MPC SYSTEM IDENTIFICATION METHOD BASED ORACLE FOR CONTROL-CPS SOFTWARE FAULT LOCALIZATION

ZHIJIAN HE

PhD

The Hong Kong Polytechnic University

2019

THE HONG KONG POLYTECHNIC UNIVERSITY

DEPARTMENT OF COMPUTING

MPC System Identification Method based Oracle for Control-CPS Software Fault Localization

ZHIJIAN HE

A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

June 2018

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signature)

HE ZHIJIAN

_____(Name of Student)

ABSTRACT

Control-CPS software fault localization (SFL, aka debugging) is of critical importance as bugs may cause major mission failures, even injuries/deaths. To locate the bugs in control-CPSs, SFL tools often demand many labeled (“correct”/“incorrect”) source code execution traces as inputs. To label the correctness of these traces, we must judge the corresponding control-CPS physical trajectories' correctness. However, unlike discrete outputs, the boundaries between correct and incorrect physical trajectories are often vague. The mechanism (aka oracle) to judge the physical trajectories' correctness thus becomes a major challenge. So far, the ad-hoc practice of “human oracles” are still widely used, whose qualities are heavily dependent upon the human experts' expertise and availability. This thesis proposes an oracle based on the system identification (SI) method used in the renowned model predictive control (MPC) technology. Originally designed for controlling black-box physical systems, the MPC-SI is adapted by us to learn the buggy control-CPS as a black-box. We use this learning result as an oracle to judge the control-CPS's behaviors, and propose a framework of methodology to prepare traces for control-CPS debugging. Evaluation results on classic control-CPSs with real-life and artificial bugs show that our proposed approach significantly outperforms the human oracle approach in SFL accuracy, recall, and latency, and in oracle false positive/negative rates.

PUBLICATIONS

1. $[InSubmission]$ **ZHIJIAN HE**, Yao Chen, Enyan Huang, Qixin Wang, "MPC System Identification Method based Oracle for Control-CPS Software Fault Localization", in submission for conference publication.

2. $[TECS18]$ **ZHIJIAN HE**, Yao Chen, Zhaoyan Shen, "Attitude Fusion of Inertial and Magnetic Sensor under Different Magnetic Filed Distortions", in *ACM Transactions on Embedded Computing Systems (TECS)*, Volume 17 Issue 2, April 2018, Article No. 48.

3. $[TECS17]$ Zhaoyan Shen, **Zhijian He**, Shuai Li, Qixin Wang, Zili Shao, "A Multi-Quadcopter Cooperative Cyber-Physical System for Timely Air Pollution Localization", in *ACM Transactions on Embedded Computing Systems (TECS)*, Volume 16 Issue 3, July 2017, Article No. 70.

4. $[MSN15]$ **Zhijian He**, Yanming Chen, Zhaoyan Shen, Enyan Huang, Shuai Li, Zili Shao, Qixin Wang, "Ard-mu-copter: A Simple Open Source Quadcopter Platform", in *Proceedings of the 11th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, Dec. 16-18, 2015, Shenzhen, China.

5. $[ICCPS15]$ **Zhijian He**, Shuai Li, Zhaoyan Shen, Muhammad Umer Khan, Zili Shao, Qixin Wang, "WiP Abstract: A quadcopter swarm for active monitoring of smog propagation", in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCP-S'15)*, Work-in-Progress Session, Seattle, USA, April, 2015.

ACKNOWLEDGEMENTS

Firstly, I offer my sincerest gratitude to my wife, Rose, who gives me a harmonious family during my Ph.D study. Without her tolerance and unquestioned supports, I would not complete my thesis. I am peculiarly fortunate to have such a good wife.

Furthermore, profound gratitude goes to my dear supervisor, Dr. Qixin Wang. Learning under Dr. Wang's guidance has been an amazing experience for his systematic research method and rigorous research attitude. I give my thank sincerely to him here, not only for his tremendous academic support, but also for giving me enough time and financial support.

I am also hugely appreciative to Dr. Shuai Li and Dr. Max Pei, who provided me a lot of guidance on software engineering and robotics, and who enlighten me the first glance of deep learning research. Without their precious support it would not be possible to conduct this thesis.

Finally, but by no means least, special mention goes to my colleagues, Enyan Huang, Yao Chen and Zhaoyan Shen. Under their selfless support, I save a lot of time and avoid many mistakes on my research. They often teach me what to program and how to program. Also, I would like to thank my badminton friends, Minglei Li, Lily, Anu and Kaining Yan. Via playing badminton together, I release my pressure and receive my friendship. They are the most important people in my world and I dedicate this thesis to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Demand

Control *Cyber-Physical Systems* (CPSs), aka control-CPSs, are growing rapidly due to the inevitable convergence of computer (i.e. cyber) and physical systems [19, 114]. Typical control-CPSs include avionics, vehicles, robotics, power grid, medical equipment etc. Many control CPSs are life/mission critical, and bugs in the cyber subsystem can cause major mission failures, even injuries/deaths [1, 13, 20, 68, 84, 108, 109]. For example, in the safety-critical applications such as "Ariane-5" (Figure 1.1), a subtle software defect inside a seldom-appear branch of the control-CPS software cause a great loss. In the hobby-like applications such as "quadcopter" (Figure 1.2), improper software design can cause dangers and injuries. Thus, control-CPS debugging therefore is of high importance.

An indispensable step for debugging is to locate bugs in the source code, aka *software fault localization* (SFL)[1]. As software complexity scales up, manual SFL no longer suffices. Over the years, many automatic SFL tools are developed. According to a recent survey [132], there are over 331 papers on SFL. The corresponding tools can be categorized into 8 families, respectively based on program spectrum, statistics, machine learning, data mining, program slice, program state, model, and others. Many main stream SFL tools and over 38% of all the tools surveyed (particularly those belong to the program spectrum, statistics, and machine learning tool families) need thousands of labeled (as "correct" or "incorrect") *source code*

[1] Though the formal name for "bugs" in the software engineering is "faults", this thesis intends to use the term "bug" and "buggy" instead of "fault" and "faulty". This is because in control theories, "fault" has other meanings.

Figure 1.1: Ariane-5 explosion (quoted from [1]), which resulted in US$500 million loss, is due to a bug in the cyber subsystem [84].

Figure 1.2: Quadcopter propellors cause serious finger injuries [2].

execution traces as input (see Fig. 1.3). Manually labeling the correctness of so many source code execution traces is impractical. Instead, we need an automatic judgement mechanism, aka *oracle*, to do the labeling.

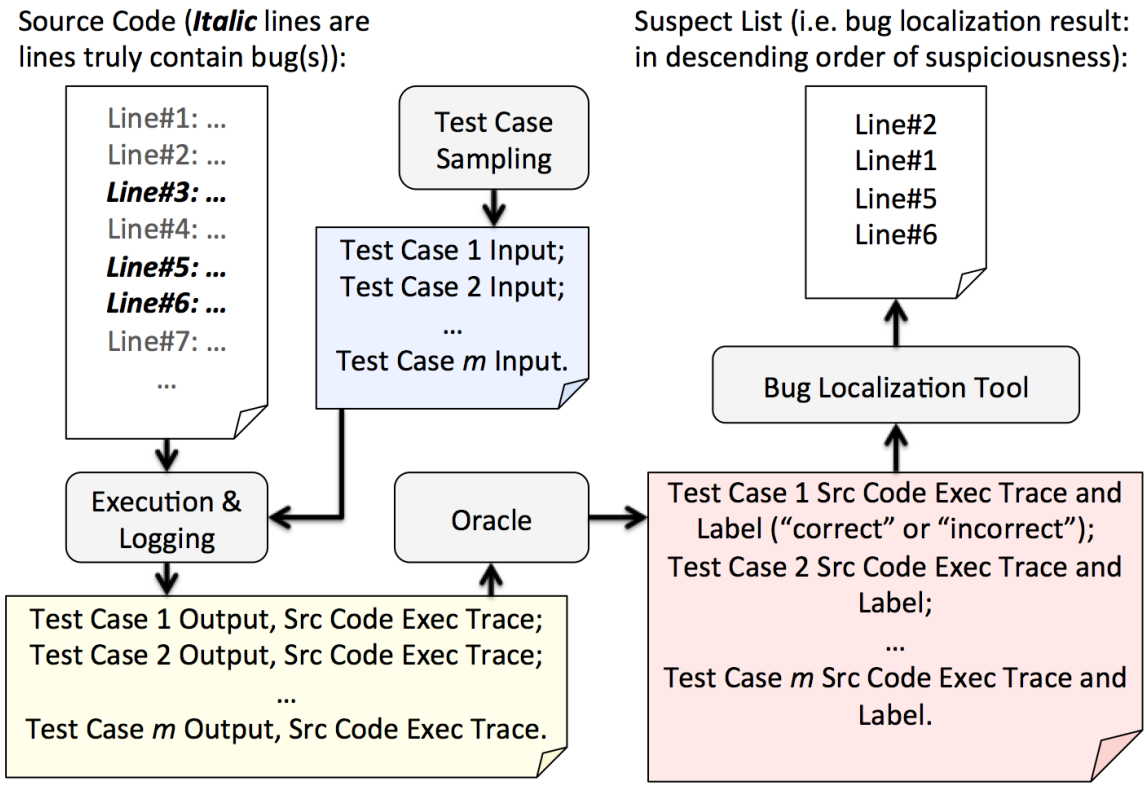


Figure 1.3: Typical work flow of (program spectrum, statistics, and machine learning based) bug localization (aka *software fault localization*, simplified as "SFL") tools.

However, how to design oracle and the corresponding source code execution trace preparation methodology is a well-known hard problem: i.e. the so-called *oracle problem* [23, 59, 61, 136]. Solutions to the oracle problem are highly application domain dependent [23,145]. For some application domains, the oracle problem is still open [23,36,72,126].

Control-CPS is one such domain, where the oracle problem faces unique challenges. Unlike the clean-cut discrete output of pure cyber systems, control-CPS outputs are continuous *physical trajectories*. At the first look, all or multiple physical trajectories in an envelope can seem correct (see Fig. 1.4). To exactly decide which one is correct/incorrect, an *ideal* black-box oracle (i.e. one without looking into the cyber source code) approach would need

a known a priori bug-free physical trajectory, i.e. the *expected physical trajectory*. To predict the expected physical trajectory, however, the approach needs an emulation[2] of the control-CPS upon the *bug-free* cyber subsystem and a correctly simulated physical subsystem[3]. This directly contradicts the fact that our cyber subsystem is yet to be debugged. Hence the ideal black-box oracle approach cannot work.

Replacing the prerequisite of the bug-free cyber subsystem with a substitute cyber subsystem implementation (e.g. those generated by model-driven development [46]) cannot solve the problem. This is because a simplified substitute implementation would not catch all the subtleties needed by debugging; while a comprehensive substitute implementation costs too much human effort (hence contradicts our goal of SFL automation), and may itself needs debugging.

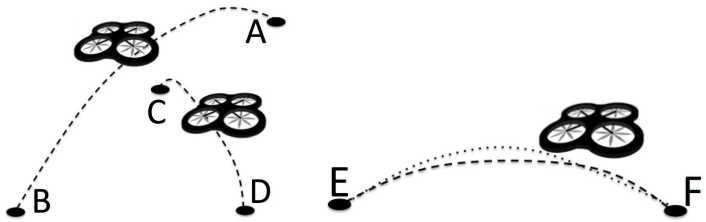
Therefore the *ideal* black-box oracle approach is unlikely to work. Meanwhile, building an oracle by analyzing the source code is neither likely to succeed. This is because the source code is yet to be debugged. Knowledge learnt from the buggy source code can be misleading. For example, we cannot use invariants found by program analysis [43, 44, 54, 102, 103] as oracles. This is because invariants must comply with all system behaviors, including the buggy behaviors. Thus invariants found before debugging will take buggy behaviors for granted as normal behaviors, hence fail as oracles.

Due to the above reality, *human oracles* are still widely used in control-CPS SFL [15, 23, 80, 86, 92, 96, 106]. Typically, a group of human experts are convened to discuss and manually design a set of assertions to judge physical trajectory correctness with best-effort. These assertions are the so-called human oracles. Apparently, the human oracle approach is fundamentally ad-hoc: it heavily depends on the human experts' expertise and availability.

To improve, this thesis aims to find another approach to deterministically and automatically generate better oracles for control-CPS SFL. Particularly, we are interested in ex-

[2] Emulation, aka half simulation, means part of the runtime system is simulated, while the other part is real.

[3] Usually, the physical subsystem model is available and is much simpler than the cyber subsystem. Hence, in this paper, we focus on control-CPSs whose physical subsystems can be correctly simulated.



(a) Is/are arc A→ B or C→D correct?

(b) Is/are the dotted or the dashed curve correct?

Figure 1.4: Control-CPS oracles are hard to design. Example application: quadcopter autopilot. The oracle needs to tell which physical trajectories are correct.

ploiting the *system identification* (SI) method used in *model predictive control* (MPC) [32]. MPC is an online control technology. It uses its SI method to learn the mathematical model of a concerned control system as a black-box; and uses the learnt model to predict future and make control decisions. MPC achieves great success in a broad range of control applications [47, 53, 74, 81, 113, 124, 138]. This empirically proves that the MPC-SI is highly capable of predicting black-box control systems behaviors, a quality critical to oracles.

1.2 Contributions of the Thesis

Based on the above intuitions, our main idea is as follows. To prepare the many source code execution traces for SFL, we emulate the control-CPS with the real cyber subsystem and the simulated physical subsystem (using the accurate physical subsystem model). While emulating a physical trajectory, we use MPC-SI to learn the control-CPS as a black-box and predict the physical trajectory in the near (emulation) future. We speculate that when all the easy-to-find bugs are removed (i.e. when automatic SFL tools are needed), physical trajectories shall usually comply with the predictions made by MPC-SI. Mismatches can be due to MPC-SI failures or bugs. But given the high empirical trustworthiness of MPC-SI, bugs are more likely. We can therefore use MPC-SI predictions as an oracle to label the

outliers of physical trajectories as incorrect, and the others as correct; and then label the corresponding source code execution traces accordingly. Guided by the above heuristics, this thesis makes the following contributions.

1. We propose a MPC-SI based oracle for control-CPS SFL.
2. We propose a corresponding source code execution trace preparation methodology, which can be carried out deterministically and automatically.
3. We compare the performance of our proposed approach (i.e. the above proposed oracle and methodology) with the main stream practice of human oracle approach. Evaluation results on classic control-CPS with real-life and artificial bugs show that our proposed approach significantly outperforms the human oracle approach in SFL accuracy, recall, and latency, and in oracle false positive/negative rates.

1.3 Thesis Outline

Based on the identified problems and objectives above, the thesis is organized as the followings:

Chapter 2 introduces background knowledge related to this thesis. We introduce a typical system architecture of control-CPS in details, and illustrate a testbed as an example. The contents of Chapter 2.2 is published in [MSN15]:

- Copyright ©2015 IEEE. Reprinted, with permission, from **Zhijian He**, Yanming Chen, Zhaoyan Shen, Enyan Huang, Shuai Li, Zili Shao, Qixin Wang, "Ard-mu-copter: A Simple Open Source Quadcopter Platform", in *Proceedings of the 11th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, Dec. 16-18, 2015, Shenzhen, China. DOI: 10.1109/MSN.2015.9.

Chapter 3 proposes an attitude fusion method of inertial and magnetic sensor. This method exploits a new fusion algorithm using various kinds of IMU sensors source, namely

gyroscope, accelerometer, and magnetometer. Compared to state-of-the-art attitude fusion approaches, this attitude fusion method addresses the indoor time-varying magnetic perturbation problem in a geometric view, which can be deployed to alleviate the bottleneck problem of acquiring precise heading in indoor attitude fusion. Through this method, we can be close to our target of considering our physical subsystem as bug-free. The contents of Chapter 3 is published in [TECS18]:

- Copyright ©2018 ACM. Reprinted, with permission, from **ZHIJIAN HE**, Yao Chen, Zhaoyan Shen, "Attitude Fusion of Inertial and Magnetic Sensor under Different Magnetic Filed Distortions", in *ACM Transactions on Embedded Computing Systems (TECS)*, Volume 17 Issue 2, April 2018, Article No. 48. DOI:10.1145/3157668.

Chapter 4 proposes our source code execution trace preparation methodology. Particularly, we explain explicitly our proposed oracle which labels the correct/incorrect physical control-CPS trajectories.

Chapter 5 evaluates our proposed oracle upon several representative control-CPSs. We discuss deeply about our evaluation results and threats to our validity.

Chapter 6 discusses related work in both control-CPS and SFL domain. The contents of Chapter 4, 5, 6 are under submission for publication:

- $[InSubmission]$ **ZHIJIAN HE**, Yao Chen, Enyan Huang, Qixin Wang, "MPC System Identification Method based Oracle for Control-CPS Software Fault Localization", in submission for conference publication.

Chapter 7 concludes the thesis and discuss the future work.

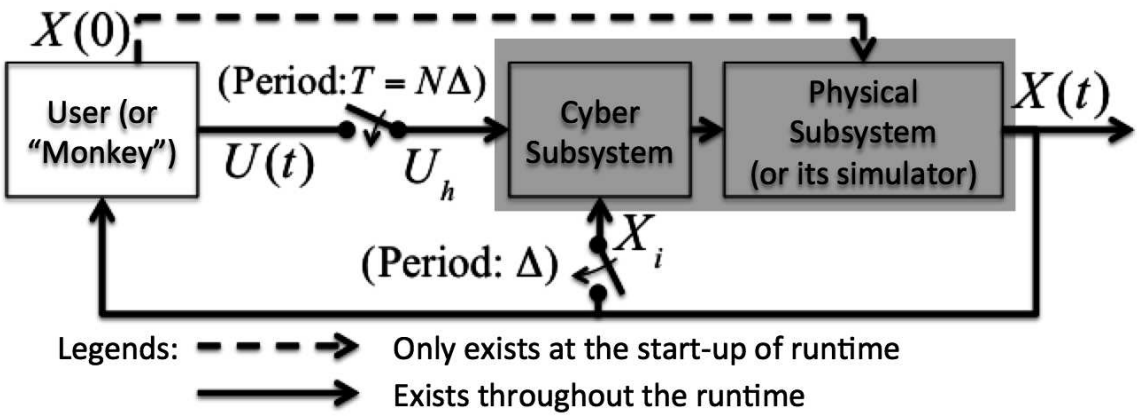
CHAPTER 2

SYSTEM ARCHITECTURE AND CONTROL-CPS TESTBED

In this chapter, we illustrate a typical control-CPS architecture and give an example to help readers better understand this architecture.

2.1 Control-CPS System Architecture

Fig. 2.1 illustrates a typical control-CPS architecture.



$U(t)$ is the user input to the control-CPS.

$X(t)$ is the state of the physical plant.

In case of emulation, the physical subsystem is replaced by its simulator, and the human user is often replaced by a human user mimicking program, aka "monkey".

Figure 2.1: A typical control-CPS architecture

In this architecture, the control-CPS consists of a *cyber subsystem* and a *physical*

subsystem (see the gray area in Fig. 2.1). User input at time $t \in [0, +\infty)$ to the control-CPS is $U(t) \in \mathbb{R}^q$. Depending on the cyber subsystem design, $U(t)$ can be an actuation signal, or a *target state* (aka *reference point*) the user wants the physical subsystem to reach. $U(t)$ is sampled before it enters the cyber subsystem. The hth ($h = 0, 1, \cdots$) sample happens at $t_h = hT$, where T is the *user input sampling period*. Correspondingly, we denote $U_h \stackrel{\text{def}}{=} U(t_h)$. The cyber subsystem zero-order holds the sample U_h for the duration $[hT, (h+1)T)$. That is, during $[hT, (h+1)T)$, the cyber subsystem takes constant U_h as the user input.

Inside of the physical subsystem lies the physical object, aka *plant*, being controlled. The state of the plant at time $t \in [0, +\infty)$ is denoted as $X(t) \in \mathbb{R}^n$. As a closed-loop system, $X(t)$ is also sampled by the cyber subsystem periodically. The ith ($i = 0, 1, \cdots$) sample happens at $t_i = i\Delta$, where Δ is the *plant sampling period* and

$$\Delta = T/N,$$

where $N \in \mathbb{Z}_{>1}$ is a preconfigured constant. Correspondingly, we denote $X_i \stackrel{\text{def}}{=} X(t_i)$. The cyber subsystem also zero-order holds the sample X_i for the duration $[i\Delta, (i+1)\Delta)$. To generate the many labeled source code execution traces for SFL, the user shall run the control-CPS many times. Usually, the human user is replaced by an automatic human-user mimicking program, aka "*monkey*", and the physical subsystem is replaced by its simulator (in this thesis, we assume the accurate model of the physical subsystem is known, hence the physical subsystem simulator is trustworthy). The cyber subsystem is still the buggy real implementation. Thus, the monkey, the buggy real cyber subsystem, and the trustworthy physical subsystem simulator form an emulation platform (see Fig. 2.1). We can run the emulated control-CPS millions of times without fatiguing any human user or real physical hardware, and we can run the emulations on parallel computers in a much faster pace than in real-time.

Each run of the emulated (or real) control-CPS generates a source code execution trace and a corresponding physical trajectory. To label the correctness of the source code

execution trace, we need to judge the correctness of the physical trajectory. This leads to the control-CPS oracle problem discussed in the next chapter.

2.2 An example of control-CPS testbed

Control-CPS links the physicals with cyber. In order to better understand a control-CPS, we need to understand its physicals.

In this section, we study the physics of quadcopter, which is an important control-CPS testbed used in our following chapters. Quadcopter is a helicopter with four equal motors placed in four corners. By maneuvering the four motors, a quadcopter control-CPS can fly to the intended location and height with intended attitudes.

The content of this section is published in [MSN15].

Location-Angular Control

The location control of a flying quadcopter is tightly coupled with the quadcopter's pitch, roll, yaw angular dynamics (sometimes the three angular dynamics are holistically called the "attitude" of the quadcopter) control [65]. As shown in Fig. 2.2, a quadcopter moves forward/backward if its pitch angle is non-zero; a quadcopter moves leftward/rightward if its roll angle is non-zero.

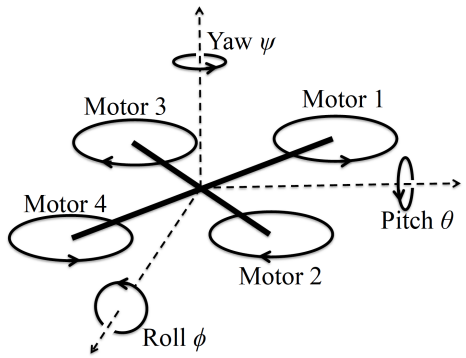


Figure 2.2: Three Angular Movement Dimensions and Propeller Motor Numbering of a Quadcopter

Therefore, the location-angular control takes a nested outer-inner control loop form, as shown in Fig. 2.3. The outer control loop is the location control loop; and the inner control loop is the angular control loop.

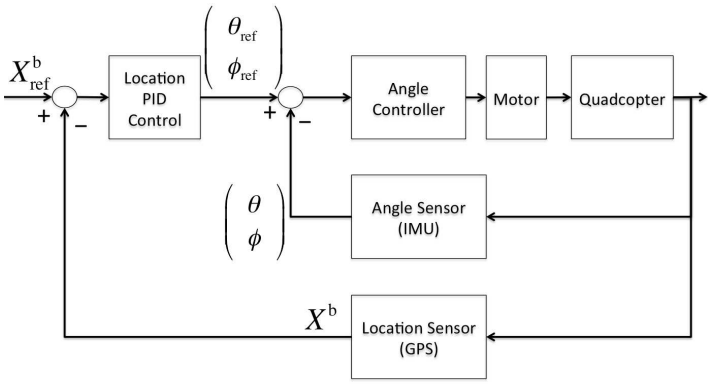


Figure 2.3: Location-Angular Nested Control Loops

The input to the location (i.e. outer) control loop is the desired location coordinates (in terms of body-oriented (x, y)-coordinates) $X^{\mathrm{b}}_{\mathrm{ref}} = (x^{\mathrm{b}}_{\mathrm{ref}}, y^{\mathrm{b}}_{\mathrm{ref}})^{\mathsf{T}}$. The control loop feedback $X^{\mathrm{b}} = (x^{\mathrm{b}}, y^{\mathrm{b}})^{\mathsf{T}}$ is the current quadcopter location coordinates (again, in body-oriented (x, y)-coordinates). The error $(X^{\mathrm{b}}_{\mathrm{ref}} - X^{\mathrm{b}})$ is fed to a PID controller to derive a desired attitude angle $(\theta_{\mathrm{ref}}, \phi_{\mathrm{ref}})^{\mathsf{T}}$, where θ_{ref} is the desired pitch angle and ϕ_{ref} is the desired roll angle. The desired pitch and roll angles serve as the input to the angular (i.e. inner) control loop, which will tilt the quadcopter's pitch angle θ and roll angle ϕ toward θ_{ref} and ϕ_{ref} respectively.

Formally, the control output of the outer control loop is described as follows.

$$\begin{aligned}
\theta_{\mathrm{ref}} &= k^{\mathrm{p}}_x(x^{\mathrm{b}}_{\mathrm{ref}} - x^{\mathrm{b}}) + k^{\mathrm{d}}_x(\dot{x}^{\mathrm{b}}_{\mathrm{ref}} - \dot{x}^{\mathrm{b}}) \\
&\quad + k^{\mathrm{i}}_x \int_0^t (x^{\mathrm{b}}_{\mathrm{ref}}(\tau) - x^{\mathrm{b}}(\tau))\mathrm{d}\tau, \\
\phi_{\mathrm{ref}} &= k^{\mathrm{p}}_y(y^{\mathrm{b}}_{\mathrm{ref}} - y^{\mathrm{b}}) + k^{\mathrm{d}}_y(\dot{y}^{\mathrm{b}}_{\mathrm{ref}} - \dot{y}^{\mathrm{b}}) \\
&\quad + k^{\mathrm{i}}_y \int_0^t (y^{\mathrm{b}}_{\mathrm{ref}}(\tau) - y^{\mathrm{b}}(\tau))\mathrm{d}\tau,
\end{aligned}$$

where k^{p}_x (also k^{p}_y), k^{i}_x (also k^{i}_y), and k^{d}_x (also k^{d}_y) are respectively the proportional, integral, and derivative control coefficients; t is the current time.

Without loss of generality, the two-level PID pitch angular control diagram is shown in Fig. 2.4 (the roll and yaw angular control follows the same principles). Again this consists of an outer and an inner control loop. The outer control loop is the pitch angle control loop. The input is the desired pitch $\theta_{\rm ref}$, the feedback is the sensed current quadcopter pitch θ. The error $(\theta_{\rm ref} - \theta)$ is fed to a PID controller to generate a desired pitch angular velocity $\omega_\theta^{\rm ref}$. The desired pitch angular velocity $\omega_\theta^{\rm ref}$ serves as the input to the inner control loop: the pitch angular velocity control loop. The feed back of the pitch angular velocity control loop is the sensed pitch angular velocity ω_θ (sensed by the gyro in the MPU-6050 IMU). The error $(\omega_\theta^{\rm ref} - \omega_\theta)$ is fed to a PID controller to generate the control signal u'_θ to be applied to quadcopter propeller motors (see Eq. (2.2) $\sim$ (2.5)). You can think of u'_θ as the component to adjust propeller motors to satisfy pitch angle control needs.

