig.4.3a. The time interval between two rising edges is the execution time Te. In this experiments, about 200 samples are collected for analysis using oscilloscope. Then count the frequency of these execution time and plot it in Fig.4.3b. The execution time of this controller is about $27\mu s$. Later we will see, compared to other latencies, this execution time is so small that it can almost be ignored.



(a) Pulses Generated at beginning and end of execution



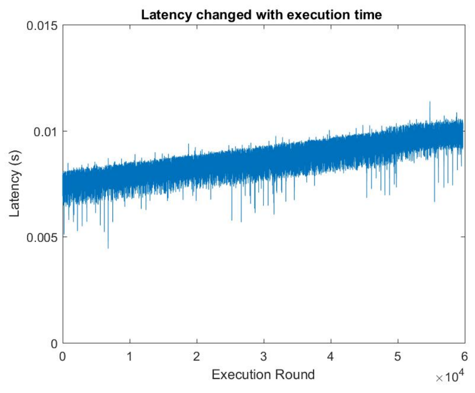
(b) Count of measured execution time

Figure 4.3: Measurement of Execution Time

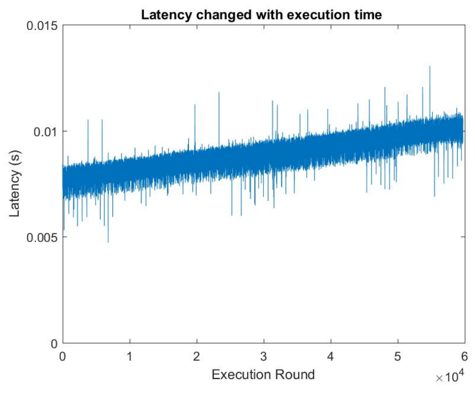
4.2.2 Delivery Time

Using methods proposed in section 3.5.2, we capture packets in both PC and BBB separately and use their timestamps to compute the delivery time. First, compute the latency Δt_B and Δt_P, which are shown in Fig.4.4a and Fig.4.4b. Then delivery time T_d is computed according to Eq.3.5 and we have Fig.4.4c. Counting the frequency of delivery time and show them in Fig.4.4d. From Fig.4.4 we are able to observe following truths:

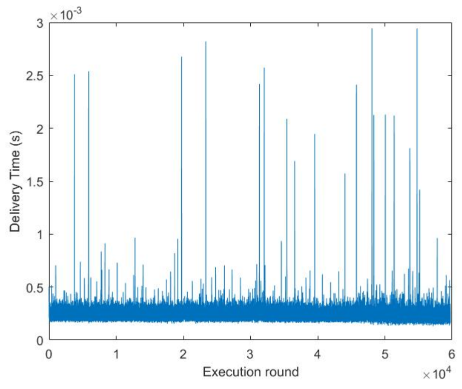
1. Most of the time, delivery time is smaller than 0.5ms.
2. The average delivery time is abot 0.2ms.
3. Sometimes delivery time can be big and reach as high as 3ms
4. Both Δt_B and Δt_P increase with the execution rounds, it is caused by clock drift, which will be explained in detail in section 4.2.3.



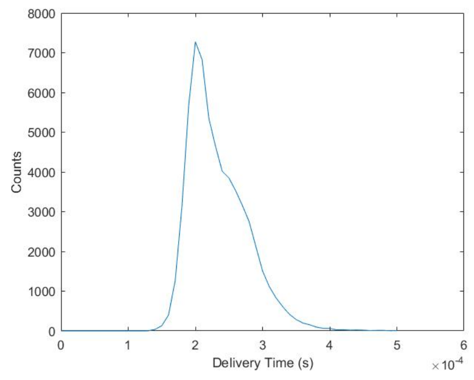
(a) Δt_B computed from packets cpatured in BBB



(b) Δt_P computed from packets cpatured in PC



(c) Delivery time



(d) Count of delivery time

Figure 4.4: Measurement of Delivery Time

4.2.3 Clock Drift

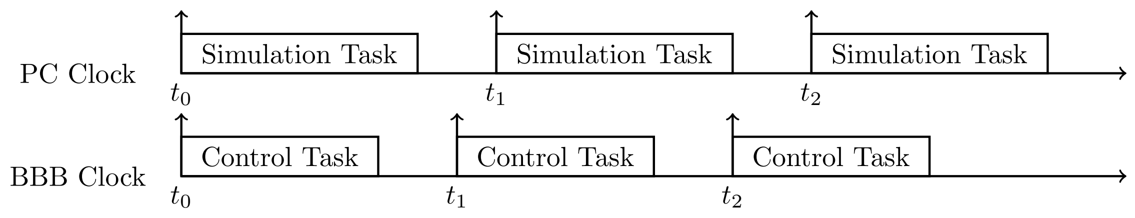
The execution time and delivery time have been measured, the round trip time is to be measured next. Before measuring roundtrip time, let's see a phenomenon called clock drift.

