



THE HONG KONG
POLYTECHNIC UNIVERSITY

香港理工大學

Pao Yue-kong Library

包玉剛圖書館

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

PHYSICAL BEHAVIOR BASED DEBUGGING:
TEST AND FAULT LOCALIZATION ON
CYBER-PHYSICAL SYSTEMS

ENYAN HUANG

M.Phil

The Hong Kong Polytechnic University

2017

The Hong Kong Polytechnic University

Department of Computing

Physical Behavior Based Debugging: Test and Fault
Localization on Cyber-Physical Systems

Enyan Huang

A thesis submitted in partial fulfilment of the requirements
for the degree of Master of Philosophy

January 2017

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.

(Signed)

Enyan Huang

(Name of student)

Abstract

The rapid growth of control cyber-physical systems (control-CPS) complexity necessitates the use of fault localization tools to locate cyber subsystems' bugs. Many mainstream fault localization tools need large number of labeled ("correct" or "faulty") execution traces as their inputs. To prepare the traces, for most control-CPSs (particularly those large or without complete/accurate models), assertions designed by domain experts are used. This approach is best-effort: often heavily subjective and ad-hoc. To make the approach more principled and systematic, we exploit the state-of-the-art hybrid systems modeling and stability theories to propose a new approach. Empirical evaluations upon commercial-product-grade large control-CPS platform show that our proposed approach achieves significant improvements (42% ~ 300% improvement in accuracy, recall, and latency medians) over the existing approach.

Acknowledgement

First of all, I would like to express my deepest gratitude to my supervisor, Dr. Qixin Wang for his patient guidance throughout my entire study. He teaches me how to do academic research from the very beginning when I have no experience. He helps me shape serious research attitude and thorough methodology, which are extremely important when doing research. He provides me with the essential guidance and knowledge which is the most relevant to my research topic. I may not complete my study without his supervision.

Next, I would like to thank many of my colleagues who help me a lot during my study. They are Dr. Yao Chen, Dr. Feng Tan, Mr. Zhijian He, Mr. Yanming Chen and Mr. Zhaoyan Shen. They share their valuable experiences with me which save me much time and effort.

I would like to thank Prof. Lui Sha from University of Illinois Urbana-Champaign and Dr. Shan Lu from The University of Chicago who give me advice for my work.

I would also like to thank Dr. Yuhong Wang and Ms. Ling Chen from Department of Civil and Environmental Engineering. They give me precious opportunities to work with real project, from which I learnt a lot.

Last I would like to thank all my friends in The Hong Kong Polytechnic University. I have had a happy time here with all of you and I am lucky to live with you for the six years of study in Hong Kong.

Contents

1	Introduction	1
1.1	Demand	1
1.2	Control Systems	2
1.3	Hybrid Automata	3
1.4	Unmanned Aerial Vehicles (UAVs)	5
1.4.1	System Dynamics of Quadcopters	5
1.5	Contributions of the Thesis	10
2	Related Work	13
2.1	Related Work in the Domain of Control	13
2.2	Related Work in the Domain of Software Engineering	15
2.2.1	Tarantula	15
2.2.2	Crosstab	16
2.2.3	BP Neural Network-based (BPNN) Approach	18
2.3	Related Work in Cross Domains	20
3	A Hybrid Systems Based Approach to Prepare Traces for Control-CPS	
	Cyber Subsystem Fault Localization	21
3.1	Overview	21
3.2	Formal Definition of Hybrid Automata	21
3.3	Physical Traces Divergence Bound Existence	23
3.4	Proposed Approach	30
3.4.1	Summary and Intuition of the Theory	30
3.4.2	Heuristics and Specifications of Proposed Approach	32
3.4.3	Rigorous Definitions of Step 4 and 9	35

4	Evaluation and Results	37
4.1	Overview	37
4.2	A Simple Example: Thermostat Control-CPS	37
4.2.1	System Model of Thermostat	37
4.2.2	Thermostat Physical Traces Divergence Bound	39
4.3	Evaluation on Ardupilot Autopilot Software	43
4.3.1	ArduPilot	44
4.3.2	Bug Injection	45
4.3.3	Fault Localization Tools	47
4.3.4	Trials and Diagnosis	48
4.3.5	Results	50
4.3.6	Discussion on Failure of BPNN Fault Localization Method . . .	55
4.3.7	Threats to Validity	56
5	Conclusions and Suggestions for Future Research	59
5.1	Conclusion	59
5.2	Suggestions for Future Research	60
	Bibliography	63

List of Figures

1.1	Example hybrid automaton: thermostat (quoted from [Che16b]). Note as oven temperature x is the only concerned physical state, the physical state vector is 1-dimensional and is hence simply represented by scalar x .	4
1.2	Inertial frame (in dash) and body frame (in solid), note the two arms of UAV are parallel to x_B, y_B axes	6
1.3	The moments of inertia of the quadcopter (quoted from [Bea08]) . . .	9
2.1	Example of Tarantula Technique (quoted from [JH05])	16
2.2	Sample Coverage Data and Execution Results (quoted from [WQ09]) .	20
3.1	Approach Heuristics. Dark shaded area is \mathcal{I}	31
4.1	Example Hybrid Automata: Thermostat. Note as oven temperature x is the only concerned physical state, the physical state vector is 1-dimensional and is hence simply represented by scalar x	38
4.2	Periodical Trace Behavior of Thermostat	41
4.3	Box-plots [16e] of diagnoses quality metrics. X-axes: execution traces preparation approach and fault localization tool combination; PA: our proposed approach; W2: Way-2 approach; TA: Tarantula; CR: Crosstab; NN: BP Neural Networks. Y-axes: respective diagnoses quality metric statistics. Bar in each box is the median of the data, box ends are 1st and 3rd quartile of the data, whisker ends are min/max of the data, outliers are dots outside the whiskers.	50
4.4	Medians of accuracy and recall over the length of the report. X-axes: length of the report; PA: our proposed approach; W2: Way-2 approach; TA: Tarantula; CR: Crosstab. Y-axes: respective medians of all subjects.	53

4.5 Fault diagnosis quality evolution over execution trace length. X-axes: length of each execution trace (unit: seconds of emulation time). Y-axes: median of accuracy (left figure), recall (middle figure), and latency (right figure) of all subjects. 54

List of Tables

2.1	Notations used in Crosstab	18
4.1	Most Representative Residual Bug Types [Nat+13]	46
4.2	Bugs Injected into ArduPilot	47
4.3	Fig. 4.3 PA vs W2 Comparisons Trustworthiness	51
4.4	Fig. 4.5 PA vs W2 Comparisons Trustworthiness	54

Introduction

1.1 Demand

With the rapid growth of embedded computing, computer systems are inevitably more tightly coupled with physical systems, resulting in the so-called *cyber-physical systems* (CPS) [Sha+08]. Control-CPS is a major category of CPS, whose physical subsystem is a control system. Typical examples of control-CPS include computer controlled aircrafts, vehicles, robotics, smart buildings etc. [EG14][Mur+15][Zhe+15].

Nowadays control-CPS cyber subsystems can be extremely complex. For example, ArduPilot [16c], a commercial-product-grade *unmanned aerial vehicles* (UAV) platform, already has over 1.4 million lines of source code. Entirely relying on manually efforts to *locate bugs* (i.e. *fault localization*) at such scale is grueling.

In fact, pure cyber system fault localization is a hot research area. Many fault localization tools are proposed in the past decades [WD09]. Several major families of fault localization tools need large set of labeled (as “correct” or “faulty”) ¹ execution traces to discover suspicious statements or source code blocks [WD09].

For control-CPSs, an execution trace has two parts: the *execution cyber trace* and the *execution physical trace*. The former is the sequence of software statements executed; and the latter is the corresponding physical subsystem’s state trajectory. To fault

¹Unlike pure cyber systems, where a trace can often be discretely labelled as “pass” or “failing”, for control-CPS, due to the ubiquitous existence of physical Gaussian noise and the use of robust control designs, a faulty output from the cyber subsystem may not always lead to failure. Hence we use the term “correct” or “faulty” trace in this paper.

localization tools, the input can only be the labeled execution cyber traces; but the execution cyber traces' labels are determined by the corresponding execution physical traces' correctness.

In current practice, the above fact leads to two ways to prepare execution traces for control-CPS cyber subsystem fault localization. **Way-1)** If the formal model of the control-CPS is known, we can use the model to generate predicted physical traces, and compare them with execution physical traces, hence to label the execution physical traces and the corresponding execution cyber traces. **Way-2)** However, for most real-world control-CPSs, their complete and accurate models are rarely known: often the models are partial, or too high level. We hence cannot expect correct execution physical traces to fully match model generated predicted physical traces. Thus Way-1 cannot work. Instead, we have to use assertions designed by domain experts to judge execution physical traces' (and hence the corresponding execution cyber traces') correctness. So far, this approach is still mainly best-effort, heavily dependent on the designer's subjective expertise and often problem specific. *We need more principled and systematic approaches.*

1.2 Control Systems

Modern control theory [Bro91][SL91][FPE93] abstracts a physical system with a vector of physical states $\vec{x} \stackrel{\text{def}}{=} (x_1, x_2, \dots, x_N)^T \in \mathbb{R}^N$, where each element x_i (where $i = 1, 2, \dots, N$) is a scalar representing one of the physical states, and “ T ” means matrix transpose. Strictly speaking x_1, x_2, \dots, x_N are all functions of continuous time t , respectively representing the corresponding physical state at time t . Let $\dot{x}_i \stackrel{\text{def}}{=} dx_i/dt$ (where $i = 1, 2, \dots, N$), i.e. the derivative of $x_i(t)$ at time t ; then the continuous time domain dynamics of the physical system is described by a differential equation

$$\dot{\vec{x}} \stackrel{\text{def}}{=} (\dot{x}_1, \dot{x}_2, \dots, \dot{x}_N)^T = f(\vec{x}), \quad (1.1)$$

where f is a function $\mathbb{R}^N \mapsto \mathbb{R}^N$ dependent on the physical system design. Exp. (1.1) means the current rate and direction of change of physical state vector \vec{x} (i.e. $\dot{\vec{x}}$) are determined by the current physical state vector via function f . Note Exp. (1.1) assumes the runtime targeted physical state vector (aka reference point) is determined by \vec{x} . For example, for an autopilot UAV, the pilot decision is made as per the UAV's current physical state vector. A specific model of UAV is given in Section. 1.4, which is widely used by many autopilot softwares, including Ardupilot, which is our main research target.

1.3 Hybrid Automata

A control-CPS consists of a physical subsystem of control and a cyber subsystem of discrete computing. The classic model for control is the differential equation based model of Exp. (1.1); and the classic model for discrete computing is automaton [HU79]. By combining the two models, hybrid automaton [AHH93][AHH96] naturally arises as a generic modeling tool for control-CPS.

A hybrid automaton is a generalization of an automaton. An automaton can be described by a graph, with nodes and edges respectively representing discrete states and events of a cyber system. In a hybrid automaton, the nodes are expanded. Each node is now associated with a continuous time domain differential equation of Exp. (1.1), which describes the dynamics of a physical subsystem. In this way, hybrid automata can describe both the discrete cyber subsystem and the continuous physical subsystem of a control-CPS.

Focusing only on the components relevant to this thesis, a simplified formal representation of a hybrid automaton A is a tuple of $(\mathcal{L}, \mathcal{E}, \vec{x}, \mathcal{I}, \rho, l^0, \text{act}, \text{inv}, \text{grd}, \text{rst})$, where a set \mathcal{L} of nodes, aka *locations*, are connected by a set \mathcal{E} of directional edges, aka *jumps* or *transitions*. A jump can be denoted by a tuple (l, l') , where $l, l' \in \mathcal{L}$ are

respectively the source and destination locations of the jump. $\vec{x} \in \mathbb{R}^N$ is the physical state vector, its set of valid initial values is denoted as \mathcal{I} .

The other components for A (i.e. $\rho, l^0, \text{act}, \text{inv}, \text{grd}, \text{rst}$) respectively define current location, initial location, \vec{x} 's dynamics, constraints, jump conditions, and new values after jumps. The rigorous definitions is presented in Section 3.2. For now, in the following, we use an example (see Fig. 1.1) to illustrate the concepts of the various hybrid automaton components.

Fig. 1.1 gives an example hybrid automaton model of the thermostat control-CPS, which automatically heats an oven to maintain oven temperature x in range $[\theta_l, \theta_h]$ ($0 < \theta_l < \theta_h$). There are two locations in the hybrid automaton: l_{heat} for heating and

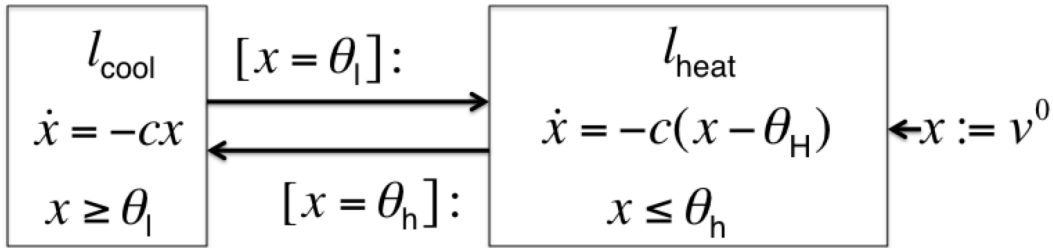


Fig. 1.1: Example hybrid automaton: thermostat (quoted from [Che16b]). Note as oven temperature x is the only concerned physical state, the physical state vector is 1-dimensional and is hence simply represented by scalar x .

l_{cool} for cooling. The initial oven temperature is $v^0 \in \mathcal{I}$ (where the valid initial value set $\mathcal{I} = [\theta_l, \theta_h]$) and the oven is initially set to location $l^0 = l_{\text{heat}}$.

In location l_{heat} , the heater is on, thus sets the outside temperature to $\theta_H > \theta_h$. This causes the oven temperature to rise at a speed proportional to inside-outside temperature difference. Thus the physical state vector dynamics at l_{heat} is $\dot{x} = -c(x - \theta_H)$. Coefficient c is a constant dependent on the type of gas in the oven and the oven wall material.

In location l_{cool} , the heater stops, hence the oven temperature drops at a speed proportional to inside-outside temperature difference (assuming outside temperature is 0). Thus the physical state vector dynamics at l_{cool} is $\dot{x} = -cx$.

Location l_{heat} has a constraint: oven temperature x must not rise over θ_h , i.e. $x \leq \theta_h$. When x reaches θ_h , guard condition $[x = \theta_h]$ is triggered and the jump to l_{cool} happens. As oven temperature x retains its old value (i.e. θ_h) immediately after the jump, the new value assignment action is not explicitly drawn in Fig. 1.1.

In contrast, location l_{cool} also has a constraint: oven temperature must not drop below θ_l , i.e. $x \geq \theta_l$. When x reaches θ_l , guard condition $[x = \theta_l]$ is triggered and the jump to l_{heat} happens. As oven temperature x retains its old value (i.e. θ_l) immediately after the jump, the new value assignment action is not explicitly drawn in Fig. 1.1.

1.4 Unmanned Aerial Vehicles (UAVs)

UAV is an abbreviation for “Unmanned Aerial Vehicle”. It is an aircraft with no pilot on board. A UAV can either fly while controlled by a pilot on ground, or it can fly fully autonomously based on pre-programmed flight plan.

UAVs have broad usage in military, civilian and academic domains. According to the different configurations of actuators and sensors, UAVs can be classified into different types. One of the most common configuration is quadcopter, which uses four vertical motors and propellers to lift the body up. A model of quadcopter will be presented in the next section to show how quadcopter control-CPS works.