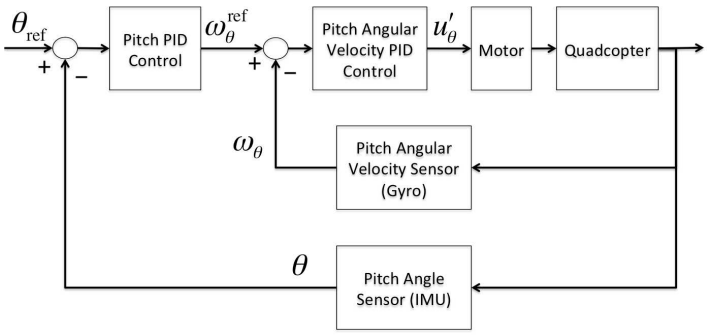


Figure 2.4: Two-Level PID Pitch Angular Control

Formally, we have

$$\begin{aligned}
\omega_\theta^{\rm ref} &= k_\theta^{\rm p}(\theta_{\rm ref} - \theta) + k_\theta^{\rm d}(\dot{\theta}_{\rm ref} - \dot{\theta}) \\
&\quad + k_\theta^{\rm i} \int_0^t (\theta_{\rm ref}(\tau) - \theta(\tau)) {\rm d}\tau, \\
u'_\theta &= k_{\omega_\theta}^{\rm p}(\omega_\theta^{\rm ref} - \omega_\theta) + k_{\omega_\theta}^{\rm d}(\dot{\omega}_\theta^{\rm ref} - \dot{\omega}_\theta) \\
&\quad + k_{\omega_\theta}^{\rm i} \int_0^t (\omega_\theta^{\rm ref}(\tau) - \omega_\theta(\tau)) {\rm d}\tau,
\end{aligned}$$

where $k_\theta^{\rm p}$ (also $k_{\omega_\theta}^{\rm p}$), $k_\theta^{\rm i}$ (also $k_{\omega_\theta}^{\rm i}$), and $k_\theta^{\rm d}$ (also $k_{\omega_\theta}^{\rm d}$) are respectively the proportional, integral, and derivative control coefficients.

Similarly, we can derive the control signal u'_ϕ and u'_ψ for roll and yaw angle control.

Height Control

The height control faces the similar challenge as angle control [65]. The quadcopter height reading is provided by the barometer (MS5611) and/or sonar (XL-MaxSonar EZ4). This reading is also unreliable. On the other hand, the vertical (i.e. height direction) acceleration readings a (provided by the accelerometer inside of the MPU-6050 IMU) is accurate. Therefore, the two-level PID angular control strategy of [65] also applies to height control [65]. We re-state the strategy as follows.

Fig. 2.5 describes the two-level height control loop. The outer loop is the height control loop. The input is the desired height h_{ref}. The feedback is the sensed quadcopter height h. The error $(h_{\text{ref}} - h)$ is fed to a PID controller to output a desired vertical acceleration a_{ref}. This a_{ref} together with gravitational acceleration g constitute the input to the inner control loop: the vertical acceleration control loop. The feedback of the vertical acceleration control loop is the sensed quadcopter's vertical acceleration a (sensed by the accelerometer inside of the MPU-6050 IMU). The error $(a_{\text{ref}} + g - a)$ is fed to a PID controller to create a control signal Δu_f to adjust the quadcopter propeller motors' throttle (see Eq. (2.1)(2.2) $\sim$ (2.5)). You can think of Δu_f as the component to adjust propeller motors to satisfy the vertical acceleration needs.

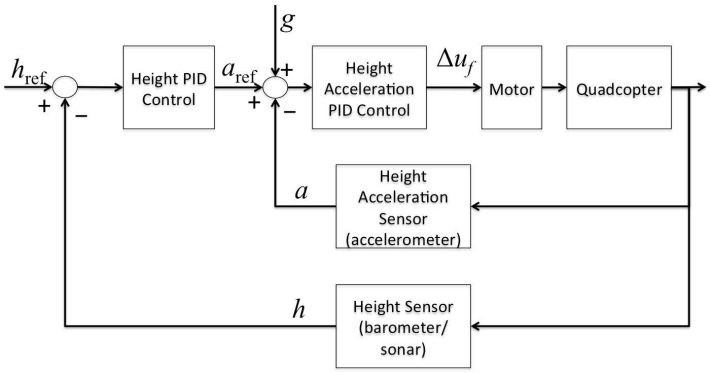


Figure 2.5: Two-Level PID Height Control

Formally, we have

$$\begin{aligned} a_{\text{ref}} &= k_h^{\text{p}}(h_{\text{ref}} - h) + k_h^{\text{d}}(\dot{h}_{\text{ref}} - \dot{h}) \\ &\quad + k_h^{\text{i}} \int_0^t (h_{\text{ref}}(\tau) - h(\tau))\mathrm{d}\tau, \\ \Delta u_f &= k_a^{\text{p}}(a_{\text{ref}} + g - a) + k_a^{\text{d}}(\dot{a}_{\text{ref}} + \dot{g} - \dot{a}) \\ &\quad + k_a^{\text{i}} \int_0^t (a_{\text{ref}}(\tau) + g(\tau) - a(\tau))\mathrm{d}\tau, \end{aligned}$$

where k_h^{p} (also k_a^{p}), k_h^{i} (also k_a^{i}), and k_h^{d} (also k_a^{d}) are respectively the proportional, integral, and derivative control coefficients.

Total Control Output

Finally, all the above control outputs converge to become the control output toward quadcopter propeller motors.

The total control output consists of two high level components: throttle and angular control adjustments. The throttle component u_f is to control the vertical acceleration (ultimately, height) of the quadcopter. It is the same to each of the four propeller motors (see Fig. 2.2). The height control output Δu_f affects the throttle component: u_f is updated as per

$$u_f(t + \mathrm{d}t) = u_f(t) + \Delta u_f(t). \tag{2.1}$$

The angular control adjustments are different to each of the four propeller motors. Without loss of generality, suppose we are adjusting the pitch. Suppose we want to increase the pitch angle (see Fig. 2.2), then propeller 1 and 3's motors should speed up, while propeller 2 and 4's motors should slow down. Meanwhile, we cannot change the total throttle component. Therefore, the pitch angle control signal u'_θ should be applied positively to motor 1 and 3, but negatively to motor 2 and 4.

Combining all the above considerations, suppose U_1, U_2, U_3, and U_4 respectively represent the raw total control signal applied to propeller motor 1, 2, 3, and 4, then the

update rules are

$$U_1(t + \mathrm{d}t) = u_f(t + \mathrm{d}t) + u'_\theta(t) + u'_\phi(t) - u'_\psi(t), \tag{2.2}$$

$$U_2(t + \mathrm{d}t) = u_f(t + \mathrm{d}t) - u'_\theta(t) + u'_\phi(t) + u'_\psi(t), \tag{2.3}$$

$$U_3(t + \mathrm{d}t) = u_f(t + \mathrm{d}t) + u'_\theta(t) - u'_\phi(t) + u'_\psi(t), \tag{2.4}$$

$$U_4(t + \mathrm{d}t) = u_f(t + \mathrm{d}t) - u'_\theta(t) - u'_\phi(t) - u'_\psi(t). \tag{2.5}$$

The above raw control signals U_1, U_2, U_3, U_4 are then range constrained and then normalized to create a PWM wave to drive the respective motors. A normalized value of 100% drives the motor to maximum speed, while 0% stops the motor.

Tuning PID Coefficients

PID controllers date back to 1890s. While proportional control provides fast response to reference set point changes and small disturbances, it cannot fully eliminate the impacts of steady disturbances, e.g. a stiff gale. To eliminate the steady disturbance impacts, we need integral control. Finally, derivative control constrains overshoot, hence improves control stability [11].

Generally, we first tune the proportional coefficients of the PID controllers to see if the quadcopter can achieve fast enough response time and acceptable overshoot. Then we increase the derivative coefficients to further reduce the overshoots. The integral control We will show the experiment result in the next section to prove our solution.

CHAPTER 3

ATTITUDE FUSION OF INERTIAL AND MAGNETIC SENSOR UNDER DIFFERENT MAGNETIC FILED DISTORTIONS

By virtue of gravity measurement from hand-held inertial measurement unit (IMU) sensor, current indoor attitude estimation algorithms can provide accurate roll/pitch dimension angles. Acquisition of precise heading is limited by the absence of accurate magnetic reference. Consequently, initial stage magnetometer calibration is deployed to alleviate this bottleneck in attitude fusion. However, available algorithms tackle magnetic distortion based on time-invariant surroundings, casting the post-calibration magnetic data into unchanged ellipsoid centered in the calibration place. Consequently, inaccurate fusion results are formulated in a more common case of random walk in time-varying magnetic indoor environment.

In this chapter, we proposes a new fusion algorithm from various kinds of IMU sensors source, namely, gyroscope, accelerometer and magnetometer. Compared to state-of-the-art attitude fusion approaches, we addresses the indoor time-varying magnetic perturbation problem in a geometric view. We propose an extend Kalman filter based (EKF-based) algorithm based on this detailed geometric model to eliminate the position-dependent affection of compass sensor. Experimental data demonstrate that, under different indoor magnetic distortion environment, our proposed attitude fusion algorithm has the maximum angle error of $2.02°$, outperforming $7.17°$ of gradient-declining-based (GD-based) algorithm. Additionally, this attitude fusion result is constructed in a low-cost hand-held arduino core based IMU device, which can be widely applied to embedded systems. This attitude fusion method can contribute to the target of considering the physical subsystem as bug-free in control-CPS.

The content of this chapter is published in in [TECS18].

3.1 Demand

Attitude and Heading Reference Systems (AHRS) hold key positions in vehicle, transportation, and unmanned aerial vehicle (UAV) designs [100], [66], [42], [31], [104]. Three-dimensional (3D) Euler angles expressing AHRS are calculated by comparing the rotation vector in a body frame with the corresponding vector represented in the earth frame. Combining the information generated from an inertial measurement unit (IMU), a quaternion is formulated from gyroscopes and accelerometers, to yield the rotation matrix which is the resource of Euler angles. Accurate pitch, roll, yaw angles are supposed to be obtained with correction from the gravity and the earth north magnetic field. However, suffering from bias error in gyroscope measurement, fusion results are influenced by static and dynamic residual errors from IMU [90]. Thus, advance calibration is performed to remove the static errors, while absolute reference from the gravity and the Earth's magnetic north direction [90] has to be leveraged to eliminate the dynamic gyroscope bias. Due to the intricate hardware drawback, these two references are exposed to noise easily [37], [110]. Elimination of the noise in absolute orientation reference, and the fusion of gyroscope, accelerometer, magnetometer data are the main challenges in AHRS [24], [93].

Motivating by wild applications in vehicle, robot and automatic control, researchers have spent a lot of effort to IMU-based devices [90], [142], [76], [91], [82]. Complementary filter algorithm proposed by [91] cast fusion problem into a weight formulation. With an initial guess of angle fusion integrated by gyroscope, this formulation contains the other guess from accelerometer, generating an optimization problem with total weight equivalent to 1. However, this approach is primarily limited by the low accuracy of dynamic angular fusion. Inspired by complementary filter, gradient declining-based (GD-based) method [90] simplifies the adjustment of parameter and models the error originating from the gradient direction of the change in quaternion. Through the correction using absolute reference from gravity and magnetometer, this algorithm achieves less than 0.6° static root mean square (RMS) error and 0.8° dynamic RMS error. The necessity of GD-based algorithm mainly comes from two aspects: (1) the post calibration magnetic field reference points to the

perfectly Earth's north, (2) dynamic magnetic sensor measurement is constructed in an time-invariant surrounding. Latter assumption easily encounters limitation because compass sensor data present an intrinsic dependence on position especially in an indoor environment. To adopt into the first-order approximation propagation error in GD-based algorithm, unscented Kalman filter (UKF) [142] approach is proposed to minimize the covariance of the state vector. Better robustness is associated with higher order approximation propagation. However, time-invariant compass north is still assumed on UKF fusion.

To avoid the hypothesis of magnetic measurement from time-invariant surrounding, practical fault models of compass sensor data are exploited in [121], [122], [135], [62], [140], [38]. These methods account for all parameters of magnetic distortion, namely, hard/soft iron error, nonorthogonality error, scaling and bias error, as well as wideband noise error [122]. In addition, they cast the real compass sensor data into a least square problem which approximates the compass data into an ellipsoid manifold. Compared with classic calibration approaches [27], [105], geometric approximation methods address the joint effect of all kinds of perturbations not limited to the xy plane, facilitating the calculation of the yaw angle without external reference. However, the fault model of the magnetic data is not clarified when sensing distance exceeds a certain range with respect to the calibration place. This phenomenon implies that the real-time magnetic distortion problem is not addressed.

In this chapter, we focus on exploiting the dynamic characters of compass measurement in an indoor environment which obtains complex unpredictable perturbations in vicinity of the sensor. Given that the non-observable state nature of yaw angle [76], an extended Kalman filter-based (EFK-based) approximation algorithm is deployed to fuse the gyroscope data and magnetic rotation data to eliminate position-dependent magnetic disturbances. Another category of navigation solution to conquer magnetic distortion is so-called biomechanics approaches [111], [40], [56] , [64], [112]. These approaches utilize pedestrian dynamic pattern, which includes walk stationary period or certain aiding events used in conjunction with IMU algorithm fusion. The main limitation of these approaches lies on the position of such portable device placed on the human body. Mounting on the foot [64] is almost compulsory for the accurate result although certain methods [49], [48] can alleviate this constraint.

However, more than 87% of intances [79], [17] the users commonly place the device in hand or in bag instead, yielding unreliable additional physical measurements or specificities of the human walk detection.

The Kalman filter has been widely accepted by the majority of fusion algorithms in automatic control [141], [94], [78]. Characterized by irregular geometric shape, compass data are not allowed to be substituted into classic Kalman filter regressions. Thus, our approach aims to present a non-linear model of dynamic fusion of accelerometer, gyroscope, and magnetometer data under time-varying magnetic disturbance. Precise roll/pitch angles are evaluated in the first step with the accurate measurement of the gravity, while a fault model of the non-observable yaw angle is formulated in the second step. Based on the variance threshold of position-dependent compass data, our proposed attitude fusion algorithm yields a set of EKF-based equations which improve the accuracy of yaw angle despite the different magnetic field distortions.

In this chapter, we develop a practical indoor attitude fusion algorithm for IMU independent of different magnetic distortion. To our best of knowledge, our geometric method is the first time to be leveraged to construct a reliable EKF-based attitude fusion. Specifically, the contributions of this work are listed as follows:

1. We deeply exploit the problem of GD-based attitude algorithm in the indoor environment. We discuss the premise of GD-based algorithm by analyzing the essentials of this algorithm. Via studying the magnet field change in the geometric view, we figure out that this premise is not reliable especially in the indoor environment.

2. We propose a robust EKF-based algorithm that eliminates complex indoor environment magnetic perturbation. Using the variance of magnetic sensor measurement, we enhance our proposed EKF-based algorithm to eliminate the unpredictable distortion of the heading.

3. We evaluate our approach on a widely used arduino core platform to verify the robustness of our algorithm.

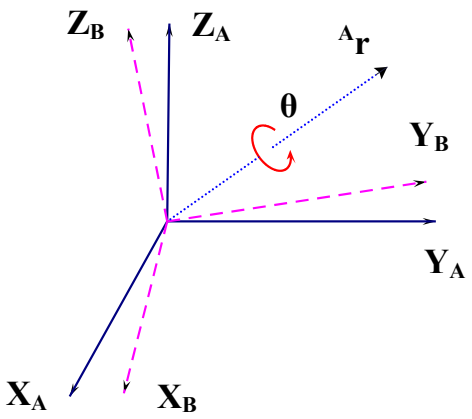


Figure 3.1: Frame A rotates around a vector r with angle θ to achieve frame B

3.2 Preliminaries

To express the rotation in 3D space (Figure 3.1), a unit quaternion is deployed to describe the property that frame A rotates around a vector ${}^{A}r$ with angle θ to achieve frame B as follows:

$$ {}^{A}_{B}q = \begin{bmatrix} \cos\frac{\theta}{2} & -r_x \sin\frac{\theta}{2} & -r_y \sin\frac{\theta}{2} & -r_z \sin\frac{\theta}{2} \end{bmatrix} \tag{3.1} $$

where $[\, r_x \; r_y \; r_z \,]$ is the rotation axis vector ${}^{A}r$ value expressed in frame A. We express frame A using the leading superscript denoting the frame with reference to. By contrast, frame B in the leading subscript denotes the frame being described respectively. θ is the rotation angle around the rotation axis vector ${}^{A}r$ in the counter-clockwise direction.

Additionally, inverse rotation can be expressed by changing the script places of A and B, denoting as ${}^{B}_{A}q$ [90], which is equivalent to the quaternion conjugate ${}^{A}_{B}q^{*}$:

$$ {}^{A}_{B}q^{*} = {}^{B}_{A}q = \begin{bmatrix} q_1 & -q_2 & -q_3 & -q_4 \end{bmatrix} \tag{3.2} $$

Using the quaternion conjugate, $*$, and cross-product of quaternions, $\otimes$, the rotation operation in 3D space can be described: A vector ${}^{A}v$, expressed in frame A, is represented in frame B as ${}^{B}v$, using the property of conjugate and cross-product of quaternions:

$$ {}^{B}v = {}^{A}_{B}q \otimes {}^{A}v \otimes {}^{A}_{B}q^{*} \tag{3.3} $$

The rotation angles are obtained from the alignment with frame A after a sequence of rotates. The Euler angles of this respective rotate operations are defined as follows:

$$\psi = \arctan(2q_2q_3 - 2q_1q_4, 2q_1^2 - 1 + 2q_2^2) \tag{3.4}$$

$$\theta = -\sin^{-1}(2q_2q_4 + 2q_1q_3) \tag{3.5}$$

$$\phi = \arctan(2q_3q_4 - 2q_1q_2, 2q_1^2 - 1 + 2q_4^2) \tag{3.6}$$

3.3 Solution Overview

In this section, an overview of our proposed attitude fusion algorithm is described, to abstract the solution which contains complex quaternion transformation and geometric approximation. We separate our solution into two steps, namely, initial calibration step (Figure 3.2) and run-time step (Figure 3.3). Initial calibration step provides an initial fault model with respect to gyroscope, accelerometer and compass, while run-time step provides precise 3D attitude fusion using these 3 kinds of data. Without loss of generality, no special platform or sensor is previously selected, implying random bias error in the gyroscope or accelerometer. These intrinsic manufacturing drawbacks are expected to be removed in the initial step. In the run-time step, dynamic errors from both gyroscope and compass are estimated and eliminated.

3.3.1 Initial calibration step

Step 1: Collected from gyroscope and accelerometer data via I2C bus [73], our proposed algorithm provides a data check interface to check whether the data are within the range of their physical constraint. Half a thousand of censored samples, stored in the buffer of microprogrammed control unit (MCU), are averaged to construct a basic estimation of sensor bias. This statistical result provides an initial compensation for the static error of both sensors, addressing the accurate measurement of the accelerometer. However, precise gyroscope measurement still requires dynamic bias elimination in the run-time step.

Step 2: Contrary to statistical approach in gyroscope and accelerometer, an initial

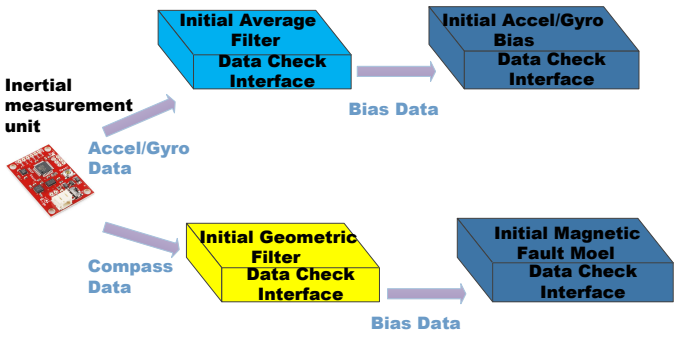


Figure 3.2: Initial calibration step of our proposed approach

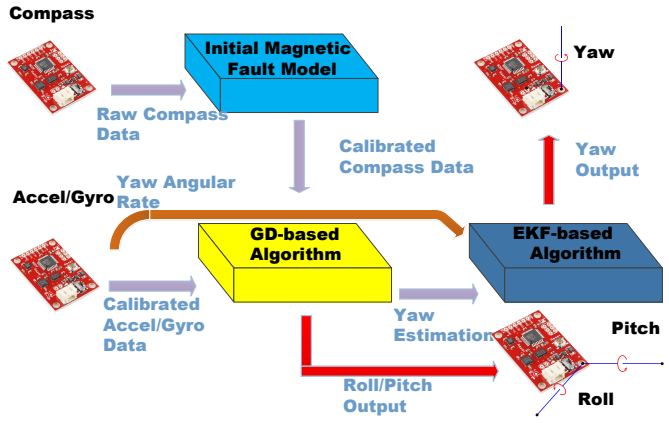


Figure 3.3: Run-time step of our proposed approach

magnetic fault model is derived geometrically due to the gauss measurement feature of a compass sensor [122]. Evaluated by data check interface of a compass, magnetic measurement is filtered if it exceeds certain range of compass sensitivity. The parameters of the initial magnetic fault model, including hard/soft iron effects, nonorthogonality, scaling and bias and wide-band noise, are calculated using maximum likelihood estimator (MLE) method [122] after the first 1000 samples are collected. Recalibration command is sent to the IMU sensor when the model construction fails.

3.3.2 Run-time step

Step 1: We input the post-calibration gyroscope and accelerometer data as the fusing resource of GD-based attitude algorithm [90]. A 3D attitude estimation is generated from the GD-based attitude fusion module under the correction of gravity. In aspect of yaw direction, raw data is derived from the compass sensor, corrected by the initial geometric fault model formed in the initial calibration step [122]. Calibrated compass measurement serves as the fusing source of GD-based attitude algorithm, forming the yaw estimation as the input of our proposed EFK-based algorithm.

Step 2: Influenced by time-varying indoor magnetic distortion, yaw estimation from GD-based algorithm obtains unpredictable error, which results in the deployment of our proposed EKF-based algorithm. The practical details of indoor magnetic distortion will be described in Section 3.4. Constrained by compass feature of measuring directional magnetic strength in gauss, raw magnetic measurement is not allowed to use in yaw angle EKF equations. We propose a non-linear EKF-based algorithm which inputs the raw yaw angular rate from IMU to construct the process function. Moreover, we derive the yaw estimation from GD-based algorithm to establish the observation function. This yaw estimation aids yaw angle correction via model predictor and data assimilation corrector. Roll/pitch angle results are the output from GD-based algorithm after the correction using gravity, while yaw angle is generated from EKF-based module with the correction of the Earth‘s north magnetic field. Details of our proposed EKF-based algorithm will be described in Section 3.5.

3.4 Attitude Representation

In this section, we use the quaternion as the agent to represent the orientation of a hand-held IMU. We list the quaternion form containing the compass, gyroscope and accelerometer data required by GD-based algorithm. Limitations of GD-based algorithm are determined under time-varying magnetic field distortion. Based on these limitations, we investigate deeply the change in the raw magnetometer measurement and the variance of such data in the disruptive

indoor environment. The results show that the yaw angle correction is unreliable if the time-invariant compass calibration is used.

3.4.1 GD-based Algorithm

Presented in quaternion form, GD-based algorithm [90] is an efficient attitude fusion algorithm that combines gyroscope data and observation data. In this section, we describe the representative parts of GD-based algorithm and formulate the hypothesis of this fusion algorithm.

North East Down coordinate

Using the capability of pointing the Earth North direction via compass sensor, we make a convention of utilizing the quaternion principles in north east down (NED) coordinate. NED coordinate consists of three axes, namely: one axis points to the direction of Earth's north, one along the Earth's eastern axis, and one points down as the same direction of gravity [127]. In this thesis, the Earth's frame refers to the NED frame, while the sensor frame stands for the frame attached at the center of the device mass.

Rotation Quaternion From A Gyroscope

Initially, the quaternion describing the orientation of a rigid body is integrated by its change rate during every sample period:

$$\overset{S}{E}q_{est,t} = \overset{S}{E}q_{est,t-1} + \overset{S}{E}\dot{q}_{est,t}\Delta t \tag{3.7}$$

where Δt is the sampling period, and $\overset{S}{E}q_{est,t-1}$ is the estimation of orientation at time t-1. $\overset{S}{E}\dot{q}_{est,t}$ denotes the change rate of the quaternion during the sample period t. This change rate value can be considered as unchanged during each sample period if proper frequency is chosen.

Given that the sample period is predefined according to particular IMU datasheet,

the initial estimated quaternion ${}^{S}_{E}\dot{\hat{q}}_{est,t}$, which describes the change rate of the Earth's frame relative to the sensor frame, is computed as the cross-product between the orientation at time $t-1$, ${}^{S}_{E}q_{est,t-1}$, and the turning rate of the IMU sensor, ${}^{S}\omega$ [90] as follows:

$${}^{S}_{E}\dot{\hat{q}}_{est,t} = \frac{1}{2}{}^{S}_{E}q_{est,t-1} \otimes {}^{S}\omega \tag{3.8}$$

Due to the measurement of IMU gyroscope sensor works in 3D space only, the first element is recast to 0 in quaternion in order to be adaptive for a unified expression:

$${}^{S}\omega = \begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \end{bmatrix} \tag{3.9}$$

where ω_x, ω_y, and ω_z indicate the ideal measurement of gyroscope in 3D space respectively. However, two categories of bias errors are obtained in gyroscope, namely, static and dynamic errors. In this paper, the terms ω_x, ω_y, ω_z express the angular rates whose initial static errors has been removed via calibration in the sensor frame. In eliminating the dynamic bias error, GD-based method is deployed ,using the correction of absolute references from gravity and the Earth's magnetic north direction, which will be explained in the following section.