Clock drift means that a clock doesn't run with the exact same rate as another one. In this case, the clock in PC and clock in BBB doesn't run at the same frequency. As we mentioned in section 2.2, one of the problems of networked control is that there are no universal clocks to synchronize controller and plant. In this case, the model in PC and controller in BBB are controlled by different clocks. Offset exists between two parts. And because of the clock drift, the offset changes over time in a fixed rate, as we can see in Fig.4.4a and Fig.4.4b, Δt_B and Δt_P are influenced by clock drift.

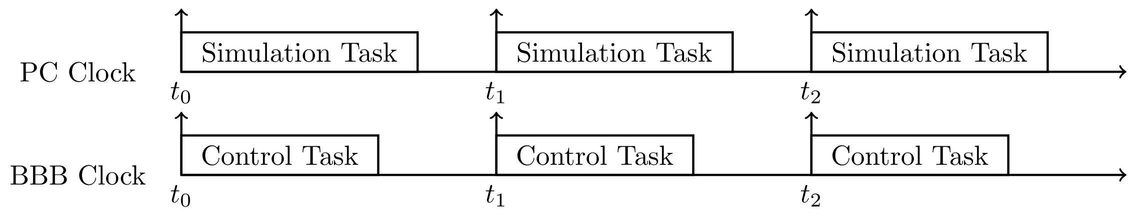
Clock drift can be a bad thing for HIL simulation because controller and simulation are all expected to be executed in real-time. If clock drift exists, these two parts runs in different frequency and make the simulation less reliable because this simulation fails to perfectly imitate the real situation. Network time protocol (NTP) can be used to synchronize multiple systems' clocks and probably can be used for synchronization in HIL simulation. However, in this case, directly applying NTP won't work. The sample time for simulation and controllers are 5 ms. In each round, the clock drift is about 1 to 20 μs. The precision of NTP is millisecond level, which is too big for simulation.

Although clock drift could influence HIL simulation's performance, in this case, it can be utilized to do timing analysis, help reveal some features of delay in HIL simulation. After many experiments, we find that clock drift is different every time a BBB boot up. 3 situations of clock drifts are shown in Fig.4.5. Later we will see that the delays are very different in these 3 situations.

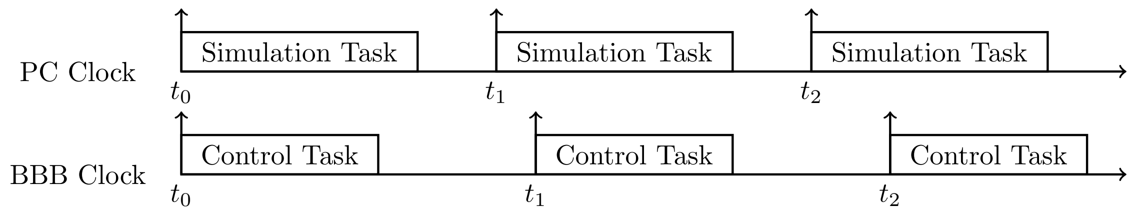
According to Fig.3.2 and equation 3.4, the roundtrip time is influenced by buffer states and the offset of simulation and controller's execution time. The clock drift will affect both buffer states and offset between two clocks. Thus in one simulation with clock drift, we can observe delays under different conditions. Using packets captured from Wireshark, we



(a) BBB clock runs faster than PC



(b) BBB clock runs in same speed as PC



(c) BBB clock runs slower than PC

Figure 4.5: Three situations of clock drift

can compute Δt_P introduced in section 3.5.2. According to our assumption, Δt_P will change because of clock drift. Fig. 4.6 shows the measurement of Δt_P in three different situations corresponding to Fig.4.5.

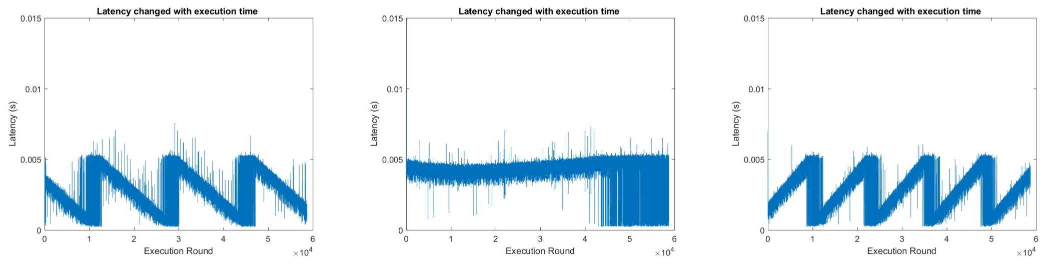
The clock drifts' influence can be observed in Fig.4.6:

1. When BBB clock runs faster than PC, executions of controller will be appear earlier in each round as simulation goes. As a result, Δt_P becomes smaller, until at some moment, BBB tries to obtain data in an empty buffer. Then Δt_P is increased by one sample time. This is the pattern that Fig.4.6a shows.