1.4.1 System Dynamics of Quadcopters

Before deriving the quadcopter dynamics, we need to introduce two different frames in which we operate. This will simplify the model expressions and the states can be easily converted between two frames through transformation matrix. The “inertial frame” is defined as the common frame we use on the ground, where the x -axis

points to the north, y -axis points to the east, and z -axis points to the opposite direction to the gravity. The origin of inertial frame can be arbitrary fixed point on the earth. For example, most quadcopters use GPS signals to acquire their position information, thus the origin is the point on the earth whose latitude and longitude are both 0° , and altitude is 0m. The “body frame” is defined by the quadcopter itself. The x -axis and y -axis point to two arms of the quadcopter, and z -axis points to the direction of the motors. The origin of body frame and inertial frame should be the same fixed point on the earth, thus any vector expressed in the body frame can be converted to a vector in the inertial frame through only rotation and vice versa. (Fig. 1.2)

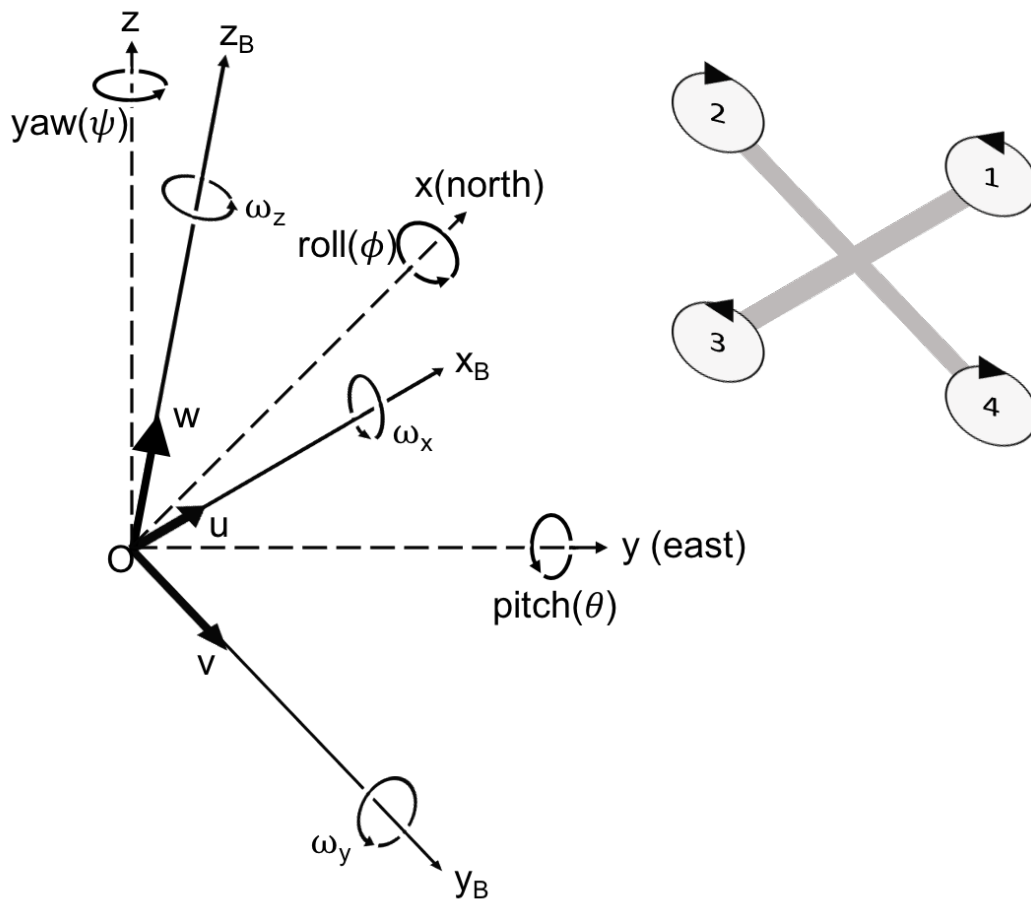


Fig. 1.2: Inertial frame (in dash) and body frame (in solid), note the two arms of UAV are parallel to x_B, y_B axes

To convert states from body frame to inertial frame and vice versa, we need a transformation matrix R such that if \vec{v} is a vector in body frame, $R\vec{v}$ is the corresponding

vector in inertial frame. Reversely, if \vec{v}' is a vector in inertial frame, $R^{-1}\vec{v}'$ is the corresponding vector in body frame. Clearly R is only related to the orientation of the quadcopter, i.e. roll(ϕ), pitch(θ) and yaw(ψ). The matrix can be derived by using Euler angle conversion:

$$R = \begin{bmatrix} c_\phi c_\psi - c_\theta s_\phi s_\psi & -c_\psi s_\phi - c_\phi c_\theta s_\psi & s_\theta s_\psi \\ c_\theta c_\psi s_\phi + c_\phi s_\psi & c_\phi c_\theta c_\psi - s_\phi s_\psi & -c_\psi s_\theta \\ s_\phi s_\theta & c_\phi s_\theta & c_\theta \end{bmatrix} \quad (1.2)$$

where c_ϕ means $\cos(\phi)$ and s_ϕ means $\sin(\phi)$.

The physical states of a quadcopter is usually expressed by a 12-element vector:

$$\vec{x} = \begin{bmatrix} x & y & z & u & v & w & \phi & \theta & \psi & \omega_x & \omega_y & \omega_z \end{bmatrix}^T \quad (1.3)$$

where x, y, z are the UAV's position in inertial frame, u, v, w are the UAV's linear velocity in body frame, ϕ, θ, ψ are the UAV's orientation in inertial frame, $\omega_x, \omega_y, \omega_z$ are the UAV's angular velocity in body frame.

The linear acceleration of quadcopter is mainly due to thrust, gravity and friction. To simplify our model, we will ignore friction in this part, then we have

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + RT_B \quad (1.4)$$

where m is the mass of quadcopter and T_B is the thrust in body frame. Thrust comes from the rotation of motors and it is proportional to the square of the motor speed (denoted by s_i):

$$T_B = k_m \sum_{i=1}^4 \begin{bmatrix} 0 \\ 0 \\ s_i^2 \end{bmatrix} \quad (1.5)$$

The rotation of motors also contributes torques to the quadcopter. For all quadcopters, two of the propellers are spinning clockwise and the other two are spinning counterclockwise (see Fig. 1.2), so the total torque applied (in body frame) on the quadcopter can be expressed as:

$$\tau_B = \begin{bmatrix} k_l(-s_1^2 + s_3^2) \\ k_l(s_2^2 - s_4^2) \\ k_b(-s_1^2 + s_2^2 - s_3^2 + s_4^2) \end{bmatrix} \quad (1.6)$$

where k_l and k_b are some constants.

In body frame, we can compute the angular acceleration from torque by Euler's equation:

$$\dot{\vec{\omega}} = \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = I^{-1}(\tau_B - \vec{\omega} \times (I\vec{\omega})) \quad (1.7)$$

where I is the inertia matrix of the quadcopter. We model the quadcopter as a sphere in the center (the control board and battery) with mass M and radius R , and four

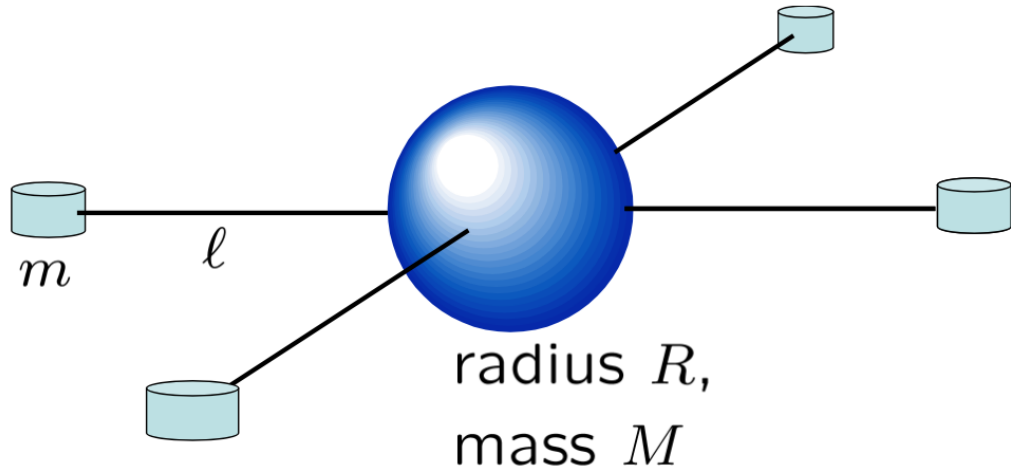


Fig. 1.3: The moments of inertia of the quadcopter (quoted from [Bea08])

point masses (the motors) of mass m at the end of each rod with length l . (Fig. 1.3)
 Therefore, I can be expressed in terms of M , m , R and l :

$$I = \begin{bmatrix} \frac{2MR^2}{5} + 2l^2m & 0 & 0 \\ 0 & \frac{2MR^2}{5} + 2l^2m & 0 \\ 0 & 0 & \frac{2MR^2}{5} + 4l^2m \end{bmatrix} \quad (1.8)$$

From all the equations above, we can finally derive the quadcopter dynamics:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (1.9)$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = R^{-1} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ k_m(s_1^2 + s_2^2 + s_3^2 + s_4^2)/m \end{bmatrix} \quad (1.10)$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = R \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad (1.11)$$

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = I^{-1} \begin{bmatrix} k_l(-s_1^2 + s_2^2 + s_3^2 - s_4^2) \\ k_l(-s_1^2 - s_2^2 + s_3^2 + s_4^2) \\ k_b(-s_1^2 + s_2^2 - s_3^2 + s_4^2) \end{bmatrix} - \vec{\omega} \times (I\vec{\omega}) \quad (1.12)$$

Note the model above is a simplified version, which aims to present the idea of how the quadcopter is controlled by the speed of four motors. In real world and most simulation environment, the dynamics will be much more complicated.

1.5 Contributions of the Thesis

In this thesis, we aim to address the demand mentioned in section 1.1. By exploiting state-of-the-art hybrid systems modeling and stability theories [GST12][Tab09][AHH93], we aim to propose a more principled and systematic approach to prepare control-CPS execution traces for the cyber subsystem fault localization.

1. We discover a mildly generic execution physical traces clustering property for hybrid systems.

2. Based on the above clustering property, we propose a more principled and systematic execution traces preparation approach for control-CPS cyber subsystem fault localization.

3. We evaluate our proposed approach on a commercial-product-grade control-CPS, comparing against the current practice of **Way-2** approach. Results show that our proposed approach achieves significant improvements (42% ~ 300% improvement in accuracy, recall, and latency medians) over the **Way-2** approach (see Fig. 4.3) for two of the three mainstream fault localization tools evaluated.

Related Work

Although control-CPS cyber subsystem fault localization is a long existing problem, few existing solutions systematically exploit and integrate cross-domain knowledge. Most solutions tend to focus on either the control or computer science/engineering perspective.

2.1 Related Work in the Domain of Control

In the domain of control, fault diagnosis in dynamic systems is a research topic which has been studied for decades. We have mature methods to handle sensing/actuating errors, parameter errors, control formulae mismatches [PFE00][Ger98][CW84], and robust/adaptive control demands [LW12][HC10]. There are four major families of the fault diagnosis approaches [Gao+15a][Gao+15b]:

1. *model-based methods*: In model-based methods, we need to know the model of the system, which can be acquired from either physical principles or system identification technique. By comparing the predicted output and the measured output, we can find faults in the system if the outputs are inconsistent. There are a bunch of fault detection methods in this category to locate the faults such as the parity relation approach [Mir79]. Some of the model-based methods can also isolate the faults, such as structure residual fault isolation [Ger88].
2. *signal-based methods*: In signal-based methods, we try to find faults by extracting and analyzing features of the signals. Abnormal patterns in the signals can

be discovered if the system contains faults. Researchers have proposed a number of features which are helpful for fault diagnosis, such as the covariance of the sensing signals [Sam+08]. Some well-known diagnostic algorithms include fast dynamic time warping (fast DTW) and correlated kurtosis (CK) [HD14].

3. *knowledge-based methods*: In knowledge-based methods, we want to know whether the system behavior is consistent over time. Such knowledge can be learned by analyzing the running history of the system through some machine learning technique. An advantage of knowledge-based methods is that they do not require any prior knowledge of the system model or the signal patterns. These methods mainly utilize the tools in other domains such as qualitative trend analysis (QTA) [MRV07] and principal component analysis (PCA) [Z+13].

4. *hybrid and active methods*: The last category contains the methods which combine different techniques together. For example, model-based method and data-driven method (knowledge-based) are integrated together to diagnose chemical reactors in [She+14]. Lastly, a special family of fault diagnose methods are so-called “active”, which diagnose the faults by significantly changing the configuration and behavior of the system. An example of active method can be found in [Che+14].

These methods, however, usually focus on continuous time domain components of the control-CPS, typically the physical subsystem.

2.2 Related Work in the Domain of Software Engineering

In the domain of computer science/engineering, software fault localization is a hot research topic for long time. Major families of tools include program spectrum analysis [Hon+15][Luc+14][DLZ05][JH05][RR03][Har+00][Ern+01], statistical analysis [B L+16][Won+08][Lib+05][Liu+06], machine learning based analysis [WQ09][BLL07][BE04], program slicing [Wei09][LW87][KL88][AH90], data mining based analysis [Cel+08][Li+06], and program state based debugging [Zh02][Zel02][ZGG06][Ors+06]. All of these tool families focus on computer science/engineering perspective, and rely on assertions designed by domain experts to bring in control domain knowledge. The approach to design such assertions are mostly best-effort. The first three families of tools heavily depend on large number of labeled execution traces as inputs, hence are included in the evaluation in Section 4.3. We select one fault localization methods in each of these families, and we give a brief introduction here to these methods.

2.2.1 Tarantula

Given a program to debug, Tarantula[JH05] runs many test cases (by varying inputs) on the program. Each run of test case is labeled as success/failure and the source code execution trace is logged. 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:

$$suspiciousness(e) = \frac{\frac{failed(e)}{totalfailed}}{\frac{passed(e)}{totalpassed} + \frac{failed(e)}{totalfailed}} \quad (2.1)$$

where $failed(e)$ is the number of failed test cases executing e , $totalfailed$ is the number of all failed test cases, $passed(e)$ is the number of passed test cases executing e and $totalpassed$ is the number of all passed test cases.

Tarantula will sort the coverage entities according to the suspiciousness value, so the programmer can look for the fault from the one with highest suspicious value. An example is given in Figure 2.1 to illustrate how Tarantula works.

	Test Cases						suspiciousness	rank	
	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3			
mid() { int x,y,z,m;									
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		

Fig. 2.1: Example of Tarantula Technique (quoted from [JH05])

2.2.2 Crosstab

Similar to Tarantula, the Crosstab (a.k.a. cross-classification table) method [Won+08] also assigns suspiciousness value to each coverage entity but with different calculation approach. Crosstab analyzes the relationship between the execution of a coverage entity and the successfulness of test cases using Chi-square test. Let the null hypothesis be

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

The Chi-square statistic is

$$\begin{aligned}\chi^2(e) = & \frac{(N_{CF}(e) - E_{CF}(e))^2}{E_{CF}(e)} \\ & + \frac{(N_{CS}(e) - E_{CS}(e))^2}{E_{CS}(e)} \\ & + \frac{(N_{UF}(e) - E_{UF}(e))^2}{E_{UF}(e)} \\ & + \frac{(N_{US}(e) - E_{US}(e))^2}{E_{US}(e)}\end{aligned}\quad (2.2)$$

where

$$E_{CF}(e) = \frac{N_C(e) \times N_F}{N} \quad (2.3)$$

$$E_{CS}(e) = \frac{N_C(e) \times N_S}{N} \quad (2.4)$$

$$E_{UF}(e) = \frac{N_U(e) \times N_F}{N} \quad (2.5)$$

$$E_{US}(e) = \frac{N_U(e) \times N_S}{N} \quad (2.6)$$

$$(2.7)$$

The N -related notions are defined in Table 2.1.

If $\chi^2(e)$ is large, it indicates that the coverage of e have some impact on the successfulness of test cases. We also need to know whether such impact is positive

Tab. 2.1: Notations used in Crosstab

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

(executing e leads to passed test cases) or negative (executing e leads to failed test cases), by introducing a new statistic $\varphi(e)$:

$$\varphi(e) = \frac{N_{CF}(e)/N_F}{N_{CS}(e)/N_S} \quad (2.8)$$

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

$$\zeta(e) = \begin{cases} \frac{\chi^2(e)}{N_F} & \text{if } \varphi(e) > 1 \\ 0 & \text{if } \varphi(e) = 1 \\ -\frac{\chi^2(e)}{N_S} & \text{if } \varphi(e) < 1 \end{cases} \quad (2.9)$$

2.2.3 BP Neural Network-based (BPNN) Approach

This fault localization method[WQ09] is based on a back-propagation (BP) neural network. Intuitively, this method uses the result of test cases as the training data, and localizes the faulty entities by feeding “virtual test cases” into the trained network.

Let m be the number of coverage entities and n be the number of test cases. BPNN first label all the coverage entities as e_1, e_2 and so on. For each test case, BPNN constructs an $m \times 1$ vector c . The value of each entry in the vector c_i is either 0 (if e_i is not covered in the test case) or 1 (if e_i is covered in the test case). Totally n vectors are constructed, one for each test case. These n vectors are the input to the BP neural network.

As for the structure of the network, the author uses a three-layer network with 3 neurons in the hidden layer and 1 neuron in the output layer. The transfer function of each neuron is the standard sigmoid function $y = 1/(1 + e^{-x})$.

The network is trained by all n test cases. For each test case, the input is the constructed vector c as mentioned above, and the output is 0 if the test case is passed or 1 if the test case is failed. BPNN uses the BP algorithm to train the network: the constructed inputs are firstly fed in the network to generate the output, and the error between the actual output and the expected output is calculated and propagated back. During the back-propagation process, the weights on the connections of neurons are adjusted to reduce the error. Bayesian regularization[BW09] is used to mitigate the overfitting problem during the training process. (Overfitting is a common problem in machine learning caused by overly emphasizing matching training data. It harms the predictive power of learnt model.) Figure 2.2 illustrates how the sample inputs and outputs are constructed.

After the network is trained. A set of m virtual test cases are constructed. Each test case in the set only covers one coverage entity so the representing vector e contains exactly one 1 and all remaining entries are 0s. By feeding the vectors to the trained network, we can get m outputs. This approach then uses the outputs as the suspiciousness values of each coverage entity.

	Statement coverage									Execution results
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	r_i
t_1	1	1	1	1	0	1	0	0	1	0
t_2	1	0	0	0	1	1	1	1	0	0
t_3	0	0	0	0	0	1	1	0	0	0
t_4	1	1	0	0	1	0	1	1	1	0
t_5	1	1	1	0	1	1	1	1	1	0
t_6	0	0	1	0	0	1	1	1	0	1
t_7	1	1	1	1	0	1	0	1	1	1

Annotations:

- "1" implies s_6 is covered by t_7 (points to s_6 in t_7)
- "0" implies s_5 is not covered by t_6 (points to s_5 in t_6)
- Five successful tests: t_1, t_2, t_3, t_4 and t_5 (points to r_i column for t_1 to t_5)
- Two failed tests: t_6 and t_7 (points to r_i column for t_6 and t_7)

Fig. 2.2: Sample Coverage Data and Execution Results (quoted from [WQ09])

2.3 Related Work in Cross Domains

Due to the recent advancement of hybrid systems theory, some cross-domain solutions start to emerge. But these solutions typically require accurate known model of the control-CPS [DPS97], or the behavior of program can be modeled with precise logic/automata [WSM02][MSW00]. Also, these solutions typically model faults as a mode or threshold in the automata [Zha+05][NB07][MB99], which is not generic or practical enough to describe all program bugs.

