# Finite-state transition models for path planning, cyber security, and control of industrial control systems

by

Kalana Pothuwila

A Dissertation

In

Mechanical Engineering

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy

Approved

Dr. Jordan M. Berg

Committee Chairman

Dr. Beibei Ren

Dr. Brian Nutter

Dr. Ismael De-Farias

Mark Sheridan
Dean of the Graduate School

August, 2017

TABLE OF CONTENTS

## Abstract

This dissertation describes the use of finite-state transition representations for path planning, cyber security, and closed-loop control of networked industrial control systems. In particular, we consider systems consisting of physical components that are controlled by networked digital devices, such as programmable logic controllers (PLCs). The physical components are typically described in continuous time using differential equations, while the communication and control components operate in discrete time. Such combinations of continuous and discrete elements are called "hybrid systems" Although there is a substantial body of literature for analyzing either continuous or discrete systems, the techniques applicable to one are generally not compatible with the other, making analysis and control of hybrid systems a challenging task. The approach taken here is to approximate the continuous dynamics by finite-state, discrete-time transition models. These can be naturally integrated with the digital control and communication infrastructure, and the resulting unified finite-state transition models can be analyzed using powerful algorithmic tools. In this work, the tasks of interest are 1) planning open-loop set-point changes of the system state, 2) detecting and neutralizing harmful control actions, and 3) implementing closed-loop control for uncertainty and disturbance rejection.

This work contains two original contributions. The first is an extension of existing methods for modeling continuous-time systems as finite-state transition systems. For existing approaches, the continuous-time system can have only a single, globally asymptotically stable, equilibrium point. That is, there must be a single equilibrium point, whose basin of attraction must comprise the entire state space. The work presented in this dissertation considers the modeling of continuous-time systems in which almost every point of the state space is contained within the basin of attraction of an asymptotically stable equilibrium point. While still not completely general, this extension allows treatment of systems with multiple equilibrium points. This extension is demonstrated on the classic inverted pendulum on a cart (IPC), including path planning and closed-loop control capabilities. The second contribution is a cyber security architecture for networked industrial control systems that significantly improves upon the current use of standard packet monitoring

approaches. Packet monitoring allows the detection and filtering of potentially dangerous commands. Standard packet monitoring methods have been developed in the context of information technology systems, where certain commands may be used as signatures for cyber attacks. However for physical systems, commands that are harmless or normal in some circumstances can cause severe damage to the system in others. The new packet monitoring system uses a finite-state transition system model to interpret system commands inputs in the context of the current system configuration.

# LIST OF FIGURES

# CHAPTER 1
# INTRODUCTION

This dissertation describes the use of finite-state transition system (FSTS) approximations for path planning, cyber security, and closed-loop control of networked industrial control systems. Roughly speaking, an FSTS consists of 1) a finite list of states that catalogs the only possible configurations that an FSTS can take, 2) a list of transitions that move the FSTS from one state to another, 3) a "label" for each transition that indexes the command action that will cause the transition to occur, and 4) an output function that maps each state to a vector-valued output. In particular, we use approximate FSTS models to represent systems consisting of physical components that are controlled by networked digital devices, such as programmable logic controllers (PLCs). The physical components themselves typically evolve in continuous time, and their dynamics are commonly modeled using differential equations. The communication and control components operate in discrete time, taking meaningful values only at multiples of some underlying "sampling time." Systems combining continuous and discrete elements are called "hybrid systems." Although there is a substantial body of literature for analyzing either continuous or discrete systems, the techniques applicable to one are generally not compatible with the other, making analysis and control of hybrid systems a challenging task. A discrete-time system can be modeled exactly by an FSTS, but an FSTS cannot exactly capture the behavior of a continuous-time system. The options for modeling hybrid systems include representing the discrete elements in a continuous-time framework, finding a new framework for describing both the discrete and continuous elements, or approximating the continuous dynamics by a finite-state model. The last approach is the one taken here, namely to approximate the continuous dynamics by an FSTS. That is, the exact FSTS description of the digital control and communication structure is augmented by an approximate FSTS model of the continuous dynamics. The resulting unified finite-state transition model can be analyzed using powerful algorithmic tools. In this work, the tasks of interest are 1) planning open-loop set-point changes of the system state, 2) detecting and neutralizing harmful control actions, and 3)

implementing closed-loop control for uncertainty and disturbance rejection.

This work contains two original contributions. The first is an extension of existing methods for modeling continuous-time systems as finite-state transition systems. For existing approaches, such as [13], the continuous-time system must have a single, globally asymptotically stable, equilibrium point. That is, there must be one and only one equilibrium point, whose basin of attraction must comprise the entire state space. The work presented in this dissertation considers the modeling of continuous-time systems in which almost every point of the state space is contained within the basin of attraction of some asymptotically stable equilibrium point, but not necessarily the same one. While still not completely general, this extension allows treatment of systems with multiple equilibrium points, allowing, for example, transitions between multiple, asymptotically stable, operating conditions.

The second contribution is a cyber security architecture for networked industrial control systems that significantly improves upon the current approach to ICS network packet monitoring [12]. Packet monitoring allows the detection and filtering of potentially dangerous commands, such as port scans or detailed configuration queries. Standard packet monitoring methods have been developed in the context of information technology systems, where certain commands may be used as signatures for cyber attacks. However for physical systems, commands that are harmless or normal in some circumstances can cause severe damage to the system in others. Under standard, context-free, packet monitoring approaches, these commands must be allowed. The new packet monitoring system uses an FSTS model to interpret system commands in the context of the current system configuration. Commands are accepted in circumstances where they are normal, and rejected or flagged in circumstances where they are inimical.

The remainder of this work is organized as follows:

Chapter 2 reviews the basic concepts and nomenclature of FSTS, in particular the notions of similarity, bisimilarity, and approximate bisimilarity. Essentially, two FSTS are bisimilar if they produce the same output sequence given analogous initial states and identical command sequences. This section concludes with the definition of an $\varepsilon$-approximate bisimulation of a continuous-time system and a survey of the relevant body of literature [5, 8, 13, 20, 21].

Chapter 3 begins by reviewing the results of [13] in detail. This paper presents an algorithm for constructing an approximate FSTS model of a continuous time system. The algorithm of [13] requires that the system has a single, globally asymptotically stable, equilibrium point. This requirement is prohibitively restrictive. Chapter 5 proceeds to extend this algorithm to cases where the state space is partitioned into regions of attraction of a finite number of equilibrium points. By assumption, almost every state is contained in exactly one of these basins. The only exceptions are the boundary points of the basins, which by assumption form a set of measure zero. By their nature, trajectories beginning near a boundary point will be sensitive to small perturbations in their initial position, especially for low-amplitude control actions. The underlying principle of the approach taken here is to increase the resolution of the state-space discretization near the basin boundaries, and to disallow control actions that have an ambiguous outcome. Finally, Chapter 5 presents a pre-processing step by which the initial dynamics may be modified to create equilibria at desired operating points, and by which the equilibrium structure of the system may be altered to satisfy the assumed requirements.

Section 5.2 presents simulations and experiments demonstrating the new algorithm on the classic inverted pendulum on a cart (IPC). Once the approximate FSTS is obtained, powerful tools from the software engineering community can be employed to execute various functions, including path planning and closed-loop control [4, 15]. These capabilities are demonstrated on the IPC system as well, by swinging the pendulum up to its unstable equilibrium point and maintaining it there.

Chapter 4 presents a state-aware packet-monitoring system that builds upon FSTS models. Several researchers have approached this problem in a theoretical optimization framework [3, 14]. By comparison, the approach presented here is heuristic, but guarantees meaningful security in a well-defined sense. The notion of security used in this section is essentially developed with packet sniffers point of view and purpose . Under the assumption that the attacker is not not hiding in the noise, the state-aware packet sniffer is shown to guarantee security in this sense. The state-aware packet monitor is demonstrated using the IPC under various system failures and cyber attacks.

Finally, Chapter 6 and 7 summarizes the conclusions of this work, and presents directions for future research.

# CHAPTER 2
# FINITE-STATE TRANSITION MODELS OF CONTINUOUS SYSTEMS

The technical contributions of this dissertation center on the modeling of continuous-time dynamical systems by finite-state transition systems. This chapter begins by reviewing the relevant mathematical concepts of simulation, bisimulation, and approximate bisimulation. Several simple examples are presented to provide concrete examples of these abstract notions. Next, this chapter presents a review of existing results for creating finite-state transition models of continuous-time systems for which the entire state space is the basin of attraction for a single equilibrium point. Finally, this chapter presents one of the main original contributions of this dissertation, namely an algorithm for creating finite-state transition models for continuous-time systems in which almost every point in the state space is contained within the basin of attraction of a stable equilibrium, where there may be finitely many such equilibrium points.