2. When BBB clock runs in the same speed as PC clock, buffer states and the offset between two parts' execution time won't change much. So Δt_P tends to remain constant just like Fig.4.6a shows. Note that it is hard to make BBB and PC run in the exactly same frequency, we use Fig.4.6b to represent this situation.

3. When BBB clock runs slower than PC, executions of controller will be put off so that Δt_P keeps growing. Then at some moment, a packet will be dropped because the input buffer of BBB is full. Δt_P will be decreased by one sample time then. This pattern is shown in Fig.4.6c

4. When Δt_P is near 0 or 5 ms, it changes between 0 and 5 frequently. This phenomenon will be explained in section 4.2.5.

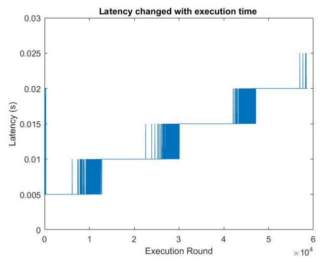


(a) BBB clock is faster than PC (b) BBB clock is same as PC (c) BBB clock is slower than PC

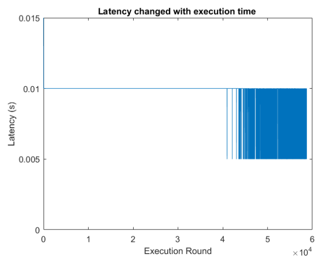
Figure 4.6: Clock drifts' influence on Δt_P

4.2.4 Roundtrip Time

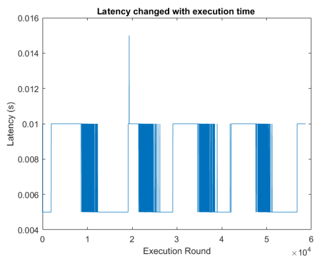
Effect of clock drift is shown above, now let's see how roundtrip time changes during the simulation in above three situations. Roundtrip time is measured by matching a received packet with a sent packet in PC and computing the difference between timestamps. The results are shown in Fig.4.7.



(a) T_r - fast BBB CLK



(b) T_r - medium BBB CLK



(c) T_r - slow BBB CLK

Figure 4.7: Round trip time in three different situations of clock drifting

By analyzing results in Fig.4.7, we know that:

1. Based on assumptions in section 3.2, when BBB clock runs faster than PC clock, BBB output packet faster than PC can consume. As a consequence, packets will gradually pile up at PC's input buffer and round trip time will keep increasing until the input buffer in PC is full. Roundtrip time in Fig.4.7a satisfies the expectation.

2. If BBB clock runs in a similar rate to PC clock and the noise's effect is ignored, the round trip time won't change. Although in Fig.4.7b BBB and PC clocks don't run in a same rate, their sampling rate is very close, thus the round trip time changes very slowly and no steep rising or falling are observed except for the beginning part. This result also fits the expectation.

3. When BBB clock runs slower than PC clock. PC will consume packets faster than BBB, so packets can't pile up at PC's input buffer. But packets may pile up at BBB's input buffer if BBB's buffer is big enough. However, BBB's buffer size is set to be

1. So whenever the packet tries to push a packet into an occupied buffer, the packet will be dropped and BBB's buffer won't be congested.

4.2.5 Queuing Delay

Now that we have know roundtrip time, execution time, delivery time, queuing delay can be computed according to equation 3.2. Furthermore, queuing delays in two buffers can be computed separately. We have measured that execution time T_e is about 27 μs, which is much smaller than other time, thus it can be ignored by assuming $T_e = 0$. Separate the queuing delay into two part, D_{q1} is the time a packet stays in BBB buffer, $D_{q2}$is the time a packet stays in PC buffer. Then equation 3.2 can be rewritten as:

$$T_r = D_{q1} + T_d + D_{q2} \tag{4.1}$$

What we know is T_d, $T_r = D_{q2} + \Delta t_P$ and $\Delta t_P = T_d + D_{q1}$. Two queuing delays thus can be computed by:

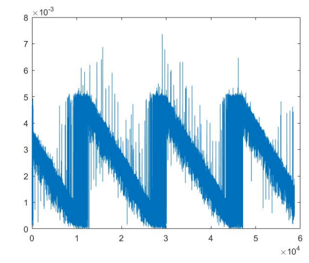
$$D_{q2} = T_r - \Delta t_P, \quad D_{q1} = \Delta t_P - T_d \tag{4.2}$$

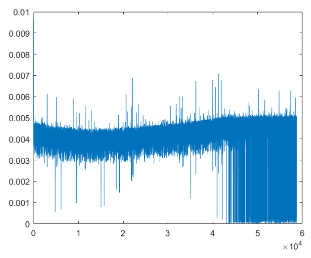
By applying above equation, queuing delays are shown in Fig.4.8.