There are works on metamorphic testing [ZXC16], which aims to discover generic output trace features to label "faulty" traces. In this sense, this paper is a step toward metamorphic testing in control-CPS using the state-of-the-art hybrid systems domain knowledge.

There are also works on hybrid automata trace convergence and stability [GST12], however, they do not discuss how to exploit these features for debugging purposes.

A Hybrid Systems Based Approach to Prepare Traces for Control-CPS Cyber Subsystem Fault Localization

3.1 Overview

From the previous discussion, we see the urgent demand on control-CPS debugging tools, but little work has been done in this area. Under this circumstance, we propose a new approach to locate faults in the cyber subsystems, based on our discovery of the existence of physical trace divergence bound. In this chapter, we will first introduce our theorem with complete proof. We will next propose our fault localization approach based on the theorem.

3.2 Formal Definition of Hybrid Automata

Before formally introducing our approach, we need to dig into the hybrid automata a bit further. First, some more details on the state vector \vec{x} . Without loss of generality, in this chapter, we further require that the elements of \vec{x} are ordered s.t. time continuous physical states are listed first. Formally, \vec{x} can be rewritten as

$$\vec{x} = (\vec{y}^T, \vec{z}^T)^T, \quad (3.1)$$

where $\vec{y} = (x_1, x_2, \dots, x_{N_{\text{cont}}})^\top$, $\vec{z} = (x_{N_{\text{cont}}+1}, x_{N_{\text{cont}}+2}, \dots, x_N)^\top$, and $\forall n \in \{1, \dots, N_{\text{cont}}\}$, physical state x_n is always continuous over time. In addition, We denote $\llbracket \vec{x} \rrbracket$ as the valid value range of \vec{x} , hence $\mathcal{I} \subseteq \llbracket \vec{x} \rrbracket$.

Besides \mathcal{L} , \mathcal{E} , \vec{x} and \mathcal{I} mentioned in Section 1.3, other components of hybrid automaton $A(\mathcal{L}, \mathcal{E}, \vec{x}, \mathcal{I}, \rho, l^0, \text{act}, \text{inv}, \text{grd}, \text{rst})$ include the following.

$\rho \in \mathcal{L}$ indicates the current location. ρ is a function of time t . $l^0 \in \mathcal{L}$ is the initial location for ρ .

Function act assigns to each location $l \in \mathcal{L}$ an *activity*, which describes the continuous time domain dynamics of \vec{x} at l . Typically, $\text{act}(l)$ is a differential equation of form Exp. (1.1).

Function inv maps each location $l \in \mathcal{L}$ to a set of assertions that \vec{x} must satisfy when at l . We denote the set of valid values of \vec{x} that satisfy $\text{inv}(l)$ as $\llbracket \text{inv}(l) \rrbracket$, note $\llbracket \text{inv}(l) \rrbracket \subseteq \llbracket \vec{x} \rrbracket$.

Function grd assigns each jump $(l, l') \in \mathcal{E}$ a *guard*, i.e. a condition on \vec{x} to trigger jump (l, l') . We denote the set of valid values of \vec{x} that satisfy $\text{grd}((l, l'))$ as $\llbracket \text{grd}((l, l')) \rrbracket$, note $\llbracket \text{grd}((l, l')) \rrbracket \subseteq \llbracket \vec{x} \rrbracket$. For convenience, we denote $\llbracket \text{grd}(l) \rrbracket \stackrel{\text{def}}{=} \bigcup_{(l, l') \in \mathcal{E}} \llbracket \text{grd}((l, l')) \rrbracket$.

Function rst assigns each jump $(l, l') \in \mathcal{E}$ a set of tuples of (\vec{v}, \vec{v}') , where $\vec{v} \in \text{closure}(\llbracket \text{inv}(l) \rrbracket) \cap \llbracket \text{grd}((l, l')) \rrbracket$ and $\vec{v}' \in \llbracket \text{inv}(l') \rrbracket$, here the ‘‘closure’’ operator gets the closure of a set. $(\vec{v}, \vec{v}') \in \text{rst}((l, l'))$ iff physical state vector \vec{x} 's value can be reset from \vec{v} immediately before the jump to \vec{v}' immediately after the jump. We also say \vec{v} is *mapped* by (l, l') to \vec{v}' . Also, iff jump (l, l') happens, ρ changes from l to l' . Note different from [AHH96], in this chapter we assume \vec{v}' is unique for a given \vec{v} in a given $\text{rst}((l, l'))$. For a given \vec{v} , multiple tuples may exist in $\text{rst}((l, l'))$; e.g., (\vec{v}, \vec{v}') and

(\vec{v}, \vec{v}'') (where $\vec{v}' \neq \vec{v}''$) may coexist in $\text{rst}((l, l'))$. In each occurrence of the jump, \vec{x} 's new value is nondeterministically chosen from all valid candidates.

Function `syn` is for hybrid automata composition, which is irrelevant to this chapter. Interested readers can refer to [AHH96] for further details. `syn` assigns each jump $(l, l') \in \mathcal{E}$ a *synchronization message* (which can be null) to define causal relations between jumps. The details are unimportant for this paper. Usually we add character ‘!’ in front of a synchronization message to indicate once the corresponding jump is triggered, the message is sent; while add character ‘?’ in front of a synchronization message to indicate once this message is received, the corresponding jump is triggered (this means a jump can be triggered by its guard condition and/or by receiving a synchronization message).

3.3 Physical Traces Divergence Bound Existence

(Acknowledgement: The mathematical analysis in this section is the result of a joint effort led by Dr. Yao Chen of our group.)

In this section, we will introduce a theorem which proves the existence of physical traces divergence bound, which is the foundation of our debugging approach. To formally describe the theory, we denote \vec{v}_n ($n \in \{1, \dots, N\}$) as the n th dimension of vector $\vec{v} \in \mathbb{R}^N$. For example, as per Exp. (3.1), \vec{x}_n ($n \in \{1, \dots, N_{\text{cont}}\}$) represents a scalar physical state that is always continuous over time. Denote \mathbb{B}_n ($n \in \mathbb{N}^+$) as the unit ball in \mathbb{R}^n . Given $a \in \mathbb{R}$, $\vec{b} \in \mathbb{R}^n$ ($n \in \mathbb{N}^+$), and set $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathbb{R}^n$, we define $a\mathcal{S}_1 \stackrel{\text{def}}{=} \{\vec{v} \in \mathbb{R}^n : \exists \vec{\mu} \in \mathcal{S}_1 \text{ s.t. } \vec{v} = a\vec{\mu}\}$, define $\vec{b} + \mathcal{S}_1 \stackrel{\text{def}}{=} \{\vec{v} \in \mathbb{R}^n : \exists \vec{\mu} \in \mathcal{S}_1 \text{ s.t. } \vec{v} = \vec{b} + \vec{\mu}\}$, and define $\mathcal{S}_1 + \mathcal{S}_2 \stackrel{\text{def}}{=} \{\vec{v} \in \mathbb{R}^n : \exists \vec{\mu}_1 \in \mathcal{S}_1 \text{ and } \vec{\mu}_2 \in \mathcal{S}_2 \text{ s.t. } \vec{v} = \vec{\mu}_1 + \vec{\mu}_2\}$. We denote $\text{boundary}(\mathcal{S})$ as the boundary of a set \mathcal{S} [16d]. Finally, a \mathcal{K} -function $f(a)$ is an increasing function for $a \geq 0$ and $f(0) = 0$. With the above notations, we propose Theorem 1.

Theorem 1 (Physical Traces Divergence Bound Existence) Given a hybrid automaton $A(\mathcal{L}, \mathcal{E}, \vec{x}, \mathcal{I}, \rho, l^0, act, inv, grd, rst)$ satisfying feature $F1 \sim F4$:

F1 $\forall l \in \mathcal{L}$, activity $act(l)$ is a differential equation $\dot{\vec{x}} = f_l(\vec{x})$, where $f_l(\vec{x})$ is continuous, differentiable, well defined on \mathbb{R}^N , and $\exists 0 < F_1 < F_2$ s.t. $\forall \vec{x} \in \mathbb{R}^N$, $\|f_l(\vec{x})\| \leq F_2$ and $\|\frac{df_l(\vec{x})}{d\vec{x}}\| \leq F_2$, and $\forall \vec{x} \in boundary(\llbracket inv(l) \rrbracket)$, $F_1 \leq \|f_l(\vec{x})\|$.

F2 $\forall (l, l') \in \mathcal{E}$, the jump (l, l') can only happen when \vec{x} penetrates the boundary of $\llbracket inv(l) \rrbracket$, i.e. reaching the boundary non-tangentially and outward closure($\llbracket inv(l) \rrbracket$).

F3 $\forall l \in \mathcal{L}$, the boundary of $\llbracket inv(l) \rrbracket$ is partitioned into one or several maximal connected regions, denote this set of partition(s) as $\mathcal{P}(boundary(\llbracket inv(l) \rrbracket))$.

F3.1 Each partition $P \in \mathcal{P}(boundary(\llbracket inv(l) \rrbracket))$ has a unique feasible jump $(l, l') \in \mathcal{E}$. The jump is feasible at every $\vec{v} \in P$, and maps \vec{v} to a unique new value $\vec{v}' \in \llbracket inv(l') \rrbracket$. Such mappings of all partitions in $\mathcal{P}(boundary(\llbracket inv(l) \rrbracket))$ collectively define a function, denoted as $\Omega_l(\vec{v}) : boundary(\llbracket inv(l) \rrbracket) \mapsto \bigcup_{l' \in \mathcal{L}} \llbracket inv(l') \rrbracket$.

F3.2 $\exists \Delta > 0$, $\forall P_1, P_2 \in \mathcal{P}(boundary(\llbracket inv(l) \rrbracket))$ and $P_1 \neq P_2$, we have $\forall \vec{v}_1 \in P_1$ and $\forall \vec{v}_2 \in P_2$, $\|\vec{v}_1 - \vec{v}_2\| \geq \Delta$.

F3.3 For each partition $P \in \mathcal{P}(boundary(\llbracket inv(l) \rrbracket))$, suppose its unique jump is $(l, l') \in \mathcal{E}$. Suppose $\vec{v} \in P$ is uniquely mapped by (l, l') to $\vec{v}' \in \llbracket inv(l') \rrbracket$. Then $\exists \mathcal{K}$ -function $B'(\cdot)$, s.t. for any sufficiently small $\varepsilon > 0$ (see footnote ¹), $\forall \vec{v} \in (\vec{v} + \varepsilon \mathbb{B}_N) \cap P$, \vec{v} is uniquely mapped by (l, l') to $\vec{v}' \in \vec{v}' + B'(\varepsilon) \mathbb{B}_N$ in $\llbracket inv(l') \rrbracket$.

F4 A is nonzeno and nonblocking.

Then given time interval $[0, T]$, $\exists \mathcal{K}$ -function $B(\cdot)$ s.t. for any physical state vector trajectory $\vec{x}(t)$ ($t \in [0, +\infty)$) starting from initial value $\vec{v}^0 \in \mathcal{I} - boundary(\llbracket inv(l^0) \rrbracket)$, we have the following. For any sufficiently small $\varepsilon > 0$, a trajectory $\vec{x}'(t)$ ($t \in [0, +\infty)$) starting from initial value $\vec{v}'^0 \in \vec{v}^0 + \varepsilon \mathbb{B}_N$ shall have $\|\vec{y}(t) - \vec{y}'(t)\| \leq B(\varepsilon)$ ($\forall t \in (0, T)$), where \vec{y} is the continuous physical states portion of \vec{x} (see Exp. (3.1)). We call $B(\cdot)$ the physical traces divergence bound.

¹This means $\exists \bar{\varepsilon} > 0$, s.t. $\forall \varepsilon \in (0, \bar{\varepsilon})$ the corresponding claim holds.

Proof: For A , due to F1, F2, F3.1, F4, given $\vec{x}(t_0) = \vec{v}^0$ and initial location $\sigma(0)$, then trajectory $\vec{x}(t)$ ($\forall t \in [t_0, +\infty)$) is determined. Denote this trajectory as $\Phi(t, t_0, \vec{v}^0, \sigma(0))$.

In case $\Phi(t, t_0, \vec{v}^0, \sigma(0))$ sequentially visits location $\sigma(0), \sigma(1), \sigma(2), \dots$, we denote the corresponding jump time instances as $\Lambda_1(t_0, \vec{v}^0, \sigma(0)), \Lambda_2(t_0, \vec{v}^0, \sigma(0)), \dots$. Due to F4, $\Lambda_1(t_0, \vec{v}^0, \sigma(0)) < \Lambda_2(t_0, \vec{v}^0, \sigma(0)) < \dots$. For a given trajectory $\vec{x}(t) = \Phi(t, t_0, \vec{v}^0, \sigma(0))$, for convenience, we also refer to the k th ($k = 1, 2, \dots$) jump time instance (if it exists) $\Lambda_k(t_0, \vec{v}^0, \sigma(0))$ as t_k . Note we assume trajectory is right continuous over time, i.e. $\vec{x}(t) = \vec{x}(t^+)$ ($\forall t \in [t_0, +\infty)$). Therefore, jump is considered to happen between t_k^- and t_k . Particularly, there is no jump during t_0 and t_0^+ . Hence $t_1 > t_0$ if there is a t_1 after all.

For $\Phi(t, t_0, \vec{v}^0, \sigma(0))$ ($t \in [t_0, +\infty)$), if its k th ($k = 1, 2, \dots$) jump exists, we also define the *jump-from state* and *jump-to state* as $\Psi_k^-(t_0, \vec{v}^0, \sigma(0))$ and $\Psi_k^+(t_0, \vec{v}^0, \sigma(0))$. Note if $\vec{x}(t) = \Phi(t, t_0, \vec{v}^0, \sigma(0))$, then $\vec{x}(t_k^+) = \vec{x}(t_k) = \Omega_{\sigma(k-1)}(\vec{x}(t_k^-))$, and $\vec{y}(t_k^+) = \vec{y}(t_k) = \vec{y}(t_k^-)$ (see Exp. (3.1)).

Furthermore, given $\approx_0 \subseteq \mathbb{R}$ and $\approx_0^N \subseteq \mathbb{R}^N$, we define

$$\begin{aligned} \Phi(t, \approx_0, \approx_0^N, \sigma(0)) &\stackrel{\text{def}}{=} \{\Phi(t, t_0, \vec{v}^0, \sigma(0)) : t_0 \in \approx_0, \vec{v}^0 \in \approx_0^N\}, \\ \Lambda_k(\approx_0, \approx_0^N, \sigma(0)) &\stackrel{\text{def}}{=} \{\Lambda_k(t_0, \vec{v}^0, \sigma(0)) : t_0 \in \approx_0, \vec{v}^0 \in \approx_0^N\}, \\ \Psi_k^-(\approx_0, \approx_0^N, \sigma(0)) &\stackrel{\text{def}}{=} \{\Psi_k^-(t_0, \vec{v}^0, \sigma(0)) : t_0 \in \approx_0, \vec{v}^0 \in \approx_0^N\}, \\ \Psi_k^+(\approx_0, \approx_0^N, \sigma(0)) &\stackrel{\text{def}}{=} \{\Psi_k^+(t_0, \vec{v}^0, \sigma(0)) : t_0 \in \approx_0, \vec{v}^0 \in \approx_0^N\}. \end{aligned}$$

We define trajectory $\Phi'(t, t_0, \vec{v}^0, \sigma(0))$ as the value of $\vec{x}(t)$ starting from $\vec{x}(t_0) = \vec{v}^0$, location $\sigma(0)$, and only under the control of differential equation $\dot{\vec{x}} = f_{\sigma(0)}(\vec{x})$. We

also define $\Phi'(t, \approx_0, \lesssim_0, \sigma(0))$ with $\Phi'(t, t_0, \vec{v}^0, \sigma(0))$, using the same way we define $\Phi(t, \approx_0, \lesssim_0, \sigma(0))$ with $\Phi(t, t_0, \vec{v}^0, \sigma(0))$.

We need the following lemmas to start proving Theorem 1.

Lemma 1 (quoted from [Ari06]) *If $f_{\sigma(0)}(\vec{x})$ and $\frac{df_{\sigma(0)}(\vec{x})}{d\vec{x}}$ are both bounded, then for any $t > t_0$ and initial value \vec{v}^0 , \exists \mathcal{K} -function $\delta(\cdot)$ s.t. for any sufficiently small $\varepsilon > 0$, $\Phi'(t, t_0, \vec{v}^0 + \varepsilon\mathbb{B}_N, \sigma(0)) \subseteq \Phi'(t, t_0 + \varepsilon\mathbb{B}_1, \vec{v}^0 + \varepsilon\mathbb{B}_N, \sigma(0)) \subseteq \Phi'(t, t_0, \vec{v}^0, \sigma(0)) + \delta(\varepsilon)\mathbb{B}_N$.*

Lemma 2 (quoted from [Che16a]) *Given F1~F4 of Theorem 1, given $\vec{x}(t) = \Phi(t, t_0, \vec{v}^0, \sigma(0))$, if jump time instance t_1 exists after all, then \exists \mathcal{K} -function $\delta(\cdot)$ s.t. for any sufficiently small $\varepsilon > 0$, we have*

$$\begin{aligned}\Psi_1^-(t_0, \vec{v}^0 + \varepsilon\mathbb{B}_N, \sigma(0)) &\subseteq \vec{x}(t_1^-) + \delta(\varepsilon)\mathbb{B}_N, \\ \Lambda_1(t_0, \vec{v}^0 + \varepsilon\mathbb{B}_N, \sigma(0)) &\subseteq t_1 + \delta(\varepsilon)\mathbb{B}_N.\end{aligned}$$

Lemma 3 *Given the same condition as Lemma 2, \exists \mathcal{K} -function $\delta(\cdot)$ s.t. for any sufficiently small $\varepsilon > 0$, we have*

$$\begin{aligned}\Psi_1^-(t_0 + \varepsilon\mathbb{B}_1, \vec{v}^0 + \varepsilon\mathbb{B}_N, \sigma(0)) &\subseteq \vec{x}(t_1^-) + \delta(\varepsilon)\mathbb{B}_N, \\ \Lambda_1(t_0 + \varepsilon\mathbb{B}_1, \vec{v}^0 + \varepsilon\mathbb{B}_N, \sigma(0)) &\subseteq t_1 + \delta(\varepsilon)\mathbb{B}_N.\end{aligned}$$

Proof: Note hybrid automaton A is time invariant. Based on Lemma 2, we derive Lemma 3. ■

Now back to the proof of Theorem 1.