Rotation Quaternion From Absolute Reference

Smooth and accurate attitude estimation trajectory should be generated with a precondition that the accumulated dynamic gyroscope bias error is eliminated. Following the transformation result of Eq. (3.3), the gravity direction can be represented in the sensor frame if no external force is added on the sensor, as the same to the Earth's north direction. This principle formulates an optimization problem in terms of orientation quaternion:

$$f({}^{S}_{E}q, {}^{E}d, {}^{S}s) = {}^{S}_{E}q^{*} \otimes {}^{E}d \otimes {}^{S}_{E}q - {}^{S}s \tag{3.10}$$

where ${}^{E}d$ indicates the absolution reference direction from both gravity and the Earth's north. ${}^{S}s$ represents the measurement in the sensor frame. ${}^{S}_{E}q$, which is the independent variable of this function, is the quaternion expressing the current orientation. With the existent of dynamic gyroscope bias error, the target of formulating Eq. (3.10) is to minimize the gap between the estimated sensor measurement inferred by absolute reference ${}^{E}d$ and actual

measured direction of the according field ${}^{S}s$ in sensor frame. Thus, an accurate estimation of IMU attitude is provided as follows:

$$\min f({}^{S}_{E}q, {}^{E}d, {}^{S}s) \tag{3.11}$$

With the independent variable ${}^{S}_{E}q$, the GD-based algorithm computes the gradient of $f({}^{S}_{E}q, {}^{E}d, {}^{S}s)$ in Eq. (3.11) as the direction of the estimated dynamic gyroscope bias error:

$$\nabla f({}^{S}_{E}q, {}^{E}d, {}^{S}s) = J^{T}({}^{S}_{E}q, {}^{E}d) f({}^{S}_{E}q, {}^{E}d, {}^{S}s) \tag{3.12}$$

$${}^{S}_{E}\dot{q}_{\epsilon,t} = \frac{\nabla f}{\|\nabla f\|} \tag{3.13}$$

Consequently, accurate gyroscope data can be accessed if the estimated quaternion derivative ${}^{S}_{E}\hat{\dot{q}}_{est,t}$ of Eq. (3.8) removes the residual in this estimated error direction:

$${}^{S}_{E}\dot{q}_{est,t} = {}^{S}_{E}\hat{\dot{q}}_{est,t} - \beta {}^{S}_{E}\dot{q}_{\epsilon,t} \tag{3.14}$$

$${}^{S}_{E}q_{est,t} = {}^{S}_{E}q_{est,t-1} + {}^{S}_{E}\dot{q}_{est,t}\Delta t \tag{3.15}$$

Eq. (15) computes the gradient of the optimising problem function $f({}^{S}_{E}q, {}^{E}d, {}^{S}s)$, using ${}^{S}_{E}\dot{q}_{\epsilon,t}$ as the description of this normalized gradient. This gradient equates to function $f({}^{S}_{E}q, {}^{E}d, {}^{S}s)$ multiple by its Jacobian $J^{T}({}^{S}_{E}q, {}^{E}d)$. Inside the function $f({}^{S}_{E}q, {}^{E}d, {}^{S}s)$, ${}^{E}d$ refers to the predefined absolute reference from both gravity and the Earth‘s north direction while ${}^{S}s$ means the real time data measured from the accelerometer and compass sensor in the sensor frame accordingly. A proper filter gain β is chosen to satisfy the elimination of the dynamic residual of gyroscope data, as shown in Eq. (14), which leads to the correct integration of attitude from each sample period in Eq. (15). Ultimately, the quaternion result ${}^{S}_{E}q_{est,t}$, which is the attitude at time t, is substituted into the rotation matrix of Eq. (10) to calculate the correct Euler angles.

3.4.2 Limitation of GD-based Algorithm

Accurate sensor measurement, ${}^{S}s$, as well as the correct reference, ${}^{E}d$, are necessary for the correct fusion result of GD-based algorithm. An accelerometer, which measures the force centered in IMU, fits this requirement after statistical calibration, while a magnetometer is difficult to guarantee the indoor measurement. Exposed to distortions such as bias, hard iron, soft iron, and nonorthogonality in the sensor frame [122],the measurement of the magnetic field is subjected to inaccurate data collection, leading to unpredictable fault in GD-based algorithm. In this section, we exploit the fault model of indoor magnetic data and prove its dependence on position.

Magnetic Fault Model

A magnetometer is a sensor measures the strength and direction of the combined magnetic fields of the Earth and objects nearby. Measurement of magnetic field is subjected to distortion such as bias, hard iron, soft iron, and nonorthogonality in the vicinity of sensor [122]. The hard iron errors, expressed as b_{HI}, refer to the presence of nearby object magnetic fields around the sensor, producing magnetic measurement offset errors which are constant in the sensor frame. The soft iron errors refer to the presence of ferromagnetic materials around the sensor, which are related to scaling offset errors. We describe soft iron errors as follows [122]:

$$h_{SI} = C_{SI}{}^{S}_{E}R^{E}h \tag{3.16}$$

where h_{SI} is the soft iron transformation matrix, C_{SI} is the soft iron scaling error matrix, ${}^{S}_{E}R$ is the rotation matrix from earth frames to sensor frame, ${}^{E}h$ is the expected original earth magnetic field data.

Nonorthogonality refers to the measurement misalignment angles between sensor frame and respective output axis. The time-invariant magnetic fault model can be transformed to a matrix [122]:

$$C_{NO} = \begin{bmatrix} 1 & 0 & 0 \\ \sin(\rho) & \cos(\rho) & 0 \\ \sin(\phi)\cos(\lambda) & \sin(\lambda) & \cos(\phi)\cos(\lambda) \end{bmatrix} \tag{3.17}$$

where ρ, ϕ, λ are the misalignment angles between the y-sensor and the y-axis, z-sensor and the x-z plane, z-sensor and the y-z plane, respectively.

In addition to the above distortion, the practical Earth's magnetic field data sample includes wide-band noise and other sensor-specific characteristics. Thus, we can model the magnetic data in a concise formula [122]:

$$h_{ri} = C^S \bar{h}_i + b + n_{mi} \tag{3.18}$$

where h_{ri} is the magnetic measurement in the sensor frame. b equates to the total bias error containing soft/hard iron bias distortion under the effect of nonorthogonality. C refers to the scaling offset errors caused by nonorthogonality and soft-iron. n_{mi} can be considered as wide-band noise and other sensor-specific characteristics.

Position-dependent Magnetometer Data

With the assumption of ideal compass measurement, $\bar{h}_i$, which fills the surface of a sphere, the fault model of magnetic filed in Eq. (3.18) implies that the compass measurement, h_{ri}, should be on the surface of an ellipsoid in the geometric view. This conclusion is validated by Figure 3.4 which shows an ellipsoid centered at $(202, 4, -108)$ except from certain noise measurements.

This proof facilitates the approximation of all measurement data into a comprehensive shape of an ellipsoid, to yield an ultimate purpose of getting the according sphere [52]:

$$(X - U)^T R^T D R (X - U) = 1 \tag{3.19}$$

In Eq. (3.19), X is a real set of points in the sphere of an ellipsoid, U is the ellipsoid center, R is the matrix representing the ellipsoid orientation, D is a diagonal matrix with

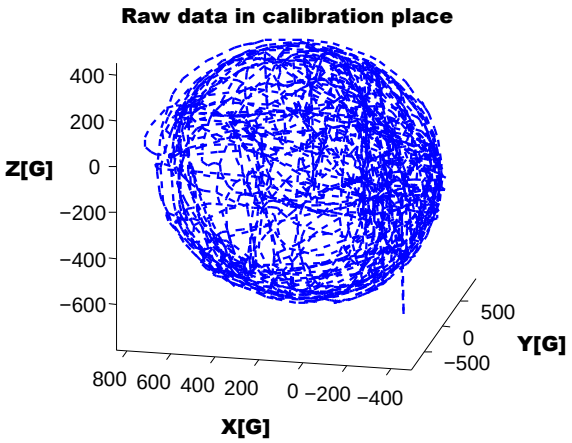


Figure 3.4: Raw magnetic data in the calibration place

diagonal numbers representing the half-lengths of axes of the ellipsoid. For example, an axis-aligned ellipsoid with the center at the origin has the equation representation as: $(x/a)^2 + (y/b)^2 + (z/c)^2 = 1$. As for this ellipsoid, U is $(0, 0, 0)$, R is the identity matrix, and D = $diag(1/a^2, 1/b^2, 1/c^2)$. However, under the perturbation of Gaussian noise, least squares method is leveraged to achieve rational approximation as follows [52]:

$$E(U, R, D) = \sum_{i=1}^{m} (L_i - r)^2 \tag{3.20}$$

where L_i is the distance from measurement data to the ellipse center U, r is the distance of respective point X_i to the center. The function E is minimized iteratively using Powell's direction-set method to search for a minimum.

3.4.3 Indoor environment affection

Explicit constraint from the parameterized magnetic fault model, which is derived from Eq. (3.20), is formed by the initial calibration only once at the beginning. With ferromagnetic materials in the vicinity of IMU, time-varying magnetic field in the indoor environment cast the measurement into inaccurate h_{ri} if the old fault model parameters are used. In this part, we investigate the heading estimation using the Earth's magnetic field in the complex indoor environment. Affection of magnetic perturbation is deeply exploited in the aspect of orientation accuracy.

Yaw Angle Calculation

With precise gravity correcting the pitch/roll angles, the bottleneck of AHRS casts caution to the calculation of an accurate yaw angle. Conventionally, post-calibration magnetometer data can provide the yaw angle information after it has been normalized as follows:

$$^{E}yaw = atan2(^{E}magy, ^{E}magx) \tag{3.21}$$

where $^{E}magx$, $^{E}magy$, $^{E}magz$ denotes the 3 axis measurements from the magnetometer in the earth frame. The yaw angle calculated by Eq. (3.21) describes the heading angle between current sensor and the Earth's north direction. To comply the principle of quaternion, we convert the Earth's north direction into quaternion form as the absolute reference as follows:

$$^{E}\hat{b}_t = \begin{bmatrix} 0 & \sqrt{^{E}magx^2 + ^{E}magy^2} & 0 & ^{E}magz \end{bmatrix} \tag{3.22}$$

$^{E}\hat{b}_t$ indicates the Earth's north direction reference, which serves as one of the components of ^{E}d in Eq. (3.11).

Disruptive indoor environment

Given the intrinsic characteristic of the magnetometer hardware, measurements from indoor environment are severely contaminated by instrumentation and environmental issues, rendering inaccurate fusion of the IMU sensors reading. In order to investigate deeply the effect of magnetic perturbation inside buildings, we construct our experiment in a room full of ferromagnetic distortion. From Figure 3.5, the experiment room is crowded with computers, ferromagnetic materials, such as electric wire and metallic boxes.

This environment is to guarantee that our portable device is surrounded by electromagnetic devices or magnetization of manmade structure in the presence of an external magnetic field. We evaluate the magnetic distortion effect in two aspects: magnetic data ellipsoid center change and heading angle change.

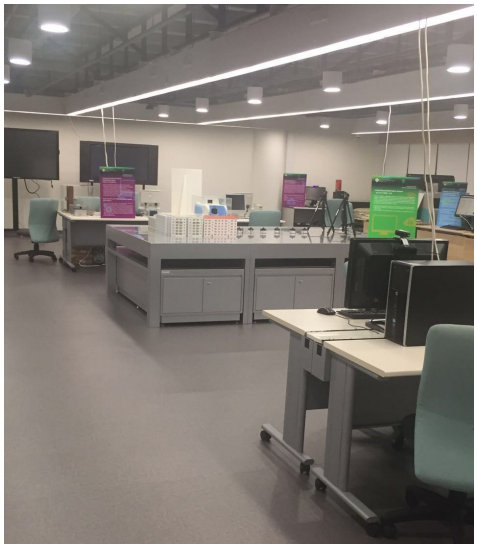


Figure 3.5: Disruptive indoor environment

Change of magnetic model

We collect raw magnetic sensor measurement data within the same room in Figure 3.4 (the calibration place), and select one representative set of data to demonstrate the position-dependent nature: From Figure 3.6, the raw measurement indicates explicitly that this el-

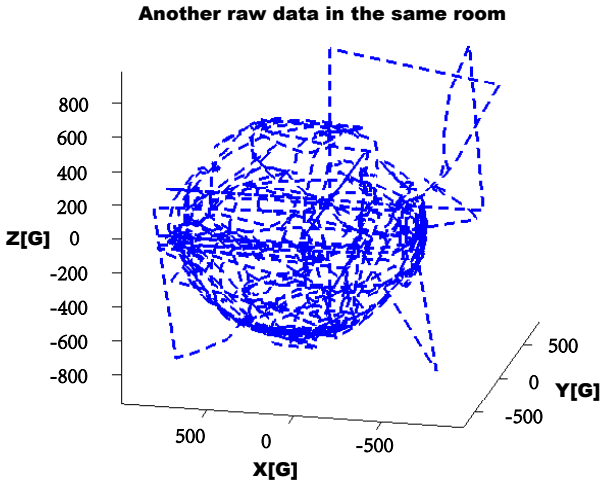


Figure 3.6: Representative set of raw magnetic data in the same room of calibration

lipsoid has a similar shape of figure 3.4 but centered at $(85, 8, -200)$. The comparison result between Figure 3.6 and Figure 3.4 indicates the magnetic distortion should not be considered as time invariant especially in an indoor environment. All kinds of magnetic pertur-

bations, such as hard/soft iron error, nonorthogonality error, and noise affection should be reappraised. If the data in Figure 3.6 uses the calibration result of Figure 3.4, GD-based fusion algorithm is bound to output error angle for the reason of having different ellipsoid centers. That is, GD-based fusion algorithm is not able to handle time-varying magnetic perturbations.

Additionally, we walk down a straight line of 70 decimeters indoors while guaranteeing the device pointing to the same direction. Using Eq. (3.21), heading angles are calculated under unpredictive magnetic affection. We demonstrate the magnetic distortion between clean heading and contaminated heading from Figure 3.7. The calculated results from measured data suffer severely by the ferromagnetic material distortion, and the effect of inaccuracy is in direct proportion to the distance from calibration place. Random results are included in the heading calculation, rendering the urgent need to rule out the magnetic perturbations. The nature of indoor and outdoor affection on magnetometer is comprehensively summarized in the survey of [16].

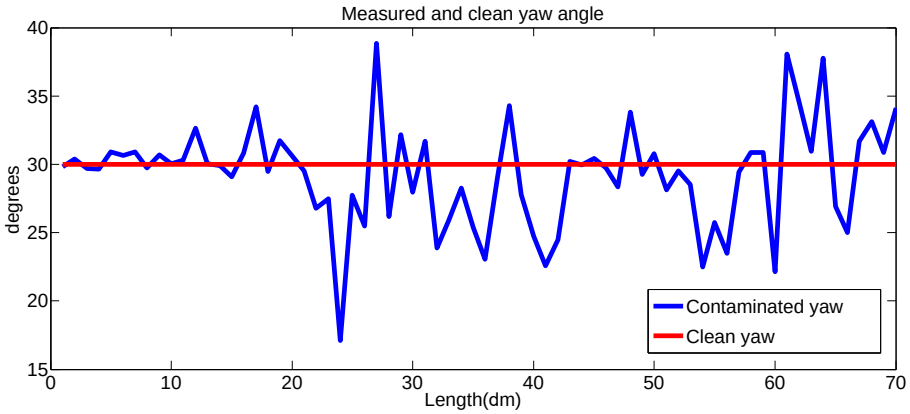


Figure 3.7: Heading affection by indoor environment

3.5 Time-varying Magnetic Distortion Solution

Based on the demonstration of the position dependence of the compass sensor, we propose an EKF-based approach to rule out unpredictable magnetic perturbation in this section. Exploiting the principles of calculating the Earth's magnetic north in both Eqs. (3.21) and (3.22),

the calculated quaternion of Earth's north direction is generated. Substituting this calculated Earth's north into GD-based attitude fusion algorithm formulates the yaw angle observation. This process facilitates the establishment of an EKF-based nonlinear system to eliminate the time-varying magnetic perturbations via its model predictor and data assimilation corrector. However, variance of magnetometer is unpredictable indoors, yielding a variance-based design in section 3.5.2 to enhance the EFK-based attitude fusion.

3.5.1 EFK-based Algorithm

In this section, we detail the process and observation equations in our proposed EKF-based approach. Additionally, a geometric approximation fault model is leveraged to solve the conversion between the measurement of the compass sensor and yaw angle observation.

We consider the following target linear system:

$$^{E}x_k = {}^{E}x_{k-1} + {}^{E}_{S}R_{k-1}{}^{S}g_{yaw,k-1}\Delta t + w_{k-1} \tag{3.23}$$

$$^{E}z_k = {}^{E}x_k + v_{k-1} \tag{3.24}$$

where state vector $^{E}x_k$ is the yaw angle expressed in the earth frame at time k, ${}^{E}_{S}R_{k-1}$ denotes the rotation matrix described the earth frame refer to sensor frame at time k-1, $^{S}g_{yaw,k-1}$ is the gyroscope measurement in yaw direction at time k-1 in the sensor frame, Δt is the time step in this EKF equations, w_{k-1} is the process noise vector from gyroscopes, $^{E}z_k$ is the observation vector which means magnetic measurement in the earth frame at time k, v_{k-1} is the measurement noise vector from compass. Basically, ${}^{E}_{S}R_{k-1}{}^{S}g_{yaw,k-1}\Delta t$ means the change in yaw angle integrated by angular rate in terms of the Earth's frame. The state vector is corrected by the observation yaw angle $^{E}z_k$.

No direct observation yaw angle is derived from compass sensor, hindering the formulation of the linear relationship between state vector $^{E}x_k$ and observation vector $^{E}z_k$. To obtain the observation yaw angle $^{E}z_k$, a conversion between magnetic field measurement from compass and the according yaw angle is necessary to be derived from the geometric approximation and the quaternion to Euler angle mapping computation as follows:

$$^{S}\bar{h}_i = C^{-1}(h_{ri} - b - n_{mi}) \tag{3.25}$$

$$\begin{bmatrix} 0 & ^{S}magx & ^{S}magy & ^{S}magz \end{bmatrix} = \frac{^{S}\bar{h}_i}{\left\| ^{S}\bar{h}_i \right\|} \tag{3.26}$$

We use the initial magnetic fault model of Eq. (3.18) to calculate the post-calibration compass measurement $^{S}\bar{h}_i$ in Eq. (3.25), to yield a unified measurement in the sensor frame in Eq. (3.26), which construct the measurement quaternion in the sensor frame also:

$$^{S}s = \begin{bmatrix} 0 & ^{S}magx & ^{S}magy & ^{S}magz \end{bmatrix} \tag{3.27}$$

Based on the Earth's magnetic north reference calculated in Eq. (3.22), both ^{S}s and $^{E}\hat{b}_t$ are substituted in the following equations to generate the yaw observation:

$$\nabla f(^{S}_{E}q, ^{E}d, ^{S}s) = J^T(^{S}_{E}q, ^{E}d) f(^{S}_{E}q, ^{E}d, ^{S}s) \tag{3.28}$$

$$^{S}_{E}\dot{q}_{\epsilon,t} = \frac{\nabla f}{\|\nabla f\|} \tag{3.29}$$

$$^{S}_{E}\dot{q}_{est,t} = \dot{^{S}_{E}\hat{q}}_{est,t} - \beta\, ^{S}_{E}\dot{q}_{\epsilon,t} \tag{3.30}$$

$$^{S}_{E}q_{est,t} = ^{S}_{E}q_{est,t-1} + ^{S}_{E}\dot{q}_{est,t}\Delta t \tag{3.31}$$

$$\psi = \arctan(2q_2q_3 - 2q_1q_4, 2q_1^2 - 1 + 2q_2^2) \tag{3.32}$$

The whole system can now use Kalman Filter iteration to get a smooth-change yaw angle under the position-dependent magnetic distortion. The initial state $^{E}x_0$ is a random yaw angle in the starting calibration place with mean $^{E}\mu_0 = E[^{E}x_0]$ and covariance $P_0 = E[(^{E}x_0 - ^{E}\mu_0)(^{E}x_0 - ^{E}\mu_0)^T]$.

According to the law of EKF iteration [60], we have the optimal yaw estimation through the following steps (Details are listed in the Appendix .3):

Initialization:

$$^{E}\mu_0 = E[^{E}x_0] \tag{3.33}$$

$$P_0 = E[(^{E}x_0 - {}^{E}\mu_0)(^{E}x_0 - {}^{E}\mu_0)^T] \tag{3.34}$$

Model Predictor:

$$^{E}x_k^f \approx {}^{E}x_{k-1}^a \tag{3.35}$$

$$P_k^f = P_{k-1} + Q_{k-1} \tag{3.36}$$

Data assimilation corrector:

$$^{E}x_k^a \approx {}^{E}x_k^f + K_k(^{E}z_k - {}^{E}x_k^f) \tag{3.37}$$

$$P_k = (I - K_k)P_k^f \tag{3.38}$$

$$K_k = P_k^f(P_k^f + R_k)^{-1} \tag{3.39}$$

The meanings of the variables are explained in the below:

$^{E}x_{k-1}^a$: estimated stated vector at time k-1;

$^{E}x_k^a$: estimated stated vector at time k;

$^{E}x_k^f$: forecast value at time k;

$^{E}z_k$: observation value at time k;

P_k: covariance of the state vector at time k;

Q_k: covariance of the process noise at time k;

R_k: covariance of the observation noise at time k;

K_k: Extend Kalman Filter gain;

P_k^f: forecast error covariance at time k.

3.5.2 Variance-based fusion

To enhance the accuracy of the EFK-based fusion on IMU, we propose a variance-based fusion function to avoid contaminated magnetic data integration. Equation 2 shows that ${}^{E}z_k$, provides absolute correction of yaw angle, rendering the gyroscope data to eliminate the severe degradations indoors. On the contrary, correction from magnetometer with random variations incorrectly integrates the yaw angle fusion. Consequently, low variance of the observation measurement is the cornerstone of correct heading.

We define an average function to detect the variance of the magnetometer sensor, which requires the mean of the signal within a window size of N samples:

$$\left\|\bar{b}\right\| = \frac{1}{N} \sum_{k=n}^{n+N-1} {}^{E}z_k \tag{3.40}$$

where $\left\|\bar{b}\right\|$ denotes the norm of averaged magnetometer observation from time step n to n+N-1. The window size N depends on the accumulating rate of the dynamic gyroscope bias error. Deviation of magnetic measurement under perturbation must be checked to guarantee low covariance of the observation, leading a threshold function we designed:

$$\Lambda({}^{E}z_k) = \frac{\left\|{}^{E}z_k - \left\|\bar{b}\right\|\right\|}{P_k} \leq \lambda \tag{3.41}$$

where $\Lambda({}^{E}z_k)$ serves as low variance measurement detector, while P_k indicates the covariance of the state vector at time k. EKF-based fusion generates rational and accurate attitude integration with the premise of threshold function, $\Lambda({}^{E}z_k)$, below a statistical boundary value λ. With respect to outage of low $\Lambda({}^{E}z_k)$ value, we utilize the dynamic bias calculated in the last average function window as a conjecture of gyroscope error, based on the absence of remarkable deviation within a short period in the physical system.

We formally specify the above heuristics into the following steps in our proposed approach:

Step 1) Given the first window size set of data from gyroscope, accelerometer and magnetometer, the according quaternion change rate is calculated resorting Eq. (13) and stored in the memory, denoted as ${}^{S}_{E}\dot{q}_{\epsilon,1}$, ${}^{S}_{E}\dot{q}_{\epsilon,2}$, ..., ${}^{S}_{E}\dot{q}_{\epsilon,N}$.

Step 2) Using Eq. (3.40), the averaged state vector is generated at time step N, rendering that the first assessment of dynamic quaternion bias is produced as ${}^{S}_{E}\dot{q}_{\epsilon,N}$. We save it as ${}^{S}_{E}error_{previous,N}$.

Step 3) New raw IMU sample triggers new iteration of averaged function in Eq. (3.40) updates, renewing the variance-based function of Eq. (3.41), and the dynamic quaternion bias assessment ${}^{S}_{E}error_{previous,N}$. Serving as a variance detector, threshold function, $\Lambda({}^{E}z_k)$ of Eq. (3.41), evaluates the variance of the magnetometer observation at each new raw IMU sample. If the value of $\Lambda({}^{E}z_k)$ satisfies the constraint of λ, EKF-based attitude fusion is allowed. That is, the controller runs the attitude fusion from Eq. (23) to Eq. (39).

Step 4) Once violation of threshold function of $\Lambda({}^{E}z_k)$ appears, we stop the correction from EKF-based attitude fusion which use observation data from magnetometer. Instead, we consider the dynamic quaternion bias, ${}^{S}_{E}error_{previous,N}$, is the same during the period of violation. We continue the attitude fusion only through Eq. (14) and Eq. (15). We avoid inaccurate integration of IMU via ruling out the the absolute magnetic correction of high variance, rendering a conservative fusion via previous quaternion change rate.

Step 5) At the moment of $\Lambda({}^{E}z_k)$ is satisfied, **Step 3** is repeated as our normal attitude fusion. Based on the clean measurement, the correction of absolute reference from magnetometer guarantees the accurate integration of EKF-based algorithm, significantly improving the counter indoor perturbation effect.

Step 6) The end.

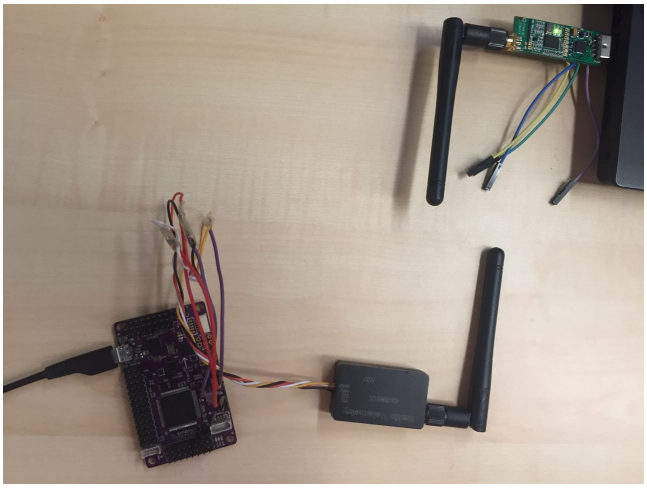


Figure 3.8: Arduino Mega 2560 platform integrated with MPU-6050 and HMC-5883

3.6 Evaluation

We construct a set of experiments to evaluate the angular accuracy in pitch, raw and yaw axis, respectively,. We first list the details of our widely used arduino core platform. Second, we compare the fusion angles using GD-based algorithm and measured angles to demonstrate the importance of magnetic calibration. We discuss the yaw angle difference before and after using our magnetic fault model in GD-based algorithm. Finally, we compare the measured angles and GD-based algorithm angles to determine the affection of different magnetic distortions as mentioned in Section 3.4. We determine the assumptions of GD-based algorithm and demonstrate that our algorithm encounters less limitation compared with the GD-based algorithm in indoor environment.

3.6.1 Equipment

We perform our experiment on a widely used arduino-based platform as shown in Figure 3.8. We choose arduino mega 2560 as our core micro-controller unit (MCU) ,which connects the IMU via SPI bus and compass via i2c bus, respectively. To avoid the turbulence of low frequency fusion, we increase the sensor data capture frequency in MCU up to 200 Hz. In the aspect of IMU, we use MPU-6050 [73] which contains 16-bits analog-to-digital conversion hardware and provides both accelerometer and gyroscope in three dimensions.

The MPU-6050 IMU provides raw information on roll and pitch angles if and only if fusion algorithm is properly constructed. This process encounters correction limitation in providing accurate yaw angle because of the absence of the indoor correction reference support. Integrating HMC-5883 [69] component on the same board with mega 2560 provides the absolute Earth‘s north as a corrector. The HMC5883L includes a 12-bit ADC which declares 1° to 2° compass heading accuracy. Our experiment requires a digital angle measuring tool to serve as an absolute comparison standard. Here, we choose an iphone 7 to transmit its angular measurement to our laptop.

To avoid the constraint of communication cable, such as usb, we utilize a pair of 433 MHz radio frequency (RF) modules to transmit data described in Figure 3.8. We send the real-time raw sensor data package back to the laptop and fuse 3D data at 200 Hz. Each package of data contains 3D gyroscope, acceleration, and magnetic gauss measurement.

3.6.2 Attitude Estimation Using Gravity

In the first step, we evaluate the GD-based fusion algorithm only using gravity as absolute reference. This step aims to discover the relationship between attitude fusion algorithm and gravity reference. With the precise capture of the accelerometers in IMU, the estimated pitch/roll angles inferred by quaternion are intended to converge to the same degree with the measured angles. Given that gravity is always pointing at negative side of z axis in Earth's frame, the estimated pitch/roll angles are supposed to be accurate if the fusion algorithm can remove the bias of gyroscopes data.