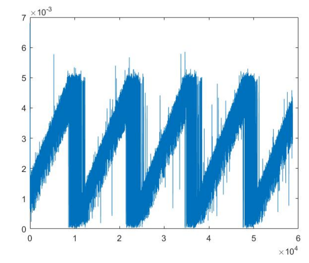
Analyze the results and we will find:

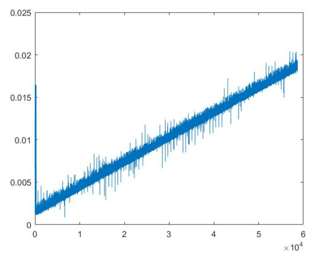
1. Queuing delays in BBB look like Δt_P's shape in Fig.4.6 and they act same as Δt_P when clock drifts are different. Theoretically the range is between 0 and 5 ms, because BBB's buffer size is 1, it is the maximum delay it can cause. But we notice that at some moment the delay is greater than 5 ms, it is probably because we assume the delivery time $T_d = 0.2ms$, which over-simplify the problem. As the measurement for delivery time showed, although most of the time delivery time is stable, sometimes delivery time can be very large. That may be the reason for the unexpected values.

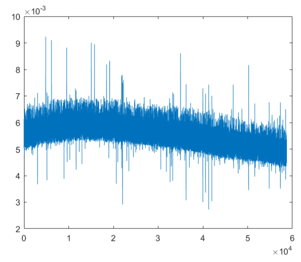
2. When BBB clock runs faster than PC, queuing delay in PC's buffer keeps growing in


(a) D_{q1} with faster BBB clock


(b) D_{q1} with BBB clock rate same as PC


(c) D_{q1} with slower BBB clock


(d) D_{q2} with faster BBB clock


(e) D_{q2} with BBB clock rate same as PC

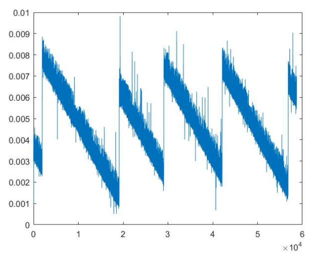
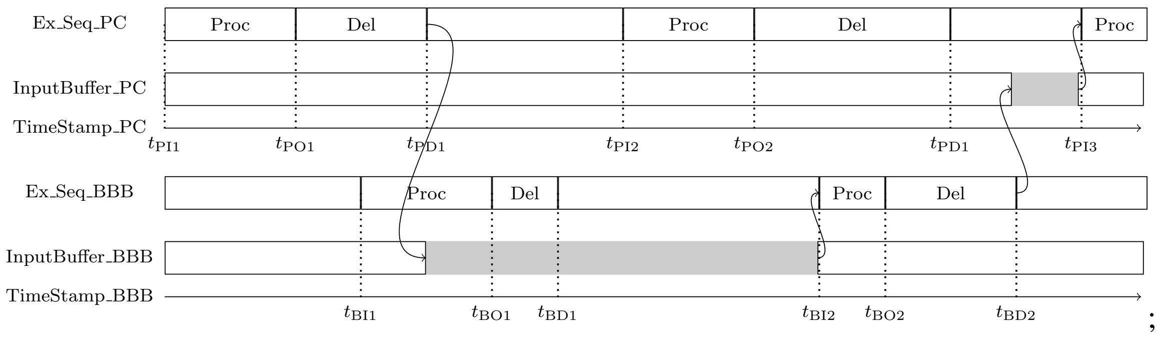

(f) D_{q2} with slower BBB clock

Figure 4.8: Queuing delay in two buffers

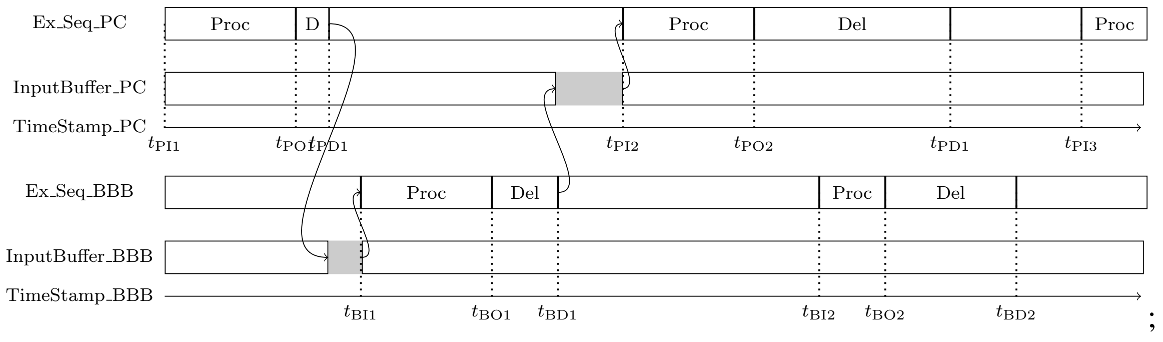
a fixed rate. Assume the difference between BBB clock and PC clock in one round is Δt_c, then after every round, a packet has to wait Δt_c more time in PC's buffer, and this time will grow until PC's buffer is full. That's why roundtrip time increases in this situation.