Given $T > 0$, given trajectory $\vec{x}(t) = \Phi(t, t_0, \vec{v}^0, \sigma(0))$ ($t \in [t_0, +\infty)$), where $\vec{v}^0 \in \mathcal{I} - \text{boundary}(\llbracket \text{inv}(\sigma(0)) \rrbracket)$ and $\sigma(0) = l^0$, let us first assume $\vec{x}(t)$ jumps $(K - 1) > 0$ times on $[t_0, t_0 + T)$, and jumps on $[T, +\infty)$.

According to Lemma 2, $\exists \mathcal{K}$ -function $\tilde{\delta}_1(\cdot)$ s.t. for sufficiently small $\varepsilon > 0$,

$$\begin{aligned}\Psi_1^-(t_0, \vec{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) &\subseteq \vec{x}(t_1^-) + \tilde{\delta}_1(\varepsilon) \mathbb{B}_N, \\ \Lambda_1(t_0, \vec{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) &\subseteq t_1 + \tilde{\delta}_1(\varepsilon) \mathbb{B}_1.\end{aligned}$$

Due to F3.3, $\exists \mathcal{K}$ -function $\hat{\delta}_1(\cdot)$ s.t.

$$\Omega_{\sigma(0)}(\Psi_1^-(t_0, \vec{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0))) \subseteq \vec{x}(t_1^+) + \hat{\delta}_1(\varepsilon) \mathbb{B}_N.$$

Denote $\delta_1 \stackrel{\text{def}}{=} \max\{\tilde{\delta}_1, \hat{\delta}_1\}$, then $\delta_1(\cdot)$ is \mathcal{K} -function, and

$$\begin{aligned}\Psi_1^+(t_0, \vec{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) &\subseteq \vec{x}(t_1^+) + \delta_1(\varepsilon) \mathbb{B}_N, \\ \Lambda_1(t_0, \vec{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) &\subseteq t_1 + \delta_1(\varepsilon) \mathbb{B}_1.\end{aligned}$$

Apply Lemma 3 to the above formulas, $\exists \mathcal{K}$ -function $\tilde{\delta}_2(\cdot)$ s.t. for sufficiently small $\varepsilon > 0$,

$$\begin{aligned}\Psi_2^-(t_0, \vec{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) &\subseteq \vec{x}(t_2^-) + \tilde{\delta}_2(\varepsilon) \mathbb{B}_N, \\ \Lambda_2(t_0, \vec{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) &\subseteq t_2 + \tilde{\delta}_2(\varepsilon) \mathbb{B}_1.\end{aligned}$$

Due to F3.3, $\exists \mathcal{K}$ -function $\hat{\delta}_2(\cdot)$ s.t.

$$\Omega_{\sigma(1)}(\Psi_2^-(t_0, \vec{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0))) \subseteq \vec{x}(t_2^+) + \hat{\delta}_2(\varepsilon) \mathbb{B}_N.$$

Denote $\delta_2 \stackrel{\text{def}}{=} \max\{\tilde{\delta}_2, \hat{\delta}_2\}$ and repeat the above process, we finally will have $\forall k \in \{1, \dots, K\}$, $\exists \mathcal{K}$ -function $\delta_k(\cdot)$ s.t. for sufficiently small $\varepsilon > 0$,

$$\Psi_k^+(t_0, \bar{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) \subseteq \bar{x}(t_k^+) + \delta_k(\varepsilon) \mathbb{B}_N,$$

$$\Lambda_k(t_0, \bar{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) \subseteq t_k + \delta_k(\varepsilon) \mathbb{B}_1.$$

Denote $\delta \stackrel{\text{def}}{=} \max\{\delta_k\}_{k=1}^K$, then $\delta(\cdot)$ is a \mathcal{K} -function, and for sufficiently small $\varepsilon > 0$, $\forall k \in \{1, \dots, K\}$,

$$\Psi_k^+(t_0, \bar{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) \subseteq \bar{x}(t_k^+) + \delta(\varepsilon) \mathbb{B}_N, \quad (3.2)$$

$$\Lambda_k(t_0, \bar{v}^0 + \varepsilon \mathbb{B}_N, \sigma(0)) \subseteq t_k + \delta(\varepsilon) \mathbb{B}_1. \quad (3.3)$$

$\forall \tau \in (t_0, T)$, suppose for trajectory $\bar{x}(t)$, the current location $\rho(\tau) = \sigma(k)$ ($k \in \{1, \dots, K-1\}$). We conduct the following discussions.

Case A): τ is not a jump time instance for $\bar{x}(t)$.

As A is nonzeno, we have $t_k < \tau < t_{k+1}$.

Based on Exp. (3.3), we have for sufficiently small $\varepsilon > 0$, any trajectory $\bar{x}'(t)$ started from $(t_0, \bar{v}^0, \sigma(0))$, where $\bar{v}^0 \in \bar{v}^0 + \varepsilon \mathbb{B}_N$, should jump to location $\sigma(k)$ and next to $\sigma(k+1)$ at t'_k and t'_{k+1} respectively. t'_k and t'_{k+1} are sufficiently close to t_k and t_{k+1} . Hence $t'_k < \tau < t'_{k+1}$, i.e. during $[t'_k, \tau]$, $\bar{x}'(t)$ is traveling in $\sigma(k)$. Meanwhile, during $[t_k, \tau]$, $\bar{x}(t)$ is also traveling in $\sigma(k)$.

Due to Exp. (3.3), $|t'_k - t_k| \leq \delta(\varepsilon)$.

If $t'_k \leq t_k$, then due to F1, $\|\bar{x}'(t_k) - \bar{x}'(t'_k)\| \leq F_2 \delta(\varepsilon)$. Due to Exp. (3.2), $\|\bar{x}'(t'_k) - \bar{x}(t_k)\| \leq \delta(\varepsilon)$. Hence $\|\bar{x}'(t_k) - \bar{x}(t_k)\| \leq (F_2 + 1)\delta(\varepsilon)$. Due to Lemma 1, $\exists \mathcal{K}$ -

function $\check{\delta}(\cdot)$ s.t. $\|\bar{x}'(\tau) - \bar{x}(\tau)\| \leq \check{\delta}((F_2 + 1)\delta(\varepsilon))$. Correspondingly $\|\bar{y}'(\tau) - \bar{y}(\tau)\| \leq \check{\delta}((F_2 + 1)\delta(\varepsilon))$. If $t'_k > t_k$, using the same way we can still prove $\|\bar{y}'(\tau) - \bar{y}(\tau)\| \leq \check{\delta}((F_2 + 1)\delta(\varepsilon))$.

Case B): τ is a jump time instance for $\bar{x}(t)$. Suppose $\tau = t_k$ ($k \in \{1, \dots, K - 1\}$). For any trajectory $\bar{x}'(t)$ with initial value $\bar{v}^0 \in (\bar{v}^0 + \varepsilon \mathbb{B}_N) \cap (\mathcal{I} - \text{boundary}(\llbracket \text{inv}(l^0) \rrbracket))$, where $\varepsilon > 0$ is sufficiently small. Due to Exp. (3.2)(3.3), $\bar{x}'(t)$ should have at least K jumps on $[t_0, +\infty)$, respectively close to $\bar{x}(t)$'s K jumps. Suppose the k th jump time instances and jump-to locations are respectively t'_k and $\sigma'(k)$ ($k = 1, \dots, K$). As per Exp. (3.2), $\sigma'(k) = \sigma(k)$. Also, let us denote $\bar{x}'(t)$'s current location as $\rho'(t)$.

Case B.1): If $t'_k \leq t_k$. As ε is sufficiently small, hence $\delta(\varepsilon)$ is sufficiently small. Due to this, Exp. (3.3), and nonzenoness of A (see F4), during $[t'_k, t_k]$, $\bar{x}'(t)$'s current location $\rho'(t)$ is always in $\sigma(k)$, and $|t'_k - t_k| \leq \delta(\varepsilon)$. Due to F1 and Exp. (3.2) $\|\bar{x}(\tau) - \bar{x}'(\tau)\| = \|\bar{x}(t_k) - \bar{x}'(t_k)\| \leq (F_2 + 1)\delta(\varepsilon)$. Therefore $\|\bar{y}(\tau) - \bar{y}'(\tau)\| \leq (F_2 + 1)\delta(\varepsilon)$.

Case B.2): if $t'_k > t_k$. As ε is sufficiently small, hence $\delta(\varepsilon)$ is sufficiently small. Due to this, Exp. (3.3), and nonzenoness of A (see F4), during $[t_k, t'_k]$, $\bar{x}'(t)$'s current location $\rho'(t)$ is always in $\sigma(k - 1)$ and $\bar{x}(t)$'s current location $\rho(t)$ is always in $\sigma(k)$. Due to Exp. (3.2), $\|\bar{y}'(t'_k) - \bar{y}(\tau)\| \leq \delta(\varepsilon)$. Due to continuity $\bar{y}'(t'_k) = \bar{y}'(t'_k^-)$. Due to F1 and Exp. (3.3), $\|\bar{y}'(\tau) - \bar{y}'(t'_k)\| = \|\bar{y}'(\tau) - \bar{y}'(t'_k^-)\| \leq F_2\delta(\varepsilon)$. Therefore $\|\bar{y}(\tau) - \bar{y}'(\tau)\| \leq (F_2 + 1)\delta(\varepsilon)$.

Summarizing **Case A** and **B**, \mathcal{K} -function $B(\varepsilon) \stackrel{\text{def}}{=} \max\{\check{\delta}((F_2 + 1)\delta(\varepsilon)), (F_2 + 1)\delta(\varepsilon)\}$ is the wanted bound function.

Finally, for the case where $K - 1 > 0$ and $\bar{x}(t)$ does not jump on $[T, +\infty)$, we can follow the same way of proof (imagine $t_K = +\infty$). For the case where $K - 1 \leq 0$, $\bar{x}(t)$ does not jump on $[t_0, T)$, we can still follow the same way of proof (now only need to use Lemma 1).

In summary, the theorem holds. ■

3.4 Proposed Approach

In this section, we aim to address the demand proposed in Section 1.1, i.e. finding a more principled and systematic approach to prepare execution traces for control-CPS cyber subsystem fault localization. The hope lies in the powerful tool of hybrid automata. With hybrid automata, we can now holistically model and analyze the cyber and physical aspects of control-CPSs. Generic properties derived from the analysis shall help us find the wanted approach. In case the model is incomplete/inaccurate, we assume the generic properties still sustain, and empirically validate our proposed approach.

3.4.1 Summary and Intuition of the Theory

In the last section, we give a solid proof of the existence of physical traces divergence bound, which is inspired by the state-of-the-art hybrid system stability research [GST12]. In this section, we would like to get rid of most of the mathematical parts and express the theory in a more intuitive way. This should help the readers see how the theory can help to build our approach.

Intuitively, we find a reasonably generic property for any hybrid automaton $A(\mathcal{L}, \mathcal{E}, \vec{x}, \mathcal{I}, \rho, l^0, \text{act}, \text{inv}, \text{grd}, \text{rst})$. If A satisfies certain mild constraints, its physical state vector trajectories started from close-by initial values shall stay close forever. This intuition is illustrated by the cluster of trajectories 1 ~ 4 and 5 ~ 8 in Fig. 3.1. We can summarize Theorem 1 to the following sketch as readers' convenience.

Sketch of Theorem 1 *Given a hybrid automaton A , if*

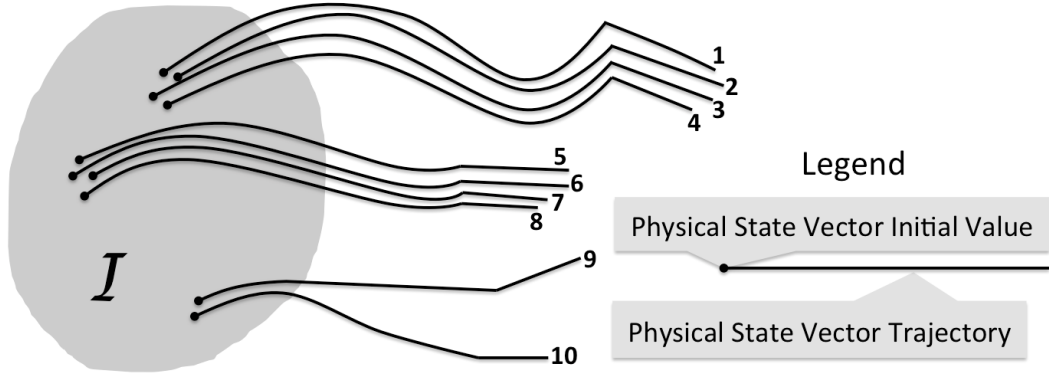


Fig. 3.1: Approach Heuristics. Dark shaded area is \mathcal{I} .

F1 in each location (i.e. node), the physical state vector $\vec{x}(t)$ dynamics are continuous, differentiable, and bounded;

F2 jump (i.e. transition) only happens when the $\vec{x}(t)$ deterministically violates current location's constraints;

F3 a location can have several constraints for $\vec{x}(t)$;

F3.1 when $\vec{x}(t)$ deterministically violates a constraint, there is always another unique location to jump to, and a unique new value for $\vec{x}(t)$ in the jump-to location;

F3.2 constraints are well separated, i.e. for $\vec{x}(t)$ there are no borderline values, at which an arbitrarily small deviation from $\vec{x}(t)$ can lead to violating a different constraint;

F3.3 a sufficiently small deviation (see footnote ¹ in Section 3.3) from $\vec{x}(t)$ leads to a same jump, and a similar new value for $\vec{x}(t)$ in the jump-to location;

F4 A is nonzero, i.e. jumps cannot happen subsequently with 0 time interval; and A is nonblocking, i.e. $\vec{x}(t)$ can go on forever once started.

Then given time interval $[0, T]$, there is a bound $B(\cdot)$ s.t. for any two physical state vector trajectories $\vec{x}_1(t)$ and $\vec{x}_2(t)$, if their initial values $\vec{x}_1(0)$, $\vec{x}_2(0)$ are sufficiently close (denote the distance as ε), and neither is on the borderline of any constraints (i.e. small deviations from $\vec{x}_1(0)$ or $\vec{x}_2(0)$ will not violate any constraint), then distance between $\vec{x}_1(t)$ and $\vec{x}_2(t)$ are bounded by $B(\varepsilon)$ throughout $(0, T)$. We call $B(\cdot)$ the physical traces divergence bound.

Note the thermostat control-CPS example in Section 1.3 can help us better understand F3.2 and F3.3. As per Fig. 4.1, at location l_{heat} , an arbitrarily small deviation $\varepsilon > 0$ of initial value, say from $x(0) = v^0$ to $x(0) = v^0 + \varepsilon$ cannot change the constraint that temperature trajectory $x(t)$ will hit: θ_h . Upon hitting the θ_h constraint, the execution of the hybrid automaton always switches to location l_{cool} with the same (hence similar) new temperature value $x(t) = \theta_h$.

3.4.2 Heuristics and Specifications of Proposed Approach

In a narrow sense, the execution/predicted physical trace of a control CPS is just the execution/predicted physical state vector trajectory. In the following, we shall use the two terms interchangeably. Intuitively, if two executions started from close-by physical state vector initial values are both correct, the physical traces should satisfy Theorem 1; otherwise, at least one of the executions is faulty. This inspires us to propose the following control-CPS execution traces preparation heuristics.

If a control-CPS's hybrid automaton model $A(\mathcal{L}, \mathcal{E}, \vec{x}, \mathcal{I}, \rho, l^0, \text{act}, \text{inv}, \text{grd}, \text{rst})$ satisfies feature F1~F4 of Theorem 1, then the execution physical traces' divergence should be bounded by $B(\varepsilon)$ as described by Theorem 1. Therefore, we can sample the physical state vector's initial value set \mathcal{I} in a clustered pattern, as shown in Fig. 3.1. That is, we sample I clusters of initial values, with J samples per cluster.

According to Fig. 3.1, within a sampling cluster, distances of physical state vector initial values are within ε , thus distances of the trajectories of these physical state vectors should be bounded by $B(\varepsilon)$. This is illustrated by the cluster of trajectory 1 ~ 4 and trajectory 5 ~ 8 in Fig. 3.1. Any pair of trajectories exceeding the distance bound means at least one of them is wrong, e.g. as illustrated by the cluster of trajectory 9 and 10 in Fig. 3.1. In case we cannot tell exactly which one is wrong, we label both as "faulty". We conjecture this pessimistic practice acceptable when $I \gg J$,

and the conjecture is corroborated by our large-scale empirical study (see Chapter 4). The intuition is as follows. Fault localization tools are the most useful for the stage of debugging *residual bugs* [Nat+13], i.e. bugs overlooked by best-effort manual debugging. As “overlooked” bugs, residual bugs rarely emerge. Correspondingly, in the residual bug debugging stage, of all I clusters of traces, only few cluster(s) may contain faulty trace(s). Even if we label all traces in those(that) cluster(s) as “faulty”, as $I \gg J$, the number of false positives is still limited and likely to be better than a subjective and ad-hoc approach.