## 2.1 Transition Systems

The concept of transition system has been introduced to describe behavior of discrete systems in computer science. Formally a transition system can be defined as in Definition: 2.1.1. It's a simple example of a transition system.An example is shown in Figure 2.1. There the transition system $T_a$ is a tuple. The set of Q is the set of states of the system. In general $Q$ can be finite or infinite. The set of possible transitions is denoted by the set $\rightarrow \subseteq Q \times Q$. A labeled transition system is shown on Definition:2.1.2. In addition a labeled transition system has the set of labels $L$. It's important when we want to make a distinction between a transition that start and end in the same state but is a result of different labels or actions. $L$ also can be infinite or finite. If both sets $Q$ and $L$ are finite the transition system is called a finite transition system.

**Definition 2.1.1.** A transition system is a pair $T_a = (Q, \rightarrow)$

$Q-$ set of finite possible states

$\rightarrow Q \times Q$ state transitions

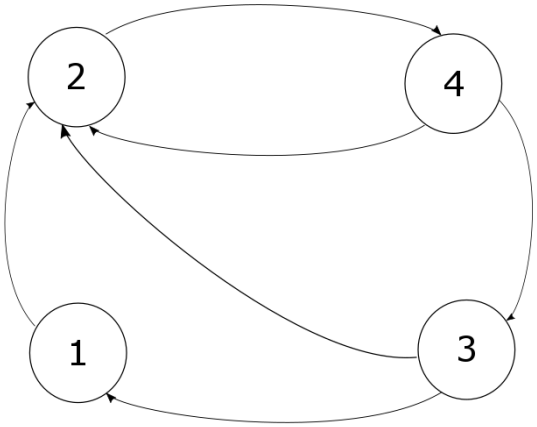


Figure 2.1. A simple transition system

**Definition 2.1.2.** A labeled transition system is a tuple $T_b = (Q, L, \rightarrow)$

$Q-$ set of finite possible states

$L-$ set of labels

$\rightarrow Q \times L \times Q$ set of state transitions

## 2.2 Simulation

An important idea we need to introduce before we talk about simulation is a binary relation. A binary relation $R$ on a set $S$, is a collection of ordered pairs of elements of $S$. In other words $R \subseteq S \times S$.

Now lets consider a transition system $T_b$ as defined in Definition: 2.1.2 and a binary relation $R \subseteq Q \times Q$. The relation $R$ is a simulation relation if for every $(q_1, q_2) \in R$, for every $l \in L$ and for every $p_1 \in Q$

$q_1 \xrightarrow{l} p_1$ implies existence of $q_2 \xrightarrow{l} p_2$ such that $(p_1, p_2) \in R$.

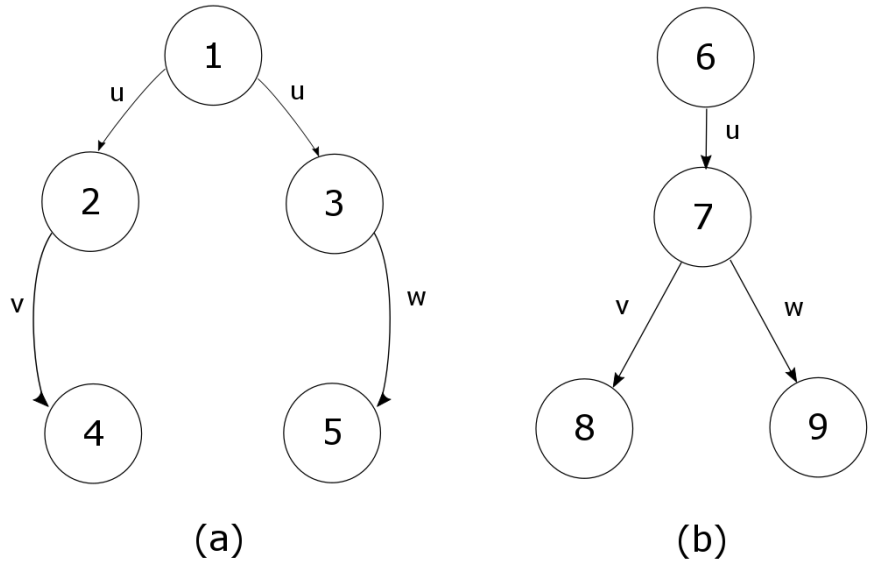


Figure 2.2. Simulation example.The transition system b) can the transition system simulate a) with relation R. $\{(1,6),(2,7),(3,7),(4,8),(5,9)\} = R$

The Figure 2.2 shows an example of simulation with a relation R,$\{(1,6),(2,7),(3,7),(4,8),(5,9)\} = R$.

## 2.3 Bisimulation

Consider again, a transition system $T_b$ as defined in Definition: 2.1.2 and a binary relation $R \subseteq Q \times Q$. The relation $R$ is a simulation relation if for every $(q_1, q_2) \in R$, for every $\in L$ and for every $p_2 \in Q$

$q_1 \xrightarrow{l} p_1$ implies existence of $q_2 \xrightarrow{l} p_2$ such that $(p_1, p_2) \in R$.

and

$q_2 \xrightarrow{l} p_2$ implies existence of $q_1 \xrightarrow{l} p_1$ such that $(p_1, p_2) \in R$.

The Figure 2.3 shows an example of bisimulation with a relation R, $\{(2,6),(4,6),(1,5),(3,5)\} = R$.

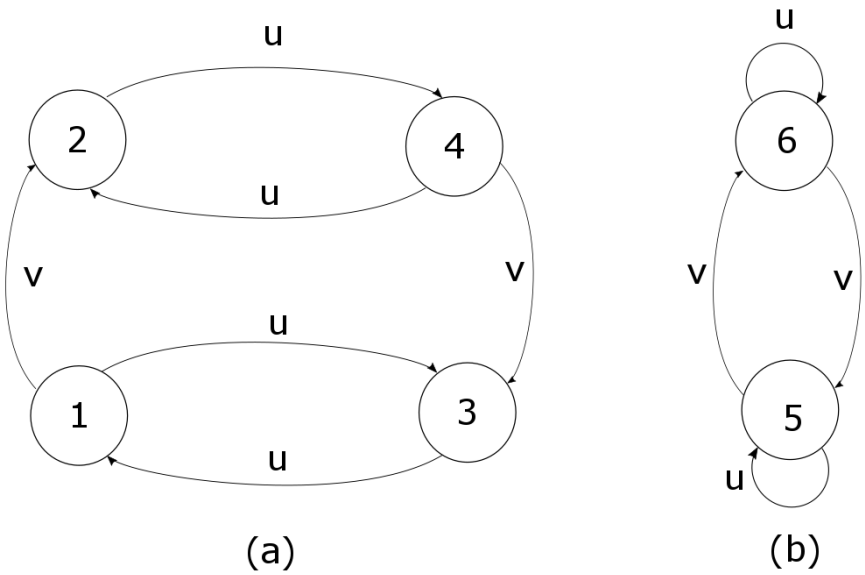


Figure 2.3. Bisimulation example. Two labeled transition systems simulating each other with relation R. $\{(2,6),(4,6),(1,5),(3,5)\} = R$.

## 2.4 Transition System Models of Continuous Dynamics

Continuous dynamics are modeled with differential equations. Any given model is not an exact replica of the real physical system. However, with in reasonable assumptions differential equations model systems quite accurately. The basic concept of modeling/abstractions is preserving some properties of interest while ignoring details less important. In this chapter we discuss how to make abstractions for continuous dynamical systems with a transition system.

## 2.5 Systems with a Single Globally Asymptotically Stable Equilibrium

A dynamical system can be described in differential equation form as shown in Equation 2.1.

$$\dot{X} = F(X, U) \tag{2.1}$$

The details of the dynamical system governed by the Equation 2.1 can be described with the description in Definition 2.5.1.

**Definition 2.5.1.** A control system described by the following set $\Sigma = (\mathbb{R}^n, U, F)$

- $\mathbb{R}^n$ is the state space
- $U \in \mathbb{R}^m$ is the input space
- $F : \mathbb{R}^n \to \mathbb{R}^n$

Let's broaden the definition of Transition systems introduced in Chapter 2 to include details of a control system.