3. When BBB clock runs in the same speed as PC, queuing delay in PC'buffer won't change much. And compared to queuing delay in BBB's buffer, there are no big oscillation between 0 and 5 ms.

4. When BBB clock runs slower than PC, queuing delay in PC's buffer will decrease. Assume PC clock is Δt_c shorter than BBB in each round, the queuing delay will then decrease by Δt_c after each round. Until this queuing delay is about to reach 0. An execution trying to pop from an empty buffer in PC will occur, then the queuing delay will be increased by 5 ms, the same procedure repeats, so the queuing delay in PC won't exceed 5 ms.



(a)



(b)

Figure 4.9: Two similar executions with only first delivery time different

5. The repeated jump between 0 and 5ms appears in queuing delay in BBB buffer. In the remaining part of this section, this phenomenon will be explained

To explain the frequent jump, we use a graph similar to Fig.3.2. In Fig.4.9 we can see two situations where the offset between PC and BBB's execution are same. In Fig.4.9a, delivery time is long, BBB executes before the packet is delivered, so it will read buffer from an empty buffer, the queuing delay will be increased by 5ms. In Fig. 4.9b, the packet is delivered before BBB's execution, the queuing delay can be as small as 0. What's more, since BBB's buffer size is 1, when a packet is delivered to a full buffer, the packet will be dropped, causing queuing delay to be decreased by 5ms. Thus the uncertainty of delivery time is responsible for the repeated jump of BBB's queuing delay.

4.3 Timing Analysis

With the latency model, through selecting proper parameters, results in Fig.4.7 can be simulated. Apply low-pass filter on Fig.4.6a, 4.6b, 4.6c we obtain Fig.4.10a, 4.10h, 4.10c, Compare them with simulated results Fig.4.10d, 4.10e, 4.10f, the simulation results are close to the actual measurement.

Fig.4.10g, 4.10b, 4.10i are the simulated round trip time corresponding to Fig.4.7a, 4.7b, 4.7c. Compare the simulated result with actual result, their performance are very similar.

The latency model built based on assumptions in section 3.2 appears to well imitate actual networked-based HIL simulation's latency. Thus the assumption of delay we proposed is reliable.

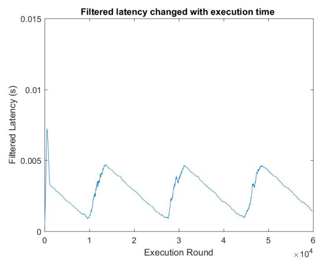
4.4 Effect on Control

The latency in network-based HIL simulation has been measured and identified above. In this section we will see the latency's effect on HIL simulation.

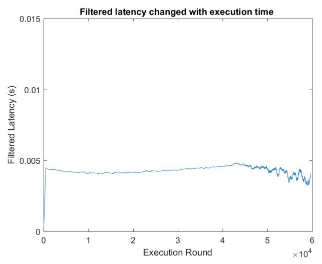
In this experiment, we will compare the performance of HIL simulation with SIL simulation. Since less latency is expected in SIL simullation, HIL simulation is expected to be less stable. SIL simulation's result was obtained in section 4.1. HIL simulation's result is compared in Fig.4.11. It's easy to notice that HIL simulation usually has larger overshoot, which corresponds to larger latency.

In the experiment, HIL simulation runs in two different clock drift situations and roundtrip time is measured. With roundtrip time and Fig.4.2, overshoot and rise time can be predicted using HIL simulation's result. Predicted result and measured result are shown in Fig.4.12.

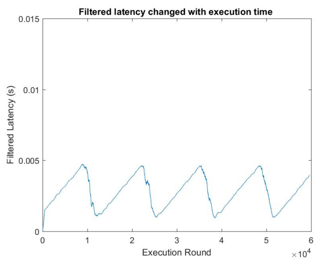
The latency of HIL simulation is as expected, when BBB clock is slower, the latency switches between two values. When BBB clock is faster, the latency keeps growing. Compare SIL and HIL simulation results, results obtained by two simulations match well. Thus