We collect raw measured data in gyroscope, accelerometer, and magnetometer from IMU. These 3 kinds of data are sent back to our laptop to do real time attitude fusion using GD-based algorithm in 200 Hz. As shown in Figure 3.9, the maximum pitch angular residual between the estimated angle of fusion algorithm and practical measured angle appears at 0° in the measured angle, and -0.65° in estimated angle, correspondingly. The maximum angular error is 0.65° which is less than 1° in dynamic error. In Figure 3.10, maximum roll angular error appears at -0.82° for the measured angle and -0.07° for the estimated angle

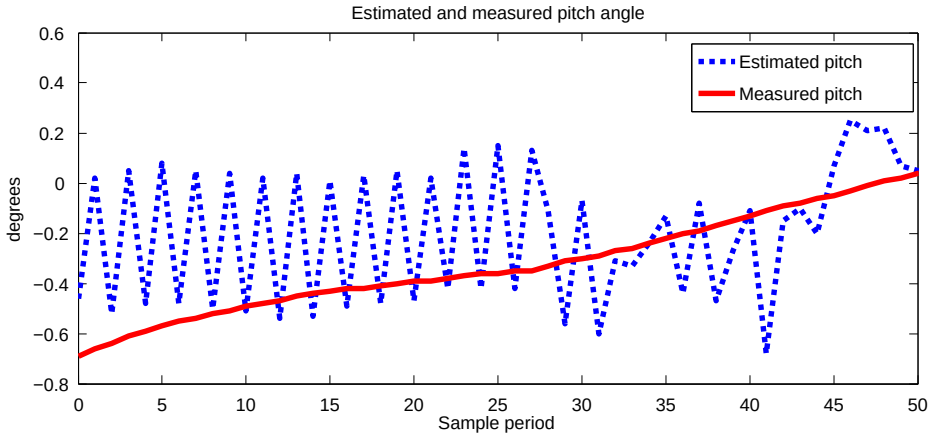


Figure 3.9: Dynamic results of measured angle and estimated angle in pitch axis

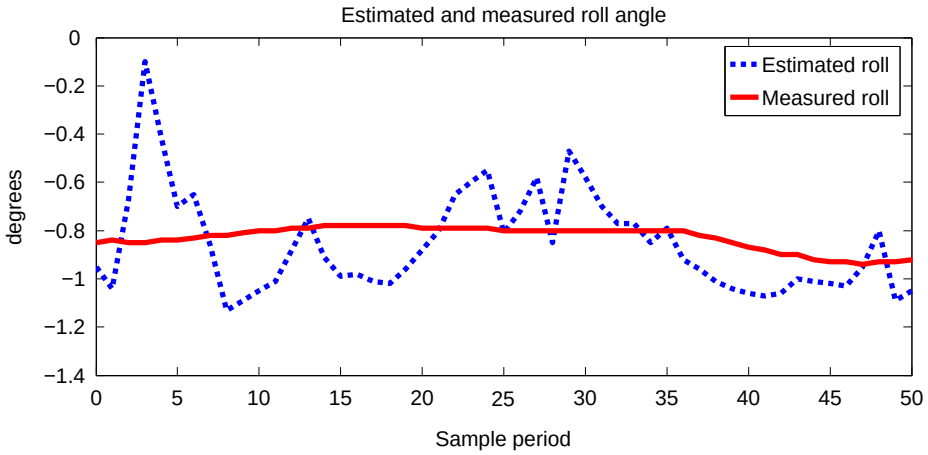


Figure 3.10: Dynamic results of measured angle and estimated angle in roll axis

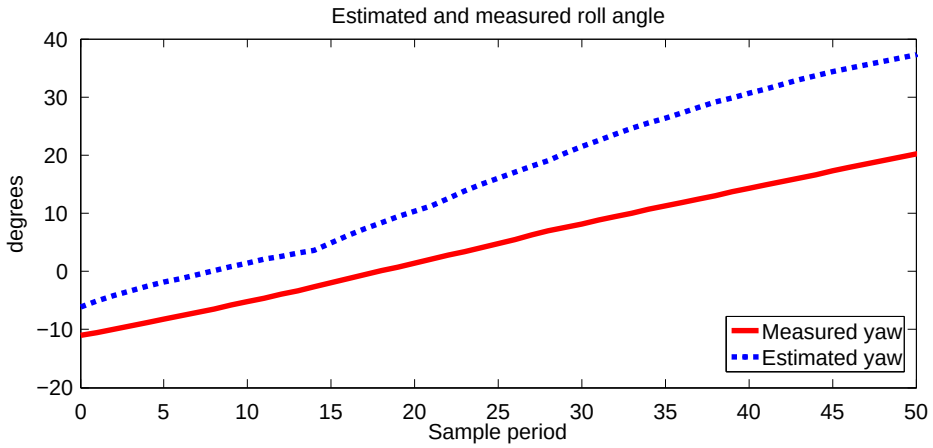


Figure 3.11: Dynamic results of measured angle and estimated angle in yaw axis

correspondingly. This maximum error in roll dimension is $0.75°$, which is less than $1°$ in dynamic error. Eliminating bias in each sample period in the roll/pitch dimension, we discover that GD-based fusion algorithm can provide accurate angle estimations with the corrector of the gravity. By contrast, from Figure 3.11, the gyroscope bias accumulates quickly because no absolute reference serves as a corrector. Fusion angle in the yaw axis cannot predict right direction and the change rate of orientation. This error implies GD-based fusion algorithm relies heavily on the correction from absolute reference in every step of gyroscopes integration. If the reference is not accurate, the GD-based fusion algorithm encounters poor robustness. Hence, the hypothesis of GD-based fusion algorithm are both accurate gravity and Earth's north references.

3.6.3 Attitude Estimation Using Gravity And Calibrated Compass

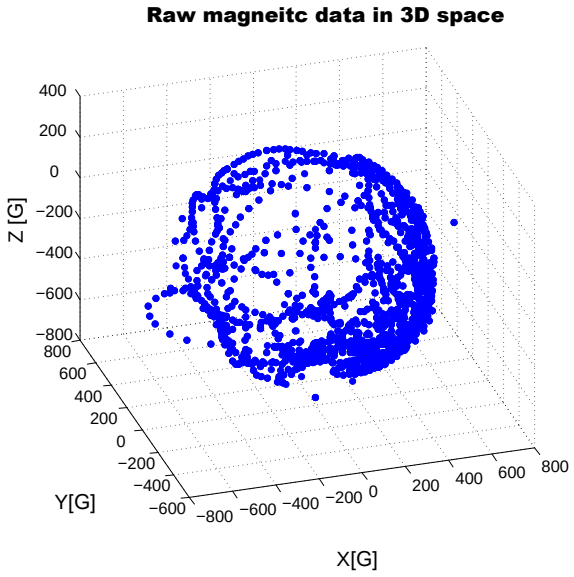


Figure 3.12: Raw data of compass in 3D space

To avoid the poor results in yaw dimension adjustment by gravity only, we combine gravity and calibrated compass in attitude estimation in the second step. We collect magnetometer only via RF modules at 200 Hz and draw the raw measured data in a geometric view. Using the model of Eq. (3.18), the raw measured compass data can be approximated as a shape of ellipsoid. This approximation is demonstrated as Figure 3.12 in an indoor

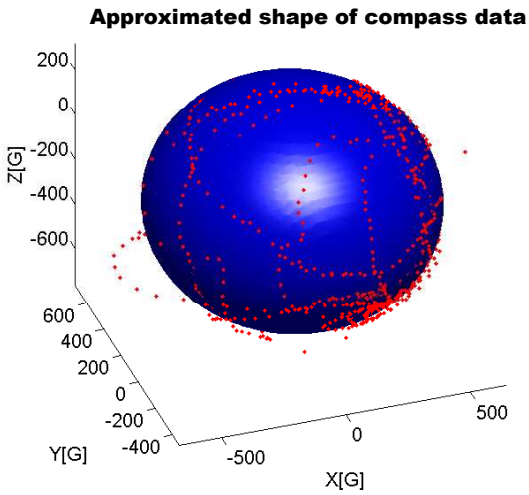


Figure 3.13: Approximated shape of compass data in 3D space

environment. The range of this magnet raw data is within (-412G, 631G), (-373G, 809G), (-761G, 247G) in the x,y,z dimensions, respectively. Presented as a ellipsoid-like shape, all the measurement are approximated into an ellipsoid via 3D least squares method mentioned in Eqs. (3.21) and (3.20). The results of least squares method are shown in Figure 3.13. Viewing this least squares model geometrically, we discover the ellipsoid center at (10.529G, 171.75G, -194.6G) with radius of (579.85, 548.64, 478.34).

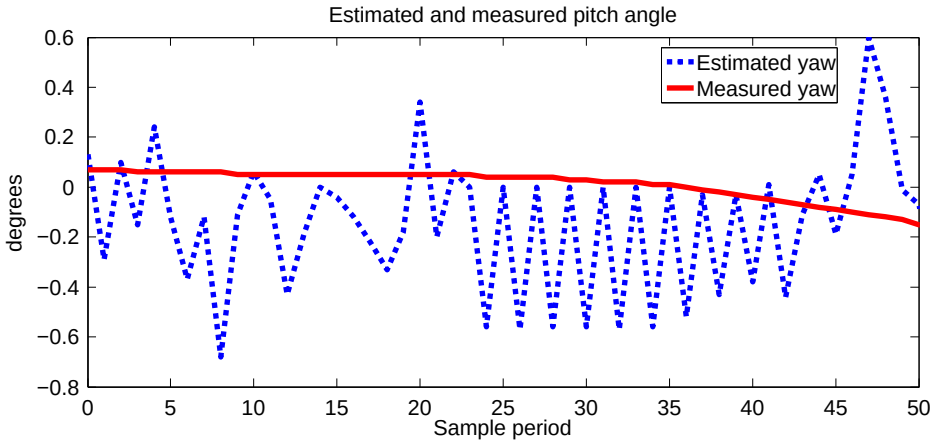


Figure 3.14: Dynamic results of measured angle and estimated angle in yaw axis using post-calibration data

Based on this accurate shape of magnetic measured data, requirements of GD-based

algorithm can be fulfilled if the accurate absolute Earth's north reference is inferred. Converting the ellipsoid into sphere geometrically, we map each measured compass information to each point in the surface of a unit standard sphere via singular value decomposition (SVD) [41] approach. Within the same place of calibration, attitude estimation shows the maximum residual between measured angle and estimated angle locations in (-0.08°, 0.6°) as shown in Figure 3.14. This error ($0.68°$) achieves the dynamic errors less than $1°$ in yaw angle ,which also validates our ellipsoid geometric approximation.

3.6.4 Attitude Estimation Under Different Magnetic Distortion

Based on the analysis of the necessary requirements of GD-based fusion algorithm above, we continue to exploit the nature of position-dependent magnetic affection on this fusion algorithm in this part. We first discover the magnetic distortions on a predefined trajectory. We try to establish the affection from compass is dependent on position. Then, we demonstrate that our proposed EKF-based algorithm improves the poor robustness of attitude fusion algorithm under dynamic unpredictable magnetic perturbations. We compare measured angles, GD-based angles, and our proposed EKF-based angles results. The comparison validates that, in an indoor environment, our proposed EKF-based attitude estimation algorithm has advantages especially in case of compass measurement exceeding a certain range from the calibration place.

To determine the the relationship between position and magnetic perturbations, we set up a trajectory in the room of Figure 3.5. By choosing the calibration place as the starting point (Figure 3.15), we predefine the trajectory which has a property of 7 meters in length and 3.5 meters in width. We rotate our IMU in hand while moving along the predefined indoor rectangle trajectory in a counter-clockwise direction. With complex environment along the trajectory, such as electricity wires or computers, unpredictable magnetic perturbations are encountered. We transfer our IMU data back to the station fixed in the starting point via RF modules at 200 Hz to view the yaw angle fusion.

We walk along the predefined trajectory for 1000 times to implement the variance-

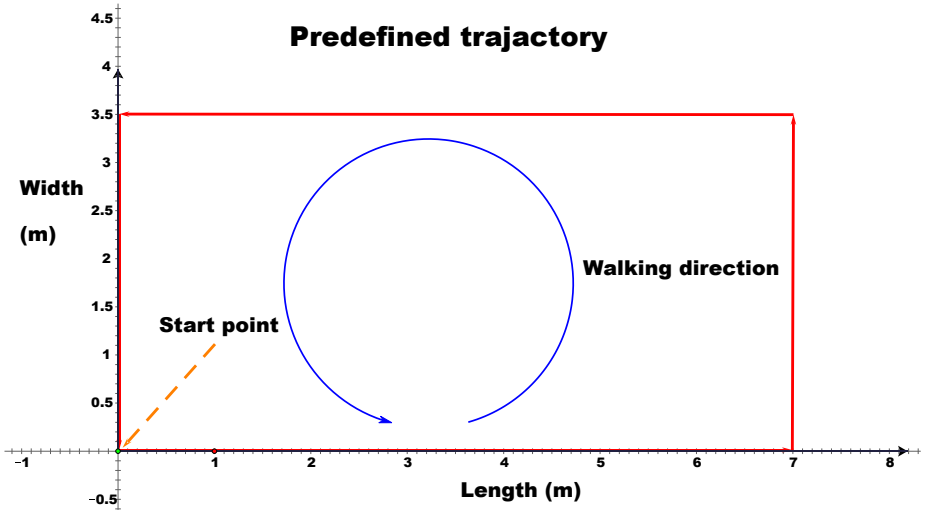


Figure 3.15: Predefined trajectory of IMU

based fusion in our proposed algorithm. Meanwhile, the sensor data, containing 3D gyroscope, accelerometer and magnetometer, is sent back to our laptop for real time angular fusion. We compare the sets of magnetic observations each time and choose size N to be 9 in Eq. (3.40), threshold function value λ to be 5.37 in Eq. (3.41), respectively. We draw the sets of results by repeating the predefined trajectory. Figure 3.16 and Figure 3.17 display one representative set of yaw angle fusion comparison results between GD-based algorithm and measured angles, the proposed EKF-based angle fusion algorithm and measured angles, respectively. Figure 3.18 calculates the angle errors. The results demonstrate that, compared to GD-based algorithm, our proposed EKF-based angle fusion algorithm has obvious advantage under different magnetic perturbations. Figure 3.18 shows that the proposed EKF-based angle fusion algorithm has maximum angle error of $2.02°$ while $7.17°$ in GD-based algorithm. Moreover, Figure 3.16 shows that the angle fusion errors can be divided into 3 stages: (1) From sample 1 to 374, angle fusion error is within $\pm 2.74°$; (2) From sample 375 to 944, angle fusion generates bigger error range of $\pm 7.17°$; (3) From sample 945 to 1500, angle fusion error becomes relatively smaller with a range of $\pm 1.57°$. This phenomenon proves magnetic distortion is dependent on position: (1) When our IMU goes along the trajectory from starting point (0, 0) in Figure 3.15 to (3.2, 0), the GD-based algorithm enters the first

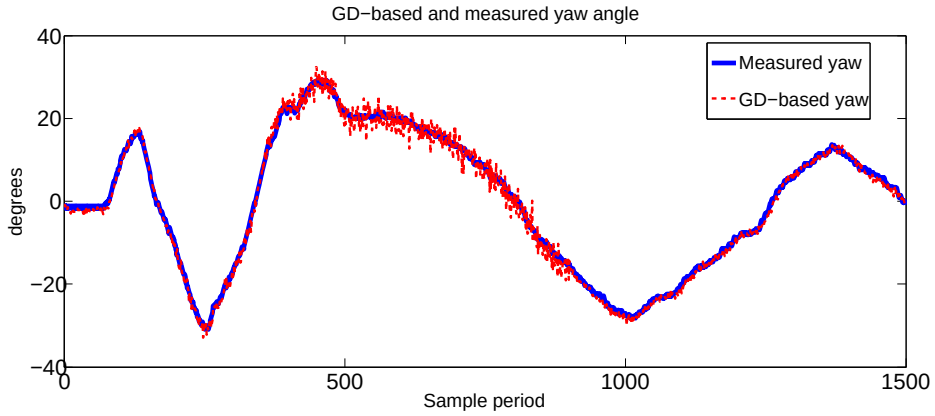


Figure 3.16: Comparison results of measured angle and GD-based angle in yaw axis

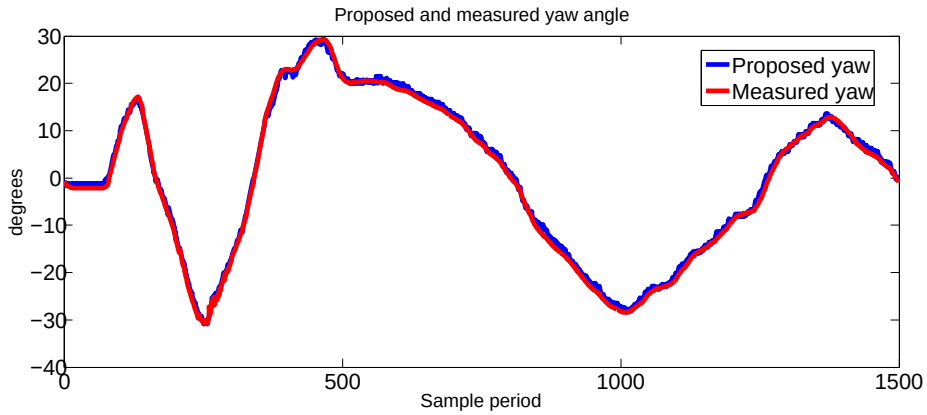


Figure 3.17: Comparison results of measured angle and proposed algorithm angle in yaw axis

stage respectively. Given its proximity to the calibration place, the residual of angle fusion is relatively small. (2) Beyond point (3.2, 0) in Figure 3.15, the GD-based algorithm enters the second stage with obvious angle errors. (3) Until the IMU returns to point (2.6, 3.5) in Figure 3.15, the GD-based algorithm enters the third stage which outputs less angle error because the place of angle fusion is close to the magnetic calibration place again. These stages of attitude fusion in yaw axis also demonstrate that the accuracy of GD-based algorithm and compass rely heavily on the position.

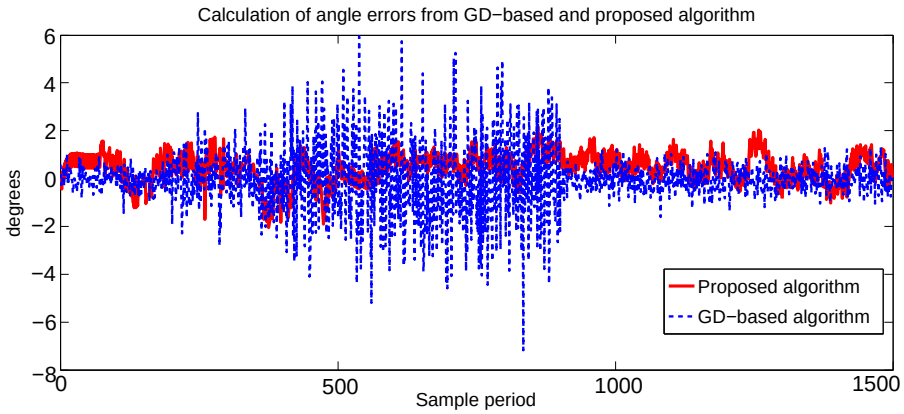


Figure 3.18: Calculation of angle errors from GD-based and proposed algorithm

CHAPTER 4

MPC-SI SOLUTION FRAMEWORK

4.1 Overview

An important component of control-CPS SFL is the oracle design which distinguishes the correct source code execution traces from incorrect ones. As proposed in Chapter 1, we import the physical trajectories and judge them using MPC-SI. MPC-SI method generates estimated trajectories via approximating the real physical trajectories. We then look at the gap between the real and estimated physical trajectories. If a gap is big, the corresponding source code execution trace is labelled as incorrect. The labelled source code execution traces are the input for SFL. Correspondingly, a suspect list of the possible buggy lines is generated as the output. Figure 4.1 illustrates the above concepts.

In this chapter, we first introduce MPC-SI. By exploiting MPC-SI, we propose an oracle for control-CPS SFL. Based on this MPC-SI oracle, we propose a framework of methodology to prepare traces for control-CPS SFL.

4.2 More on the Heuristics

Chapter 1 points out that due to the empirically proven capability of MPC-SI in predicting control systems' behaviors, we shall exploit it as an oracle for control-CPS SFL.

MPC is an online control strategy for systems hard to model analytically. It regards the control system as a black-box and focuses on learning the relationship between its sampled input and output online using its SI method. The MPC-SI learning result is called the *identified model*. Specifically, if we regard the the gray area of Fig. 2.1 as a black-box control

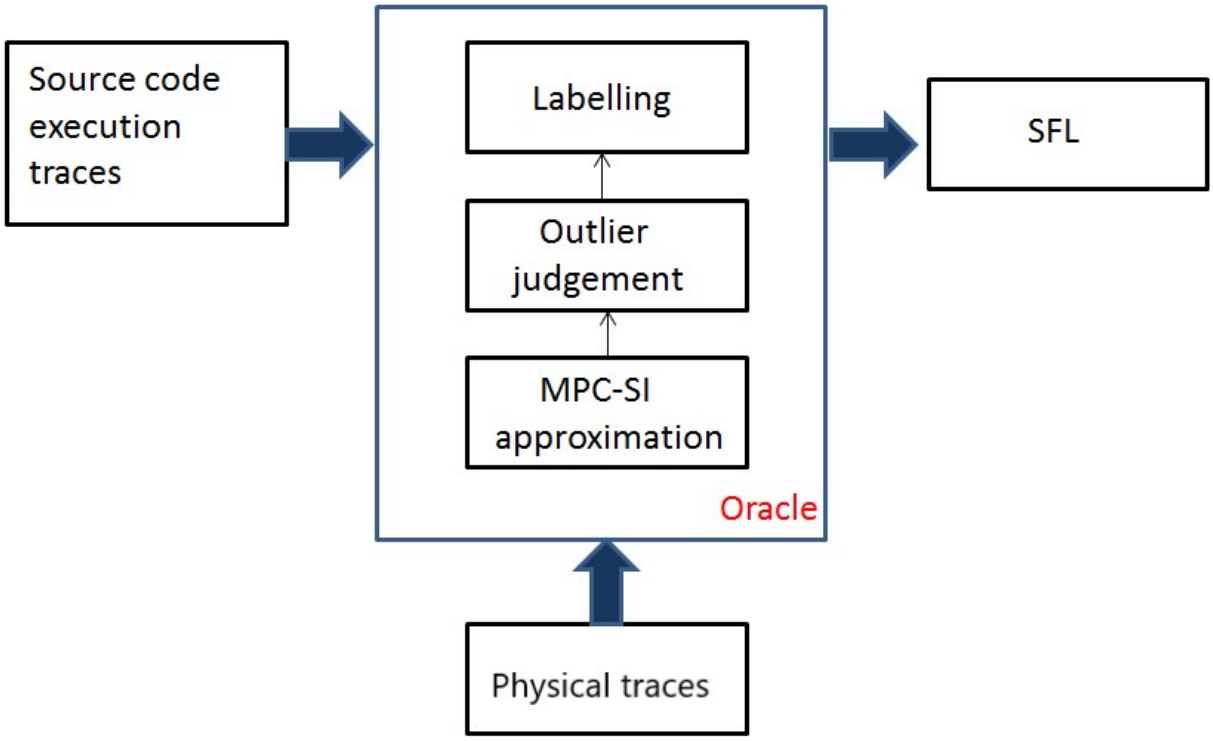


Figure 4.1: Proposed oracle overview

system, whose input is $U_h \in \mathbb{R}^q$ and output is plant state $X_i \in \mathbb{R}^n$, then MPC-SI typically models the relationship between U_h and X_i ($\forall i = p, p+1, \cdots$) as

$$X_i = (\sum_{j=1}^{p} A_j X_{i-j}) + BU_h + \xi_i, , \tag{4.1}$$

where $h = \lfloor i\Delta/T \rfloor = \lfloor i/N \rfloor$; $p \in \{1, \ldots, N-1\}$ is a preconfigured constant; A_1, A_2, $\cdots$, $A_p \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times q}$ are the model parameters to be learnt; and $\xi_i \in \mathbb{R}^n$ is the *SI error*. Appendix A elaborates the details of the above MPC-SI technology.[1]

As mentioned in Chapter 2, during runtime, in each user input sampling period, the user input sample U_h holds constant throughout $[hT, (h+1)T)$. In such a user input sampling period, once $i+1$ ($p \leqslant i < N$) consecutive plant states (without loss of generality, denote

[1] There are other forms of the identified model for MPC-SI, but the model of Exp. (4.1) is the most widely used. In this thesis, unless otherwise denoted, MPC-SI identified model by default refers to the model described by Exp. (4.1).

them as X_0, ..., $X_{i-(p-1)}$, ..., X_{i-1}, X_i) become available, the MPC-SI method optimizes the values of A_1, A_2, ..., A_p and B in Exp. (4.1), so that the runtime accumulated SI error energy is minimized (interested readers can refer to the seminal textbook of Camacho et al. [32] for the details of the algorithm). Suppose the optimal A_1, A_2, ..., A_p, and B values are $A_1^{(*,i)}$, $A_2^{(*,i)}$, ..., $A_p^{(*,i)}$, and $B^{(*,i)}$, then MPC-SI method predicts X_{i+1} with $\hat{X}_{i+1}$:

$$\hat{X}_{i+1} \stackrel{def}{=} (\sum_{j=1}^{p} A_j^{(*,i)} X_{i+1-j}) + B^{(*,i)} U_h. \tag{4.2}$$

When X_{i+1} becomes available, we know the *MPC-SI prediction error*:

$$e_{i+1} \stackrel{def}{=} \hat{X}_{i+1} - X_{i+1}. \tag{4.3}$$

Usually the MPC-SI prediction error magnitude should be small. When it becomes extremely large, something may be wrong: intuitively, the control system execution may be incorrect. In this sense, we can use the MPC-SI method as an oracle. For example, as Figure 4.2 shows, the estimated trajectory of control-CPS (dot line) approximates the real trajectory (solid line) closely when the code is executed normally. Once a large gap between these two appears, we suspect a bug takes effect. Thus, our MPC-SI oracle labels the corresponding source code execution trace as "incorrect".

The MPC-SI method has three advantages, which makes it an ideal oracle candidate for control-CPS SFL:

First, the method procedure is deterministic, and is readily implementable as an automatic program.

Second, both the method procedure and the identified model are simple, and are independent of the internal complexity of the control-CPS. Hence it is easy to implement and use.