In case a complete and sound hybrid automaton model A that satisfies feature F1~F4 of Theorem 1 is unavailable, which is common for a real-world control-CPS, we conjecture the above heuristics can still work. The rationale is that man-made control-CPSs are intentionally designed not to become chaos systems [HSD12], where infinitesimal deviation from initial value can cause drastic trace differences. Thus in control-CPSs, borderline behaviors violating F1~F4 of Theorem 1, though possible, are statistically rare under sufficiently random sampling. Thus we can imagine a hidden Theorem 1-complying hybrid automaton model \tilde{A} still exists; and regard violation of Theorem 1 as faults. Results of our evaluation in Chapter 4 give important empirical evidence for the correctness of the conjecture.

We formally specify the above heuristics into the following approach steps. Note rigorous definitions of **Step 4** and **9** contain too much math and the we put the details into a separate section.

Step 1) Given a control-CPS implementation, confirm if this control-CPS has a hybrid automaton model $A = (\mathcal{L}, \mathcal{E}, \vec{x}, \mathcal{I}, \rho, l^0, \text{act}, \text{inv}, \text{grd}, \text{rst})$. If not, go to **Step 7**.

Step 2) Confirm if hybrid automaton model A satisfies feature F1~F4 of Theorem 1. If cannot confirm, go to **Step 7**.

Step 3) As per Theorem 1, a physical traces divergence bound $B(\varepsilon)$ exists. Try to find this bound, i.e. the closed-form $B(\varepsilon)$. If this fails, go to **Step 7**.

Step 4) We randomly sample $I \in \mathbb{Z}^+$ vectors $\{\vec{p}_i\}_{i=1}^I$ uniformly distributed in the valid initial value set \mathcal{I} (but excluding all borderline values for location constraints); then for each \vec{p}_i , we randomly sample $J \in \mathbb{Z}^+$ ($J \ll I$) vectors in its $\frac{\varepsilon}{2}$ vicinity (where $\varepsilon > 0$ is sufficiently small, see footnote ¹ in Section 3.3), denote them as $\{\vec{v}^{i,j}\}_{j=1}^J$. Thus for given i , $\vec{v}^{i,j}$ s ($j = 1, \dots, J$) are in ε distance to each other.

Step 5) For each $\vec{v}^{i,j}$ ($i \in \{1, \dots, I\}$, $j \in \{1, \dots, J\}$), we set physical state vector \vec{x} 's initial value to $\vec{v}^{i,j}$ and run the control-CPS for T seconds (where $T > 0$ is a pre-configured constant) to get an execution physical trace $\vec{x}^{i,j}(t)$ ($t \in [0, T]$). Meanwhile, we also log the execution cyber trace, denoted as $\text{cybtrc}(i, j)$, and label it as “correct” for now.

Step 6) For each $i \in \{1, \dots, I\}$, for each pair of execution physical traces $\vec{x}^{i,j}(t)$ and $\vec{x}^{i,\ell}(t)$ (where $j, \ell \in \{1, \dots, J\}$ and $j \neq \ell$; $t \in [0, T]$), if their distance ever exceeds $B(\varepsilon)$ during $(0, T)$, then Theorem 1 is violated, we label both $\text{cybtrc}(i, j)$ and $\text{cybtrc}(i, \ell)$ as “faulty” (note the cyber traces were labeled “correct” by default in **Step 5**). Go to **Step 10**.

Step 7, 8) Respectively the same as **Step 4** and **5**.

Step 9) For each $i \in \{1, \dots, I\}$, for each pair of execution physical traces $\vec{x}^{i,j}(t)$ and $\vec{x}^{i,\ell}(t)$ (where $j, \ell \in \{1, \dots, J\}$ and $j \neq \ell$; $t \in [0, T]$), if their distance ever becomes statistically too separated during $(0, T)$ (see Section 3.4.3 for rigorous definitions on “statistically too separated”), then we consider a violation of the hidden Theorem 1-complying hybrid automaton model \tilde{A} happens, and label both $\text{cybtrc}(i, j)$ and $\text{cybtrc}(i, \ell)$ as “faulty” (note the cyber traces were labeled “correct” by default in **Step 8**). Go to **Step 10**.

Step 10) The End.

For convenience, we call the above **Step 1~10** our “*proposed approach*” in the following.

3.4.3 Rigorous Definitions of **Step 4** and **9**

Step 4) We uniformly sample $I \in \mathbb{Z}^+$ vectors in \mathcal{I} -boundary($\llbracket \text{inv}(l^0) \rrbracket$). Denote each sample as \vec{p}_i ($i = 1, \dots, I$). For each \vec{p}_i , we define an \vec{c} -hyper-cube as $\mathbb{C}(\vec{c}, \vec{p}_i) \stackrel{\text{def}}{=} \{\vec{v} \in \mathbb{R}^N : \forall n \in \{1, \dots, N\}, |\vec{v}_n - \vec{p}_{i,n}| \leq \frac{\vec{c}_n}{2}\}$, where $\vec{c} \in (\mathbb{R}^+)^N$ is a pre-configured vector; $\vec{c}_n, \vec{v}_n, \vec{p}_{i,n}$ are respectively the n th dimension of \vec{c}, \vec{v} and \vec{p}_i . Furthermore, \vec{c} satisfies $\|\vec{c}\| \leq \varepsilon$, where ε is a sufficiently small real number. We then uniformly sample $J \in \mathbb{Z}^+$ vectors from $\mathbb{C}(\vec{c}, \vec{p}_i) \cap \mathcal{I}$, and denote each sample as $\vec{v}^{i,j}$ ($j = 1, \dots, J$). As $\|\vec{c}\| \leq \varepsilon$, for the given \vec{p}_i , the *cluster* of vectors $\{\vec{v}^{i,j}\}_{j=1}^J$ are all within ε distance to each other.

Step 9) For each $i \in \{1, \dots, I\}$, for each $j, \ell \in \{1, \dots, J\}$ and $j \neq \ell$, compute the n th dimension divergence ratio $\gamma_n^{i,j,\ell}$ (where $n \in \{1, \dots, N_{\text{cont}}\}$) as $\gamma_n^{i,j,\ell} \stackrel{\text{def}}{=} \max_{t \in [0, T]} \left\{ \left| \vec{x}_n^{i,j}(t) - \vec{x}_n^{i,\ell}(t) \right| / \left| \vec{x}_n^{i,j}(0) - \vec{x}_n^{i,\ell}(0) \right| \right\}$.

We pick a statistical confidence percentile α (as per convention, we pick $\alpha = 95\%$). For each dimension $n \in \{1, \dots, N_{\text{cont}}\}$, we estimate the α -th percentile of all observed $\gamma_n^{i,j,\ell}$ ($i \in \{1, \dots, I\}, j, \ell \in \{1, \dots, J\}$, and $j \neq \ell$) as the *estimated bound for the n th dimension divergence ratio*, denoted as $\hat{\Gamma}_n$.

We label $\text{cybtrc}(i, j)$ ($i \in \{1, \dots, I\}, j \in \{1, \dots, J\}$) as “*faulty*” if $\exists n, \ell$ ($n \in \{1, \dots, N_{\text{cont}}\}, \ell \in \{1, \dots, J\}$, and $j \neq \ell$) s.t. $\gamma_n^{i,j,\ell} > \hat{\Gamma}_n$; and as “*correct*” otherwise.

Evaluation and Results

4.1 Overview

In this chapter, we will show two use cases of our proposed approach. The first use case demonstrates how to apply our approach for control-CPS with known model, which is a thermostat to maintain the temperature in an oven within a certain range. In the second use case, we conduct experiments on a widely used control-CPS, Ardupilot, which is an open source controller for UAVs. We evaluate our approach by injecting bugs in to the source code, and locating the faults using three fault localization tools. We compare our approach with currently commonly used **Way-2** approach (see 1.1), and the result shows our approach achieves significant improvements.

4.2 A Simple Example: Thermostat Control-CPS

4.2.1 System Model of Thermostat

In this section, we will re-visit the thermostat control-CPS and see how our proposed approach can be applied to debug such system. Fig. 4.1 gives an example hybrid automaton model of the thermostat control-CPS, which automatically heats an oven to maintain oven temperature x in range $[\theta_l, \theta_h]$ ($0 < \theta_l < \theta_h$).

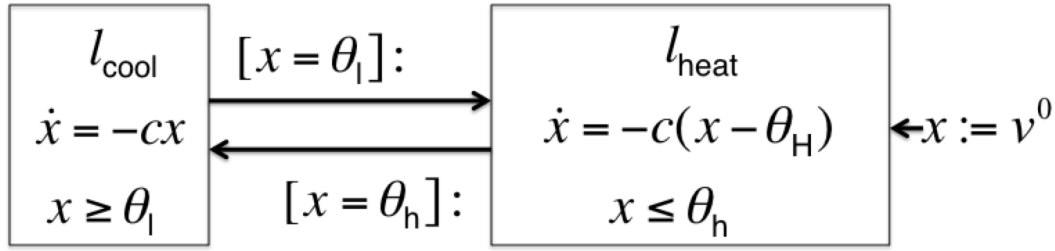


Fig. 4.1: Example Hybrid Automata: Thermostat. Note as oven temperature x is the only concerned physical state, the physical state vector is 1-dimensional and is hence simply represented by scalar x .

There are two locations in the hybrid automaton: l_{heat} for heating and l_{cool} for cooling. The initial oven temperature is v^0 and the oven is initially set to location l_{heat} .

In l_{cool} , the heater stops, hence the oven temperature drops at a speed proportional to inside-outside temperature difference (assuming outside temperature is 0), i.e. $\text{act}(l_{\text{cool}})$ is $\dot{x} = -cx$. In l_{heat} , the heater is on, turning the outside temperature to $\theta_H > \theta_h$, hence the oven temperature rises at a speed proportional to inside-outside temperature difference, i.e. $\text{act}(l_{\text{heat}})$ is $\dot{x} = -c(x - \theta_H)$. Coefficient c is a constant dependent on the type of gas in the oven and the oven wall material.

In l_{cool} , oven temperature must not drop below θ_l , i.e. $\text{inv}(l_{\text{cool}})$ is $x \geq \theta_l$. When x reaches θ_l , $\text{grd}(l_{\text{cool}})$ is triggered, which is denoted in Fig. 4.1 as $[x = \theta_l] :$, and the jump to l_{heat} takes place. As oven temperature x retains its old value immediately after the jump, $\text{rst}((l_{\text{cool}}, l_{\text{heat}}))$ is not explicitly drawn in Fig. 4.1. In contrast, in l_{heat} , oven temperature must not rise over θ_h , i.e. $\text{inv}(l_{\text{heat}})$ is $x \leq \theta_h$. When x reaches θ_h , $\text{grd}(l_{\text{heat}})$ is triggered, which is denoted in Fig. 4.1 as $[x = \theta_h] :$, and the jump to l_{cool} takes place. As oven temperature x retains its old value immediately after the jump, $\text{rst}((l_{\text{heat}}, l_{\text{cool}}))$ is not explicitly drawn in Fig. 4.1.

4.2.2 Thermostat Physical Traces Divergence Bound

(Acknowledgement: The mathematical analysis of this sub-section is the result of a joint effort led by Dr. Yao Chen of our group.)

Let $x(t)$ ($t \geq t_0$) denote the oven temperature (i.e. the only physical state interested in this control-CPS) trajectory. When current location is l_{heat} and l_{cool} , based on the corresponding differential equation given in Fig. 4.1, the dynamics of $x(t)$ ($t \geq t_0$) is respectively

$$x(t - t_0) = x(t_0)e^{-c(t-t_0)} + \theta_{\text{H}}(1 - e^{-c(t-t_0)}), \quad (4.1)$$

$$x(t - t_0) = x(t_0)e^{-c(t-t_0)}. \quad (4.2)$$

Correspondingly, given initial oven temperature $v^0 \in [\theta_l, \theta_h]$, in l_{heat} (l_{cool}), the time needed to reach θ_h (θ_l) is respectively

$$t_{\uparrow}(v^0) = \frac{1}{c} \ln \frac{\theta_{\text{H}} - v^0}{\theta_{\text{H}} - \theta_{\text{h}}} \quad (4.3)$$

$$t_{\downarrow}(v^0) = \frac{1}{c} \ln \frac{v^0}{\theta_l}. \quad (4.4)$$

Particularly, we denote

$$T_{\uparrow} \stackrel{\text{def}}{=} t_{\uparrow}(\theta_l) = \frac{1}{c} \ln \frac{\theta_{\text{H}} - \theta_l}{\theta_{\text{H}} - \theta_{\text{h}}} \quad (4.5)$$

$$T_{\downarrow} \stackrel{\text{def}}{=} t_{\downarrow}(\theta_{\text{h}}) = \frac{1}{c} \ln \frac{\theta_{\text{h}}}{\theta_l}. \quad (4.6)$$

Let $\Phi(t, t_0, v^0, l_{\text{h}})$ ($t \geq t_0$) be the oven temperature trajectory starting from time t_0 , initial temperature $v^0 \in [\theta_l, \theta_h]$, and location l_{h} in the hybrid automaton of Fig. 4.1. Suppose $x_1(t) = \Phi(t, t_0, v_1^0, l_{\text{h}})$ and $x_2(t) = \Phi(t, t_0, v_2^0, l_{\text{h}})$, where without loss of generality, we assume

$$\theta_l \leq v_1^0 \leq v_2^0 \leq \theta_{\text{h}}. \quad (4.7)$$

Furthermore, let $\varepsilon \stackrel{\text{def}}{=} v_2^0 - v_1^0 \geq 0$. In the following we assume

$$\varepsilon < \min\left\{\frac{\theta_h - \theta_l}{\theta_h}(\theta_H - \theta_h), \frac{\theta_h - \theta_l}{\theta_H - \theta_l}(\theta_H - \theta_h)\right\}. \quad (4.8)$$

Then we have the following lemma.

Lemma 4 *Suppose t_1 and t_2 are respectively the first time instances that $x_1(t)$ and $x_2(t)$ hits θ_h , then under Assumption (4.7)(4.8), we have $t_2 \leq t_1$ and $t_1 - t_2 < \min\{T_\uparrow, T_\downarrow\}$.*

Proof: Due to Exp. (4.4), we get $t_1 = \frac{1}{c} \ln \frac{\theta_H - v_1^0}{\theta_H - \theta_h}$, $t_2 = \frac{1}{c} \ln \frac{\theta_H - v_2^0}{\theta_H - \theta_h}$. As $v_1^0 \leq v_2^0$, we have $t_2 \leq t_1$.

$$\begin{aligned} \varepsilon = v_2^0 - v_1^0 < \frac{\theta_h - \theta_l}{\theta_h}(\theta_H - \theta_h) &\Rightarrow v_2^0 - v_1^0 < \frac{\theta_h - \theta_l}{\theta_h}(\theta_H - v_1^0) \Leftrightarrow \frac{\theta_H - v_1^0}{\theta_H - v_2^0} < \frac{\theta_h}{\theta_l} \Leftrightarrow \frac{1}{c} \ln \frac{\theta_H - v_1^0}{\theta_H - v_2^0} < \\ \frac{1}{c} \ln \frac{\theta_h}{\theta_l} &\Leftrightarrow t_1 - t_2 < T_\downarrow. \end{aligned}$$

$$\begin{aligned} \varepsilon = v_2^0 - v_1^0 < \frac{\theta_h - \theta_l}{\theta_H - \theta_l}(\theta_H - \theta_h) &\Rightarrow v_2^0 - v_1^0 < \frac{\theta_h - \theta_l}{\theta_H - \theta_l}(\theta_H - v_1^0) \Leftrightarrow \frac{\theta_H - v_1^0}{\theta_H - v_2^0} < \frac{\theta_H - \theta_l}{\theta_H - \theta_h} \Leftrightarrow \\ \frac{1}{c} \ln \frac{\theta_H - v_1^0}{\theta_H - v_2^0} < \frac{1}{c} \ln \frac{\theta_H - \theta_l}{\theta_H - \theta_h} &\Leftrightarrow t_1 - t_2 < T_\uparrow. \end{aligned}$$

■

As per the Thermostat hybrid automaton (see Fig. 4.1), once x_2 (x_1) hits θ_h for the first time, it will start cooling as per Exp. (4.2), until x_2 (x_1) hits θ_l . Denote the time instance as t'_2 (t'_1). Then $t'_2 = t_2 + T_\downarrow$ ($t'_1 = t_1 + T_\uparrow$). Right after t'_2 (t'_1), x_2 (x_1) will start heating as per Exp. (4.1), until x_2 (x_1) hits θ_h . Denote the time instate as t''_2 (t''_1). Then $t''_2 = t'_2 + T_\uparrow = t_2 + T_\downarrow + T_\uparrow$ ($t''_1 = t'_1 + T_\uparrow = t_1 + T_\downarrow + T_\uparrow$). Right after t''_2 (t''_1), the above period restarts.