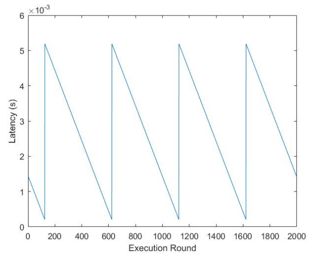
(a) Filtered Δt_out fast BBB CLK



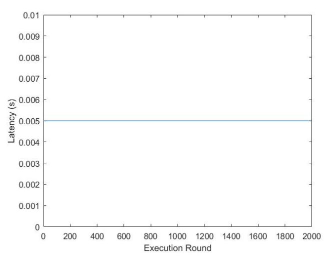
(b) Simulated T_r medium BBB CLK



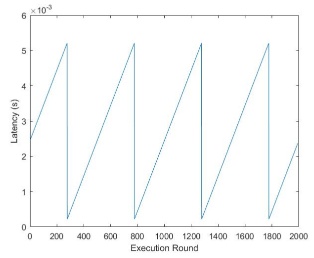
(c) Filtered Δt_out slow BBB CLK



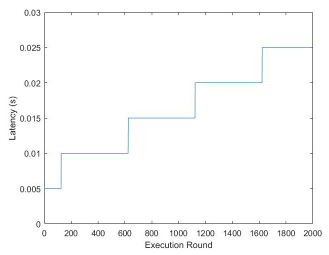
(d) Simulated Δt_out fast BBB CLK



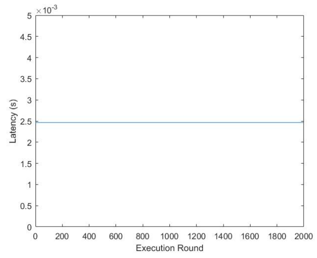
(e) Simulated Δt_out medium BBB CLK



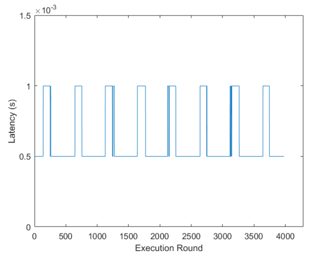
(f) Simulated Δt_out slow BBB CLK



(g) Simulated T_r fast BBB CLK

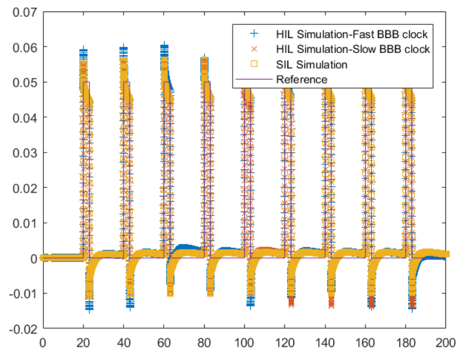


(h) Filtered Δt_out medium BBB CLK

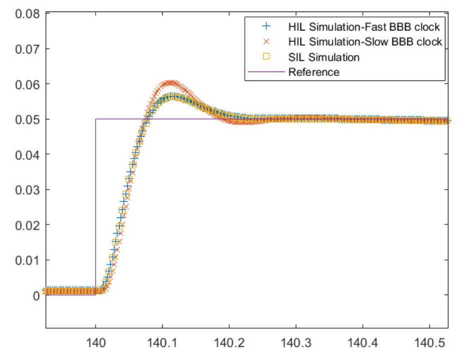


(i) Simulated T_r slow BBB CLK

Figure 4.10: Simulation of Latency



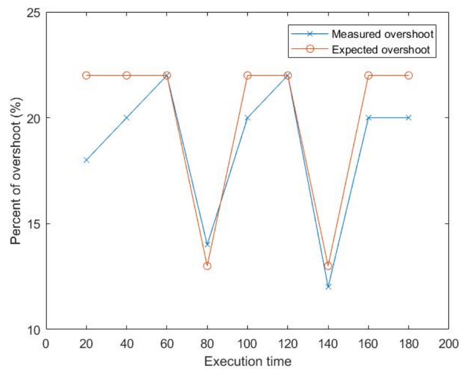
(a)



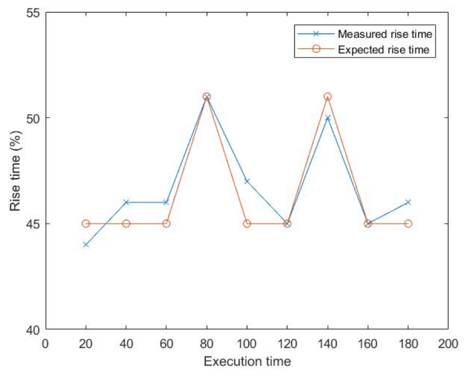
(b)

Figure 4.11: Step response

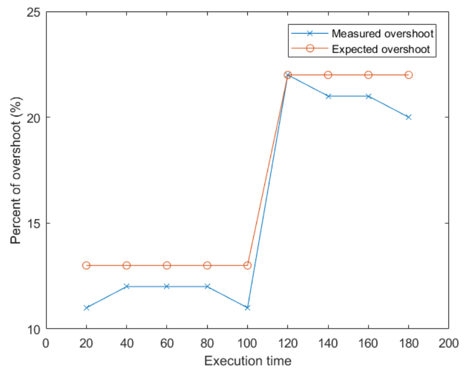
SIL and HIL have similar performance with same latency. Even though packet loss and clock asynchronizezation exist in HIL simulation, latency is the main factor that affects control performance.