Third, though the method is simple, we have strong belief that it can effectively catch subtleties of various control-CPS physical trajectories. This is empirically evidenced by the success of MPC in broad variety of applications over the decades: industrial, chemical,

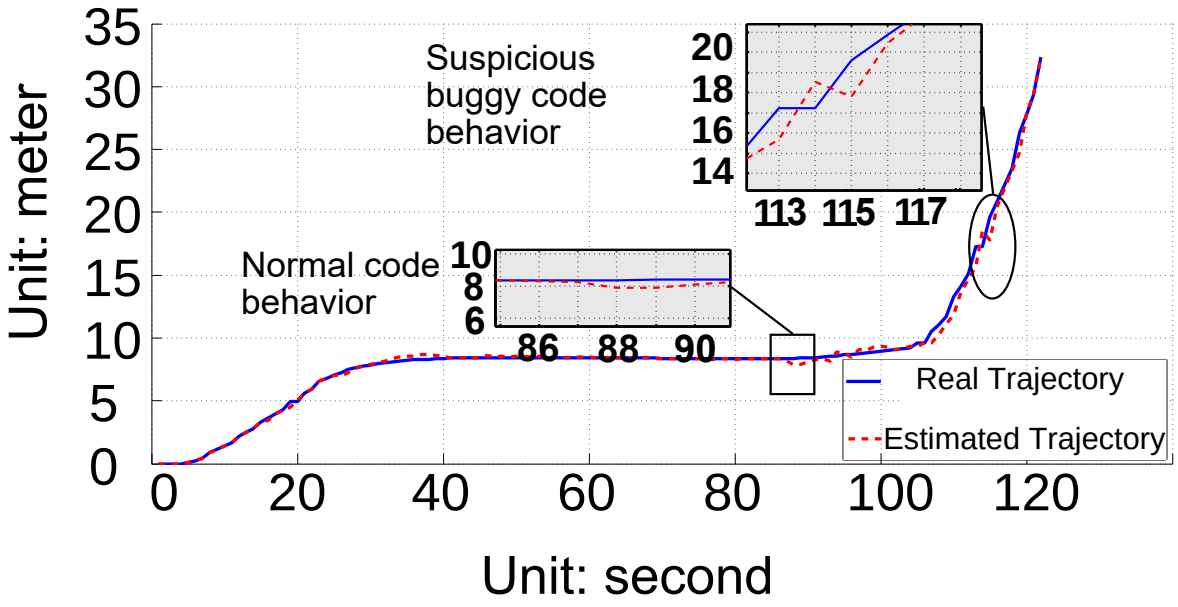


Figure 4.2: Distinguishing correct/incorrect trajectories via MPC-SI oracle example

vehicular, robotics, medical etc. [32,47,53,81,124]. In these MPC applications, the MPC-SI method is actually used to control life/mission critical plants. Serious accidents would have happened if it is untrustworthy.

4.3 Proposed Oracle and Source Code Execution Trace Preparation Methodology

Based on the heuristics of Section 4.2, we propose to use the MPC-SI method as our oracle, and propose the corresponding source code execution trace preparation methodology for control-CPS SFL. Our proposed oracle and source code execution trace preparation methodology (referred to as "*our proposed approach*" in the following) are summarized by Fig. 4.3

Note in Fig. 4.3-**Step1**, we leave the sampling strategy open. By default, we shall randomly sample from the valid set of initial states as per uniform distribution(In Chapter 5, to increase the number of test cases, after uniformly sampled n test cases' initial states, we add disturbances to these n initial states to create another n test cases initial states, and repeat

Step1 Sample a valid initial state X_0 of the plant.

Step2

Task1 Emulate the control-CPS using the emulation platform of Fig. 2.1 starting from $X(0) := X_0$. Log the physical trajectory $\{X_i\}$, user input trace $\{U_h\}$, and source code execution trace θ.

Task2 Run the MPC-SI method in parallel with the control-CPS emulation. For each new physical trajectory sample X_{i+1} from the emulation, calculate the MPC-SI prediction error e_{i+1} via Exp. (4.3) and log it.

Step3 When the emulation ends, check if the MPC-SI prediction error trace $\{e_i\}$ contains outlier(s). If so, label θ as "incorrect"; otherwise label θ as "correct".

Step4 If enough source code execution traces are collected, terminate; otherwise go to **Step1**.

Figure 4.3: Our proposed oracle and methodology to prepare source code execution traces (this oracle and methodology are referred to as "*our proposed approach*").

this until the total number of test cases is quadrupled: 4n.). Our proposed MPC-SI method based oracle is embodied by Fig. 4.3-**Step3**, which labels the correctness of the source code execution trace. Also in Fig. 4.3-**Step3**, note in statistics, currently there is no consensus on the strict definition of an "outlier" [9]. In this thesis, we regard a data point outside of (mean $\pm$ 6std) of statistics as an outlier.

Compared to our proposed approach, the other approach is the so-called *human oracle approach* depicted by Fig. 4.4.

Step1 Same as Fig. 4.3-**Step1**.

Step2 Same as Fig. 4.3-**Step2**-**Task1**.

Step3 Use human oracle to judge the correctness of physical trajectory $\{X_i\}$, and label the corresponding source code execution trace θ.

Step4 Same as Fig. 4.3-**Step4**.

Figure 4.4: Human oracle and methodology to prepare source code execution traces (this oracle and methodology are referred to as the "*human oracle approach*").

CHAPTER 5

EVALUATION AND RESULTS

Next, we shall evaluate our proposed approach (see Fig. 4.3).

5.1 Evaluation Metrics and Research Questions

Our proposed approach (see Fig. 4.3) mainly serve three main stream families of SFL tools: respectively based on program spectrum, statistics, and machine learning. As shown by Fig. 1.3, an SFL tool of such types takes in a large number of labeled source code execution traces, and outputs a *suspect list*. This list shows suspected buggy *source code entities* (such as lines of source code, blocks of function definition, blocks of class definition, etc.) in descending order of suspiciousness.

We have three metrics to measure the quality of the suspect list. Suppose the suspect list is $\mathcal{S} = (s_1, s_2, \cdots, s_l)$, where s_i ($i = 1, \cdots, l$) is the ith suspected (s_1 is the most suspected) source code entity. Suppose $\mathcal{B}$ is the set of truly buggy source code entities. Then *accuracy* refers to $|\{s_i | s_i \in \mathcal{B}\}|/l$, i.e. coverage of true bugs by $\mathcal{S}$; *recall* refers to $|\{s_i | s_i \in \mathcal{B}\}|/|\mathcal{B}|$, i.e. coverage of true bugs in $\mathcal{B}$; *latency* refers to $\min\{i | s_i \in \mathcal{B}\}$, i.e. starting from s_1, the index of the first suspect in $\mathcal{S}$ that is truly buggy. In case none of the items in $\mathcal{S}$ belong to $\mathcal{B}$, we define latency as the length of suspect list $\mathcal{S}$, i.e. l, as the programmer needs to investigate all l items in the suspect list before s/he stops.

As the suspect list is the SFL result, the above suspect list quality metrics are henceforth also the SFL quality metrics. Our evaluation intend to clarify the following research question.

Q1 How does our proposed approach impact SFL quality (measured by accuracy, recall, and latency)?

Besides the above research questions, we also notice that oracles may have other usage than SFL. Therefore, we are also interested in the raw quality of the oracle itself. Specifically, suppose we have collected a set of source code execution traces Θ; let

$$\mathcal{P} \stackrel{def}{=} \{\theta \in \Theta | \theta \text{ is labeled by the oracle as buggy (i.e. "incorrect")}\},$$
$$\mathcal{P}_{\mathsf{T}} \stackrel{def}{=} \{\theta \in \Theta | \theta \text{ truly includes buggy entity of the source code}\},$$
$$\mathcal{N} \stackrel{def}{=} \Theta - \mathcal{P}, \text{ and } \mathcal{N}_{\mathsf{T}} \stackrel{def}{=} \Theta - \mathcal{P}_{\mathsf{T}};$$

we can then define the *false positive rate* R_{FP} and the *false negative rate* R_{FN} of the oracle as

$$R_{\mathsf{FP}} \stackrel{def}{=} |\mathcal{P} \cap \mathcal{N}_{\mathsf{T}}|/|\mathcal{N}_{\mathsf{T}}|, \text{ and } R_{\mathsf{FN}} \stackrel{def}{=} |\mathcal{N} \cap \mathcal{P}_{\mathsf{T}}|/|\mathcal{P}_{\mathsf{T}}|. \tag{5.1}$$

We intend to clarify the following research question.

Q2 How does our proposed approach impact raw oracle quality (measured by R_{FP}, R_{FN})?

In the following, we shall carry out evaluations on two contro-CPS test-beds with various bugs from real-life and/or artificial injection to answer the above research questions.

5.2 Control-CPS Test-beds

We carry out evaluations on two control-CPS test-beds.

The first test-bed is ArduPilot [5]. ArduPilot is a state-of-the-art open platform to build consumer-product-grade *unmanned aerial vehicles* (UAVs) [3, 4]. The cyber subsystem of ArduPilot is mostly written in C++. It consists of over 1.4 million lines of source code from more than 9,600 files, with a total size of over 380MB. To debug the entirety of

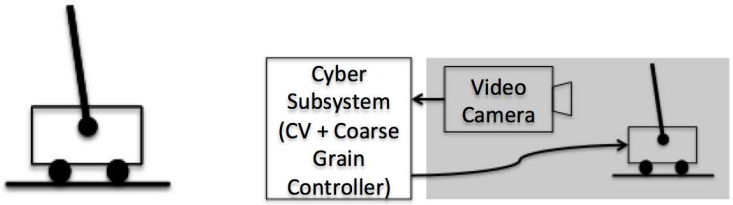
ArduPilot contains too much work. In this thesis, we focus on debugging the application layer of ArduPilot, which already consists of 27,000 lines of source code from 160 files, with a total size of over 1.06MB.

Besides the practical value and size of ArduPilot, it also features a physical subsystem simulator: the so-called *software in the loop* (SITL) simulator. As the UAV physical subsystem model is simple and well established, the SITL simulator is relatively simple (only consists of about 6 thousand lines of source code). Also, the SITL simulator is already widely used by the community, even for the development of consumer-grade-products [3, 4, 26]. Based on these, we reasonably regard the SITL simulator as bug-free.

By connecting a (possibly buggy) ArduPilot cyber subsystem with the SITL physical subsystem simulator, we build the control-CPS emulation platform required by our proposed approach and the human oracle approach (see Fig. 2.1, Fig. 4.3-**Step2**, and Fig. 4.4-**Step2**). The monkey (see Fig. 2.1) we adopt is a program that steers the ArduPilot UAV from randomly sampled initial conditions. The initial x, y, z location and pitch, roll, yaw of the UAV are sampled uniformly from range $[0, 111.3\text{km}]$, $[0, 111.3\text{km}]$, $[0.4\text{km}, 0.5\text{km}]$, $[-20°, 20°]$, $[-20°, 20°]$, $[0°, 360°)$ respectively. Each emulation spans an emulated time duration of 12 seconds. In the emulated time, the plant sampling period $\Delta = 100\text{ms}$, and the user input sampling period $T = 2\text{s}$.

The second control-CPS test-bed is one that combines *inverted pendulum and computer vision* (IP+CV). An *inverted pendulum* (IP) is a classic generic purpose test-bed for control theories [29, 57, 115]. It consists of a cart moving along the x-axis and a metal rod (i.e. the pendulum) with one end hinged on the cart (see Fig. 5.1(A)). The other end of the metal rod is free to rotate. The IP's on-cart controller carries out fine-grain control: it moves the cart back and forth along the x-axis to keep the metal rod standing up-right still. In our IP+CV control-CPS test-bed, the plant of the physical subsystem is an IP (see Fig. 5.1(B)). The state of the IP is captured by a video camera. The video captured, i.e. a sequence of pictures taken every 8s, is sent to the cyber subsystem. The cyber subsystem takes charge of coarse-grain control. Using the pictures from the video camera, the cyber subsystem rec-

ognizes the current x-axis location of the IP via *computer vision* (CV), and then decides the next target x-axis location (aka *reference location*) for the IP. The cyber subsystem of the IP+CV control-CPS consists of about 4,000 lines of source code from 11 files, with a total size of 161.9KB.



(A) an IP

(B) The entire control-CPS with *computer vision* (CV) library in the cyber subsystem

Figure 5.1: *Inverted pendulum and computer vision* (IP+CV) control-CPS

To build the emulation platform of Fig. 2.1 for the IP+CV control-CPS test-bed, we need a physical subsystem simulator. For analytical convenience, we consider the video camera of Fig. 5.1(B) as part of the physical subsystem, i.e. the IP+CV control-CPS's physical subsystem is the gray area of Fig. 5.1(B). We use a virtual reality software, Unity [14], to simulate the gray area components of Fig. 5.1(B). The simulator consists of 1464 lines of code. Due to its small scale, the simulator can be manually thoroughly debugged. We therefore regard the simulator as bug-free.

By connecting a (possibly buggy) IP+CV cyber subsystem with the physical subsystem simulator, we can build the control-CPS emulation platform required by our proposed approach and the human oracle approach (see Fig. 2.1, Fig. 4.3-**Step2**, and Fig. 4.4-**Step2**). The monkey (see Fig. 2.1) we adopt is a program that steers the IP from randomly sampled initial conditions. The initial x-axis location of the IP cart is sampled uniformly from range $[-0.32, 0]$(m). Each emulation trial spans an emulated time duration of 280 seconds. In the emulated time, the plant sampling period $\Delta = 40$ms, and the user input sampling period $T = 8$s.

5.3 Evaluations with Real-life Bugs

ArduPilot maintains a released version log in GitHub [6]. From it we can find real-life bugs appeared in earlier released versions. Of these bugs, 8 belong to the application layer source code that we focus on. We inject these bugs back into our ArduPilot test-bed, as described by Table 5.1.

Table 5.1: Real-life Bugs to be Injected into ArduPilot

File-Function-Line#	**Bug Description**
mode_steering.cpp-update-56	wrong stop throttle calculation
mode_steering.cpp-update-6	wrong navigation throttle calculation when reversing
mode_auto.cpp-update-56	wrong acceleration calculation when reversing
mode_auto.cpp-_enter-12	wrong initialization in auto mode
mode_rtl.cpp-_enter-56	wrong initialization in rtl mode
mode_guided.cpp-update-28	wrong mavlink signal message
mode_guided.cpp-_enter-6	wrong initialization in guided mode
mode.cpp-calc_throttle-79	wrong braking throttle value calculation

Guided by Table 5.1, we create 40 versions of buggy ArduPilot cyber subsystems, each called a *subject*. Eight of the 40 subjects respectively contain one of the candidate real-life bugs in Table 5.1. The other 32 subjects each contains 5 candidate real-life bugs selected from Table 5.1.

For each buggy ArduPilot cyber subsystem subject, our emulation platform runs 3 trials. In each trial, the emulation platform generates 2665 source code execution traces (and the corresponding physical trajectories, user input traces) by repeating Fig. 4.3-**Step1**

to **Step2**-**Task1** 2665 times. These source code execution traces are then labeled by our MPC-SI oracle as described by Fig. 4.3-**Step2**-**Task2** and **Step3**.

The 2665 labeled source code execution traces are then sent to three different SFL tools to find bugs: *Tarantula* (TA) [77], *Crosstab* (CR) [130], and *BP Neural Network* (NN) [133]. These three tools are respectively the well-known representative from the program spectrum based, statistics based, and machine learning based SFL tool families. As mentioned before, such tools respectively output a suspect list of l items (i.e. l possibly buggy source code entities) in descending order of suspiciousness. We assume a human programmer's effort is limited and can only investigate the top 10 items in a suspect list. Therefore, our TA, CR, and NN SFL tools respectively output an $l = 10$ item suspect list.

In comparison, we also carry out the human oracle approach (see Fig. 4.4). Simply put, in the human oracle approach evaluation, each buggy ArduPilot subject is also given 3 trials. In each trial, the emulation platform generates 2665 source code execution traces (and the corresponding physical trajectories, user input traces)[1]. These source code execution traces are then labeled by the human oracle, and then sent to TA, CR, and NN for SFL.

Specifically, the human oracle for ArduPilot runs as follows. A source code execution trace is labeled "correct" iff its corresponding physical trajectory obeys all the rules **H1.1**$\sim$**H1.3** listed below; and labeled "incorrect" otherwise.

H1.1 The UAV velocity shall never exceed ± 20m/s.

H1.2 The UAV velocity component along longitude, latitude, and altitude shall never exceed ± 31.5m/s, ± 28.16m/s, ± 30m/s respectively.

[1] Note for both our proposed approach and the human oracle approach, the emulation methods are the same (see Fig. 4.3-**Step1** to **Step2**-**Task1**, Fig. 4.4-**Step1** to **Step2**). Therefore, in fact, the (unlabeled) source code execution traces (and the corresponding physical trajectories, user input traces) generated for our proposed approach evaluation are reused.

H1.3 The UAV angular velocity along pitch and roll shall never exceed ± 3.5rad/s and ± 3.6rad/s respectively.

Rules **H1.1** $\sim$ **H1.3** are given by a panel of four domain experts via thorough discussions. Two of the four domain experts have PhD degrees, and respectively have over 15 and 9 years of experiences on control systems and CPS research and development. Three of the four domain experts have over two years of ArduPilot development experiences.

The evaluation results[2] are plotted in Fig. 5.2. In the figure:

X-axis for (a)$\sim$(i): oracle approach + SFL tool. X-axis for (j)$\sim$(l): oracle approach. PA: our proposed approach; HA: human oracle approach. TA: Tarantula; CR: Crosstab; NN: BP Neural Network. FP: oracle false positive rate; FN: oracle false negative rate.

Y-axis for (a)$\sim$(i): statistics of per trial SFL quality metrics. Y-axis for (j)$\sim$(l): statistics of per trial oracle false positive/negative rates.

For each given oracle approach and SFL tool, there are $8 \times 3 = 24$ one-bug subject trials and $32 \times 3 = 96$ five-bug subject trials, hence a total of 120 trials. Each trial corresponds to 2665 source code execution traces.

In each boxplot [7], the thick bar in each box is the median of the data, the box bottom/top are the 1st and 3rd quartile of the data (see footnote 2 for some subtle details). In (a)$\sim$(i), the "+"s outside the boxes are data outside of the 1st and 3rd quartiles (as they have relatively discrete values, we plot them individually instead of using whiskers to provide more information).

[2] The results include statistical medians and quartiles. Subtle details on how to calculate medians and quartiles are elaborated in [7, 12]. Particularly, when there are $2n$ data points, median is the average of the nth and the $(n+1)$th data point values; when there are $(4n+1)$ (or, respectively, $(4n+3)$) data points, then the 1st quartile is 25% (or, respectively, 75%) of the nth (or, respectively, $(n+1)$th) data point value plus 75% (or, respectively, 25%) of the $(n+1)$th (or, respectively, $(n+2)$th) data point value, while the 3rd quartile is 75% (or, respectively, 25%) of the $(3n+1)$th (or, respectively, $(3n+2)$th) data point value plus 25% (or, respectively, 75%) of the $(3n+2)$th (or, respectively, $(3n+3)$th) data point value [12].

This evaluation will be discussed in Section 5.5.

Ideally, we would also like to carry out real-life bug evaluation on the IP+CV control-CPS test-bed. Unfortunately, this test-bed is a legacy of our lab. Its debugging history is not well maintained. Meanwhile, its smaller size (compared to ArduPilot) and long usage history make it unlikely to contain more bugs. Thererfore, we do not include the IP+CV test-bed in the real-life bug evaluation.

5.4 Evaluations with Artificial Bugs

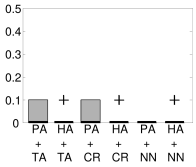
Besides the real-life bugs discovered in our control-CPS test-beds, there are other commonly seen bugs. Natella et al. conducted a comprehensive survey, and publish a list of commonly seen bugs-in-the-field as shown in Table 5.2 [101]. In our evaluation, we also inject such bugs (referred to as "*artificial bugs*" in the following) into our test-beds, and evaluate our proposed approach in the corresponding SFL.

Table 5.2: Common Bugs-in-the-Field (quoted from [101])

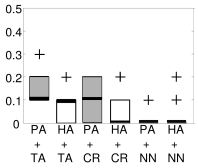
Type	**Description**
WPFV	Wrong Variable used in Parameter of Function call
WVAV	Wrong Value Assigned to Variable
MVAE	Missing Variable Assignment using Expression
MFC	Missing Function Call
MIA	Missing IF construct Around statements
MVIV	Missing Variable Initialization using a Value
MVAV	Missing Variable Assignment using a Value
MIFS	Missing IF construct plus Statements
MIEB	Missing IF construct plus statements plus ELSE Before statements
MLC	Missing a Logic Clause in branch condition
MLPA	Missing small and Localized Part of the Algorithm
WAEP	Wrong Arithmetic Expression in Parameter of function call

We run evaluations on both the ArduPilot and the IP+CV test-beds.

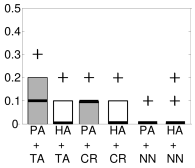
Accuracy (the higher the better)



(a) 1-bug subjects' trials

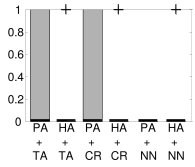


(b) 5-bug subjects' trials

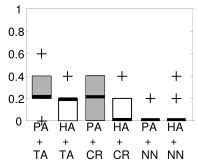


(c) all subjects' trials

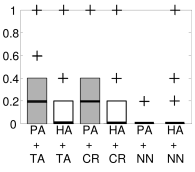
Recall (the higher the better)



(d) 1-bug subjects' trials

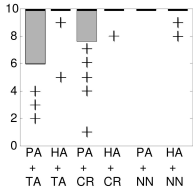


(e) 5-bug subjects' trials

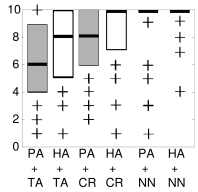


(f) all subjects' trials

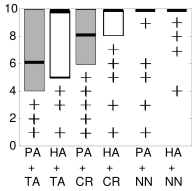
Latency (the shorter the better)



(g) 1-bug subjects' trials

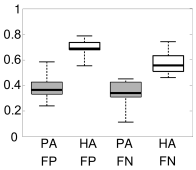


(h) 5-bug subjects' trials

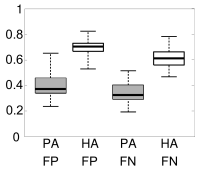


(i) all subjects' trials

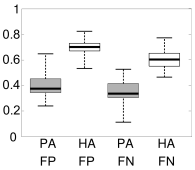
Oracle false positive rate (R_{FP}) and false negative rate (R_{FN})



(j) 1-bug subjects' trials



(k) 5-bug subjects' trials



(l) all subjects' trials

Figure 5.2: Evaluation Results: ArduPilot with real-life bugs

For the ArduPilot test-bed, guided by Table 5.2, we inject various artificial bugs into the cyber subsystem source code. These candidate artificial bugs and injection locations are listed in Table 5.3. We create 61 versions (i.e. 61 subjects) of buggy ArduPilot cyber subsystems. 17 of the 61 subjects respectively contain one of the candidate artificial bugs in Table 5.3. The other 44 subjects each contains 5 candidate artificial bugs selected from Table 5.3.

For each buggy ArduPilot cyber subsystem subject, our emulation platform runs 3 trials. Each trial generates 2665 source code execution traces (and the corresponding physical trajectories and user input traces). These source code execution traces are then labeled by our MPC-SI oracle as described by Fig. 4.3-**Step2-Task2** and **Step3**.

The 2665 labeled source code execution traces are then sent to the TA, CR, and NN SFL tools to find bugs. Same as Section 5.3, the SFL tools respectively output an $l = 10$ item suspect list.

In comparison, we also carry out the human oracle approach (see Fig. 4.4). Similarily, in the human oracle approach evaluation, each buggy ArduPilot subject is also given 3 trials. In each trial, the emulation plaform generates 2665 source code execution traces (and the corresponding physical trajectories, user input traces)[3]. These source code execution traces are then labeled by the human oracle (see Section 5.3 **H1.1**$\sim$**H1.3** for the details), and then sent to TA, CR, and NN for SFL.

The evaluation results are plotted in Fig. 5.3. In the figure:

X-axis for (a)$\sim$(i): oracle approach + SFL tool. X-axis for (j)$\sim$(l): oracle approach. PA: our proposed approach; HA: human oracle approach. TA: Tarantula; CR: Crosstab; NN: BP Neural Network. FP: oracle false positive rate; FN: oracle false negative rate.

[3] Due to the same reason as footnote 1, we actually reuse the emulated (unlabeled) source code execution traces (and the corresponding physical trajectories, user input traces) generated in the comparison evaluation for our proposed approach.

Y-axis for (a)∼(i): statistics of per trial SFL quality metrics. Y-axis for (j)∼(l): statistics of per trial oracle false positive/negative rates.

For each given oracle approach and SFL tool, there are $17 \times 3 = 51$ one-bug subject trials and $44 \times 3 = 132$ five-bug subjects trials, hence a total of 183 trials. Each trial corresponds to 2665 source code execution traces.

In each boxplot [7], the thick bar in each box is the median of the data, the box bottom/top are the 1st and 3rd quartile of the data (see footnote 2 for some subtle details). In (a)∼(i), the "+"s outside the boxes are data outside of the 1st and 3rd quartiles (as they have relatively discrete values, we plot them individually instead of using whiskers to provide more information).

This evaluation results will be discussed in Section 5.5.

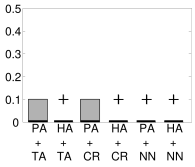
For the IP+CV test-bed, we do the same thing. Guided by Table 5.2, we inject various artificial bugs into the cyber subsystem source code. These candidate artificial bugs and injection locations are listed in Table 5.4. We create 12 versions (i.e. 12 subjects) of buggy IP+CV cyber subsystems. 6 of the 12 subjects respectively contain one of the candidate artificial bugs in Table 5.4. The other 6 subjects each contain 5 candidate artificial bugs selected from Table 5.4.

For each buggy IP+CV cyber subsystem subject, our emulation platform runs 3 trials. Each trial generates 65 source code execution traces (and the corresponding physical trajectories and user input traces). These source code execution traces are then labeled by our MPC-SI oracle as described by Fig. 4.3-**Step2**-**Task2** and **Step3**.

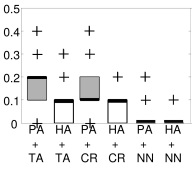
The 65 labeled source code execution traces are then sent to the TA, CR, and NN SFL tools to find bugs. Same as Section 5.3, the SFL tools respectively output an $l = 10$ item suspect list.

In comparison, we also carry out the human oracle approach (see Fig. 4.4). Simply put, in the human oracle approach evaluation, each buggy IP+CV subject is also given 3 trials. In each trial, the emulation platform generates 65 source code execution traces (and

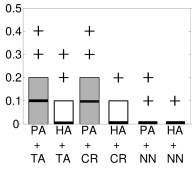
Accuracy (the higher the better)



(a) 1-bug subjects' trials

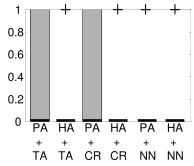


(b) 5-bug subjects' trials

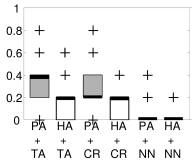


(c) all subjects' trials

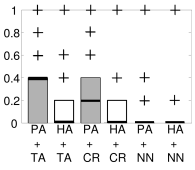
Recall (the higher the better)



(d) 1-bug subjects' trials

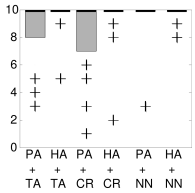


(e) 5-bug subjects' trials

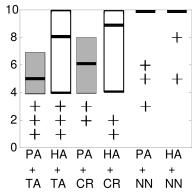


(f) all subjects' trials

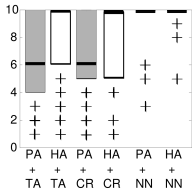
Latency (the shorter the better)



(g) 1-bug subjects' trials

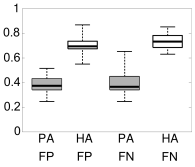


(h) 5-bug subjects' trials

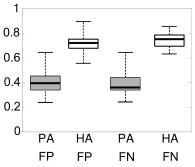


(i) all subjects' trials

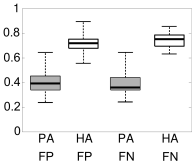
Oracle false positive rate (R_{FP}) and false negative rate (R_{FN})



(j) 1-bug subjects' trials



(k) 5-bug subjects' trials



(l) all subjects' trials

Figure 5.3: Evaluation Results: ArduPilot with artificial bugs

the corresponding physical trajectories, user input traces)[4]. These source code execution traces are then labeled by the human oracle, and then sent to TA, CR, and NN for SFL.