**Definition 2.5.2.** Finite state transition system is a tuple $T = (Q, L, \to, Q^0)$
$Q-$ State space
$L-$ Input space
$\to Q\ X\ Q$ state transitions
$Q^0-$ set of possible initial states

A control system can be put into the transition system form without any discretization with almost no harm. This form is useful for some type of work similar to the methods described in this text and not a good form to be useful in some other analysis. Lets put the control system described in Definition 2.5.1 and Equation: 2.1 into the transition system form [15].

$T(\Sigma) := (Q, L, \to, Q_0)$
where:

- $Q = \mathbb{R}^n$
- $L = \mathbb{U}$, a subset of all locally essentially bounded functions of time.
- $q \xrightarrow{u} p$ , if $x(\tau, q, u) = p$ for some $\tau \in \mathbb{R}^+$

- $Q_0 = Q$

Above mentioned transition system $T$ describes the system in Definition 2.5.1 and Equation: 2.1 without any desctreization. Following transition system $T_\tau$ is its time discretized transition system.

$T_\tau(\Sigma) := (Q, L, \rightarrow, Q_0)$
where:

- $Q = \mathbb{R}^n$
- $L = \mathbb{U}$, a subset of all locally essentially bounded functions of time.
- $q \xrightarrow{u} p$ , if $x(\tau, q, u) = p$ for some pre selected $\tau \in \mathbb{R}^+$
- $Q_0 = Q$

A transition system can be infinite or finite. However, above transition systems are infinite as their state spaces and input spaces have infinite amount of elements.

## 2.6 Approximate bisimulation

As described above control systems can be put into Infinite transition system. For computational purposes Finite State Transitions Systems(FSTS) are practically more useful. FSTS is a transition system which has a finite state space and a finite input space. This discretization of spaces comes with a cost and we loose some accuracy. Therefore we try to come up with approximate FSTS for control system. Then to achieve some properties curtain classes of control systems are attractive. In the following section let us introduce certain classes of control systems.

### 2.6.1 Incremental stability

Consider a system and two trajectories of it which starts from two different points in the state space with the same input. If the deference of such all such trajectories

of the system are guaranteed to be bounded by a $KL$ function, such systems are called to be Incrementally Globally Asymptotically Stable. The formal definition is the following.

**Definition 2.6.1.** [4, 22] A system is Incrementally Globally Asymptotically Stable if for any $t \in \mathbb{R}_0^+$, $x, y \in \mathbb{R}^n$ and $u \in U$ the following condition is satisfied with a $KL$ function $\beta$:

$$\|x(t, x, u) - x(t, y, u)\| \leq \beta(\|x - y\|, t)$$

Next, consider two trajectories which starts from two different points in the state space with the two different inputs. In this case we can achieve Incrementally Input to state stability. The formal definition is the following.

**Definition 2.6.2.** [4, 22] A system is Incrementally Input to state stable if for any $t \in \mathbb{R}_0^+$, $x, y \in \mathbb{R}^n$ and $u, v \in U$ the following condition is satisfied with a $KL$ function $\beta$ and $K_\infty$ function $\gamma$:

$$\|x(t, x, u) - x(t, y, v)\| \leq \beta(\|x - y\|, t) + \gamma(\|u - v\|_\infty)$$

### 2.6.2 The concept of approximate bisimulation

Exact bisimulation is very restrictive. Therefor the concept of approximate bisimulation was invented. The concept has arose through mainly the work of [5, 8, 20, 21]. Relaxing the exact observations to close enough observations with a desired precision is the main idea in approximate bisimulation. Following definitions define the concept formally.

**Definition 2.6.3.** [4] Let $T_1 = Q_1, \rightarrow_1, Q_1^0$ and $T_2 = Q_2, \rightarrow_2, Q_2^0$ be transition systems. A relation $R \subseteq Q_1 \ X \ Q_2$ is a $\delta$-approximation bisimulation relation between $T_1$ and $T_2$, if following conditions are satisfied for all $(q_1, q_2) \in R$

1. $\|q_1 - q_2\| \leq \delta$
2. $q_1 \ \rightarrow_1 \ q_1'$ implies existence of $q_2 \ \rightarrow_2 \ q_2'$, such that $(q_1', q_2') \in R$
3. $q_2 \ \rightarrow_2 \ q_2'$ implies existence of $q_1 \ \rightarrow_1 \ q_1'$, such that $(q_1', q_2') \in R$

**Definition 2.6.4.** [4] $T_1$ and $T_2$ are approximately bisimilar with precision $\delta$, if there exists a $\delta-$ approximate bisimulation relation between $T_1$ and $T_2$ such that:

1. $q_1 \in Q_1^0$ implies existence of $q_2 \in Q_2^0$, such that $(q_1, q_2) \in R$
2. $q_2 \in Q_2^0$ implies existence of $q_1 \in Q_1^0$, such that $(q_1, q_2) \in R$

# CHAPTER 3
# FSTS MODEL FOR (PATH PLANNING/PATH APPROXIMATION) HYBRID PLC CONTROL

## 3.1 Introduction

Various methods are used in the literature to analyze ICS. Most of them analyze software, embedded systems, Programmable Logic Controllers(PLC) and their control software with very higher level of abstractions of the continuous dynamics of the system [1, 2, 7, 11, 19]. This is due to the limitation of tools available to handle heterogeneity of ICS. On the other hand various studies on simulating hybrid systems with state transition systems can be found on the literature [4, 6, 15]. We identify suitable state transition systems for classes of ICS and combine that information with the rest of the discrete information of ICSs.

## 3.2 FSTS models for various classes of ICS

Considering all kinds of ICS in a single analysis is very difficult and not necessary. We limit our analysis for certain stable classes of ICS. We identify physical dynamics of ICSs as a system that can be described by the following equations.

$$\begin{aligned} \dot{X} &= F_M(X, U) \\ M &= M(X, U) \end{aligned} \tag{3.1}$$

$M$ is the mode of the system which can be switched by the PLC depending on the needs of the ICS and state of the system.

However to simplify our analysis we start with simpler systems and continue to increase complexity.

**Case 1:** ICS with only continuous dynamics

Let a control system be described by the following set of deferential equations. The details of the system is desctribed in Definition 3.2.1.

$$\dot{X} = f(X, U) \tag{3.2}$$

**Definition 3.2.1.** [15] A control system described by the following set $\Sigma = (\mathbb{R}^n, U, P, F)$

- $\mathbb{R}^n$ is the state space
- $U$ set of constant
- $U$ is s subset of
- $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ continuous Lipschitz function. For every compact set $K \subset \mathbb{R}^n$, there exists a constant $k > 0$ such that $||f(x, u) - f(y, u)|| \leq k||x - y||$ for all $x, y \in K$ and for all $u \in U$

Let's define a FSTS with more detail than the definitions we used in previous chapters. Additional information about observations provide structure for approximate bisimulation relation definition.

**Definition 3.2.2.** Finite state transition system is a set $T = (Q, L, \to, Q^0, O, H)$
$Q-$ set of finite possible states
$L-$ A set of labels
$\to Q\ X\ Q$ transition relation
$Q^0-$ set of finite possible initial states
$O-$ An output set
$H-$ An output function

Let's construct two transition systems which can be tested for approximate bisimilarity. The first transition system defined in definition 3.2.3 is only a time discretization. Therefor it's almost similar to the system described by the differential equations. The second transition system defined in definition 3.2.4 is discretized in both input and state spaces.

**Definition 3.2.3.** $T_\tau(\Sigma) = (Q, L, \rightarrow, Q^0, O, H)$

$Q - \mathbb{R}^n$

$L - \{l_1 \in U_\tau\}$

$q \xrightarrow{l} q$ if $x(\tau, q, l) = p$

$Q^0 - \mathbb{R}^n$

$H - 1$

**Definition 3.2.4.** $T_{\tau,\eta,\mu}(\Sigma) = (Q, L, \rightarrow, Q^0, O, H)$

$Q - [\mathbb{R}^n]_\eta$

$L - [L]_\mu$

$q \xrightarrow{l} q$ if $p - x(\tau, q, l) \leq \eta/2$

$Q^0 - \mathbb{R}^n$

$H - Q \hookrightarrow O$

To talk about bisimulation or approximate bisimlation we need a relation that describes the relationship of state spaces of the considered transition systems. For approximate buisimulation we can define a relation with $\delta$ precision as in Definition 3.2.5.