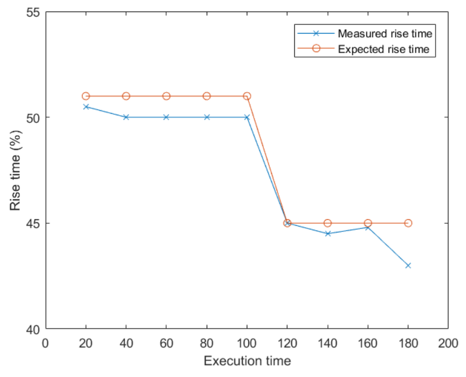
(a) Overshoot when BBB clock is slow



(b) Rise time when BBB clock is slow



(c) Overshoot when BBB clock is fast



(d) Rise time when BBB clock is fast

Figure 4.12: Step response in SIL simulation

Chapter 5

Summary and Future Work

In this thesis, network-based HIL simulation has been evaluated. The latency in HIL simulation is located and identified. The results shows that latency in network-based HIL simulation is mainly caused by network delay, especially queuing delay. Delivery time also has some influence on latency, especially when sample time is smaller. Queuing delay largely depends on buffer size, if buffer size is n, sample time is T_s, maximum queuing delay is nT_s.

Control performance of HIL simulation has been also evaluated. HIL simulation has problems like packet loss, network delay, clock asynchronization. But network delay may be the most important factor. When an HIL simulation's latency is found, it is control performance can be predicted by results of SIL simulation.

To improve HIL simulation performance the main job is to decrease the latency. It can be achieved using following methods:

1. Reduce receive buffer size. One of the main reason of long latency is that packets may congest in input buffer. If receiver buffer size is small (e.g. can only store one packet), packets can't pile up and the maximum queuing delay is one sample time, which is acceptable.

2. Modify sampling rate of sending and receiving block. As we can see in the experiment result of Fig.4.7, if receiving speed is faster than sending speed, most of the time there would be no packet waiting in buffer, thus the queuing delay can be smaller.

This thesis mainly focuses on the latency of network-based HIL simulation and its effect. There are much more work can be done to improve the results. For example, experiments in this thesis are done when execution time is too small to be considered, more experiments can be done to observe the results when execution is large. As for the network latency, the buffer size's influence has been found, but the sampling rate may also have some influence and more experiments can be done to study sampling rate's effect. In this thesis, we only studied relation between latency and control performance. More experiments can be done to find relation between packet loss and clock asynchronization to support the conclusion. The control effect of HIL simulation is studied by comparing HIL and SIL simulation results. But to verify the reliability of HIL simulation, comparing it with real physical devices is a better way.

BIBLIOGRAPHY

[1] Edward A Lee. Cyber physical systems: Design challenges. In *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on*, pages 363–369. IEEE, 2008.

[2] Emeka Eyisi, Zhenkai Zhang, Xenofon Koutsoukos, Joseph Porter, Gabor Karsai, and Janos Sztipanovits. Model-based control design and integration of cyberphysical systems: an adaptive cruise control case study. *Journal of Control Science and Engineering*, 2013:1, 2013.

[3] Sanja Lazarova-Molnar, Hamid Reza Shaker, and Nader Mohamed. Reliability of cyber physical systems with focus on building management systems. In *Performance Computing and Communications Conference (IPCCC), 2016 IEEE 35th International*, pages 1–6. IEEE, 2016.

[4] Lothar Thiele, Felix Sutton, Romain Jacob, Roman Lim, Reto Da Forno, and Jan Beutel. On platforms for cps-adaptive, predictable and efficient. In *Rapid System Prototyping (RSP), 2016 International Symposium on*, pages 1–3. IEEE, 2016.

[5] Peter Marwedel. *Embedded system design*, volume 1. Springer, 2006.

[6] Jean Blanger, P Venne, and Jean-Nicolas Paquin. The what, where, and why of real-time simulation. pages 37–49, 01 2010.

[7] Di Shang, Emeka Eyisi, Zhenkai Zhang, Xenofon Koutsoukos, Joseph Porter, Gabor Karsai, and Janos Sztipanovits. A case study on the model-based design and integration of automotive cyber-physical systems. In *Control & Automation (MED), 2013 21st Mediterranean Conference on*, pages 483–492. IEEE, 2013.

[8] Jeff C Jensen, Danica H Chang, and Edward A Lee. A model-based design methodology for cyber-physical systems. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1666–1671. IEEE, 2011.

[9] Fabio Paterno. *Model-based design and evaluation of interactive applications*. Springer Science & Business Media, 2012.