Specifically, the human oracle for IP+CV runs as follows. A source code execution trace is labeled "correct" iff its corresponding physical trajectory obeys all the rules **H2.1**$\sim$**H2.3** listed below; and labeled "incorrect" otherwise.

H2.1 When the IP velocity is within ± 0.01837m/s, the IP angular velocity shall never exceed ± 0.0731rad/s.

H2.2 The IP angular displacement shall never exceed ± 0.09rad.

H2.3 The IP velocity shall never exceed ± 0.4545m/s.

Rules **H2.1** $\sim$ **H2.3** are given by a panel of four domain experts via thorough discussions. two of the four domain experts have PhD degrees and respectively have over 15 and 9 years of experiences on control systems (including IP) and CPS research and development. Two of the four domain experts have over three years of computer vision program development experiences.

The evaluation results are plotted in Fig. 5.4. In the figure:

X-axis for (a)$\sim$(i): oracle approach + SFL tool. X-axis for (j)$\sim$(l): oracle approach. PA: our proposed approach; HA: human oracle approach. TA: Tarantula; CR: Crosstab; NN: BP Neural Network. FP: oracle false positive rate; FN: oracle false negative rate.

Y-axis for (a)$\sim$(i): statistics of per trial SFL quality metrics. Y-axis for (j)$\sim$(l): statistics of per trial oracle false positive/negative rates.

For each given oracle approach and SFL tool, there are $6 \times 3 = 18$ one-bug subject trials and $6 \times 3 = 18$ five-bug subject trials, hence a total of 32 trials. Each trial corresponds to 65 source code execution traces.

[4] Due to the same reason as footnote 1, we actually reuse the emulated (unlabeled) source code execution traces (and the corresponding physical trajectories, user input traces) generated in the comparison evaluation for our proposed approach.

In each boxplot [7], the thick bar in each box is the median of the data, the box bottom/top are the 1st and 3rd quartile of the data (see footnote 2 for some subtle details). In (a)∼(i), the "+"s outside the boxes are data outside of the 1st and 3rd quartiles (as they have relatively discrete values, we plot them individually instead of using whiskers to provide more information).

This evaluation results will be discussed in Section 5.5.

5.5 Discussions on Evaluation Results

In this section, we discuss the results of our evaluations, aiming to answer the research questions **Q1** and **Q2** (see Section 5.1).

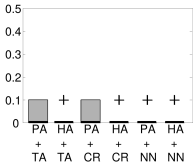
Corresponding to **Q1**, Fig. 5.2(a)∼(i), Fig. 5.3(a)∼(i), and Fig. 5.4(a)∼(i) all show our proposed approach outperforms the human oracle approach in accuracy, recall, and latency in the TA and CR SFL. The significance and magnitude of the improvement are respectively quantified by the p-values [10,39] and *effect size* (ES) [8,129] values in Table **??**. Note p-values estimate the probabilities that the comparison differences are only by chance [39], hence the smaller the more significant are the comparison differences. ES measures the magnitude of the difference. Our ES are quantified by Cohen's d [129]: a magnitude (i.e. the absolute value of ES) of over 0.4 indicates the difference magnitude is at least medium.

As shown in Table 5.5, 5.6 and, 5.7, for accuracy, recall, and latency comparisons related to the TA and CR SFL, the p-values are nearly[5] always below 5%, hence the improvement of our proposed approach over the human oracle approach is significant. Besides, the corresponding ES value magnitudes are nearly[6] always above 0.4, and often exceed 0.8,

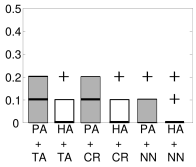
[5] Except for 4 cases (out of the total 54 cases) slightly over 5%, respectively for the accuracy and recall of 1-bug CR SFL of ArduPilot artificial bugs (see Fig. 5.3), and for the accuracy and recall of 1-bug CR SFL of IP+CV artificial bugs (see Fig. 5.4).

[6] Except for 4 cases (out of the total 54 cases) slightly below 0.4, respectively for the accuracy and recall of 1-bug CR SFL of ArduPilot artificial bugs (see Fig. 5.3), and for the latency of 5-bug and all subjects TA SFL

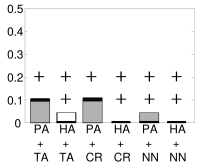
Accuracy (the higher the better)



(a) 1-bug subjects' trials

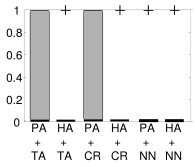


(b) 5-bug subjects' trials

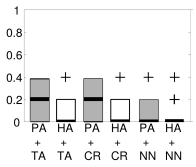


(c) all subjects' trials

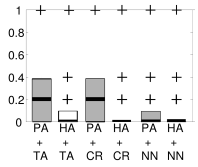
Recall (the higher the better)



(d) 1-bug subjects' trials

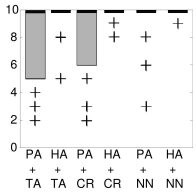


(e) 5-bug subjects' trials

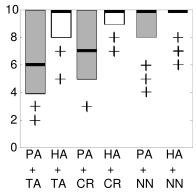


(f) all subjects' trials

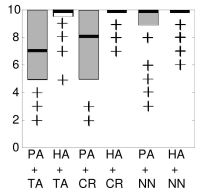
Latency (the shorter the better)



(g) 1-bug subjects' trials

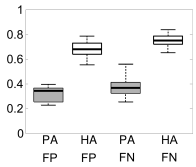


(h) 5-bug subjects' trials

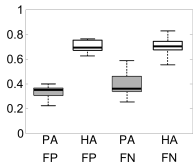


(i) all subjects' trials

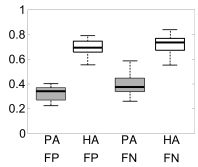
Oracle false positive rate (R_{FP}) and false negative rate (R_{FN})



(j) 1-bug subjects' trials



(k) 5-bug subjects' trials



(l) all subjects' trials

Figure 5.4: Evaluation Results: IP+CV with artificial bugs

implying the magnitude of the improvement is medium to large.

Table 5.5: Quality of Fig. 5.2 Statistics

Fig. 5.2 PA vs HA Comparisons (on ArduPilot, with real-life bugs)							
metric		**1-bug sbj**		**5-bug sbj**		**all sbj**	
		p-value	**ES**	**p-value**	**ES**	**p-value**	**ES**
accuracy	**TA**	2.9%	0.65	<1‰	0.53	<1‰	0.51
	CR	0.7%	0.82	<1‰	0.82	<1‰	0.77
	NN	15.5%	-	36.4%	-	24.0%	-
recall	**TA**	2.9%	0.65	<1‰	0.53	<1‰	0.50
	CR	0.7%	0.82	<1‰	0.82	<1‰	0.69
	NN	15.5%	-	36.4%	-	9.5%	-
latency	**TA**	1.4%	-0.74	4.4%	-0.32	0.5%	-0.37
	CR	0.9%	-0.79	0.6%	-0.43	<1‰	-0.47
	NN	17.9%	-	37.7%	-	46.3%	-
R_{FP}		<1‰	-4.9	<1‰	-4.5	<1‰	-4.6
R_{FN}		<1‰	-3.0	<1‰	-3.6	<1‰	-3.4

of ArduPilot real-life bugs (see Fig. 5.2).

Table 5.6: Quality of Fig. 5.3 Statistics

Fig. 5.3 PA vs HA Comparisons (on ArduPilot, with artificial bugs)							
metric		**1-bug sbj**		**5-bug sbj**		**all sbj**	
		p-value	**ES**	**p-value**	**ES**	**p-value**	**ES**
accuracy	**TA**	3.3%	0.43	<1‰	0.98	<1‰	0.73
	CR	5.5%	0.38	<1‰	0.72	<1‰	0.57
	NN	65.0%	-	54.7%	-	46.3%	-
recall	**TA**	3.3%	0.43	<1‰	0.98	<1‰	0.64
	CR	5.5%	0.38	<1‰	0.72	<1‰	0.52
	NN	65.0%	-	54.7%	-	50.8%	-
latency	**TA**	0.4%	-0.59	<1‰	-0.68	<1‰	-0.59
	CR	3.4%	-0.43	<1‰	-0.44	<1‰	-0.40
	NN	14.0%	-	30.8%	-	11.3%	-
R_{FP}		<1‰	-5.4	<1‰	-4.8	<1‰	-4.9
R_{FN}		<1‰	-5.2	<1‰	-5.9	<1‰	-5.6

Table 5.7: Quality of Fig. 5.4 Statistics

Fig. 5.4 PA vs HA Comparisons (on IP+CV, with artificial bugs)							
metric		**1-bug sbj**		**5-bug sbj**		**all sbj**	
		p-value	**ES**	**p-value**	**ES**	**p-value**	**ES**
accuracy	**TA**	2.5%	0.78	1.2%	0.85	0.2%	0.75
	CR	5.6%	0.66	2.2%	0.88	0.3%	0.72
	NN	38.6%	-	42.9%	-	37.2%	-
recall	**TA**	2.5%	0.78	1.2%	0.85	0.4%	0.70
	CR	5.6%	0.66	2.2%	0.88	0.9%	0.63
	NN	38.6%	-	42.9%	-	32.3%	-
latency	**TA**	1.5%	-0.86	2.3%	-1.05	<1‰	-0.94
	CR	1.4%	-0.86	0.2%	-1.24	<1‰	-1.03
	NN	8.6%	-	20.5%	-	7.0%	-
R_{FP}		<1‰	-6.1	<1‰	-7.4	<1‰	-6.6
R_{FN}		<1‰	-5.5	<1‰	-4.4	<1‰	-4.8

For the NN SFL, neither our proposed approach nor the human oracle approach works well. This is possibly because NN, being a neural network solution, needs big data to really work. As big data solutions may often need millions to trillions of items in the training set, the tens or thousands of labeled source code execution traces that we generate for each trial are probably too small.

Anyway, for NN, the lack of difference between our proposed approach and the human oracle approach is reflected in the corresponding p-values in Table 5.5, 5.6 and, 5.7. For accuracy, recall, and latency comparisons related to the NN SFL, the p-values are big.

This also means that it is meaningless to evaluate the magnitude of improvement, i.e. the corresponding ES values (which are hence omitted in the table).

Corresponding to **Q2**, Fig. 5.2(j)∼(l), Fig. 5.3(j)∼(l), and Fig. 5.4(j)∼(l) all show our proposed approach outperforms the human oracle approach in R_{FP} and R_{FN}. The significance and magnitude of the improvement are respectively quantified by the p-values and ES values in Table 5.5, 5.6 and, 5.7.

As shown in Table 5.5, 5.6 and, 5.7, for R_{FP} and R_{FN} comparisons, the p-values are all much smaller than 5%, hence the improvement of our proposed approach over the human oracle approach is significant. Besides, the corresponding ES value magnitudes are all over 0.8, indicating the magnitude of improvement is large.

In summary, we have the following answers to **Q1** and **Q2**.

*Answer to **Q1**: In our evaluation, compared to the human oracle approach, our proposed approach significantly improves the quality of TA and CR SFL (in terms of accuracy, recall, and latency). The magnitude of improvement is medium to large.*

*Answer to **Q2**: In our evaluation, compared to the human oracle approach, our proposed approach significantly improves the raw quality of the oracle (in terms of false positive/negative rates). The magnitude of improvement is large.*

5.6 Threats to Validity

Construct validity: Threats to construct validity are mainly concerned with whether the measurements used in the evaluation reflect real-world situations.

Human Oracle Validity. The main threat to construct validity lies in the way to construct the human oracle. Individual experts can be biased due to his/her knowledge and skill set differences. To mitigate the bias introduced by individuals, we form panels of at least four domain experts to design the human oracle via thorough discussions.

Golden Oracle Validity. Another threat is in measuring the oracle false positive/negative

rates. To judge the raw correctness (in terms of oracle false positive/negative rates) of oracles, we need a "golen oracle" that tells if the control-CPS behavior is truly buggy or not. Unfortunately, for control-CPSs, what shall be the golden oracle is indeed also an open problem. As shown by Fig. 1.4(b), a so-called "correct" physical trajectory may not be unique. We cannot, for example, use the behavior of a specific implementation (e.g. ArduPilot without bug injection) as the golden oracle, and deny the correctness of other behavioral possibilities.

Given this, we adopt a quasi-golden oracle. We conservatively classify a code trace as truly buggy as long as it covers a buggy source code entity. Such approximation is acceptable considering that our ultimate goal is to locate the buggy source code entity instead of detecting buggy behavior. For this ultimate goal, it makes sense that a code trace covering buggy entities is always reported to SFL tools, even if it may indeed cause no buggy behaviors.

Also, note the golden oracle is not used to label code traces for SFL in our evaluations. Those code traces are labeled by our MPC-SI oracle and/or human oracle. Hence the golden oracle has nothing to do with TA, CR, and NN SFL evaluations.

SFL Tool Representativeness. SFL tools selected for the study may pose a third threat to construct validity. To reduce this threat, we used three representatives from different families of SFL tools and compared their performance in terms of accuracy, recall, and latency in the same evaluation settings. We also plan to conduct extensive evaluations on more SFL tools to further substantiate our findings.

Better SI Methods. We choose the MPC-SI of Exp. (4.1) because it is deterministic, automatic, simple, and empirically well tested by broad adoption in industry [75] [125]. However, MPC-SI of Exp. (4.1) is not the only system identification (SI) method. There are many other SI methods. Nevertheless, the value of this thesis lies in that it reveals the great potential of SI based oracles for control-CPS SFL [119]. Using MPC-SI oracle already outperforms the mainstream human oracle approach significantly. Comprehensive survey of other SI methods may find SI oracles that outperform human oracles even more.

Internal validity: Threats to internal validity are mainly concerned with uncontrolled or uncontrollable factors that may affect the evaluation results.

Evaluation Platform Implementation Correctness. In our evaluations, the main threat to internal validity is the possible bugs in the implementation of the oracle approaches (our proposed approach and the human oracle approach) and the reimplementation of existing SFL tools. To address the threat, we review our code to ensure their correctness before conducting the evaluations.

Setting of p. Another threat is the setting of the MPC model order: p (see Exp. (4.1), (4.2)). In our evaluations, p is set to 7. This is an empirical setting made by our engineers based on their experiences. How to optimize the setting of p is still an open problem. Fortunately, our evaluations show that even with this empirical setting, our proposed approach already outperforms the human oracle approach. With an optimized p, our proposed approach shall outperform the human oracle apporoach even more.

Bug Injection Locations and Combinations. The locations of injected real-life bugs are determined by the public archives (see the File-Function-Line# columns of Table 5.1). While the locations of injected artificial bugs are randomly selected (see the Function-Line# columns of Table 5.3, 5.4). Due to combinatorial explosion and feasibility constraints, we cannot try every possible injection location/combination. But we are tring our best to randomize these selections.

External validity: Threats to external validity are mainly concerned with whether the findings in our evaluations are generalizable to other situations.

Variety of Test-beds and Injected Bugs. The limited types of control-CPSs and bugs used in our evaluations may threaten the external validity. To mitigate the problem, we deploy two widely used control-CPSs as test-beds, and deploy real-life, as well as artificial bugs reflecting commonly seen bugs in field. Such test-beds and bugs, however, may still be unable to represent all control-CPSs and bugs. As future work, we plan to carry out more evaluations on other test-beds and deploy more types of bugs.

Table 5.3: Artificial Bugs to be Injected into ArduPilot

Type	**Function-Line#**	**Bug Description**
WVAV	pos_to_rate_xy-845	mistake ”multiply” by ”divide”
MVAV	calc_leash_length-999	an assignment deleted
WVAV	pos_to_rate_z-365	assigned value negated
MVAE	pos_to_rate_z-360	an assignment deleted
MVAE	pos_to_rate_xy-798	an assignment deleted
WPFV	pos_to_rate_z-378	swapped parameter x,y
MLPA	get_stopping_point_z-287	wrong expression used
MLPA	get_stopping_point_z-289	wrong expression used
MVAV	advance_wp _target_along_track-610	an assignment deleted
MVAV	calculate_wp _leash_length-794	an assignment deleted
MFC	set_wp_origin _and_destination-487	immediate function return
MVIV	calc_slow_down _distance-1232	initialization deleted
MIEB	pos_to_accel_z-410	missing if construct plus statements plus else before ”!_flags.freeze_ff_z”
MIA	pos_to_rate_xy-847	missing if construct
MIFS	calc_leash_length-1000	missing if construct plus ”leash_length ” statements
WAEP	get_stopping_point_z-290	wrong arithmetic expression
MLC	set_alt_target_from_climb _rate_ff-211	a logic clause in branch condition deleted

Table 5.4: Artificial Bugs to be Injected into IP+CV

Type	**Function-Line#**	**Bug Description**
WVAV	cvFindChessboardCorners-260	assigned value negated
MVAE	cvFindChessboardCorners-516	an assignment deleted
WVAV	cvCreateMatHeader-130	assigned value negated
MVIV	cvCreateMatHeader-137	initialization deleted
WVAV	cvInitImageHeader-2973	mistake ”0” by ”-1”
WAEP	cvInitImageHeader-2980	wrong arithmetic operation

CHAPTER 6

RELATE WORK

This thesis focuses on specialized oracles that exploit cross-domain knowledge (particularly control theories) to judge control-CPS *physical* trajectory correctness. Hence the topic is orthogonal to general-purpose oracle design for pure cyber systems [23, 59, 61, 136].

6.1 Related Work in the Domain of Control

The specific cross-domain tool used in this thesis is MPC-SI of Exp. (4.1). We choose MPC-SI because it is deterministic, automatic, simple, and empirically well tested by broad adoption in industry. However, MPC-SI of Exp. (4.1) is not the only system identification (SI) method. In fact, SI is a vast research domain in control theories [120]. There are many other SI methods. But to our best knowledge, our thesis is the first to apply SI in control-CPS SFL oracle design, and reveals the great potential of using SI to build oracles for control-CPS SFL. As our future work, we are interested in applying various other SI methods to build control-CPS SFL oracles. One particular set of SI methods of interest are those identify hybrid automata models [22, 97] from control-CPS behaviors. But to use these SI methods, there are still open issues to be addressed. For example, the work of [22] focuses on converging behaviors when mining hidden hybrid automata, which is not always the case for generic control-CPS behaviors. The work of [97] needs domain knowledge of the control-CPS (e.g. knowing the semantics of feature vectors), hence is not entirely black-box.

Because of the hardness of the oracle problem, people no longer insist on designing a perfect oracle that identifies every correct and incorrect execution trace. Instead, people

now focus more on the so-called *imperfect oracles*, which identify wrong execution traces as much as possible, and choose test cases accordingly. Metamorphic testing [34–36, 137] is one such endeavor. It exploits necessary conditions governing outputs' interrelationships as imperfect oracles. Chen et al. [35] propose a metamorphic testing oracle for control-CPS with PID controller software. Modern control-CPSs, however, are often more complex than PID control. Recently, Goebel et al. [63] propose a set of hybrid systems [25, 51, 116, 118] stability theories for control-CPSs. The stability rules can be used as a metamorphic testing oracle. However, Goebel et al.'s theories require the definition of a "Lyapunov" function for the concerned hybrid system. For a given generic control-CPS, the existence of a Lyapunov function is not guaranteed, neither is there a generic routine to decide its existence. Even if the Lyapunov function exists, there is no generic mechanical routine to find it. Therefore Goebel et al.'s hybrid systems stability theories are mainly for designing new control-CPSs, instead of for designing imperfect oracles for existing and/or generic control-CPSs. Nonetheless, the above endeavors inspire us to propose this paper's MPC based oracle, which is indeed an imperfect oracle in a general sense.

Generally speaking, control-CPS debugging is a relatively open area with huge problem space. Matinnejad et al. [95] discuss automated test case generation for control-CPS debugging. The idea is to adjust test case inputs so as to maximally diverge control-CPS physical trajectories' shapes. Test case generation, however, is not the focus of this thesis. Liu et al. [85] propose a SFL tool specialized for control-CPS whose cyber and physical models are known accurately. However, this thesis focuses on control-CPSs whose accurate cyber model is unavailable.

6.2 Related Work in the Domain of Software Engineering

A research area in software engineering that is closely related to SI is specification inference. Various approaches combining static and dynamic analysis techniques have been developed to infer specifications in the form of discrete automata [67, 98, 107] and behavioral models [45]. Significant effort has also been invested in detecting invariants that govern program

variables and their interrelationships across different program executions. There are two big families of invariant detection techniques, based on static and dynamic program analysis, respectively. Static program analysis based invariant detectors have been successfully used in Airbus avionics [43]. Dynamic trace analysis based invariant detectors can now discover fairly complex math formulae upon program variables [44,54,102,103]. Others have devised ways to construct assertions summarizing program behaviors [58]. The result specifications, however, are derived from all the behaviors used for the inference, and naturally incorporate also the incorrect one, if not pruned out. Such specifications can be used to help programmers understand program behaviors, but not as the oracle to decide whether a behavior is correct or not. Recently, the division between specification inference and control system SI starts to blur. Representative works crossing this division are the aforementioned hybrid automata SI methods [22, 97].

Alippi et al. [18] designed an oracle for control-CPS fault detection. However their focus is on control-CPS sensor faults instead of software faults.

There are other efforts to solve the oracle problem. Several modeling/programming languages are developed to formally describe oracles [50, 107, 117, 128]; in addition, assertions and contracts [99] are forms to describe oracles. But they do not answer the oracle problem itself: what oracles to describe.

We can use N-version [89] or previous versions [139] to generate trajectories for comparisons, but such methods are orthogonal to the oracle problem solutions for debugging a single version of a control-CPS.

Software fault localization is a very important research topic. Due to the feature of time-consuming and expensive of manual intervene, much effort has been spent on developing SFL tools. Major families of tools include program spectrum analysis [77] [70] [87] [55], statistical analysis [21] [131], machine learning based analysis [134] [28] [30], program slicing [123] [88], data mining based analysis [33] [83], and program state based debugging [143] [144]. The first three families of tools, which rely on large number of labeled correct/incorrect traces as inputs, are leveraged to evaluate our proposed oracle. Thus, we

select one representative method in each of these three families to describe the principles of SFL tools.

6.2.1 Tarantula

As a typical spectrum analysis software fault localization tool, Tarantula [77] analyzes the program using the relationship between program execution results and code execution traces. The intuition of Tarantula is that statements which are executed primarily by failed test cases are highly suspicious of being faulty. Thus, Tarantula gathers large amounts of data about a software system under test (by varying inputs). These data include each test result (success/failure) and code execution log traces. Tarantula assigns "suspiciousness value" to each coverage entity e of the program source code (a "coverage entity" can be a line of code, a block of code, a function, etc.), using the following formula:

$$suspiciousness(e) = \frac{\frac{failed(e)}{totalfailed}}{\frac{passed(e)}{totalpassed} + \frac{failed(e)}{totalfailed}} \tag{6.1}$$

where failed(e) is the number of failed test cases which execute entity e, totalfailed refers to the number of all failed test cases, passed(e) means the number of passed test cases executing e, and totalpassed is the number of all passed test cases.

After each entity gets its suspicious value, Tarantula will sort the score to form a suspiciousness list. This list guides the programmer to search for the software fault one by one from the top of suspicious list. This can be illustrated by an example from Fig. 6.1:

As we see, line 7, which has the suspiciousness value 0.83, is the first fault Tarantula locates. Line 6 needs the second round check with a score of 0.71.

6.2.2 Crosstab

Besides spectrum-based SFL method, the Crosstab [131] fault localization tool is a representative statistics-based SFL. Similar to Tarantula, Crosstab also utilizes the coverage infor-

	Test Cases						suspiciousness	rank
mid() { int x,y,z,m;	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3		
1: read("Enter 3 numbers:",x,y,z);	●	●	●	●	●	●	0.5	7
2: m = z;	●	●	●	●	●	●	0.5	7
3: if (y<z)	●	●	●	●	●	●	0.5	7
4: if (x<y)	●	●			●	●	0.63	3
5: m = y;		●					0.0	13
6: else if (x<z)	●				●	●	0.71	2
7: m = y; // *** bug ***	●					●	0.83	1
8: else			●	●			0.0	13
9: if (x>y)			●	●			0.0	13
10: m = y;			●				0.0	13
11: else if (x>z)				●			0.0	13
12: m = x;							0.0	13
13: print("Middle number is:",m);	●	●	●	●	●	●	0.5	7
} Pass/Fail Status	P	P	P	P	P	F		

Figure 6.1: Tarantula example

mation of each executable statement and the execution result (success or failure). Crosstab analyzes that whether the positive execution result relates to the entity using Chi-square test. Here we list the notations as table. 6.1 in the first part:

Table 6.1: Notations in crosstab method

N	total number of test cases
N_F	total number of failed test cases
N_S	total number of passed cases
$N_C(e)$	number of test cases covering e
$N_{CF}(e)$	number of failed test cases covering e
$N_{CS}(e)$	number of passed test cases covering e
$N_U(e)$	number of test cases not covering e
$N_{UF}(e)$	number of failed test cases not covering e
$N_{US}(e)$	number of passed test cases not covering e

Here, we assume the null hypothesis is:

H_0: Program execution is independent of the coverage of e.

Using the chi-square value, we can point out this hypothesis can be rejected. The following equation illustrate the chi-square statistic:

$$\chi^2(e) = \frac{\big(N_{CF}(e) - E_{CF}(e)\big)^2}{E_{CF}(e)} + \frac{\big(N_{CS}(e) - E_{CS}(e)\big)^2}{E_{CS}(e)} + \frac{\big(N_{UF}(e) - E_{UF}(e)\big)^2}{E_{UF}(e)} + \frac{\big(N_{US}(e) - E_{US}(e)\big)^2}{E_{US}(e)} \tag{6.2}$$

where

$$E_{CF}(e) = \frac{N_C(e) \times N_F}{N} \tag{6.3}$$

$$E_{CS}(e) = \frac{N_C(e) \times N_S}{N} \tag{6.4}$$

$$E_{UF}(e) = \frac{N_U(e) \times N_F}{N} \tag{6.5}$$

$$E_{US}(e) = \frac{N_U(e) \times N_S}{N} \tag{6.6}$$

If $\chi^2(e)$ is large, it means the entity e has some impact on the program execution result. The question lies on whether executing e leads to passed program execution result or the fail ones. Thus, statistic $\varphi(e)$ is leveraged to analyze this question:

$$\varphi(e) = \frac{\frac{N_{CF}(e)}{N_F}}{\frac{N_{CS}(e)}{N_S}} \tag{6.7}$$

If $\varphi(e)$ ¿ 1, the coverage of e is more related to the failed result. If $\varphi(e)$ ¡ 1, the coverage of e is more related to the passed result. Finally the Crosstab method defines the suspiciousness value $\zeta(e)$ as:

$$\zeta(e) = \begin{cases} \frac{\chi^2(e)}{N_P} & \text{if } \varphi(e) > 1 \\ 0 & \text{if } \varphi(e) = 1 \\ -\frac{\chi^2(e)}{N_S} & \text{if } \varphi(e) < 1 \end{cases} \tag{6.8}$$

Using this suspiciousness value, programmer can locate the fault from the code execution log owning the highest score.

6.2.3 BP Neural Network-based (BPNN) Approach

A category of fault localization methods is using machine learning to locate the software defects. Here we choose a representative method [134] based on a back-propagation (BP) neural network. BPNN approach bridges the coverage information and program execution result via neural network training, and locates the bugs by feeding “virtual test cases” into the trained network.