**Definition 3.2.5.** [4] Let $T_1 = Q_1, \rightarrow_1, Q_1^0$ and $T_2 = Q_2, \rightarrow_2, Q_2^0$ be transition systems. A relation $R \subseteq Q_1 \ X \ Q_2$ is a $\delta$-approximation bisimulation relation between $T_1$ and $T_2$, if following conditions are satisfied for all $(q_1, q_2) \in R$

1. $\|q_1 - q_2\| \leq \delta$
2. $q_1 \ \rightarrow_1 \ q_1'$ implies existence of $q_2 \ \rightarrow_2 \ q_2'$, such that $(q_1', q_2') \in R$
3. $q_2 \ \rightarrow_2 \ q_2'$ implies existence of $q_1 \ \rightarrow_1 \ q_1'$, such that $(q_1', q_2') \in R$

**Definition 3.2.6.** [4] $T_1$ and $T_2$ are approximately bisimilar with precision $\delta$, if there exists a $\delta-$ approximate bisimulation relation between $T_1$ and $T_2$ such that:

1. $q_1 \in Q_1^0$ implies existence of $q_2 \in Q_2^0$, such that $(q_1, q_2) \in R$
2. $q_2 \in Q_2^0$ implies existence of $q_1 \in Q_1^0$, such that $(q_1, q_2) \in R$

Let $T_1(\Sigma)$ be $T_1 = (Q_1, \rightarrow_1, Q_1^0)$ and $T_2(\Sigma)$ be $T_2 = (Q_2, \rightarrow_2, Q_2^0)$. Then $T_1(\Sigma)$ and $T_2(\Sigma)$ will be bisimilar systems according to the following theorem.

**Theorem 3.2.1.** *Control system $\Sigma$ and any desired precision $\epsilon \in \mathbb{R}^+$, If $\Sigma$ is Incrementally asymptotically input to state stable then for any $\tau \in \mathbb{R}^+$,$\eta \in \mathbb{R}^+$ and $\mu \in \mathbb{R}^+$ satisfying the following inequality,*

$\beta(\epsilon, \tau) + \nu + \gamma(\mu) + \eta/2 \leq \epsilon$

*$T_1$ is $\epsilon-$bisimilar to $T_2$*

**Proof.** refer [6]

**Theorem 3.2.2.** *Control system $\Sigma$ and any desired precision $\epsilon \in \mathbb{R}^+$, If $\Sigma$ is Incrementally asymptotically input to state stable then for any $\tau \in \mathbb{R}^+$,$\eta \in \mathbb{R}^+$ and $\mu \in \mathbb{R}^+$ with bounded noise and satisfying the following inequality,*

$\beta(\epsilon, \tau) + \nu_x + \nu_x + \gamma(\mu) + \eta/2 \leq \epsilon$

*$T_\tau$ is $\epsilon-$bisimilar to $T_{\tau,\eta,\mu}$*

**Proof.**

By definition of the relation R the (i) condition of Definition 3.2.5 is satisfied for any $(x, q) \in R$.

Consider any transition of $T_\tau$, $x \xrightarrow{l} y$, whether $l \in L_1$ and $x, y \in q_1$. However due to state noise the final output may be with in a $a$ distance form y.

$$\left\|y - y^{'}\right\| \leq a$$

As input space of both transition systems are assumed to be the same, we can find $z = x(\tau, q, l)$. As $Q_2$ cover the whole space $Q_1$ with maximum $\eta/2$ distance to any location, we can find $p \in Q_2$ such that:

$$\|z - p\| \leq \eta/2$$

therefore $q \xrightarrow{l} p$ in $T_{\tau,\eta,\mu}(\Sigma)$. As $\Sigma$ is incrementally globally asymptotically stable the following is true.

$\|y - p\| = \|y - z + z - p\|$

$\leq \|y - z\| + \|z - p\|$

$\leq \beta(\|x - q\|, \tau) + a + \eta/2$
$= \beta(\|\epsilon\|, \tau) + a + \eta/2 \leq \epsilon$

Therefor $(y, p) \in R$. Condition (ii) in Definition 3.2.5 is satisfied. Now consider the other way. As All the points in the state space of $T_{\tau,\eta,\mu}(\Sigma)$ are in $T_\tau(\Sigma)$ every transition of $T_{\tau,\eta,\mu}(\Sigma)$ can be simulated with $T_\tau(\Sigma)$. This completes the proof.

**Corollary.** *All the infinitely many solutions of the ICS $\Sigma$ can be approximated by a finite number of paths of state transitions of FSTS $T_2$. Further more if the time $t \leq T_w$, time window there is only finite number of transitions in each FSTS path.*

This is because the theorem guarantees any path of the original system can be $\delta$ approximated by the second state and input discretized transition system and the transition system is finite.

**Case 2:** Synchronized centralized control of ICS

Some of the PLC controlled task can be easily synchronized. Here we consider synchronized PLC driven industrial control systems. Such a control system can be described with the Equation:3.3 and Definition:3.2.7.

$$\begin{aligned} \dot{X} &= F_M(X, U) \\ M &= M(X, U) \end{aligned} \tag{3.3}$$

**Definition 3.2.7.** ICS with PLCs operate on sequential logic is a system $\Sigma = (\mathbb{R}^n, U, P, F)$

- $\mathbb{R}^n$ is the state space
- $U \in \mathbb{R}^m$ is the input space
- $P = \{1, ..., m\}$ is the finite set of modes
- $\mathtt{p} \in \mathbb{P}$, where P is a subset of $S(\tau, P)$ which is the set of piecewise constant functions from $\tau$ to $P$. These mode changes are done by the PLC.

- $F = \{f_1, ..., f_m\}$ where $f_m : \mathbb{R}^n \times U \to \mathbb{R}^n$

Here also, we can consider two ways of operation. The mode change by the PLC can be done open loop(without observing states) and closed loop(with state feedback). In the case of Synchronized centralized open loop control of ICS, we can design approximate bisimilar transition systems.

$T(\Sigma) := (Q, L, \to, Q_0, O, H)$
where:

- $Q = \mathbb{R}^n$
- $L = U$
- $q \xrightarrow{u} p$ , if $x(\tau, q, \mathtt{p}, u) = p$ for some $\tau \in \mathbb{R}^+$
- $Q_0 = Q$
- $O-$ Output set
- $H-$ Output function

Following is its time discretized transition system.

$T_\tau(\Sigma) := (Q, L, \to, Q_0)$
where:

- $Q = [\mathbb{R}^n]_\eta$
- $L = U$
- $q \xrightarrow{u} p$ , if $x(\tau, q, \mathtt{p}, u) = p$ for some $\tau \in \mathbb{R}^+$
- $Q_0 = Q$
- $O-$ Output set

- $H-$ Output function

These transition systems can be made bisimilar with a relation R if the ICS ,$\Sigma = (\mathbb{R}^n, p, \mathbb{P}^n, F)$ is Incrementally Globally Uniformly Asymptotically Stable(IGUAS). The concept of IGUAS is defined in the following definition. The requirements of IGUAS and for other details of approximate bisimulation of switched systems we refer the reader to [6].

**Definition 3.2.8.** [6]
A system $\Sigma = (\mathbb{R}^n, p, \mathbb{P}^n, F)$ is incrementally globally uniformly asymptotically stable if there exits a $KL$ functoin $\beta$ such that for all $t \in \mathbb{R}^+$, for all $x, y \in \mathbb{R}^n$, for all switching signals $P \in \mathbb{P}$, the following condition is satisfied:

$$||x(t, x, p) - x(t, y, p)|| \leq \beta(||x - y||, t) \tag{3.4}$$

**Case 3:** ICS Input to state stable under bounded noise

ICS with Input to state stable dynamics under bounded noise is considered here. Systems with noise has been studied with transition systems in the literature [16, 17]. Systems of this type can be modeled and achieve alternating approximate bisimulation.

**Definition 3.2.9.** [17] ICS with PLCs operate on sequential logic and bounded noise is a system $\Sigma = (\mathbb{R}^n, W, U, P, F)$

- $\mathbb{R}^n$ is the state space
- $W = U \times V$ is the input space, where:$U \in \mathbb{R}^m$ is the input space, $V \subseteq \mathbb{R}^s$ is the disturbance input space
- $P = \{1, ..., m\}$ is the finite set of modes

- $\mathbf{p} \in \mathbb{P}$, where P is a subset of $S(\tau, P)$ which is the set of piecewise constant functions from $\tau$ to $P$. These mode changes are done by the PLC.
- $F = \{f_1, ..., f_m\}$ where $f_m : \mathbb{R}^n \times U \to \mathbb{R}^n$

The basic definition for alternating approximate bisimulation is the following. For further detail we refer the reader to [17].