[10] Jim A Ledin. Hardware-in-the-loop simulation. *Embedded Systems Programming*, 12:42–62, 1999.

[11] James F Kurose and Keith W Ross. *Computer networking: A top-down approach featuring the internet, 6/E*. Pearson Education Inc, 2013.

[12] Gabe Hoffmann, Dev Gorur Rajnarayan, Steven L Waslander, David Dostal, Jung Soon Jang, and Claire J Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). In *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, volume 2, pages 12–E. IEEE, 2004.

[13] Introduction to modeling and simulation systems. https://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf, 2000.

[14] PM Menghal and A Jaya Laxmi. Real time simulation: Recent progress & challenges. In *Power, Signals, Controls and Computation (EPSCICON), 2012 International Conference on*, pages 1–6. IEEE, 2012.

[15] R Kuffel, J Giesbrecht, T Maguire, RP Wierckx, and P McLaren. Rtds-a fully digital power system simulator operating in real time. In *WESCANEX 95. Communications, Power, and Computing. Conference Proceedings., IEEE*, volume 2, pages 300–305. IEEE, 1995.

[16] I Etxeberria-Otadui, Vincent Manzo, Seddik Bacha, and F Baltes. Generalized average modelling of facts for real time simulation in arene. In *IECON 02 [Industrial*

Electronics Society, IEEE 2002 28th Annual Conference of the], volume 2, pages 864–869. IEEE, 2002.

[17] Simon Abourida, Christian Dufour, Jean Bélanger, Guillaume Murere, Nicolas Lechevin, and Biao Yu. Real-time pc-based simulator of electric systems and drives. In *Applied Power Electronics Conference and Exposition, 2002. APEC 2002. Seventeenth Annual IEEE*, volume 1, pages 433–438. IEEE, 2002.

[18] Lutz Köster, Thomas T Thomsen, and Ralf Stracke. Connecting simulink to osek: Automatic code generation for real-time operating systems with targetlink. 2001.

[19] MathWorks® (2018). Simulink® user's guide (r2018a). https://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf, Mar 2018.

[20] Marko Bacic. On hardware-in-the-loop simulation. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 3194–3198. IEEE, 2005.

[21] Wei Ren, Michael Steurer, and Thomas L Baldwin. Improve the stability and the accuracy of power hardware-in-the-loop simulation by selecting appropriate interface algorithms. *IEEE Transactions on Industry Applications*, 44(4):1286–1294, 2008.

[22] Rainer Krenn. Limitations of hardware-in-the-loop simulations of space robotics dynamics using industrial robots.

[23] Saffet Ayasun, Robert Fischl, Sean Vallieu, Jack Braun, and Dilek Cadırlı. Modeling and stability analysis of a simulation–stimulation interface for hardware-in-the-loop applications. *Simulation Modelling Practice and Theory*, 15(6):734–746, 2007.

[24] Chinchul Choi and Wootaik Lee. Analysis and compensation of time delay effects in hardware-in-the-loop simulation for automotive pmsm drive system. *IEEE Transactions on industrial electronics*, 59(9):3403–3410, 2012.

[25] Wei Zhang, Michael S Branicky, and Stephen M Phillips. Stability of networked control systems. *IEEE Control Systems*, 21(1):84–99, 2001.

[26] Gregory C Walsh, Hong Ye, and Linda G Bushnell. Stability analysis of networked control systems. *IEEE transactions on control systems technology*, 10(3):438–446, 2002.

[27] Mei Yu, Long Wang, Tianguang Chu, and Guangming Xie. Stabilization of networked control systems with data packet dropout and network delays via switching system approach. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 4, pages 3539–3544. IEEE, 2004.

[28] Johan Kraft. Rtssim-a simulation framework for complex embedded systems. *Technical Report*, 2009.

[29] Markus Bohlin, Yue Lu, Johan Kraft, Per Kreuger, and Thomas Nolte. Simulation-based timing analysis of complex real-time systems. In *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA'09. 15th IEEE International Conference on*, pages 321–328. IEEE, 2009.

[30] Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22, 2011.

[31] Atheer L Salih, M Moghavvemi, Haider AF Mohamed, and Khalaf Sallom Gaeid. Flight pid controller design for a uav quadrotor. *Scientific research and essays*, 5(23):3660–3667, 2010.

[32] Tarek Madani and Abdelaziz Benallegue. Backstepping control for a quadrotor helicopter. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3255–3260. IEEE, 2006.

[33] Guilherme V Raffo, Manuel G Ortega, and Francisco R Rubio. An integral predictive/nonlinear h ∞ control structure for a quadrotor helicopter. *Automatica*, 46(1):29–39, 2010.

[34] Myungsoo Jun, Stergios I Roumeliotis, and Gaurav S Sukhatme. State estimation of an autonomous helicopter using kalman filtering. In *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 3, pages 1346–1353. IEEE, 1999.