BPNN first label all the coverage entities using s_1, s_2 as and so on. Each entity is marked as either 0 (if s_i is not covered in the test case) or 1 (if s_i is covered in the test case). For example, in the Fig. 6.2, there are 7 test cases. In each test case, 9 coverage entities are included. If the program covers certain coverage entities, the according labels become 1 or otherwise 0 instead. Let m be the number of coverage entities and n be the number of test cases. Thus, n vectors serve as the input to a BP neural network. This BP neural network is a typical neural network which has a three-layer network with 3 neurons in the hidden layer and 1 neuron in the output layer. The according transfer function of each neuron is sigmoid function $y = \frac{1}{1+e^{-x}}$. After the preparation of the neural network, the BPNN approach use these n vectors and n program execution results (0 if the test case is passed or 1 if the test case is failed) as the material to train the neural network.

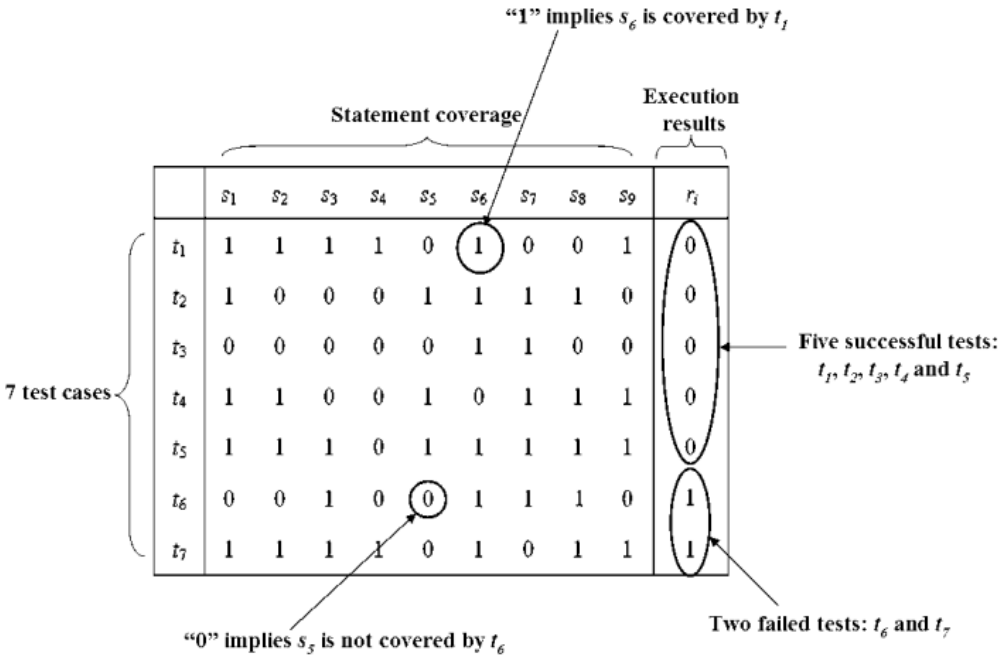


Figure 6.2: BPNN test cases example

The basic idea of BPNN is to use the constructed input and known output to fed into the network. The error between the actual output and the expected output is calculated and propagated back. Similar to back-propagation algorithm, the weights of connecting all the neurons are adjusted in order to reduce the error. Bayesian regularization is leveraged to

prevent the overfitting problem in the training process. Fig. 6.3 explains the generation of the estimated execution results.

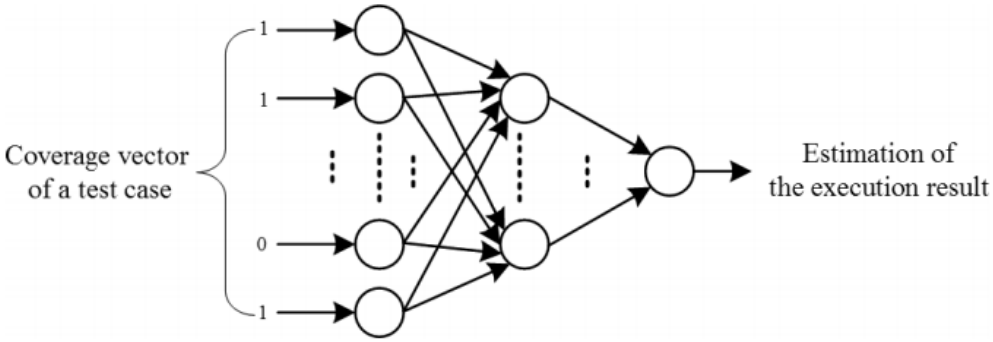


Figure 6.3: BPNN constructure

After the network is trained, virtual test cases are constructed to serve as input of the network (Fig. 6.4). In order to evaluate the weight of each coverage entity in the vector, each input vector only covers one coverage entity (label as 1) and exclude others (label as 0). Via feeding the input into this neural network, we can get the suspiciousness values of each coverage entity in the output vectors. This process is explained in the Fig. 6.4.

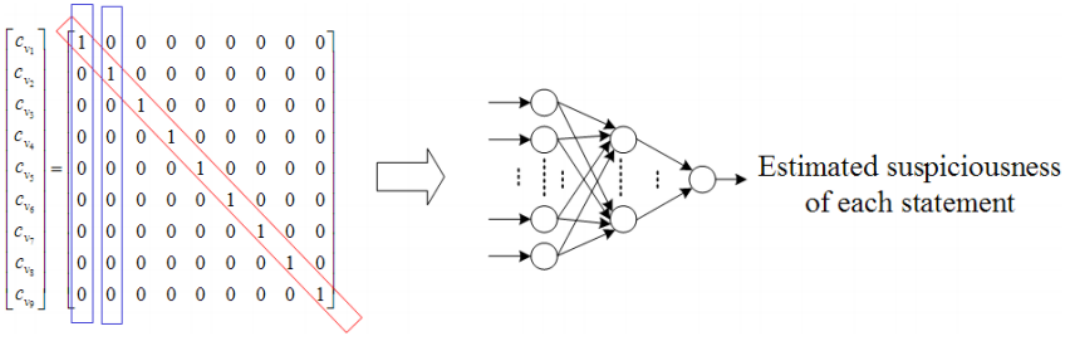


Figure 6.4: BPNN suspiciousness estimation

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusion

Rapid convergence of computer and physical system results in increasing demand on control-CPSs functionalities, which requires more and more software design transferring to cyber-subsystem. Thus, the scale of control-CPSs software are inevitable to become huge, and the complexity becomes high. Debugging of control-CPSs software is an important topic in the rapidly growing scale and complexity of control-CPSs software. However, only a few research has been focus on how to test control-CPSs or locate the defects of cybersystem design. Currently, manual debugging is mainly leveraged to locate faults, which indicates the fault localization process is still time consuming and expensive.

In this thesis, we adapted the MPC-SI method to oracle design, and proposed an oracle and corresponding source code execution trace preparation methodology for control-CPS SFL. Our proposed approach can be carried out deterministically and automatically. We evaluated our proposal on classic control-CPS test-beds with real-life and artificial bug SFL. The evaluation results show that our proposed approach significantly outperforms the human oracle approach, achieving medium to large improvements on SFL accuracy, recall, and latency, as well as oracle false positive/negative rates.

7.2 Future Work

In this thesis, the theory of MPC-SI is leveraged to evaluate the physical behaviors of control-CPSs. This oracle serves as the labelling machine for SFL so that the software fault localiza-

tion can be generated automatically and systematically. We can improve this MPC-SI based oracle approach in the following aspects.

7.2.1 Golden Oracle Selection

An important aspect of evaluating oracle is whether its judgements are correct. In this thesis, we measure this performance using false positive/negative rates. The false positive/negative rates consider an source code execution trace as truly buggy as long as it covers a buggy entity. This statement can be improved because the influence of covering buggy entity on physical behavior needs to be evaluated. The circumstance that buggy entity execution code generates no consequence on control-CPSs behavior is rational in certain initial settings. As a result, what type of buggy entity can generate turbulence on control-CPSs behavior and how does it affect the behavior should be considered.

Another aspect of evaluating oracle is whether it approximates the "golden oracle". However, how to select a "golden oracle" in control-CPSs domain is still an open question. Some research select a set of principles to work as a "god". But the "god" judgement on control-CPSs is difficult to be generalized due to specific features in different hardware. Thus, what is the evaluating matrix on specific control-CPSs should be discussed in the future.

7.2.2 Target Systems Selection

We build our experiment on both ArduPilot (commercial-grade control-CPSs), and IP+CV (classical control-CPSs). In order to prove our MPC-SI oracle approach can be generalized, more target systems should be selected. Via constructing more experiments on different CPS platforms, we are able to point out what is the constraint on using MPC-SI theory to work as an oracle. Also, more control-CPSs can improve the rationality of source code execution trace preparation methodology, which can enhance the software fault localization ability of MPC-SI oracle.

7.2.3 Empirical Study on MPC-SI Model

This thesis discuss the MPC-SI based oracle performance comparing to human oracle. Thus, the target focus on whether our proposed oracle can outperform human in the aspects of accuracy, recall, and latency. In the prospect of fetching the most rational MPC-SI model, this thesis lacks the empirical optimization settings. In our evaluation, p is set to 7. But whether this value can be changed to other value in order to get a better oracle performance needs to be discuss in the future. A more rational MPC-SI model can help us improve the accuracy of labelling physical trajectories.

Appendix

.1 Appendix A: A Formal Description of MPC-SI

The concept of MPC-SI is well-known in control-CPSs [32]. The following formulates the classic MPC model prediction technology upon the control-CPS architecture of Fig. 2.1.

Without loss of generality, we only consider the model building process during $[0, T)$. Given U_0 (which is constant throughout $[0, T)$), $p, i \in \mathbb{Z}_{>0}$ (where $0 < p \leqslant i < N$), and X_j ($j = 0, 1, \cdots, i$), the ith p-order linear model $\mathcal{M}_i$ to predict X_{i+1} has the following form.

$\forall k \in \{p, p+1, \cdots, i+1\}$, we have

$$X_k = (\sum_{j=1}^{p} A_j X_{k-j}) + BU_0 + \xi_k, \tag{1}$$

where ξ_k is the *plant state prediction error*. Note $X_k, \xi_k \in \mathbb{R}^n$, $U_0 \in \mathbb{R}^q$. A_1, A_2, $\cdots$, $A_p \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times q}$ are constant matrices to be determined. Indeed, the task of our linear model prediction is to decide the value of A_1, A_2, $\cdots$, A_p, and B. Specifically, the values shall be set to minimize a cumulative metric on the plant state estimation errors. The following elaborates this idea.

Denote

$$\bar{A} \stackrel{\mathsf{def}}{=} (A_1, A_2, \cdots, A_p, B) \in \mathbb{R}^{n \times (np+q)}, \tag{2}$$

$$Y_k \stackrel{\mathsf{def}}{=} (X_{k-1}^{\mathsf{T}}, X_{k-2}^{\mathsf{T}}, \cdots, X_{k-p}^{\mathsf{T}}, U_0^{\mathsf{T}})^{\mathsf{T}} \in \mathbb{R}^{(np+q) \times 1}, \tag{3}$$

then Exp. (1) becomes $X_k = \bar{A} Y_k + \xi_k$. Let

$$\hat{X}_k \stackrel{\mathsf{def}}{=} \bar{A} Y_k, \tag{4}$$

be the *prediction of plant state* X_k. Then the plant state prediction error can be rewritten as

$$\xi_k = X_k - \hat{X}_k. \tag{5}$$

We define the ith *cumulative plant state prediction error* as

$$\mathcal{J}_i \stackrel{\text{def}}{=} \sum_{k=p}^{i} \lambda^{i-k} ||\xi_k||^2 = \sum_{k=p}^{i} \lambda^{i-k} \xi_k^{\mathsf{T}} \xi_k, \tag{6}$$

where preconfigured constant $\lambda \in (0, 1)$ is the forgetting factor. As X_k, Y_k, λ ($k = p, \cdots, i$) are given, Exp. (4)(5)(6) imply $\mathcal{J}_i$ is a function of $\bar{A}$. That is, the objective of "model building" is to find the optimal $\bar{A}$ to minimize $\mathcal{J}_i$.

Suppose the optimal $\bar{A}$ that minimizes $\mathcal{J}_i$ is $\bar{A}_i^*$. Then $\bar{A}_i^*$ can be obtained via the following procedure.

$\forall k \in \{p, p+1, \cdots, i\}$, denote

$$\xi'_{i,k} \stackrel{\text{def}}{=} \sqrt{\lambda^{i-k}}\xi_k, \quad X'_{i,k} \stackrel{\text{def}}{=} \sqrt{\lambda^{i-k}}X_k, \quad Y'_{i,k} \stackrel{\text{def}}{=} \sqrt{\lambda^{i-k}}Y_k.$$

Thus $\mathcal{J}_i = \sum_{k=p}^{i} (\xi'^{\mathsf{T}}_{i,k} \xi'_{i,k})$ and $\xi'_{i,k} = X'_{i,k} - \bar{A} Y'_{i,k}$.

The latter implies that $(\xi'_{i,i}, \xi'_{i,i-1}, \cdots, \xi'_{i,p}) = (X'_{i,i}, X'_{i,i-1}, \cdots, X'_{i,p}) - \bar{A}(Y'_{i,i}, Y'_{i,i-1}, \cdots, Y'_{i,p})$.

Based on least-square approximation theories [71], we have

$$\bar{A}_i^* = \bar{X}'_i \bar{Y}'^{\mathsf{T}}_i (\bar{Y}'_i \bar{Y}'^{\mathsf{T}}_i)^{-1}, \tag{7}$$

where

$$\begin{aligned} \bar{X}'_i &\stackrel{\text{def}}{=} (X'_{i,i}, X'_{i,i-1}, \cdots, X'_{i,p}) \in \mathbb{R}^{n \times (i-p+1)}, \\ \bar{Y}'_i &\stackrel{\text{def}}{=} (Y'_{i,i}, Y'_{i,i-1}, \cdots, Y'_{i,p}) \in \mathbb{R}^{(np+q) \times (i-p+1)}. \end{aligned}$$

Note that computation of $(\bar{Y}'_i \bar{Y}'^{\mathsf{T}}_i)^{-1}$ in Exp. (7) is very time-consuming when $np+q$ is large. A faster way is to calculate $\bar{A}_i^*$ with an iterative formula. Details are as follows.

In case we have already built the linear model $\mathcal{M}_{i-1}$, and have derived its corre-

sponding $\bar{X}'_{i-1}$, $\bar{Y}'_{i-1}$. Then because

$$\begin{aligned} \bar{X}'_i &= (X'_{i,i}, \sqrt{\lambda}\bar{X}'_{i-1}) = (X_i, \sqrt{\lambda}\bar{X}'_{i-1}) \\ \text{and} \quad \bar{Y}'_i &= (Y'_{i,i}, \sqrt{\lambda}\bar{Y}'_{i-1}) = (Y_i, \sqrt{\lambda}\bar{Y}'_{i-1}), \\ \text{we have } \Psi_i &\stackrel{\text{def}}{=} \bar{X}'_i\bar{Y}'^{\mathsf{T}}_i = \lambda\bar{X}'_{i-1}\bar{Y}'^{\mathsf{T}}_{i-1} + X_iY_i^{\mathsf{T}} \\ \text{and } \Phi_i &\stackrel{\text{def}}{=} \bar{Y}'_i\bar{Y}'^{\mathsf{T}}_i = \lambda\bar{Y}'_{i-1}\bar{Y}'^{\mathsf{T}}_{i-1} + Y_iY_i^{\mathsf{T}}. \end{aligned}$$

We have the following lemma [71].

Lemma .1.1 (Matrix Inverse Formula [71])**.** *Given matrices A, B, C, D with appropriate dimensions, if $A = B + CDC^{\mathsf{T}}$ and A, B, D are invertible, then*

$$A^{-1} = B^{-1} - B^{-1}C(D^{-1} + C^{\mathsf{T}}B^{-1}C)^{-1}C^{\mathsf{T}}B^{-1}.$$

Due to Lemma .1.1 and note $\Phi_{i-1} \stackrel{\text{def}}{=} \bar{Y}'_{i-1}\bar{Y}'^{\mathsf{T}}_{i-1}$, we have

$$\begin{aligned} \Phi_i^{-1} &= \lambda^{-1}\Phi_{i-1}^{-1} - \\ &\quad \lambda^{-2}\Phi_{i-1}^{-1}Y_i(I + \lambda^{-1}Y_i^{\mathsf{T}}\Phi_{i-1}^{-1}Y_i)^{-1}Y_i^{\mathsf{T}}\Phi_{i-1}^{-1}. \end{aligned}$$

Since $Y_i^{\mathsf{T}}\Phi_{i-1}^{-1}Y_i \in \mathbb{R}$ while $\bar{Y}'_i\bar{Y}'^{\mathsf{T}}_i \in \mathbb{R}^{(np+q)\times(np+q)}$, when $np + q > 1$, the computation of $(I + \lambda^{-1}Y_i^{\mathsf{T}}\Phi_{i-1}^{-1}Y_i)^{-1}$ is much faster than $(\bar{Y}'_i\bar{Y}'^{\mathsf{T}}_i)^{-1}$ (see Exp. (7)).

Therefore, our iterative formulae for fast calculation of $\bar{A}_i^*$ are

$$\begin{aligned} \bar{A}_i^* &= \Psi_i\Phi_i^{-1}, \qquad (8) \\ \Psi_i &= \lambda\Psi_{i-1} + X_iY_i^{\mathsf{T}}, \\ \Phi_i &= \lambda\Phi_{i-1} + Y_iY_i^{\mathsf{T}}, \\ \Phi_i^{-1} &= \lambda^{-1}\Phi_{i-1}^{-1} - \\ &\quad \lambda^{-2}\Phi_{i-1}^{-1}Y_i(I + \lambda^{-1}Y_i^{\mathsf{T}}\Phi_{i-1}^{-1}Y_i)^{-1}Y_i^{\mathsf{T}}\Phi_{i-1}^{-1}. \end{aligned}$$

To summarize, during interval $[0, T)$, given U_0, and initial training data X_0, X_1, $\cdots$, X_p, the MPC model prediction procedure is depicted by Fig. 1.

Step1 $i := p$ (model prediction starts from X_{p+1}).

Step2 Use λ, U_0, X_i, X_{i-1}, $\cdots$, X_0 to derive $\bar{A}_i^*$ via Exp. (7).

Step3 Plug $\bar{A}_i^*$'s evaluation of A_1, A_2, $\cdots$, A_p, B into Exp. (1) to create the the ith p-order linear model $\mathcal{M}_i$. Use $\mathcal{M}_i$ to predict X_{i+1} (i.e. calculate $\hat{X}_{i+1}$) via Exp. (4).

Step4 When X_{i+1} becomes available, use Exp. (5) to calculate plant state prediction error ξ_{i+1}.

Step5 If $i + 1 \geqslant N$, i.e. the current user input sampling period ends, terminate (to be restarted by the next user input sampling period); otherwise, $i := i + 1$ and go to **Step2**.

Figure 1: MPC model prediction procedure. Note due to the causality of our emulation, X_N is still affected by U_0, hence is included in the prediction comparison. Meanwhile, X_0 is included in the learning of the MPC model as it is the initial state of the plant.

.2 Details of quaternion computation

For two quaternions $a = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix}$ and $b = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix}$, their cross-product $a \otimes b$ is defined as:

$$a \otimes b = \begin{bmatrix} a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4 \\ a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3 \\ a_1b_3 - a_2b_4 + a_3b_1 - a_4b_2 \\ a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1 \end{bmatrix}^T \tag{9}$$

To get the Euler angles ψ, θ and ϕ, one quaternion is used to map the respective rotation matrix ${}^A_B R$:

$${}^A_B R = \begin{bmatrix} 2q_1^2 - 1 + 2q_2^2 & 2(q_2q_3 + q_1q_4) & 2(q_2q_4 - q_1q_3) \\ 2(q_2q_3 - q_1q_4) & 2q_1^2 - 1 + 2q_3^2 & 2(q_3q_4 + q_1q_2) \\ 2(q_2q_4 + q_1q_3) & 2(q_3q_4 - q_1q_2) & 2q_1^2 - 1 + 2q_4^2 \end{bmatrix} \tag{10}$$

.3 Details of EKF-based algorithm

We consider the following nonlinear system [60]:

$$^{E}x_k = f(^{E}x_{k-1}) + w_{k-1} \tag{11}$$

$$^{S}z_k = h(^{E}x_k) + v_{k-1} \tag{12}$$

where state vector $^{E}x_k$ is yaw angle at time k, w_{k-1} is the process noise vector from gyroscopes, $^{S}z_k$ is the observation vector which means magnetic measurement in the sensor frame at time k, v_{k-1} is the measurement noise vector from compass.

From Eq. (11), we can substitute yaw direction gyroscopes into process function f(.) to get the process equation:

$$^{E}x_k = {}^{E}x_{k-1} + {}^{E}_{S}R_{k-1}{}^{S}g_{yaw,k-1}\Delta t + w_{k-1} \tag{13}$$

Variable ${}^{E}_{S}R_{k-1}$ denotes the rotation matrix described the earth frame refer to sensor frame at time k-1, $^{S}g_{yaw,k-1}$ is the gyroscope measurement in yaw direction at time k-1 in the sensor frame, Δt is the time step in this EKF equations. Basically, ${}^{E}_{S}R_{k-1}{}^{S}g_{yaw,k-1}\Delta t$ means the change of yaw angle integrated by gyroscopes in terms of the earth frame. Obviously, Eq. (13) is a linear description of state vector $^{E}x_k$.

In the sake of non-observable nature between the magnetic measurement in the sensor frame $^{S}z_k$, and the state vector $^{E}x_k$, we choose the observation $^{S}z_k$ convert to the yaw angle calculated by post-calibration magnetic data:

$$\nabla f({}^{S}_{E}q, {}^{E}d, {}^{S}s) = J^{T}({}^{S}_{E}q, {}^{E}d) f({}^{S}_{E}q, {}^{E}d, {}^{S}s) \tag{14}$$

$${}^{S}_{E}\dot{q}_{\epsilon,t} = \frac{\nabla f}{\|\nabla f\|} \tag{15}$$

$${}^{S}_{E}\dot{q}_{est,t} = {}^{S}_{E}\dot{\hat{q}}_{est,t} - \beta {}^{S}_{E}\dot{q}_{\epsilon,t} \tag{16}$$

$${}^{S}_{E}q_{est,t} = {}^{S}_{E}q_{est,t-1} + {}^{S}_{E}\dot{q}_{est,t}\Delta t \tag{17}$$

$$\psi = \arctan(2q_2q_3 - 2q_1q_4, 2q_1^2 - 1 + 2q_2^2) \tag{18}$$

As a result, nonlinear observation Eq. 47 can be converted to the following linear equation:

$$^{E}z_k = {}^{E}x_k + v_{k-1} \tag{19}$$

The whole system now is able to use Kalman Filter iteration to get a smooth-change yaw angle under the position-dependent magnetic distortion. The initial state $^{E}x_0$ is a random yaw angle in the starting calibration place with mean $^{E}\mu_0 = E[^{E}x_0]$ and covariance $P_0 = E[(^{E}x_0 - {}^{E}\mu_0)(^{E}x_0 - {}^{E}\mu_0)^T]$. We assume the Gaussian noise in the process function w_k, and in the observation function v_k are both random sequences data with zero-mean and known covariances. Both of them are uncorrelated with the state vector ^{E}x [60]:

$$E[w_k] = 0; E[w_k w_k^T] = Q_k; E[^{E}x w_k^T] = 0 \tag{20}$$

$$E[v_k] = 0; E[v_k w_k^T] = R_k; E[^{E}x v_k^T] = 0 \tag{21}$$

According to the law of Kalman Filter iteration, we have the optimal estimate $^{E}x^a_{k-1}$ with P_{k-1} covariance at time t-1 after the observation from compass measurement $^{E}z_{k-1}$. The next predictable state vector $^{E}x^f_k$ can be generated by Eq. (13):

$$^{E}x^f_k = E[^{E}x_k|^{E}z_{k-1}] \tag{22}$$

$$^{E}x^f_k = E[^{E}x_{k-1} + {}^{E}_{S}R_{k-1}{}^{S}g_{yaw,k-1}\Delta t + w_{k-1}|^{E}z_{k-1}] \tag{23}$$

We simplify the component $H.O.T_p = {}^{E}_{S}R_{k-1}{}^{S}g_{yaw,k-1}\Delta t + w_{k-1}$ and expand the equation in Taylor Series at the point $^{E}x^a_{k-1}$ we get:

$$^{E}x_{k-1} + H.O.T_p = {}^{E}x^a_{k-1} + {}^{E}x_{k-1} - {}^{E}x^a_{k-1} + H.O.T_p \tag{24}$$

Because the Eq. (24) is linear function related to ${}^{E}x_{k-1}$, Jacobian of the function f(${}^{E}x_{k-1}$) is equivalent to 1 respectively. The element ${}^{E}x_{k-1} - {}^{E}x^{a}_{k-1}$ is the error between estimate and real state vector value whose expectation is equivalent to 0. As a result, we can get the forecast value ${}^{E}x^{f}_{k}$ is:

$$ {}^{E}x^{f}_{k} \approx {}^{E}x^{a}_{k-1} \tag{25} $$

The forecast error covariance is given by the Kalman Filter law:

$$ P^{f}_{k} = E[({}^{E}x_{k-1} - {}^{E}x^{a}_{k-1})({}^{E}x_{k-1} - {}^{E}x^{a}_{k-1})^{T}] \tag{26} $$

$$ P^{f}_{k} = P_{k-1} + Q_{k-1} \tag{27} $$

The next step is so-called data assimilation step. After getting forecast value ${}^{E}x^{f}_{k}$ with the covariance P^{f}_{k} and the measurement ${}^{E}z_{k}$, the goal of this data assimilation step is to approximate the unbiased estimation ${}^{E}x^{a}_{k}$ of ${}^{E}x_{k}$. Using the prove in , we have:

$$ {}^{E}x^{a}_{k} = {}^{E}x^{f}_{k} + K_{k}({}^{E}z_{k} - E[h({}^{E}x_{k})|{}^{E}z_{k}]) \tag{28} $$

K_{k} denotes the Kalman gain while ${}^{E}z_{k}$ is the yaw angle calculated from post-calibration compass measurement at time k. Similar to the process function, we expand the observation function in Taylor Series about ${}^{E}x^{f}_{k}$ because Jacobian of the function h(${}^{E}x_{k}$) is equivalent to 1:

$$ {}^{E}x_{k} + H.O.T_{o} = {}^{E}x^{f}_{k} + {}^{E}x_{k} - {}^{E}x^{f}_{k} + H.O.T_{o} \tag{29} $$

In order to without loss of generality, we denote $H.O.T_{o} = v_{k-1}$ where v_{k-1} means observation function Gaussian noise. Given $E[h({}^{E}x_{k} - {}^{E}x^{f}_{k})|{}^{E}z_{k}]) = 0$, the state estimate ${}^{E}x^{a}_{k}$ is:

$$^{E}x_k^a \approx {}^{E}x_k^f + K_k({}^{E}z_k - {}^{E}x_k^f) \tag{30}$$

We denote the covariance of value $^{E}x_k^a$ as P_k. Under the constrain of K_k minimizing the trace of P_k as described in , P_k can be described as:

$$P_k = (I - K_k)P_k^f \tag{31}$$

The summary of the above deduction theorem of Kalman Filter iteration can be expressed as:

Model and Observation:

$$^{E}x_k = {}^{E}x_{k-1} + {}^{E}_{S}R_{k-1}{}^{S}g_{yaw,k-1}\Delta t + w_{k-1} \tag{32}$$

Initialization:

$$^{E}\mu_0 = E[^{E}x_0] \tag{33}$$

$$P_0 = E[(^{E}x_0 - {}^{E}\mu_0)(^{E}x_0 - {}^{E}\mu_0)^T] \tag{34}$$

Model Predictor:

$$^{E}x_k^f \approx {}^{E}x_{k-1}^a \tag{35}$$

$$P_k^f = P_{k-1} + Q_{k-1} \tag{36}$$

Data assimilation Corrector:

$$^{E}x_k^a \approx {}^{E}x_k^f + K_k({}^{E}z_k - {}^{E}x_k^f) \tag{37}$$

$$P_k = (I - K_k)P_k^f \tag{38}$$

$$K_k = P_k^f(P_k^f + R_k)^{-1} \tag{39}$$

REFERENCES

[1] Longer video of 'ariane 5' rocket first launch failure/explosion. http://www.youtube.com, September 2010. [Online: accessed Jan-2018].