Apparently

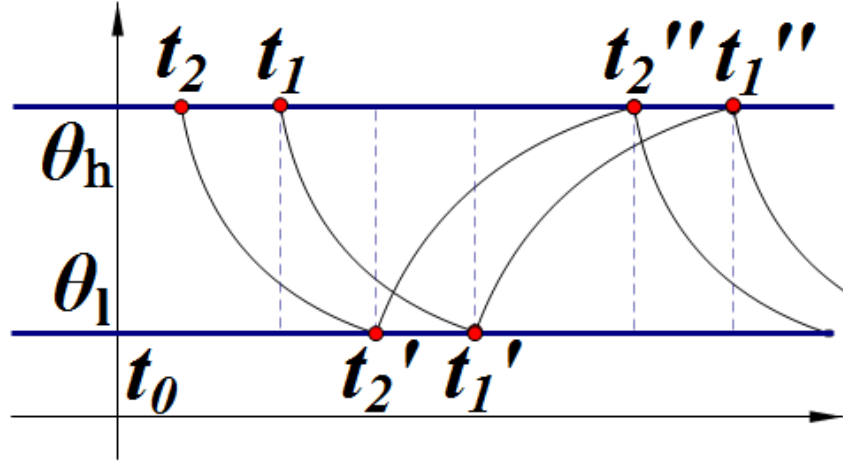


Fig. 4.2: Periodical Trace Behavior of Thermostat

$$t_1 - t_2 = t'_1 - t'_2 = t''_1 - t''_2. \quad (4.9)$$

Also, as per Lemma 4, $t_2 \leq t_1$, $t_1 - t_2 < T_{\downarrow} \Rightarrow t_1 < t'_2$, and $t_1 - t_2 = t'_1 - t'_2 < T_{\uparrow} \Rightarrow t'_1 < t''_2$. Therefore, we have

$$t_2 \leq t_1 < t'_2 \leq t'_1 < t''_2 \leq t''_1. \quad (4.10)$$

Thus the periodical behaviors of the pair x_2 and x_1 are shown in Fig. 4.2. Due to the periodicity, we only need to analyze one period, e.g. $[t_1, t_1 + T_{\downarrow} + T_{\uparrow} = t''_1]$ of x_2 and x_1 (see Fig. 4.2), to fully understand them (note the behaviors of x_2 and x_1 during $[t_0, t_1]$ are the same as those during $[t''_1 - (t_1 - t_0), t''_1]$).

According to Fig. 4.2, the period of $[t_1, t''_1]$ has four partitions: $[t_1, t'_2]$, $[t'_2, t'_1]$, $[t'_1, t''_2]$, and $[t''_2, t''_1]$. Use Exp. (4.1)(4.2), we can respectively get that $|x_1(t) - x_2(t)| \leq c\theta_h|t_1 - t_2|$ ($\forall t \in [t_1, t'_2]$), $|x_1(t) - x_2(t)| \leq c(\theta_H + \theta_h)|t_1 - t_2|$ ($\forall t \in [t'_2, t'_1]$), $|x_1(t) -$

$x_2(t) \leq c(\theta_H - \theta_l)|t_1 - t_2|$ ($\forall t \in [t'_1, t''_1]$), and $|x_1(t) - x_2(t)| \leq c(2\theta_H - \theta_l)|t_1 - t_2|$ ($\forall t \in [t''_2, t''_1]$).

Meanwhile, as per Exp. (4.4), $|t_1 - t_2| = \frac{1}{c} n \frac{\theta_H - v_1^0}{\theta_H - v_2^0} = \frac{1}{c} n (1 + \frac{\varepsilon}{\theta_H - v_2^0}) \leq \frac{1}{c} \frac{\varepsilon}{\theta_H - v_2^0} \leq \frac{1}{c} \frac{\varepsilon}{\theta_H - \theta_h}$ (as $v_2^0 \leq \theta_h$).

Therefore, $\forall t \geq t_0$, we have

$$\begin{aligned} |x_1(t) - x_2(t)| &\leq c(\theta_H + \max\{\theta_h, \theta_H - \theta_l\})|t_1 - t_2| \\ &\leq (2\theta_H + \theta_h - \theta_l) \frac{\varepsilon}{\theta_H - \theta_h} \stackrel{\text{def}}{=} B(\varepsilon). \end{aligned} \quad (4.11)$$

$B(\varepsilon)$ in (4.11) is the bound that we aim to find. Note the bound $B(\varepsilon)$ only has to be valid for “sufficiently small” $\varepsilon > 0$. Assumption (4.8) automatically holds if our “sufficiently small” means at least smaller than the right hand side of (4.8). ■

For example, given a thermostat conforming to this model with parameters $c = 0.01$, $\theta_H = 50$, $\theta_h = 30$, $\theta_l = 20$, we can calculate

$$\begin{aligned} \varepsilon &< \min\left\{\frac{\theta_h - \theta_l}{\theta_h}(\theta_H - \theta_h), \frac{\theta_h - \theta_l}{\theta_H - \theta_l}(\theta_H - \theta_h)\right\} \\ &= \min\left\{\frac{30 - 20}{30}(50 - 30), \frac{30 - 20}{50 - 20}(50 - 30)\right\} \\ &= \frac{20}{3}. \end{aligned} \quad (4.12)$$

Thus for any two initial states v_1^0, v_2^0 such that $|v_1^0 - v_2^0| \leq \frac{20}{3}$, the divergence bound exists. Let $\varepsilon = 1 < \frac{20}{3}$, we have

$$\begin{aligned}
|x_1(t) - x_2(t)| &\leq (2\theta_H + \theta_h - \theta_l) \frac{\varepsilon}{\theta_H - \theta_h} \\
&= (2 \times 50 + 30 - 20) \frac{1}{50 - 30} \\
&= 5.5.
\end{aligned} \tag{4.13}$$

Suppose we have an implementation of the thermostat according to the model, the temperature divergence should not go above $5.5K$ if the initial temperature difference is within $1K$. If some physical trajectories violate this property, we can confidently assert the implementation contains faults. By inspecting which statements are covered in these violative test cases, we may locate the faults by using fault localization tools.

4.3 Evaluation on Ardupilot Autopilot Software

To evaluate our proposed approach, we notice that execution traces preparation approaches must be used along with fault localization tools. There are many families of mainstream fault localization tools (see Section 2). Among them, three families demand large number of randomly sampled and labeled execution traces: program spectrum analysis, statistical analysis, and machine learning based analysis. Typically, a fault localization tool from these three families intakes a large number of labeled execution traces of a program; and outputs a ranked list of program entities in decreasing order of their likelihood of being buggy, aka suspiciousness values. The list, aka *diagnosis*, is then used to guide manual debugging.

Given a diagnosis $D = (d_1, d_2, \dots, d_m)$ and the set F of all buggy entities of the program, we define several diagnosis quality metrics. The *accuracy* metric measures the percentage of buggy entities contained in a diagnosis, i.e. $|\{d_i : d_i \in F\}|/m$. The *recall* metric measures the percentage of reported buggy entities of all buggy entities,

i.e. $|\{d_i : d_i \in F\}|/|F|$. The *latency* metric measures the number of entities one has to examine before reaching the first truly buggy one, assuming the entities are examined in their listed order. The latency metric can be defined as $\min\{i : d_i \in F\}$. A high quality diagnosis shall have high accuracy, high recall, and low latency. To empirically evaluate our proposed execution traces preparation approach, we compare our proposed approach against the **Way-2** approach in preparing execution traces of a commercial-product-grade large control-CPS (note the **Way-1** approach is impractical for most real-world control-CPSs, see Section 1.1). We feed the prepared traces to fault localization tools; and compare the generated diagnoses. Particularly, we compare two aspects. The *effectiveness* aspect concerns the quality, i.e. *accuracy*, *recall*, and *latency* of the diagnoses. The *efficiency* aspect concerns the length of execution traces needed for achieving good quality diagnoses.

Our evaluation shall answer the following *research questions* (RQs):

RQ1 How our proposed approach impacts accuracy?

RQ2 How our proposed approach impacts recall?

RQ3 How our proposed approach impacts latency?

RQ4 How do the trace lengths impact diagnoses quality?

4.3.1 ArduPilot

We choose ArduPilot [16c] as our target control-CPS for evaluation. ArduPilot is an open source platform for UAVs. It is successfully used in many commercial products, including 3DRobotics [16a] and Aeromao [16b], and represents the state-of-the-art of consumer UAV control-CPSs.

The ArduPilot cyber subsystem is mostly written in C++. It consists of over 1,400,000 lines of code from more than 9,600 source files; and its total size is over 380MB. We focus on cyber subsystem modules responsible for waypoint navigation and position/attitude control. The waypoint navigation module breaks the route from source end to destination end into consecutive line segments, the vertices of these line segments are called *way points*. Position/attitude control maneuvers the UAV from one way point to the next-hop way point. These modules alone contain over 3,000 lines of code.

ArduPilot's distribution includes a physical subsystem simulator: the *software in the loop* (SITL) simulator. As this simulator is a well-known and widely used ArduPilot physical subsystem simulator, we reasonably assume it is bug-free. With this simulator serving as the physical subsystem, we can run ArduPilot *emulations*: by connecting the actual authentic/buggy-ArduPilot cyber subsystem to this simulator. We call this emulation set-up *authentic/buggy-ArduPilot SITL emulation set-up*.

Execution physical traces of our emulation are the trajectories of the physical state vector \vec{x} . For ArduPilot UAV, \vec{x} is a 6-dimension-tuple. The 6 dimensions respectively represent the UAV's current x-location (i.e. longitude location), y-location (i.e. latitude location), altitude, and pitch, roll, yaw angles. The valid initial value set \mathcal{I} for \vec{x} is respectively $[0, 111.3\text{km}]$, $[0, 111.3\text{km}]$, $[0.4\text{km}, 0.5\text{km}]$, $[-20^\circ, 20^\circ]$, $[-20^\circ, 20^\circ]$, $[0^\circ, 360^\circ]$ for the 6 dimensions of \vec{x} .

4.3.2 Bug Injection

Following common practice in the field [Foo+15][LH06][Hut+94], we inject bugs into the authentic ArduPilot cyber subsystem to create buggy versions for evaluation. Using injected bugs, we are able to control the type and location of bugs in system, so as to evaluate fault localization results precisely. This decision, however,

Tab. 4.1: Most Representative Residual Bug Types [Nat+13]

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

presents threats to the generalization of our evaluation results, which we discuss in Section 4.3.7.

As our proposed approach mainly deals with residual bugs (i.e., bugs overlooked by best-effort manual debugging, see Section 3.4.2), to make our evaluation as close to real-world scenario as possible, the types of our injected bugs are selected from Tab. 4.1, which lists 12 most common types of residual bugs in the field [Nat+13][DM06]. The specific injected bugs are listed in Tab. 4.2.

By injecting bugs into the authentic ArduPilot cyber subsystem, we create 47 buggy versions of ArduPilot. Among these, 35 versions are 5-bug-versions, i.e. each version contains 5 bugs randomly picked (as per uniform distribution) from Tab. 4.2; and 12 versions are 1-bug-versions, i.e. each version contains respectively one bug listed in Tab. 4.2. By default, all these 47 buggy versions has UAV hovering configuration enabled. By merely disabling the hovering configuration, we create another 47 buggy versions.

We call the combination of a buggy cyber subsystem and the corresponding physical subsystem (or physical subsystem simulator) a *subject*. In total, we have 94 subjects for our evaluation.

Tab. 4.2: Bugs Injected into ArduPilot

Type	Bug Location: Function.Line	Bug Description
WVAV	pos_to_rate_xy.845	“multiply” becomes “divide”
MVAV	clac_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_z.378	an assignment deleted
WPFV	pos_to_rate_xy.798	swapped parameter x, y
MLPA	get_stopping_point_z.287	wrong expression used
MLPA	get_stopping_point_z.289	wrong expression used
WVAV	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

4.3.3 Fault Localization Tools

To mitigate bias introduced by a specific fault localization tool, we employ three classic fault localization tools: *Tarantula* (TA) [JH05], *Crosstab* (CR) [Won+08], and *BP Neural Network* (NN) [WQ09]. The three tools are representatives from three major fault localization tool families, respectively the program spectrum analysis tool family, the statistics analysis tool family, and the machine learning based analysis tool family. Other families of fault localization tools are not selected as they do not heavily depend on large number of labeled execution traces (see Section 2).

In our evaluation, we consider program entities at the level of *basic blocks* (i.e. a block of code that has no branch). Unless otherwise noted, we refer to “basic blocks” simply as “*blocks*” in the following. We also assume a programmer only has the man-power to examine the top 10 blocks in turn in a diagnosis to pinpoint the bugs. That is, we restrict all diagnoses to list only 10 items.

4.3.4 Trials and Diagnosis

Given a subject and an execution traces preparation approach, we call the complete process from preparing execution traces of the subject to getting the 3 diagnoses (respectively using TA, CR, and NN) a *trial*. Our evaluation is empirical and statistical: it runs a Monte Carlo of many trials. To further randomize our Monte Carlo evaluation, we run 4 trials on each given subject: 2 using our proposed execution traces preparation approach, and 2 using the **Way-2** approach.

For a trial using the **Way-2** approach, we randomly sample 4000 initial values (as per uniform distribution) from the valid initial value set \mathcal{I} for ArduPilot UAV physical state vector \vec{x} (see Section 4.3.1 last paragraph). Using the 4000 initial values, we respectively run the subject 4000 times, retrieving 4000 execution traces. For a trial using our proposed approach, we randomly sample $I = 1000$ clusters of $J = 4$ initial states from \mathcal{I} for \vec{x} (see **Step-7** in Section 3.4 on how to uniformly distribute the random samples: the sampling cluster size is $\pm 11.13\text{m}$, $\pm 11.13\text{m}$, $\pm 10\text{m}$, $\pm 5^\circ$, $\pm 5^\circ$, $\pm 10^\circ$ respectively for the 6 dimensions of \vec{x}). Using the $I \times J = 4000$ initial values, we also respectively run the subject 4000 times, retrieving 4000 execution traces.

For each trial, the emulated subject execution time is $T = 120\text{sec}$. The emulated ArduPilot control duty-cycle is 40ms. The physical state vector logging period is every 300ms of emulation time. So every execution trace logs $120 \times 1000/300 = 400$ physical state vector values.

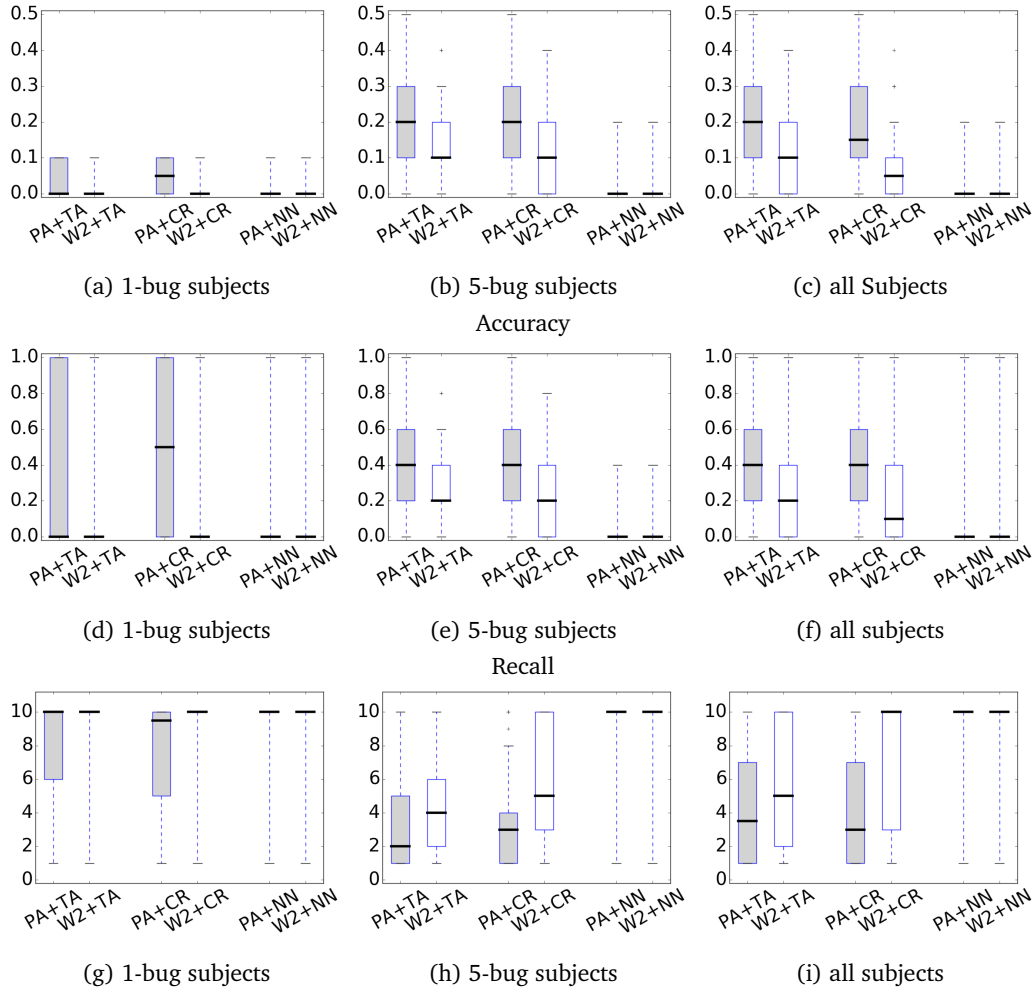
Traces in a trial are then labeled as per **Step 9** in Section 3.4, in case our proposed approach is used; or by checking if they satisfy a set of assertions proposed by a panel of domain experts, in case the **Way-2** approach is used. The panel consists of three domain experts: two have PhD degrees and over 15 and 9 years of experiences in control systems and CPS R&D respectively; two have over 2 years of ArduPilot development experiences. The panel proposes the following assertions.