**Definition 3.2.10.** [4]
Let $T_1 = (Q_1, A_1 \times B_1, \to_1, Q_1^0)$ and $T_2 = (Q_2, A_1 \times B_1, \to_2, Q_2^0)$ be transition systems. A relation $R \subseteq Q_1 \ X \ Q_2$ is a $\delta$-approximation alternating bisimulation relation between $T_1$ and $T_2$, if following conditions are satisfied for all $(q_1, q_2) \in R$

1. $\|q_1 - q_2\| \leq \delta$
2. $\forall a_1 \in A_1 \exists a_2 \in A_2 \forall b_2 \in B_2 \exists b_1 \in B_1$ such that
   $q_1 \ \to_1 \ q_1'$ implies existence of $q_2 \ \to_2 \ q_2'$, such that $(q_1', q_2') \in R$
3. $q_2 \ \to_2 \ q_2'$ implies existence of $q_1 \ \to_1 \ q_1'$, such that $(q_1', q_2') \in R$

### 3.2.1 PLC Feedback control with state approximations

PLC switching can be based on time or states of the system. In the case the PLC is based on time switching happens in both transition systems we approximate in the same manner. However when switching is sensitive to States, we need to consider all the possible cases that could be a result of approximation. All these possibilities are important for the attack detection methods in the next chapter. Computations of these cases can be doe with the following algorithm.

**Algorithm 1:**
Input:$T$ (Transition system),$\epsilon$ accuracy,PLC Input logic function, ints
Output: state estimations, SE
C$\Omega = Null$
$SE_0 = int \times mode$
for i=1...n
  for each element in SE
  compute possible mode switching $M$, with PLC Input logic function
  $\Omega = \psi(T, M, SE_{i-1}, \epsilon)$
  C$\Omega = C\Omega + \Omega$
  end
  $SE_i$=C$\Omega$
end

The above algorithm finds the $i + 1$ possible states given the $i$ possible states. This algorithm can be used as the core of any algorithm that computes all the possible state paths of the system that can be approximated with the transition system for a set of given initial conditions. In the algorithm $\psi$ computes State Estimations(SE) for the $i + 1$ time step once $T, M, SE_{i-1}$ and $\epsilon$ are given.

# CHAPTER 4
# FSTS BASED PACKET DATA SCREENING

## 4.1 Introduction

Safety and security of ICS has been very important issue all the time. The new issue is, added vulnerabilities due to the communication network that has become a part of the system. On the other hand information security is also studied for years. They provide authentication, encryption and various other access control methods etc. Then it is important to understand what are the major characteristic of the problem of security of modern ICS. The major deference respect to the information security of computer networks is availability of the model of the control system describing the physics of the system.

In literature various model and various types of attacks are discussed [9, 10, 14]. Our study is how we can use a FSTS model to find solutions for the problem of security of modern ICS.

## 4.2 Method

Studying various attack detectors/monitors in the literature, we can generalize a model for attack detectors. [14] provides a generalized detector model for some problems discussed in the literature. It's shown in the following definition.

**Definition 2: [14]** A monitor is a deterministic algorithm with access to continuous-time measurements and knowledge of the system dynamics. The input of a monitor is $A = \{E, A, C, y(t) \forall t \in \mathbb{R}_{\geq 0}\}$. The output of a monitor is $\Psi(A) = \{\Psi_1(A), \Psi_2(A)\}$ with $\Psi_1(A) = \{True, False\}$ stating whether there is an attack, and $\Psi_2(A)-$ affected set of sensors and actuator.

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{4.1}$$

However, in a modern ICS we don't have continuous measurements of all the variables. For this reason and to adopt the concept of the monitor for ICSs, we define the following definition.

**Definition 3:** A monitor is a deterministic algorithm with access to data packets

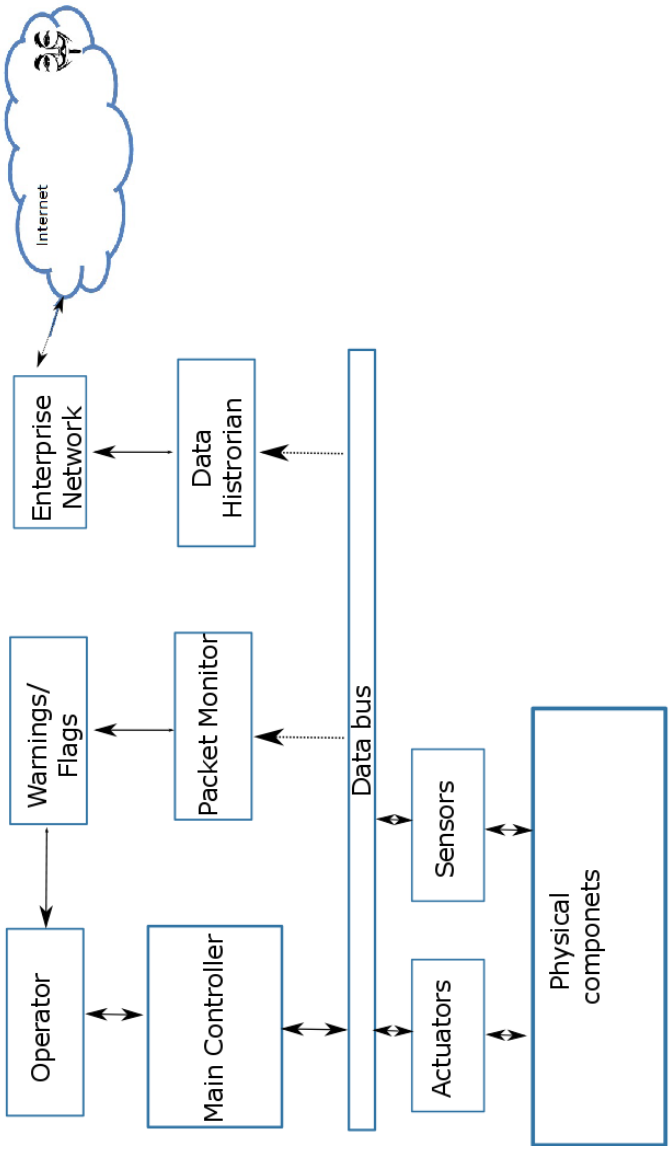


Figure 4.1. Industrial Control System Layout

communicated through the communication network of the control system containing measurements and knowledge of the system dynamics. The input of a monitor is $A = \{System\ model,\ Sniffed\ packets\}$ at $T \subset \{t \in \mathbb{R}_{\geq 0}\}$. The output of a monitor is $\Psi(A) = \{\Psi_1(A), \Psi_2(A)\}$ with $\Psi_1(A) = \{True, False\}$, and $\Psi_2(A)-$ affected set of sensor and actuator data packets.

Using either Definitions or something similar in the literature, lets try to

generalized core of the problem or the algorithm. As described in [3, 14, 18] the core of the problem can be identified as the following optimization problem.

**Definition 4.2.1.** [14] Core of the attack monitor problem

$$\begin{aligned} \underset{v_x, v_y, x_0}{\text{minimize}} \quad & ||v_x|| + ||v_y|| \\ \text{subject to} \quad & E\dot{x}(t) = Ax(t) + v_x(t) \\ & y(t) = Cx(t) + v_y(t) \\ & x(0) = x_0 \in \mathbb{R}^n \end{aligned}$$

Above definition is defined in the continuous domain for linear systems. As we discus before, modern ICS mostly communicate data in packet form. Therefor we propose and introduce "Packet sniffer's attack monitor problem" as one of the main contributions of this work .

**Definition 4.2.2.** Packet sniffer's attack monitor problem for state attacks

$$\begin{aligned} \underset{X_0}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f_M(X_n, U_n) \\ & M = M(X_n, U_n) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F(x_{i,n})) \leq \delta \\ & N_f = \sum F(x_{i,n}) \end{aligned}$$

$N_f$ − Number of flagged packets
$X_n$ − State of the system at step n
$U_n$ − Input of the system at step n
$M$ − Mode of the system controlled by the PLC
$x_{i,n}$ − $i^{th}$ element of $X_n$
$F(x_{i,n})$ − Flag status function,1 −Flagged, 0 −Otherwise
$y_{i,n}$ − Measurement of $i^{th}$ element of $X_n$,assume $C = 1$
$\delta$ − Accuracy threshold

**Definition 4.2.3.** Packet sniffer's attack monitor problem for state and input attacks

$$\begin{aligned} \underset{X_0}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f_M(X_n, U_n) \\ & M = M(X_n, U_n) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F_n(x_{i,n})) \leq \delta \\ & N_f = w_s \sum F(x_{i,n}) + w_u \sum F(u_{i,n}) \end{aligned}$$

$w_s$ − Weight for state packets
$w_u$ − Weight for input packets

We study the problems defined in Definition 4.2.2 and 4.2.3 with respect to the model we discussed in the previous chapter. We expect to identify what solutions can be given to the main problem. Also we expect to identify what can not be done by solving the above minimization problem with a model in a FSTS.