[2] Drone dangers, risks and injuries. https://www.youtube.com/watch?v=ch1SnP41tx0, November 2015. [Online: accessed Jan-2018].

[3] 3dr site scan. https://3dr.com, 2017. [Online; accessed Jan-2018].

[4] Aeromao. http://www.aeromao.com, January 2018. [Online: accessed Jan-2018].

[5] ARDUPILOT. http://ardupilot.org/, 2018. [Online; accessed Jan-2018].

[6] Ardupilot github repository. https://github.com/ArduPilot/ardupilot, January 2018. [Online: accessed Jan-2018].

[7] Box plot. https://en.wikipedia.org/wiki/Box_plot/, 2018. [Online; accessed Jan-2018].

[8] Effect size. https://en.wikipedia.org/wiki/Effect_size, 2018. [Online; accessed Jan-2018].

[9] Outlier. https://en.wikipedia.org/wiki/Outlier, January 2018. [Online; accessed 27-Jan-2018].

[10] p-value. https://en.wikipedia.org/wiki/P-value, January 2018. [Online; accessed 28-jan-2018].

[11] Pid controller wiki. https://en.wikipedia.org/wiki/PIDcontroller, January 2018. [Online; accessed Jan-2018].

[12] Quartile. https://en.wikipedia.org/wiki/Quartile, January 2018. [Online; accessed Jan-2018].

[13] Therac-25. https://en.wikipedia.org/wiki/Therac-25, January 2018. [Online; accessed Jan-2018].

[14] Unity - game engine. https://unity3d.com, 2018. [Online; accessed Jan-2018].

[15] Sheeva Afshan, Phil McMinn, and Mark Stevenson. Evolving readable string test inputs using a natural language model to reduce human oracle cost. *Proc. of the 6th IEEE Intl. Conf. on Software Testing, Verification and Validation*, pages 352–361, 2013.

[16] Muhammad Haris Afzal, Valérie Renaudin, and Gérard Lachapelle. Assessment of indoor magnetic field anomalies using multiple magnetometers. In *ION GNSS*, volume 10, pages 21–24, 2010.

[17] Muhammad Haris Afzal, Valérie Renaudin, and Gérard Lachapelle. Use of earths magnetic field for mitigating gyroscope errors regardless of magnetic perturbation. *Sensors*, 11(12):11390–11414, 2011.

[18] Cesare Alippi, Stavros Ntalampiras, and Manuel Roveri. Model-free fault detection and isolation in large-scale cyber-physical systems. *IEEE Trans. on Emerging Topics in Computational Intelligence*, 1(1):61-71, February 2017.

[19] M. Anitha, R. Sanjai, W. Michael, and H. Mats PE. Design considerations for modeling modes in cyber–physical systems. *IEEE Design & Test*, 32(5):66–73, 2015.

[20] James Ayre. 16,000 fiat 500e electric cars recalled for software issue. *Clean Technica*, June 6 2016.

[21] Tien-Duy B Le, David Lo, Claire Le Goues, and Lars Grunske. A learning-to-rank based fault localization approach using likely invariants. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 177–188. ACM, 2016.

[22] Ayca Balkan, Paulo Tabuada, Jyotirmoy V. Deshmukh, Xiaqing Jin, and James Kapinski. Underminer: a framework for automatically identifying non-converging behaviors in black box system models. *Proc. of EMSOFT'16*, October 2016.

[23] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: a survey. *IEEE TSE*, 41(5):507-525, May 2015.

[24] Pedro Batista, Carlos Silvestre, Paulo Oliveira, and Bruno Cardeira. Low-cost attitude and heading reference system: Filter design and experimental evaluation. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2624–2629. IEEE, 2010.

[25] Selcuk Bayraktar, Georgios E. Fainekos, and George J. Pappas. Hybrid modeling and experimental cooperative control of multiple unmanned aerial vehicles. *Tech. Rprt. MS-CIS-04-32, Dept. of CIS, U. of Penn.*, December 2004.

[26] Darko Bozhinoski, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Massimo Tivoli. FLYAQ: enabling non-expert users to specify and generate missions of autonomous multicopters. *Proc. of the 30th IEEE/ACM Intl. Conf. on Automated Software Engineering*, 2015.

[27] Brainybit. Build an electronic compass using the hmc5883l module. `https://brainy-bits.com/tutorials/find-your-way-using-the-hmc5883l`, 2016.

[28] Lionel C Briand, Yvan Labiche, and Xuetao Liu. Using machine learning to support debugging with tarantula. In *Software Reliability, 2007. ISSRE'07. The 18th IEEE International Symposium on*, pages 137–146. IEEE, 2007.

[29] William L. Brogan. *Modern control theory (3rd ed.)*. Pearson, 1990.

[30] Yuriy Brun and Michael D Ernst. Finding latent code errors via machine learning over program executions. In *Proceedings of the 26th International Conference on Software Engineering*, pages 480–490. IEEE Computer Society, 2004.

[31] Pasquale Buonocunto and Mauro Marinoni. Tracking limbs motion using a wireless network of inertial measurement units. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pages 66–76. IEEE, 2014.

[32] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[33] Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. Formal concept analysis enhances fault localization in software. In *International Conference on Formal Concept Analysis*, pages 273–288. Springer, 2008.

[34] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing: a new approach for generating next test cases. *The HKUST CS Tech. Report HKUST-CS98-01*, 1998.

[35] T. Y. Chen, F.-C Kuo, and W. K. Tam. Testing a software-based pid controller using metamorphic testing. *Proc. of the 1st Intl. Conf. on Pervasive and Embedded Computing and Communication Systems*, pages 387–396, 2011.

[36] Tsong Yueh Chen, T. H. Tse, and Zhi Quan Zhou. Semi-proving: an integrated method for program proving, testing, and debugging. *IEEE TSE*, 37(1):109-125, 2011.

[37] Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N. Padmanabhan. Indoor localization without the pain. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking (MobiCom)*, pages 173–184. ACM, 2010.

[38] Jaewoo Chung, Matt Donahoe, Chris Schmandt, Ig-Jae Kim, Pedram Razavai, and Micaela Wiseman. Indoor location sensing using geo-magnetism. In *Proceedings of*

the 9th international conference on Mobile systems, applications, and services (MobiSys), pages 141–154. ACM, 2011.

[39] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 2013.

[40] Ionut Constandache, Romit Roy Choudhury, and Injong Rhee. Towards mobile phone localization without war-driving. In *Infocom, 2010 proceedings ieee*, pages 1–9. IEEE, 2010.

[41] Robert M Corless, Patrizia M Gianni, Barry M Trager, and Stephen M Watt. The singular value decomposition for polynomial systems. In *Proceedings of the 1995 international symposium on Symbolic and algebraic computation*, pages 195–207. ACM, 1995.

[42] Gustavo A. Peláez Coronado, Fernando García, Arturo de la Escalera, and José María Armingol. Driver monitoring based on low-cost 3-d sensors. *IEEE Trans. Intelligent Transportation Systems*, 15(4):1855–1860, 2014.

[43] Patrick Cousot, Radhia Cousot, Jorome Feret, Laurent Mauborgne, Antoine Mine, David Monniaux, and Xavier Rival. The astree analyzer. *Proc. of ESOP*, 2005.

[44] Christoph Csallner, Nikolai Tillmann, and Yannis Smaragdakis. DySy: dynamic symbolic execution for invariant inference. *Proc. of ICSE*, pages 281–290, 2008.

[45] Valentin Dallmeier, Christian Lindig, Andrzej Wasylkowski, and Andreas Zeller. Mining object behavior with adabu. pages 17–24, 2006.

[46] Ewen Denney, Ganesh Pai, and Iain Whiteside. Model-driven development of safety architectures. *Proc. of ACM/IEEE 20th Intl. Conf. on Model Driven Engineering Languages and Systems (MODELS)*, pages 156–166, 2017.

[47] Jan Dentler, Somasundar Kannan, Miguel Angel Olivares Mendez, and Holger Voos. A tracking error control approach for model predictive position control of a quadrotor

with time varying reference. *Proc. of the IEEE Intl. Conf. on Robotics and Biomimetics*, Dec. 3-7 2016.

[48] Estefania Munoz Diaz and Ana Luz Mendiguchia Gonzalez. Step detector and step length estimator for an inertial pocket navigation system. In *Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on*, pages 105–110. IEEE, 2014.

[49] Estefania Munoz Diaz, Ana Luz Mendiguchia Gonzalez, and Fabian de Ponte Muller. Standalone inertial pocket navigation system. In *Position, Location and Navigation Symposium-PLANS 2014, 2014 IEEE/ION*, pages 241–251. IEEE, 2014.

[50] Roong-Ko Doong and Phyllis G. Frankl. The ASTOOT approach to testing object-oriented programs. *ACM TSEM*, 3(2):101-130, April 1994.

[51] Parasara Sridhar Duggirala, Le Wang, Sayan Mitra, Mahesh Viswanathan, and Cesar Munoz. Temporal precedence checking for switched models and its application to a parallel landing protocol. *Proc. of Formal Methods (FM)*, LNCS 8442:215-229, May 2014.

[52] D.A Eberly. Least squares fitting of data. In *Magic Software, Inc, Chapel Hill, N.C.*, 2001.

[53] U. Eren, A. Prach, B.B. Kocer, S.V. Rakovic, E. Kayacan, and B. Acikmese. Model predictive control in aerospace systems: Current state and opportunities. *AIAA Journal of Guidance, Control, and Dynamics*, 2017.

[54] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE TSE*, 27(2):99-123, February 2001.

[55] Michael D Ernst, Jake Cockrell, William G Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, 2001.

[56] Raúl Feliz Alonso, Eduardo Zalama Casanova, and Jaime Gómez García-Bermejo. Pedestrian tracking using inertial sensors. 2009.

[57] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback control of dynamic systems (7th ed.)*. Pearson, 2014.

[58] Gordon Fraser and Andrea Arcuri. Evosuite: Automatic test suite generation for object-oriented software. *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSC'11)*, pages 416–419, 2011.

[59] Gordon Fraser and Andreas Zeller. Mutation-driven generation of unit tests and oracles. *IEEE Trans. on Software Engineering*, 38(2):278-292, Mar/Apr 2012.

[60] Terejanu GA. Extended kalman filter tutorial. `http://usersicesutexasedu/~terejanu/files/tutorialEKFpdf`, 2008.

[61] Gregory Gay, Matt Staats, Michael Whalen, and Mats P.E. Heimdahl. Automated oracle data selection support. *IEEE Trans. on Software Engineering*, 41(11), November 2015.

[62] Demoz Gebre-Egziabher, GH Elkaim, J David Powell, and BW Parkinson. A non-linear, two-step estimation algorithm for calibrating solid-state strapdown magnetometers. In *8th International St. Petersburg Conference on Navigation Systems (IEEE/AIAA)*, 2001.

[63] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. *Hybrid dynamical systems: modeling, stability, and robustness*. Princeton University Press, 2012.

[64] H Guo, M Uradzinski, H Yin, and M Yu. Indoor positioning based on foot-mounted imu. *BULLETIN OF THE POLISH ACADEMY OF SCIENCES*, 2015.

[65] Zhijian He, Yanming Chen, Zhaoyan Shen, Enyan Huang, Shuai Li, Zili Shao, and Qixin Wang. Ard-mu-copter: A simple open source quadcopter platform. In *Mobile*

Ad-hoc and Sensor Networks (MSN), 2015 11th International Conference on, pages 158–164. IEEE, 2015.

[66] Zhijian He, Yanming Chen, Zhaoyan Shen, Enyan Huang, Shuai Li, Zili Shao, and Qixin Wang. Ard-mu-copter: A simple open source quadcopter platform. In *11th International Conference on Mobile Ad-hoc and Sensor Networks, MSN 2015, Shenzhen, China, December 16-18, 2015*, pages 158–164, 2015.

[67] M. H. Heule and S. Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 13(4):825-856, August 2013.

[68] Jerry Hirsch and Ken Bensinger. Toyota settles acceleration lawsuit after $3-million verdict. *Los Angeles Times*, Oct 25 2013.

[69] Honeywell. 3-axis digital compass ic hmc5883l. `https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf`, 2013.

[70] Shin Hong, Byeongcheol Lee, Taehoon Kwak, Yiru Jeon, Bongsuk Ko, Yunho Kim, and Moonzoo Kim. Mutation-based fault localization for real-world multilingual programs (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 464–475. IEEE, 2015.

[71] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge Univ. Press, 2012.

[72] William E. Howden. Introduction to the theory of testing. *Edward Miller and William E. Howden, editors, Tutorial: Software Testing and Validation Techniques*, pages 16–19, 1978.

[73] InvenSense. Mpu-6000 and mpu-6050 product specification revision 3.4. `https://www.cdiweb.com/datasheets/invensense/MPU-6050_DataSheet_V3`, 2013.

[74] A. Jain, E. Biyik, and A. Chakrabortty. A model predictive control design for selective modal damping in power systems. *Proc. of American Control Conference*, 2015.

[75] Abhishek Jain, Emrah Biyik, and Aranya Chakrabortty. A model predictive control design for selective modal damping in power systems. In *American Control Conference (ACC), 2015*, pages 4314–4319. IEEE, 2015.

[76] Antonio Ramón Jiménez, Fernando Seco Granja, José Carlos Prieto, and Jorge I. Guevara Rosas. Indoor pedestrian navigation using an INS/EKF framework for yaw drift reduction and a foot-mounted IMU. In *7th Workshop on Positioning Navigation and Communication, WPNC 2010, Dresden Germany, 11-12 March 2010, Proceedings*, pages 135–143, 2010.

[77] James A Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering*, pages 273–282. ACM, 2005.

[78] Simon J. Julier and Joseph J. LaViola. On kalman filtering with nonlinear equality constraints. *IEEE Trans. Signal Processing*, 55:2774–2784, 2007.

[79] Daisuke Kamisaka, Shigeki Muramatsu, Takeshi Iwamoto, and Hiroyuki Yokoyama. Design and implementation of pedestrian dead reckoning system on a mobile phone. *IEICE transactions on information and systems*, 94(6):1137–1146, 2011.

[80] Aaron Kane, Thomas Fuhrman, and Philip Koopman. Monitor based oracles for cyber-physical system testing. *Proc. of the 44th Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 148–155, 2014.

[81] Somasundar Kannan, Seyed Amin Sajadi Alamdari, Jan Dentler, Miguel A. Olivares-Mendez, and Holger Voos. Model predictive control for cooperative control of space robots. *AIP Conf. Proc.*, 1798(1), January 2017.

[82] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp)*, pages 421–430. ACM, 2012.

[83] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on software Engineering*, 32(3):176–192, 2006.

[84] J. L. Lions. *Ariane 5 flight 501 failure: report by the inquiry board.* Paris, July 19 1996.

[85] Bing Liu, Lucia, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Simulink fault localization: an iterative statistical debugging approach. *Software Testing, Verification and Reliability*, 26:431-459, May 11 2016.

[86] Pablo Loyola, Matt Staats, In-Young Ko, and Gregg Rothermel. Dodona: automated oracle data set selection. *Proc. of ISSTA'14*, pages 193–203, 2014.

[87] L. Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi. Extended comprehensive study of association measures for fault localization. *Journal of Software: Evolution and Process*, 26(2):172–219, 2014.

[88] James R Lyle. Automatic program bug location by program slicing. In *The Second International Conference on Computers and Applications*, pages 877–883, 1987.

[89] Michael R. Lyu. *Software fault tolerance*. Wiley, 1995.

[90] Sebastian OH Madgwick, Andrew JL Harrison, and Ravi Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. In *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–7. IEEE, 2011.

[91] Robert E. Mahony, Tarek Hamel, and Jean Michel Pflimlin. Nonlinear complementary filters on the special orthogonal group. *IEEE Trans. Automat. Contr.*, 53:1203–1218, 2008.

[92] Haroon Malik, Hadi Hemmati, and Ahmed E. Hassan. Automatic detection of performance deviations in the load testing of large scale systems. *Proc. of ICSE'13*, pages 1012–1021, 2013.

[93] Renato Mancuso, Or D Dantsker, Marco Caccamo, and Michael S Selig. A low-power architecture for high frequency sensor acquisition in many-dof uavs. In *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, pages 103–114. IEEE, 2014.

[94] Joao Luis Marins, Xiaoping Yun, Eric R. Bachmann, Robert B. McGhee, and Michael Zyda. An extended kalman filter for quaternion-based orientation estimation using MARG sensors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2001: Expanding the Societal Role of Robotics in the the Next Millennium, Maui, HI, USA, October 29 - November 3, 2001*, pages 2003–2011, 2001.

[95] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Automated test suite generation for time-continuous simulink models. *Proc. of ICSE'16*, pages 595–606, May 14-22 2016.

[96] Phil McMinn, Mark Stevenson, and Mark Harman. Reducing qualitative human oracle costs associated with automatically generated test data. *ACM ISSTA'10*, July 12-16 2010.

[97] Ramy Medhat, S. Ramesh, Borzoo Bonakdarpour, and Sebastian Fischmeister. A framework for mining hybrid automata from input/output traces. *Proc. of the 12th Intl. Conf. on Embedded Software (EMSOFT'15)*, pages 177–186, 2015.

[98] Maik Merten, Falk Howar, Bernhard Steffen, Patrizio Pellicione, and Massimo Tivoli. Automated inference of models for black box systems based on interface descriptions. *Proc. of the 5th Intl. Conf. Leveraging Appl. of Formal Methods, Verification and Validation: Tech. for Mastering Change*, LNCS 7609:79-96, 2012.

[99] Bertrand Meyer. Eiffel: a language and environment for software engineering. *J. of Systems and Software*, 8(3):199-246, June 1988.

[100] Bojan Milosevic, Roberto Naldi, Elisabetta Farella, Luca Benini, and Lorenzo Marconi. Design and validation of an attitude and heading reference system for an aerial

robot prototype. In *American Control Conference, ACC 2012, Montreal, QC, Canada, June 27-29, 2012*, pages 1720–1725, 2012.

[101] Roberto Natella, Domenico Cotroneo, Joao A Duraes, and Henrique S Madeira. On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering*, 39(1):80–96, 2013.

[102] ThanhVu Nguyen, Deepak Kapur, Westley Weimer, and Stephani Forrest. Using dynamic analysis to discover polynomial and array invariants. *Proc. of ICSE*, pages 683–693, 2012.

[103] ThanhVu Nguyen, Deepak Kapur, Westley Weimer, and Stephani Forrest. Using dynamic analysis to generate disjunctive invariants. *Proc. of ICSE*, pages 608–619, 2014.

[104] Hyduke Noshadi, Foad Dabiri, Shaun Ahmadian, Navid Amini, and Majid Sarrafzadeh. Hermes: mobile system for instability analysis and balance assessment. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(1s):57, 2013.

[105] Anthony Odin. Arduino tutorial: Hmc5883l compass magnetometer i2c. `https://www.youtube.com/watch?v=VVlwIRTiHTQ/`, 2016.

[106] Fabrizio Pastore, Leonardo Mariani, and Gordon Fraser. CrowdOracle: can the crowd solve the oracle problem? *Proc. of Intl. Conf. on Software Testing, Verification, Validation*, pages 342–351, 2013.

[107] Dennis K. Peters and David Lorge Parmas. Using test oracles generated from program documentation. *IEEE Trans. on Software Engineering*, 24(3):161-173, March 1998.

[108] Kevin Poulsen. Software bug contributed to blackout. `Securityfocus.com`, Feb 11 2004. [Online; accessed Jan-2018].

[109] Kevin Poulsen. Tracking the blackout bug. `Securityfocus.com`, April 7 2004. [Online; accessed Jan-2018].

[110] Anshul Rai, Krishna Kant Chintalapudi, Venkata N Padmanabhan, and Rijurekha Sen. Zee: Zero-effort crowdsourcing for indoor localization. In *Proceedings of the 18th annual international conference on Mobile computing and networking (MobiCom)*, pages 293–304. ACM, 2012.

[111] Valérie Renaudin, Bertrand Merminod, and Michel Kasser. Optimal data fusion for pedestrian navigation based on uwb and mems. In *Position, Location and Navigation Symposium, 2008 IEEE/ION*, pages 341–349. IEEE, 2008.

[112] Patrick Robertson, Michael Angermann, and Bernhard Krach. Simultaneous localization and mapping for pedestrians using only foot-mounted inertial sensors. In *Proceedings of the 11th international conference on Ubiquitous computing (UbiComp)*, pages 93–96. ACM, 2009.

[113] L. Sebeke, Xi Luo, B. de Jager, W. P. M. H. Heemels, E. Heijman, and H. Gruell. Model predictive control algorithm for large-area regional hyperthermia. *Journal of Therapeutic Ultrasound*, 4:A173, 2016.

[114] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. In *Machine Learning in Cyber Trust*, pages 3–13. Springer, 2009.

[115] Jean-Jacques E. Slotine and Weiping Li. *Applied nonlinear control*. Pearson, 1991.

[116] O. Sokolsky and H. S. Hong. Qualitative modeling of hybrid systems. *Proc. of Workshop on Formal Models in Software Development*, June 2001.

[117] J. M. Spivey. *Z notation - a reference manual (2nd ed.)*. Westview, 2014.

[118] Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.

[119] Arun K Tangirala. *Principles of system identification: Theory and practice*. Crc Press, 2014.

[120] Arun K. Tangirala. *Principles of System Identification: Theory and Practice*. CRC Press, 2015.

[121] JF Vasconcelos, G Elkaim, C Silvestre, P Oliveira, and B Cardeira. A geometric approach to strapdown magnetometer calibration in sensor frame. volume 41, pages 172–177. Elsevier, 2008.

[122] JF Vasconcelos, G Elkaim, C Silvestre, P Oliveira, and B Cardeira. Geometric approach to strapdown magnetometer calibration in sensor frame. volume 47, pages 1293–1306. IEEE, 2011.

[123] Mark Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, pages 439–449. IEEE Press, 1981.

[124] Avishai Weiss, Morgan Baldwin, Richard Scott Erwin, and Ilya Kolmanovsky. Model predictive control for spacecraft rendezvous and docking: strategies for handling constraints and case studies. *IEEE Trans. on Control Systems Tech.*, 23(4):1638-1647, 2015.

[125] Avishai Weiss, Morgan Baldwin, Richard Scott Erwin, and Ilya Kolmanovsky. Model predictive control for spacecraft rendezvous and docking: Strategies for handling constraints and case studies. *IEEE Transactions on Control Systems Technology*, 23(4):1638–1647, 2015.

[126] Elaine J. Weyuker. On testing non-testable programs. *The Computer J.*, 25(4):465-470, 1982.

[127] Wikipedia. North east down. `https://en.wikipedia.org/wiki/North_east_down/`, 2016.

[128] Jeannette M. Wing. A specifier's introduction to formal methods. *IEEE Computer*, 23(9):8-24, September 1990.

[129] Claes Wohlin, Per Runeson, Martin Host, Magnus C. Ohlsson, Bjoorn Regnell, and Anders Wesslen. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.

[130] Eric Wong, Tingting Wei, Yu Qi, and Lei Zhao. A crosstab-based statistical method for effective fault localization. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 42–51. IEEE, 2008.

[131] Eric Wong, Tingting Wei, Yu Qi, and Lei Zhao. A crosstab-based statistical method for effective fault localization. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 42–51. IEEE, 2008.

[132] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. on Software Engineering*, 42(8):707-740, August 2016.

[133] W Eric Wong and Yu Qi. Bp neural network-based effective fault localization. *International Journal of Software Engineering and Knowledge Engineering*, 19(04):573–597, 2009.

[134] W Eric Wong and Yu Qi. Bp neural network-based effective fault localization. *International Journal of Software Engineering and Knowledge Engineering*, 19(04):573–597, 2009.

[135] Zhitian Wu, Yuanxin Wu, Xiaoping Hu, and Meiping Wu. Calibration of three-axis strapdown magnetometers using particle swarm optimization algorithm. In *Robotic and Sensors Environments (ROSE), 2011 IEEE International Symposium on*, pages 160–165. IEEE, 2011.

[136] Tao Xie. Augmenting automatically generated unit-test suits with regression oracle checking. *Proc. of the 20th European Conf. on Object-Oriented Programming (ECOOP 2006)*, pages 380–403, July 2006.

[137] Xiaoyuan Xie, W. Eric Wong, Tsong Yueh Chen, and Baowen Xu. Metamorphic slice: an application in spectrum-based fault localization. *Info. & Software Tech.*, 55:866-879, 2013.

[138] J. Yang, R. Grosu, S. A. Smolka, and A. Tiwari. Love thy neighbor: V-formation as a problem of model predictive control. *Proc. of the 27th Intl. Conf. on Concurrency Theory (CONCUR)*, LIPICS 59:4:1-5, August 2016.

[139] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *J. of Software Testing, Verification & Reliability*, 22(2):67-120, March 2012.

[140] Qiuyue Yu, Lei Cheng, Qinyan Zhang, Yang Chen, Huaiyu Wu, Quanmin Zhu, Yongji Wang, and Nian Liu. Research on magnetic compass calibration for air-ground amphibious robot system. In *Control Conference (CCC), 2016 35th Chinese*, pages 6172–6177. TCCT, 2016.

[141] Xiaoping Yun and Eric R. Bachmann. Design, implementation, and experimental results of a quaternion-based kalman filter for human body motion tracking. *IEEE Trans. Robotics*, 22:1216–1227, 2006.

[142] Francisco Zampella, Antonio Ramón Jiménez, Fernando Seco, José Carlos Prieto, and Jorge Guevara. Simulation of foot-mounted IMU signals for the evaluation of PDR algorithms. In *2011 International Conference on Indoor Positioning and Indoor Navigation, IPIN 2011, Guimaraes, Portugal, September 21-23, 2011*, pages 1–7, 2011.

[143] Andreas Zeller. Isolating cause-effect chains from computer programs. In *Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering*, pages 1–10. ACM, 2002.

[144] Xiangyu Zhang, Neelam Gupta, and Rajiv Gupta. Locating faults through automated predicate switching. In *Proceedings of the 28th international conference on Software engineering*, pages 272–281. ACM, 2006.

[145] Wujie Zheng, Hao Ma, Michael R. Lyu, Tao Xie, and Irwin King. Mining test oracles of web search engines. *Proc. of the 26th IEEE/ACM Intl. Conf. on Automated Software Engineering*, pages 408–411, November 2011.