- A1** Flight's deviation from the straight line linking the current and next waypoints is bounded by 10m.
- A2** Pitch and roll angle fluctuation during flight is bounded by $\pm 52^\circ$.
- A3** Velocity along x-location, y-location, and altitude axes are respectively bounded by $\pm 47.2\text{m/sec}$, $\pm 47.2\text{m/sec}$, $\pm 68.35\text{m/sec}$.
- A4** Angular velocity along pitch, roll, and yaw are respectively bounded by $\pm 1.3\text{rad/sec}$, $\pm 1.3\text{rad/sec}$, $\pm 6.28\text{rad/sec}$.
- A5** Overshoot from a fixed reference point (i.e. the targeted physical state) should decrease along time.

Note the ranges listed in A1~A4 are derived from 267 emulation hours of the authentic-ArduPilot SITL emulation set-up.

Next, the labeled traces are fed to the 3 fault localization tools respectively to produce three diagnoses.

In total, we ran $94 \times 4 = 376$ trials, using $376 \times 4000 = 1504000$ traces from $1504000 \times 120/3600 = 50133.33$ emulation hours of ArduPilot, and produced $94 \times 4 \times 3 = 1128$ fault diagnoses. This gives us $1128/2 = 564$ pairs of diagnoses to compare, where each pair are produced using the same subject and fault localization tool but with different execution traces preparation approaches. These pairs provide a good collection of data points for us to statistically analyze the differences introduced by the execution traces preparation approaches.



Latency. Note when data set size is an even number, median is the average of the middle two data points, hence .5 latency median is possible.

Fig. 4.3: Box-plots [16e] of diagnoses quality metrics. X-axes: execution traces preparation approach and fault localization tool combination; PA: our proposed approach; W2: **Way-2** approach; TA: Tarantula; CR: Crosstab; NN: BP Neural Networks. Y-axes: respective diagnoses quality metric statistics. Bar in each box is the median of the data, box ends are 1st and 3rd quartile of the data, whisker ends are min/max of the data, outliers are dots outside the whiskers.

4.3.5 Results

In this section, we report the results of our evaluations, with the aim of answering the research questions R1~R4 (see the beginning of Section 4.3).

RQ1 (Diagnosis Accuracy): Fig. 4.3(a)~(c) compare the diagnoses accuracy statistics (in form of box-plots [16e]) for 1-bug subjects, 5-bug subjects, and all subjects

Tab. 4.3: Fig. 4.3 PA vs W2 Comparisons Trustworthiness

metric	subjects	TA		CR		NN	
		p-value	es	p-value	es	p-value	es
accuracy	1-bug	8.08e-4	0.59	1.37e-4	0.70	0.35	8.02e-2
	5-bug	1.81e-9	0.73	2.35e-13	0.94	0.50	7.59e-17
	all	3.20e-11	0.56	1.74e-15	0.76	0.45	1.26e-2
recall	1-bug	8.08e-4	0.59	1.37e-4	0.70	0.35	8.02e-2
	5-bug	1.81e-9	0.73	2.35e-13	0.94	0.50	7.59e-17
	all	2.84e-11	0.62	1.05e-15	0.78	0.41	3.46e-2
latency	1-bug	7.76e-3	-0.37	5.25e-4	-0.64	0.26	-1.56e-1
	5-bug	1.51e-4	-0.41	1.14e-11	-0.90	0.34	-5.11e-2
	all	8.94e-6	-0.33	5.90e-14	-0.74	0.24	-6.94e-2

respectively. According to the figures, our proposed execution traces preparation approach significantly raises diagnoses accuracy for the TA and CR fault localization tools.

The figures also show that most trials using NN fault localization tool fail to report any buggy block in the diagnoses, no matter using our proposed approach or the **Way-2** approach to prepare execution traces. A plausible explanation is that neural networks are sensitive to false positive/negative labeling of training sets. Hence NN suits **Way-1** approach more, where traces can have nearly zero false labels. But as explained in Section 1.1, **Way-1** approach is impractical for control-CPSs, hence is not included in our comparison evaluation.

To quantify the trustworthiness of Fig. 4.3’s comparisons, we run the Wilcoxon signed rank test [Coh13] on all PA vs W2 comparisons in Fig. 4.3(a)~(c). This is a paired non-parametric difference test, which estimates the probability, aka *p-value*, that the comparison differences are only by chance. We also compute the Cohen’s d [Woh+00] as *effect size* (es) to estimate the magnitude of the comparison differences.

The “accuracy” rows of Tab. 4.3 report the *p-values* (column **p-value**) and the effect sizes (column **es**) for all PA vs W2 comparisons in Fig. 4.3(a)~(c). The results (*p-value* \ll 0.05) confirm that our proposed execution traces preparation approach

(i.e. PA) significantly improves the TA and CR diagnoses accuracy. Cohen's d statistics ($es > 0.4$) also reveal that the improvements' magnitude on TA and CR diagnoses are medium to large.

*Answer to RQ1: In our evaluation, compared to the **Way-2** approach, our proposed execution traces preparation approach significantly and substantially increases the accuracy of TA and CR fault localization diagnoses.*

RQ2 (Diagnosis Recall): Similarly, Fig. 4.3(d)~(f) compare the diagnoses recall statistics. Following the same analysis workflow, we derive the following answer to RQ2.

*Answer to RQ2: In our evaluation, compared to the **Way-2** approach, our proposed execution traces preparation approach significantly and substantially increases the recall of TA and CR fault localization diagnoses.*

We restrict the length of the diagnoses report to 10 in all the figures we reported above. To mitigate the possible bias introduced by the length of the report, we use Fig. 4.4 to show how the median accuracy, recall, and latency of diagnoses evolve over the length of the diagnoses report. Since the accuracy and recall of NN is constantly poor for both approaches (PA and W2), we exclude the NN result in this figure. We can see the proposed approach (PA) can always achieve a better performance regardless the length of the suspect list, when the length of the suspect list varies from 1 to 15.

RQ3 (Diagnosis Latency): Also similarly, Fig. 4.3(g)~(i) compare the diagnoses latency statistics. Following the same analysis workflow, we derive the following answer to RQ3.

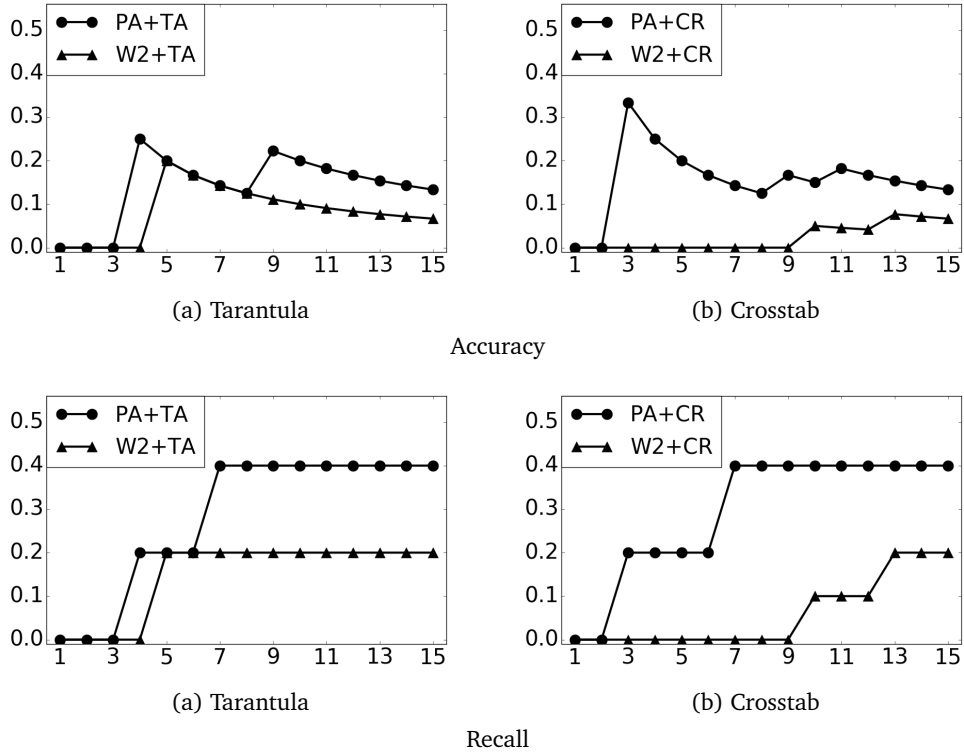


Fig. 4.4: Medians of accuracy and recall over the length of the report. X-axes: length of the report; PA: our proposed approach; W2: **Way-2** approach; TA: Tarantula; CR: Crosstab. Y-axes: respective medians of all subjects.

*Answer to RQ3: In our evaluation, compared to the **Way-2** approach, our proposed execution traces preparation approach significantly and substantially reduces the latency of TA and CR fault localization diagnoses.*

Note the following. First, if a diagnosis has accuracy of 0, i.e., no block reported by the diagnosis is actually buggy, we designate the diagnosis' latency to be 10, as the programmer needs to use up his/her 10 item man-power. Second, in latency comparison, the smaller the latency the better (whereas for accuracy and recall, the bigger the better). Thus for “es” statistics, a more negative magnitude means more improvement. However, for “p-values”, still a p-value $\ll 0.05$ means the comparison is trustworthy.

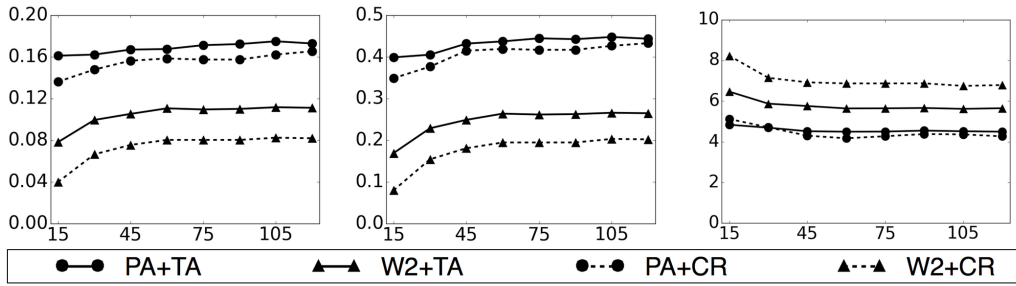


Fig. 4.5: Fault diagnosis quality evolution over execution trace length. X-axes: length of each execution trace (unit: seconds of emulation time). Y-axes: median of accuracy (left figure), recall (middle figure), and latency (right figure) of all subjects.

Tab. 4.4: Fig. 4.5 PA vs W2 Comparisons Trustworthiness

metric	TA		CR	
	p-value	es	p-value	es
median accuracy	5.81e-3	7.78	5.86e-3	7.20
median recall	5.86e-3	7.32	5.66e-3	6.89
median latency	5.81e-3	-5.89	5.86e-3	-6.78

RQ4 (Overall Efficiency): Since the effectiveness of NN is constantly poor for both approaches (PA and W2), suggesting NN may be unsuitable for control-CPS debugging, we exclude the NN data from the overall efficiency study.

Fig. 4.5 shows how the median accuracy, recall, and latency of diagnoses evolve over the length of the execution traces used. Two things are immediately noticeable from the figure. First, medians resulted from our proposed approach are far better than those from **Way-2** approach, even when the execution traces are as short as 15sec. Second, the medians tend to stabilize once the execution traces are longer than 30sec, and the improvement achieved by **Way-2** approach from using longer execution traces is not enough to beat our proposed approach. Furthermore, the trustworthiness of Fig. 4.5 is supported by the p-value and es data listed in Tab. 4.4.

*Answer to RQ4: In our experiment, compared to the **Way-2** approach, our proposed execution traces preparation approach can use shorter execution traces to achieve better quality TA and CR diagnoses.*

4.3.6 Discussion on Failure of BPNN Fault Localization

Method

In our experiment result, a notable phenomenon is that the performance of the BPNN fault localization method is constantly poor for accuracy, recall and latency. As Neural Network is currently one of the most popular machine learning methods, and it has many significant applications in various areas, it is worth to discuss why it fails to localize the bugs in our experiments. In the original paper[WQ09] which proposed this method, it actually produced satisfactory fault localization report, so we believe the concept of BPNN is feasible. However, in our experiment, the labelling of execution traces is not absolutely accurate due to the nature of Cyber-Physical systems, so there are significant numbers of mislabelling execution traces as the training data of the Neural Network. In the original paper, BPNN is used to debug pure cyber systems so the labelling of each test case is almost perfect, and the Neural Network is fed by much higher-quality training data. This may be the main reason of the result difference between our experiment and the original experiment.

There are some other reasons of the poor performance of BPNN fault localization method in our experiment. For example, training a Neural Network is a much more complicated optimization problem than other machine learning models such as linear regression, so it requires a larger data size. The data size is determined by the actual number of layers and neurons in the network. For example, in computer vision, the MNIST dataset[LeC98] is a set of handwritten digits which is used by many researchers to train different models. It contains 60000 training images and 10000 test images. In our experiment, there are only 8000 test cases in each subject, which may be not enough to train a useful model to localize the bugs.

The conclusion is that BPNN-based fault localization method may not be suitable in our experiment, because currently we cannot acquire enough and high-quality labelled execution traces as the data to train the network. However, it is still possible

to locate faults in control CPS by means of other machine learning based methods. Additionally, we aim to find more accurate execution trace labelling methods in our future research.

4.3.7 Threats to Validity

In this section, we outline possible threats to the validity of our study and show how we mitigate them.

Construct: We evaluated the effectiveness of our proposed approach and **Way-2** approach based on fault localization quality. While fault localization is an important application that needs our proposed approach, other automatic debugging applications, e.g., automated program repair, may also benefit from our proposed approach.

Also, we did not evaluate the **Step 4~6** branch of our proposed approach, because i) the validity of this branch is already supported by Theorem 1; ii) most control-CPSs used in real-world cannot satisfy conditions set in **Step 1~3**, hence will not use the **Step 4~6** branch.

Internal: We endeavored to minimize threats to internal validity of our evaluation. We designed the experimental protocol to strictly follow recommended guidelines [AB11] to achieve statistically significant results. Most analysis results show high statistical significance, hence support soundness of the evaluation. We have also reviewed all our source code and tested the implementation on a simple thermostat control-CPS [Che16b] before conducting the evaluations.

External: Main threat to external validity is the representativeness of our evaluation subjects. Bugs used in this study are injected instead of original, and the types of bugs injected are mainly mutation of expressions (like multiplication to division)

instead of logical bugs (like mutating the condition in IF statements). However, the injected bugs are carefully selected from representative residual bugs reported by well recognized studies [Nat+13], so as to make the subjects as close to real-world cases as possible. This helps reducing the threat.

Also, fault localization tools used in the evaluation play a major role in the evaluation results. To reduce the threat of bias caused by a specific tool, we used three classic tools from different families and compared their results.

Due to time limits, the evaluation was conducted only on the ArduPilot control-CPS. More evaluations using other real-world control-CPSs as subjects would help us further reduce the threat.

Conclusions and Suggestions for Future Research

5.1 Conclusion

Control-CPS is growing rapidly nowadays. Autonomous cars and UAVs undoubtedly change many people's lifestyle in the recent years. Thanks to the development in the areas of modern control theory, signal processing and sensing technology, we are able to design very advanced models with countless useful functionalities. As the cyber subsystems become larger and more complex, testing and debugging become more and more important in the development cycle to guarantee the implementation indeed meets the design requirements. However, there are only a few works focusing on testing and debugging control-CPS cyber subsystems and the common practice is still quite subjective and brute-force.

By exploiting the state-of-the-art hybrid systems modeling and stability theories, in this research, we propose a principled and systematic approach to prepare execution traces for control-CPS cyber subsystem fault localization. We evaluated our proposed approach on a commercial-product-grade large control-CPS platform. The evaluation results show that that our proposed approach significantly improves the existing best-effort approach.

5.2 Suggestions for Future Research

In this research, we propose a new approach to prepare execution traces for control-CPS cyber subsystem fault localization. This new approach can be extended in various dimensions for future research.

First, in this research, we pick the confidence percentile $\alpha = 95\%$ for control-CPS with unknown hybrid automaton model. The confidence percentile will determine the number of traces labelled as “faulty” and it will influence the final fault localization result. It is possible to find a systematic method choosing the confidence percentile by further studying the control-CPS. For example, we can refer to the historical data of similar systems if such data is available.

Second, as mentioned in Section 4.3.7, the evaluation was conducted only on the ArduPilot control-CPS. Experiments on different platforms can be conducted in the future to support our proposed approach. We have explored some existing open source control-CPS libraries like ROS and AUTOSAR, and evaluations on these platforms can be a part of future work.

Last but not least, we suggest further theoretical research on control-CPS, especially dealing with complicated systems with unclear system dynamics. Specifically, we wonder whether it is possible to find the divergence bound without clearly knowing the model. For example, in the previous example of thermostat, the divergence bound is

$$(2\theta_H + \theta_h - \theta_l) \frac{\varepsilon}{\theta_H - \theta_h} \tag{5.1}$$

It turns out that the divergence bound is independent of c , which means we can find the bound without knowing c . It shows the possibility that we can get the divergence bound even for more complicated system with unknown parameters. Actually similar ideas have been proposed in some other areas such as adaptive control [ÅW13] and probabilistic model checking [KNP02]. It is possible to migrate the ideas to help us improve the accuracy of labelling physical traces.