**Case 1:**Observe state packets only, dynamics are Incrementally globally asymptotically stable

For this case attack monitor problem can be put into the following form.

**Definition 4.2.4.** Packet sniffer's attack monitor problem for state attacks

$$\begin{aligned} \underset{X_0,\ U}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f(X_n, U_n) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F_n(y_{i,n})) \leq \epsilon \\ & N_f = \sum F(x_{i,n}) \end{aligned}$$

**Theorem 4.2.1.** *Consider an ICS $\Sigma$(Definition 4.2) and any desired precision $\epsilon \in \mathbb{R}^+$. If there exists an approximate bisimulation relation $R$ making $T_1$ and $T_2$ $\epsilon$*

*approximately bisimilar and measurements are not manipulated bounded, then there exists at least one path with $\epsilon$ accuracy which satisfies the problem defined in Definition:4.2.4.*

**Proof.**

In chapter 4, the Corollary of Theorem 3.2.2 it was shown that FSTSs can cover all the solution space.

Let the set of all observed state measurements be $Y$. Consider $\Pi_i$ the set of paths that satisfy $abs(x_{i,n} - y_{i,n}) \leq \epsilon$ for $y_{i,n} \in Y$ and $y_{i,n}$ not flagged ($F_n(y_{i,n}) = 0$), then $\Pi_i$ contains all the paths that go through a $\epsilon$ neighborhood of $y_{i,n}$. This is true for all $i, y_{i,n} \in Y$. The correct solution also must go through $y_{i,n}$ for all $i, y_{i,n} \in Y$. As the FSTS cover the whole solution space, there exists at least one path with $\epsilon$ accuracy which satisfies the optimization problem.

**Corollary.** *If the number of packets flagged is non zero, received packets are guaranteed manipulated(or corrupted).*

**Proof.**

This means that there was no path that $\epsilon$ approximate that takes the given measurements Y. However, according to chapter 4 the Corollary of Theorem 3.2.2, FSTS can $\epsilon$ approximate all the possible paths in the time window. Therefor there is at least one measurement $y_{i,n} \in Y$, such that $abs(x_{i,n} - y_{i,n}) > \delta$.

**Lemma 4.2.2.** *The attack was done with J packets. The attack can be solved with K packets, $J > K$. The attack is undetectable*

**Proof.**

$Y_J$ can be solved with flagging $J$ packets with $N_f = J$. There is a path that satisfies the optimization problem with flagging $k$ packets,$N_f = K$. Since $J > K$, K packet attack is the minimum solution to the optimization problem.

**Remarks**

In other words this explains the idea if the attacker can take control of more than enough number of packets, attacker can get attack identifiers to believe the manipulated packets and reject the true packets.

It's also worth noticing here is if we know some of the $J$ packets are very suspicious with other information(network packet losing data etc). We may still be able to correctly identify the $J$ number of packets which is theoretically impossible only considering the the physical dynamical system.

**Case 2:** Observe state packets only ,Continuous dynamics are incrementally globally asymptotically stable. There is mode switching controlled by the PLC.

For this case attack monitor problem can be put into the following form.

**Definition 4.2.5.** Packet sniffer's attack monitor problem for state attacks

$$\begin{aligned} \underset{X_0,\ U}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f_m(X_n, U_n) \\ & m = M(U_n) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F_n(y_{i,n})) \leq \epsilon \\ & N_f = \sum F(x_{i,n}) \end{aligned}$$

**Theorem 4.2.3.** *Consider an ICS and any desired precision $\epsilon \in \mathbb{R}^+$. If there exists an approximate bisimulation relation $R$ making $T_1$ and $T_2$ $\epsilon$ approximately bisimilar and measurements are not manipulated, then there exists at least one path with $\epsilon$ accuracy which satisfies the problem defined in Definition:4.2.5.*

**Proof.**
The proof in Theorem 4.2.1 still valid for this case.

**Corollary.** *If the number of packets flagged is non zero, received packets are guaranteed manipulated(or corrupted).*

**Proof.**
The proof for the Corollary of Theorem 4.2.1 still valid for this case.

**Case 3:** Observe state packets only ,Continuous dynamics are incrementally globally asymptotically stable. There is mode switching controlled by the PLC. Measurements are noisy.

For this case attack monitor problem can be put into the following form. The new addition to this case is noise. The offset resulted by the disturbance input, $d$ is assumed to be bounded, $-a \leq d \leq a$, $a \in \mathbb{R}^+$.

**Definition 4.2.6.** Packet sniffer's attack monitor problem for state attacks

$$\begin{aligned} \underset{X_0,\ U}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f_m(X_n, U_n) \\ & m = M(U_n) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F_n(y_{i,n})) \leq \delta \\ & N_f = \sum F(x_{i,n}) \end{aligned}$$

here $\delta = \epsilon + a$.

**Theorem 4.2.4.** *Consider an ICS $\Sigma$(Definition:4.2.6) and any desired precision $\epsilon \in \mathbb{R}^+$. If there exists an approximate bisimulation relation $R$ making $T_1$ and $T_2$ $\delta$ approximately bisimilar and measurements are not manipulated bounded, then there exists at least one path with $\epsilon$ accuracy which satisfies the problem defined in Definition:4.2.4.*

**Proof.**

In chapter 4, the Corollary of Theorem 3.2.2 it was shown that FSTSs can cover all the solution space.

Let the set of all observed state measurements be $Y$. Consider $\Pi_i$ the set of paths that satisfy $abs(x_{i,n} - y_{i,n}) \leq \delta$ for $y_{i,n} \in Y$ and $y_{i,n}$ not flagged ($F_n(y_{i,n}) = 0$), then $\Pi_i$ contains all the paths that go through a $\delta$ neighborhood of $y_{i,n}$. This is true for all $i, y_{i,n} \in Y$. The correct solution also must go through a $\epsilon$ neighborhood of $y_{i,n}$ for

all $i, y_{i,n} \in Y$. As the FSTS cover the whole solution space, there exists at least one path with $a$ accuracy which satisfies the optimization problem.

**Remarks**

Here even though the FSTS can provide $\epsilon$ accuracy, we only can achieve $\delta$ accurate paths. Hoever when the number of packets observed increase, we get more and more accurate results with best accuracy bound $\epsilon$.

**Corollary.** *If the number of packets flagged is non zero, received packets are guaranteed manipulated(or corrupted).*

**Proof.**

This means that there was no path that $\delta$ approximate that takes the given measurements Y. However, according to chapter 4 the Corollary of Theorem 3.2.2, FSTS can $\epsilon$ approximate all the possible paths in the time window. Therefor there is at least one measurement $y_{i,n} \in Y$, such that $abs(x_{i,n} - y_{i,n}) > \delta$.

**Case 4:** Observe state packets and input packets ,Continuous dynamics are incrementally globally asymptotically stable. There is mode switching controlled by the PLC. Measurements are not noisy.

For this case attack monitor problem can be put into the following form. The most important addition we consider in this case is input packet observation.

**Definition 4.2.7.** Packet sniffer's attack monitor problem for state and input attacks

$$\begin{aligned} \underset{X_0}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f_m(X_n, U_n) \\ & m = M(U_n) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F_n(x_{i,n})) \leq \epsilon \\ & N_f = w_s \sum F(x_{i,n}) + w_u \sum F(u_{i,n}) \end{aligned}$$

In all the previous cases we only observed state packets. We checked whether state packets matched with a possible path of the control system with some level of accuracy. Now we have addition information of observed inputs. Previously the optimization had to determine the possible input according to the possible state paths. However here we use input observed as additional information.

**Theorem 4.2.5.** *Consider an ICS $\Sigma$(Definition:4.2.7) and any desired precision $\epsilon \in \mathbb{R}^+$. If there exists an approximate bisimulation relation $R$ making $T_1$ and $T_2$ $\delta$ approximately bisimilar and measurements are not manipulated bounded, then there exists at least one path with $\epsilon$ accuracy which satisfies the problem defined in Definition:4.2.4.*

**Proof.**

In this case we have a redundancy of information. We can find a non empty set of paths as we proved in the previous cases. Those paths also determined a possible sequence of inputs. As the FSTS cover all the solution space, there must be at least one solution that matches with the observed input sequence.