Bibliography

- [16a] *3D Robotics*. <https://3dr.com/>, 2016 (cit. on p. 44).
- [16b] *Aeromao*. <http://www.aeromao.com/>, 2016 (cit. on p. 44).
- [16c] *ArduPilot Mega -Home*. <http://www.ardupilot.co.uk>, 2016 (cit. on pp. 1, 44).
- [16d] *Boundary (topology)*. [https://en.wikipedia.org/wiki/Boundary%5F\(topology\)](https://en.wikipedia.org/wiki/Boundary%5F(topology)). 2016 (cit. on p. 23).
- [16e] *Box plot*. <https://en.wikipedia.org/wiki/Box%5Fplot>. 2016 (cit. on p. 50).
- [AB11] Andrea Arcuri and Lionel Briand. „A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering“. In: *Proc. of the 33rd Intl. Conf. on Software Engineering (ICSE'11)* (2011), pp. 1–10 (cit. on p. 56).
- [AH90] Hiralal Agrawal and Joseph R. Horgan. „Dynamic Program Slicing“. In: *Proc. of the ACM SIGPLAN'90 Conf. on Programming Language Design and Implementation* (June 1990), pp. 246–256 (cit. on p. 15).
- [AHH93] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. „Automatic Symbolic Verification of Embedded Systems“. In: *RTSS* (1993), pp. 2–11 (cit. on pp. 3, 10).
- [AHH96] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. „Automatic Symbolic Verification of Embedded Systems“. In: *IEEE Trans. on Software Engineering* 22(3):181-201 (Mar. 1996) (cit. on pp. 3, 22, 23).
- [Ari06] Julien Arino. *Fundamental Theory of Ordinary Differential Equations*. Lecture Notes of Dept. of Mathematics. Univ. of Manitoba, Fall 2006 (cit. on p. 26).
- [ÅW13] Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013 (cit. on p. 61).
- [B L+16] 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* (2016), pp. 177–188 (cit. on p. 15).
- [BE04] Yuriy Brun and Michael D. Ernst. „Finding Latent Code Errors via Machine Learning over Program Executions“. In: *Proc. of the 26th Intl. Conf. on Software Engineering* (May 2004), pp. 480–490 (cit. on p. 15).

- [Bea08] Randal Beard. „Quadrotor Dynamics and Control Rev 0.1“. In: (2008) (cit. on p. 9).
- [BLL07] Lionel C. Briand, Yvan Labiche, and Xuetao Liu. „Using Machine Learning to Support Debugging with Tarantula“. In: *Proc. of the 18th IEEE Intl. Symp. on Software Reliability* (Nov. 2007), pp. 137–146 (cit. on p. 15).
- [Bro91] William L. Brogan. *Modern Control Theory (3rd Ed.)* Prentice Hall, 1991 (cit. on p. 2).
- [BW09] Frank Burden and Dave Winkler. „Bayesian regularization of neural networks“. In: *Artificial Neural Networks: Methods and Applications* (2009), pp. 23–42 (cit. on p. 19).
- [Cel+08] Peggy Cellier, Mireille Ducasse, Sebastien Ferre, and Olivier Ridoux. „Formal Concept Analysis Enhances Fault Localization in Software“. In: *Proc. of the 4th Intl. Conf. on Formal Concept Analysis* (Feb. 2008), pp. 273–288 (cit. on p. 15).
- [Che+14] Ziqiang Chen, Feng Lin, Caisheng Wang, Yi Le Wang, and Min Xu. „Active diagnosability of discrete event systems and its application to battery fault diagnosis“. In: *IEEE Transactions on control systems technology* 22.5 (2014), pp. 1892–1898 (cit. on p. 14).
- [Che16a] Yao Chen. *A Note on Hybrid Systems Traces Distances*. <http://www.comp.polyu.edu.hk/%7Eecsqwang/research/techreports.html>. 2016 (cit. on p. 26).
- [Che16b] Yao Chen. *Hybrid Automata Tutorial: Analysis of Thermostat Hybrid Automaton*. <http://www.comp.polyu.edu.hk/%7Eecsqwang/research/tutorials.html>. 2016 (cit. on pp. 4, 56).
- [Coh13] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 2013 (cit. on p. 51).
- [CW84] Edward Y. Chow and Alan S. Willsky. „Analytical Redundancy and the Design of Robust Failure Detection Systems“. In: *IEEE Trans. on Automatic Control* AC-29(7):603-614 (1984) (cit. on p. 13).
- [DLZ05] Valentin Dallmeier, Christian Lindig, and Andreas Zeller. „Lightweight Defect Localization for Java“. In: *Proc. of the 19th European Conf. on Objective-Oriented Programming* (July 2005), pp. 528–550 (cit. on p. 15).
- [DM06] J.A. Duraes and H. Madeira. „Emulation of Software Faults: A Field Data Study and A Practical Approach“. In: *IEEE Trans. on Software Engineering* 32(11):849-867 (Nov. 2006) (cit. on p. 46).
- [DPS97] Richard A. DeMillo, Hsin Pan, and Eugene H. Spafford. „Failure and Fault Analysis for Software Debugging“. In: *Proc. of the 21st Intl. Computer Software and Applications Conf.* (Aug. 1997), pp. 515–521 (cit. on p. 20).
- [EG14] Christian von Essen and Dimitra Giannakopoulou. „Analyzing the next generation airborne collision avoidance system“. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2014), pp. 620–635 (cit. on p. 1).
- [Ern+01] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. „Dynamically Discovering Likely Program Invariants to Support Program Evolution“. In: *IEEE Trans. on Software Engineering* 27(2):99-123 (Feb. 2001) (cit. on p. 15).

- [Foo+15] King Chun Foo, Zhen Ming (Jack) Jiang, Bram Adams, et al. „An Industrial Case Study on the Automated Detection of Performance Regressions in Heterogeneous Environments“. In: *Proc. of the 37th Intl. Conf. on Software Engineering - Vol. 2 (ICSE'15)* (2015), pp. 159–168 (cit. on p. 45).
- [FPE93] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. „Feedback Control of Dynamic Systems“. In: Addison-Wesley Publishing Company, Nov. 1993 (cit. on p. 2).
- [Gao+15a] Zhiwei Gao et al. „A Survey of Fault Diagnosis and Fault-Tolerant Techniques Part I: Fault Diagnosis with Model-Based and Signal-Based Approaches“. In: *IEEE Trans. on Ind. Electronics* ((preprint) 2015) (cit. on p. 13).
- [Gao+15b] Zhiwei Gao et al. „A Survey of Fault Diagnosis and Fault-Tolerant Techniques Part II: Fault Diagnosis with Knowledge-Based and Hybrid/Active Approaches“. In: *IEEE Trans. on Ind. Electronics* ((preprint) 2015) (cit. on p. 13).
- [Ger88] Janos J Gertler. „Survey of model-based failure detection and isolation in complex plants“. In: *IEEE Control systems magazine* 8.6 (1988), pp. 3–11 (cit. on p. 13).
- [Ger98] Janos Gertler. *Fault Detection and Diagnosis in Engineering Systems*. CRC Press, May 1998 (cit. on p. 13).
- [GST12] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton Univ. Press, 2012 (cit. on pp. 10, 20, 30).
- [Har+00] Mary Jean Harrold, Gregg Rothermel, Kent Sayre, Rui Wu, and Liu Yi. „An Empirical Investigation of the Relationship between Spectra Differences and Regression Faults“. In: *Journal of Software Testing, Verification and Reliability* 10(3):171-194 (Sept. 2000) (cit. on p. 15).
- [HC10] Naira Hovakimyan and Chengyu Cao. *\mathcal{L}_1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation*. SIAM, 2010 (cit. on p. 13).
- [HD14] Liu Hong and Jaspreet Singh Dhupia. „A time domain approach to diagnose gearbox fault based on measured vibration signals“. In: *Journal of Sound and Vibration* 333.7 (2014), pp. 2164–2180 (cit. on p. 14).
- [Hon+15] Shin Hong, Byeongcheol Lee, Taehoon Kwak, et al. „Mutation-Based Fault Localization for Real-World Multilingual Programs (T)“. In: *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on* (2015), pp. 464–475 (cit. on p. 15).
- [HSD12] Morris W. Hirsch, Stephen Smale, and Robert L. Devaney. *Differential Equations, Dynamic Systems, and an Introduction to Chaos (3rd Ed.)* Academic Press, Mar. 2012 (cit. on p. 33).
- [HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Apr. 1979 (cit. on p. 3).
- [Hut+94] Monica Hutchins, Herb Foster, Tarak Goradia, and Thomas Ostrand. „Experiments of the Effectiveness of Dataflow- and Controlflow-based Test Adequacy Criteria“. In: *Proc. of the 16th Intl. Conf. on Software Engineering (ICSE'94)* (1994), pp. 191–200 (cit. on p. 45).

- [JH05] James A Jones and Mary Jean Harrold. „Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique“. In: *Proc. of the 20th IEEE/ACM Intl. Conf. on Automated Software Engineering* (2005), pp. 273–282 (cit. on pp. 15, 16, 47).
- [KL88] Bogdan Korel and Janusz Laski. „Dynamic Program Slicing“. In: *Information Processing Letters* 29(3):155-163 (Oct. 1988) (cit. on p. 15).
- [KNP02] Marta Kwiatkowska, Gethin Norman, and David Parker. „PRISM: Probabilistic symbolic model checker“. In: *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer. 2002, pp. 200–204 (cit. on p. 61).
- [LeC98] Yann LeCun. „The MNIST database of handwritten digits“. In: <http://yann.lecun.com/exdb/mnist/> (1998) (cit. on p. 55).
- [LH06] Chao Liu and Jiawei Han. „Failure Proximity: A Fault Localization-based Approach“. In: *Proc. of the 14th ACM SIGSOFT Intl. Symp. on Foundations of Software Engineering (SIGSOFT'06/FSE-14)* (2006), pp. 46–56 (cit. on p. 45).
- [Li+06] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. „CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code“. In: *IEEE Trans. on Software Engineering* 32(3):176-192 (2006) (cit. on p. 15).
- [Lib+05] Ben Liblit, Mayur Naik, Alice X. Zheng, Alex Aiken, and Michael I. Jordan. „Scalable Statistical Bug Isolation“. In: *Proc. of the 2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation* (June 2005), pp. 15–26 (cit. on p. 15).
- [Liu+06] Chao Liu, Long Fei, Xifeng Yan, Jiawei Han, and Samuel P. Midkiff. „Statistical Debugging: A Hypothesis Testing-Based Approach“. In: *IEEE Trans. on Software Engineering* 32(10):831-848 (Oct. 2006) (cit. on p. 15).
- [Luc+14] Lucia Lucia, David Lo, Lingxiao Jiang, Ferdian Thung, and Aditya Budi. „Extended comprehensive study of association measures for fault localization“. In: *Journal of Software: Evolution and Process* 26.2 (2014), pp. 172–219 (cit. on p. 15).
- [LW12] Eugene Lavretsky and Kevin Wise. *Robust and Adaptive Control: With Aerospace Applications, Advanced Textbooks in Control and Signal Processing*. Springer, Nov. 2012 (cit. on p. 13).
- [LW87] James R. Lyle and Mark Weiser. „Automatic Program Bug Location by Program Slicing“. In: *Proc. of the 2nd Intl. Conf. on Computer and Applications* (June 1987), pp. 877–883 (cit. on p. 15).
- [MB99] P. J. Mosterman and G. Biswas. „Diagnosis of Continuous Valued Systems in Transient Operating Regions“. In: *IEEE Trans. on Systems, Man, and Cybernetics -Part A* 29(6):554-565 (1999) (cit. on p. 20).
- [Mir79] Leonid Alekseevich Mironovskii. „Functional diagnosis of linear dynamic systems“. In: *Avtomatika i Telemekhanika* 8 (1979), pp. 120–128 (cit. on p. 13).
- [MRV07] Mano Ram Maurya, R Rengaswamy, and V Venkatasubramanian. „A signed directed graph and qualitative trend analysis-based framework for incipient fault diagnosis“. In: *Chemical Engineering Research and Design* 85.10 (2007), pp. 1407–1422 (cit. on p. 14).

- [MSW00] Cristinel Mateis, Markus Stumptner, and Franz Wotawa. „Modeling Java Programs for Diagnosis“. In: *Proc. of the 14th European Conf. on Artificial Intelligence* (Aug. 2000), pp. 171–175 (cit. on p. 20).
- [Mur+15] Anitha Murugesan, Sanjai Rayadurgam, Michael W Whalen, and Mats PE Heimdahl. „Design Considerations for Modeling Modes in Cyber–Physical Systems“. In: *IEEE Design & Test* 32.5 (2015), pp. 66–73 (cit. on p. 1).
- [Nat+13] Roberto Natella, Domenico Cotroneo, Joao A. Duraes, and Henrique S. Madeira. „On Fault Representativeness of Software Fault Injection“. In: *IEEE Trans. on Software Engineering* 39(1):80-96 (Jan. 2013) (cit. on pp. 33, 46, 57).
- [NB07] S. Narasimhan and G. Biswas. „Model-Based Diagnosis of Hybrid Systems“. In: *IEEE Trans. on Systems, Man and Cybernetics -Part A* 37(3):348-361 (2007) (cit. on p. 20).
- [Ors+06] Alessandro Orso, Shrinivas Joshi, Martin Burger, and Andreas Zeller. „Isolating relevant component interactions with JINSI“. In: *Proceedings of the 2006 international workshop on Dynamic systems analysis* (2006), pp. 3–10 (cit. on p. 15).
- [PFE00] Ron J. Patton, Paul M. Frank, and Robert N. Clark (Eds.) *Issues of Fault Diagnosis for Dynamic Systems*. Springer, 2000 (cit. on p. 13).
- [RR03] Manos Renieris and Steven P. Reiss. „Fault Localization with Nearest Neighbor Queries“. In: *Proc. of the 18th IEEE Intl. Conf. on Automated Software Engineering (ASE)* (Oct. 2003), pp. 30–39 (cit. on p. 15).
- [Sam+08] Paraskevi A Samara, George N Fouskitakis, John S Sakellariou, and Spilios D Fassois. „A statistical method for the detection of sensor abrupt faults in aircraft control systems“. In: *IEEE Transactions on Control Systems Technology* 16.4 (2008), pp. 789–798 (cit. on p. 14).
- [Sha+08] Lui Sha et al. „Cyber-Physical Systems: A New Frontier“. In: *IEEE SUTC* (2008), pp. 1–9 (cit. on p. 1).
- [She+14] Nida Sheibat-Othman, Nassim Laouti, Jean-Pierre Valour, and Sami Othman. „Support vector machines combined to observers for fault diagnosis in chemical reactors“. In: *The Canadian Journal of Chemical Engineering* 92.4 (2014), pp. 685–695 (cit. on p. 14).
- [SL91] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice Hall, 1991 (cit. on p. 2).
- [Tab09] Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009 (cit. on p. 10).
- [WD09] W. Eric Wong and Vidroha Debroy. „A Survey of Software Fault Localization“. In: *Tech. Report UTDCS-45-09* (Nov. 2009) (cit. on p. 1).
- [Wei09] Mark Weiser. „Program Slicing“. In: *IEEE Trans. on Software Engineering* SE-10(4):352-357 (July 2009) (cit. on p. 15).
- [Woh+00] Claes Wohlin, Per Runeson, Martin Host, et al. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000 (cit. on p. 51).

- [Won+08] W. Eric Wong, Tingting Wei, Yu Qi, and Lei Zhao. „A Crosstab-Based Statistical Method for Effective Fault Localization“. In: *Proc. of the 1st Intl. Conf. on Software Testing, Verification and Validation* (Apr. 2008), pp. 42–51 (cit. on pp. 15, 16, 47).
- [WQ09] W. Eric Wong and Yu Qi. „BP Neural Network-Based Effective Fault Localization“. In: *Intl. Journal of Software Engineering and Knowledge Engineering* 19(4):573-597 (2009) (cit. on pp. 15, 18, 20, 47, 55).
- [WSM02] Franz Wotawa, Markus Stumptner, and Wolfgang Mayer. „Model-Based Debugging or How to Diagnose Programs Automatically“. In: *Proc. of the 15th Intl. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Developments in Applied Artificial Intelligence* (June 2002), pp. 746–757 (cit. on p. 20).
- [Z+13] Yu Zhang, Chris Bingham, Michael Gallimore, et al. „Fault detection and diagnosis based on extensions of PCA“. In: *Advances in Military Technology* 8.2 (2013), pp. 27–41 (cit. on p. 14).
- [Zel02] Andreas Zeller. „Isolating Cause-Effect Chains from Computer Programs“. In: *Proc. of the 10th ACM SIGSOFT Symp. on Foundations of Software Engineering* (Nov. 2002), pp. 1–10 (cit. on p. 15).
- [ZGG06] Xiangyu Zhang, Neelam Gupta, and Rajiv Gupta. „Locating Faults through Automated Predicate Switching“. In: *Proc. of the 28th Intl. Conf. on Software Engineering* (May 2006), pp. 272–281 (cit. on p. 15).
- [Zh02] Andreas Zeller and Ralf hildebrandt. „Simplifying and Isolating Failure-Inducing Input“. In: *IEEE Trans. on Software Engineering* 28(2):183-200 (Feb. 2002) (cit. on p. 15).
- [Zha+05] Feng Zhao, Xenofon Koutsoukos, Horst Haussecker, Jim Reich, and Patric Cheung. „Monitoring and Fault Diagnosis of Hybrid Systems“. In: *IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics* 35(6):1225-1240 (Dec. 2005) (cit. on p. 20).
- [Zhe+15] Xi Zheng, Christine Julien, Miryung Kim, and Sarfraz Khurshid. „Perceptions on the state of the art in verification and validation in cyber-physical systems“. In: (2015) (cit. on p. 1).
- [ZXC16] Zhi Quan Zhou, Shaowen Xiang, and Tsong Yueh Chen. „Metamorphic Testing for Software Quality Assessment: A Study of Search Engines“. In: *IEEE Trans. on Software Engineering* 42(3):264-284 (Mar. 2016) (cit. on p. 20).