**Corollary.** *If the number of packets flagged is non zero, received packets are guaranteed manipulated(or corrupted).*

**Proof.**

There are two ways a packet can be flagged. First the packet got flagged can be a input data packet. This means for every state path such that $abs(x_{i,n} - y_{i,n}) > \delta$ for all $i$, there is at least one input observed that can not be accepted with the path. That means either flagged input is manipulated or the state path is manipulated.

Second, the packet got flagged can be a input data packet. This means there is no state path such that $abs(x_{i,n} - y_{i,n}) > \delta$ for all $i$ for all $i$, that accept all the observed inputs. This also means either flagged state packet is manipulated or at least one input packet is manipulated.

**Lemma 4.2.6.** *The attack was done with packets can make a weight of $N_{f_J} = w_s \sum F(x_{i,n}) + w_u \sum F(u_{i,n})$. The attack can be solved with packet weight of $N_{f_K} = w_s \sum F(x_{i,n}) + w_u \sum F(u_{i,n})$, $N_{f_J} > N_{f_K}$. The attack is undetectable.*

**Proof.**

The attack $J$ can be solved with flagging $N_{f_J}$ worth packets. There is a path that satisfies the optimization problem with flagging $N_{f_K}$ worth packets,$N_f = K$. Since $N_{f_J} > N_{f_K}$, $N_{f_K}$ packet attack is the minimum solution to the optimization problem.

**Case 5:** Observe state packets and input packets ,Continuous dynamics are incrementally globally asymptotically stable. There is mode switching controlled by the PLC and PLC logic is also given. Measurements are not noisy.

For this case attack monitor problem can be put into the following form. The most important addition we consider in this case is PLC input logic.

**Definition 4.2.8.** Packet sniffer's attack monitor problem for state and input attacks

$$\begin{aligned} \underset{X_0}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f_M(X_n, U_n) \\ & M = M(X_n, U_n) \\ & U_n = g(X_n, t) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F_n(x_{i,n})) \leq \epsilon \\ & N_f = w_s \sum F(x_{i,n}) + w_u \sum F(u_{i,n}) \end{aligned}$$

Here we consider the the PLC logic which is a function of state and possibly time. FSTS only give as an approximation to the state therefor some ambiguity can take place while finding the control input. To assist with we use the Algorithm presented in the previous chapter to propagate state and possible inputs. With the help of the algorithm we solve thew above optimization problem.

**Case 6:** Observe state packets and input packets ,Continuous dynamics are incrementally globally asymptotically stable. There is mode switching controlled by the PLC and PLC logic is also given. Measurements are noisy.

**Definition 4.2.9.** Packet sniffer's attack monitor problem for state and input

attacks

$$\begin{aligned} \underset{X_0}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f_M(X_n, U_n) \\ & M = M(X_n, U_n) \\ & U_n = g(X_n, t) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F_n(x_{i,n})) \leq \delta \\ & N_f = w_s \sum F(x_{i,n}) + w_u \sum F(u_{i,n}) \end{aligned}$$

here $\delta = \epsilon + a$.

**Case 7:** Observe state packets and input packets ,Continuous dynamics are incrementally globally asymptotically stable. Consider state disturbances. There is mode switching controlled by the PLC and PLC logic is also given. Measurements are noisy.

**Definition 4.2.10.** Packet sniffer's attack monitor problem for state and input attacks

$$\begin{aligned} \underset{X_0}{\text{minimize}} \quad & N_f \\ \text{subject to} \quad & X_{n+1} := f_M(X_n, U_n) \\ & M = M(X_n, U_n) \\ & U_n = g(X_n, t) \\ & abs(x_{i,n} - y_{i,n}) \times (1 - F_n(x_{i,n})) \leq \delta \\ & N_f = w_s \sum F(x_{i,n}) + w_u \sum F(u_{i,n}) \end{aligned}$$

here $\delta = \epsilon + a + b$.

## 4.3 Simulation results

The tank system used in the simulation is shown in Figure 4.2. Their dynamics are modeled by following differential equations.

$$\begin{aligned} \dot{h_1} &= k_0 f_{in} - k_1\sqrt{h_1} \\ \dot{h_2} &= k_2 h_1 - k_3\sqrt{h_2} \\ \dot{h_3} &= k_4 h_2 - k_0 f_{in} \end{aligned} \tag{4.2}$$

In the simulations constants of the differential equations were chosen to be $k_0 = 0.083$, $k_1 = 0.1083$, $k_2 = 0.1083$, $k_3 = 0.0667$, $k_4 = 0.0667$. A transition system was developed of the form in Definition 3.2.4 with $\eta = .5$ and $\tau = 60$. In put space had only 3 elements(3 flow rates) namely, $U = \{0, 20, 30\}$.

The optimization problem in Definition 4.2.4 was solved. Implementation was done with Answer Set Prolog(ASP). A trail was run with initial conditions $h_1(0) = 11.5$, $h_2(0) = 3.5$ and $h_3(0) = 4$ and u=30. Then the algorithm was given correct measurements and the result was $N = 0$ optimal solution. That means no packet was flagged. Then at $t = 60$ and $t = 540$ two state measurement packets were corrupted artificially to be $h_2(60) = 10$ and $h_1(60) = 5$. The optimization algorithm resulted a solution with $N = 2$ and the corresponding corrupted two state measurements were flagged. These results are shon in Figure 4.3.

Next the optimization problem defined in Definition 4.2.7 was implemented with ASP. However no switching was tested. Additional state limit constraints were put to the ASP program. The tank 2 maximum allowable height was set to less than nine. Then a trial was run as shown in Figure 4.4 where the simulation flagged the input packets as they were guiding the system to an unsafe state.

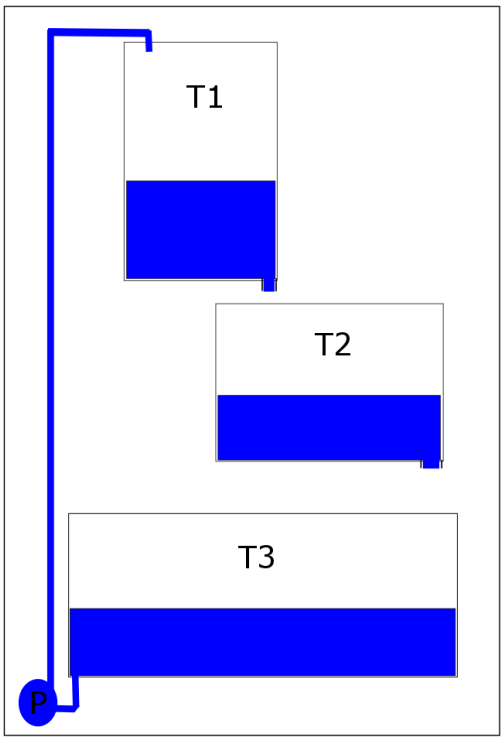


Figure 4.2. Tank system diagram

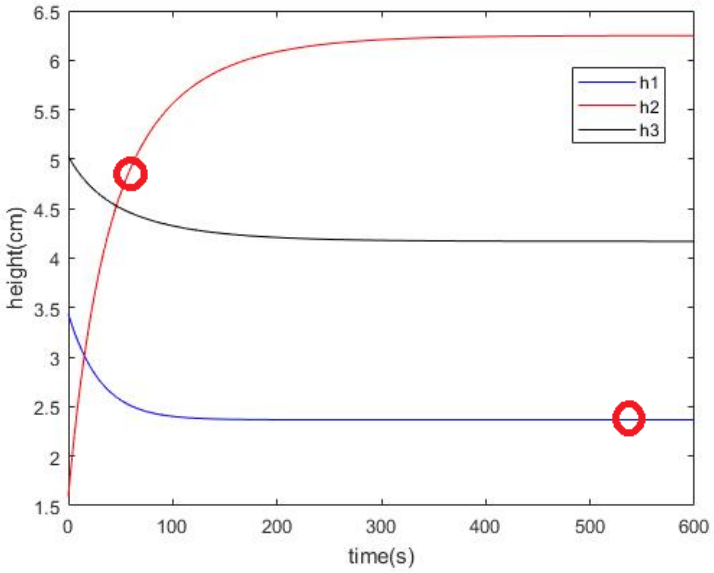


Figure 4.3. Simulation results: The graph of Heights of the tank system vs Time. State packet flagging by the packet monitor's attack detection optimization problem due to state observation mismatches.

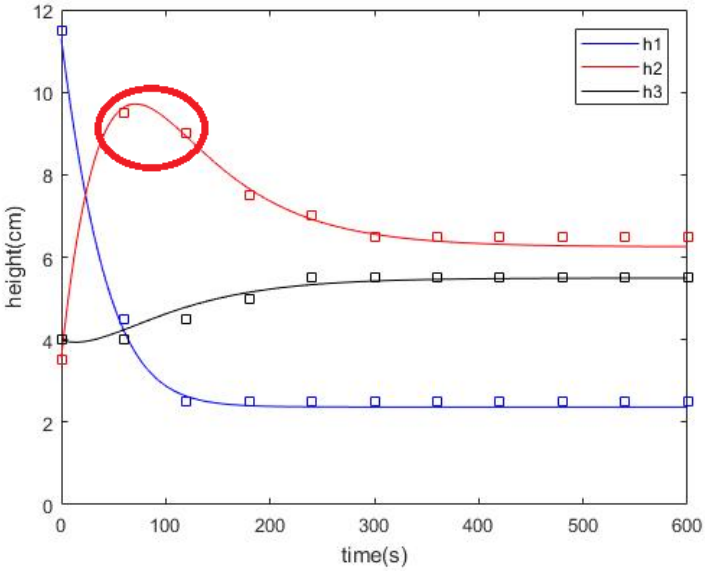


Figure 4.4. Simulation results: The graph of Heights of the tank system vs Time. Input packet flgging due to tank overflow at 60 seconds and 120 seconds.

## 4.4 Contributions

Contributions of this chapter can be summarized as following.

- Conversion of fundamental attack identification problem to packet sniffer's optimization problem
- Study of eligibility of the packet sniffer's optimization problem
  - Study what can be done and what can't be done in the attack detection process
  - Theorem proofs of what can be done and what can't be done in the attack detection process

# CHAPTER 5
# FSTD BASED CONTROL WITH FEEDBACK

## 5.1 Introduction

Up to this chapter we have been discussing Finite State Transition System(FSTS) and their use in cyber security of ICS. In this chapter we look at FSTS based control. Control synthesis with FSTS is usually open loop. [4, 15] provides examples for them. The main idea of approximate bisimulation is explained in Figure: 5.1(a). With approximately bisimilar systems when one system take a certain path on the state space, the other system also guaranteed to go within the given precision to the path of the first system. Therefor in this case feedback is not necessary. However when noise is getting larger bisimulation would not be possible. After this limit with feedback FSTS control can be achieved as described in the following subsection.

## 5.2 Method

With limited amount of noise bisimulation can be achieved. When systems accept bisimilar Transition Systems(TS) we can achieve open loop control without feedback from the system. However most of the practical systems contain considerable amount of noise. Higher levels of noise can make this systems not to accept bisimilar TS. We identify, systems which accept bisimilar transition systems (without considering noise), can be controlled with noise with feedback. This process is shown in Figure: 5.1(b).

Due to noise the system shown with dotted lines in Figure: 5.1(b) is deviating away from the expected $\epsilon$ accuracy with respect to the second system shown in solid line. However if it is still limited by $\epsilon'(> \epsilon)$ accuracy, we can still achieve a trajectory $\epsilon'$ accurate for some systems. We work on an algorithm to implement this type of feedback controlling with TS. We expect to run experiments with a simple pendulum apparatus to test the algorithm.

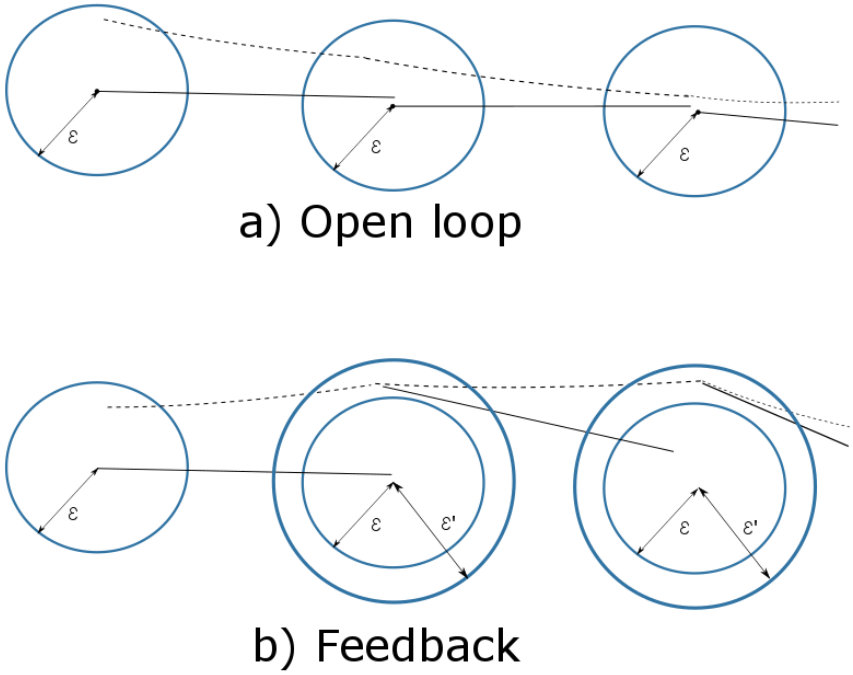


Figure 5.1. Trajectories of systems with (a) open loop and (b) feedback, control. State transition system 1 state Trajectory in dotted line, State transition system 2 state Trajectory in solid line

### 5.2.1 Transition system

Let's construct transition systems for a given control system.

**Definition 5.2.1.** A control system described by the following set $\Sigma = (\mathbb{R}^n, U, P, F)$

- $\mathbb{R}^n$ is the state space
- $U \in \mathbb{R}^m$ is the input space
- Mode changing law (input)
- $F = \{f_1, ..., f_m\}$ where $f_m : \mathbb{R}^n \times U \to \mathbb{R}^n$

### 5.2.2 Time discretization

A control system can be put into the transition system form without any discretization. Lets put the control system described in Definition 5.2.1 into the transition system form [15].

$T(\Sigma) := (Q, L, \rightarrow, O, H)$
where:

- $Q = \mathbb{R}^n$
- $L = U$
- $q \xrightarrow{u} p$ , if $x(\tau, q, u) = p$ for some $\tau \in \mathbb{R}^+$
- $O = \mathbb{R}^n$
- $H = 1_{\mathbb{R}^n}$

Following is its time discretized transition system.

$T_\tau(\Sigma) := (Q, L, \rightarrow, O, H)$
where:

- $Q = [\mathbb{R}^n]_{e_1..e_n}$
- $L = [U]_\mu$
- $q \xrightarrow{u} p$ , if $x(\tau, q, u) = p$ for some $\tau \in \mathbb{R}^+$
- $O = \mathbb{R}^n$
- $H = 1_{\mathbb{R}^n}$

### 5.2.3 Input desensitization

Input discretization can be achieved in few ways [15]. All locally essentially bounded functions can be approximated with a set of discretized inputs functions. Input functions can be approximated with a linear combination of a set of basis functions, a set piecewise constant functions or path end point discretization in state space. For more detal, we refer the reader to [15].

### 5.2.4 Feed back control

The concept of feedback control with FSTS was explained before. Here is the complete process of path planing with FSTS.

First we need to discretize the state space into a sparse grid of point the interested ares of the state space. $Q_1 = [\mathbb{R}^n]_{c_1..c_n}$. Then we can define a set of balls around each element of $Q_1$. These balls can be identified with $B_{d_1}(q)$, for any $q \in Q$.

Each pair of these balls are then tested whether to be an edge of candidate path map, $G_c \subseteq Q_1 \times Q_1$. If there is an input $l \in L_2$ such that $z = x(\tau, l, p)$ and $|z - q| \leq d_1$ where $p, q \in Q_1$, then $(p, q) \in G_c$.

Let $g \in Q_1$ be the goal state and $s \in Q_1$ be the start location. Then with the map $G_c$ we can find a candidate path from $s$ to $g$. Let a path be $(p_1, p_2...p_L) \in Q_1^L$. $L$ is the length of the path.

Consider balls $B_{\acute{d}_1..\acute{d}_n}(p_i)$ and $B_{d_1..d_n}(p_{i+1})$ and a second discretization of the state space $Q_2 = [\mathbb{R}^n]_{e_1..e_n}$. Then from every point on the path, $p_i \in Q_1$ and every point $q \in B_{\acute{d}_1..\acute{d}_n}(p_i)$ and $q \in Q_2$, we find a control such that output location is guaranteed to be in $B_{d_1}(p_{i+1})$. We do this process for all $i$, thus a control is found. In the execution of the control law, we first sense where the system is in the current $i_{th}$ ball $B_{\acute{d}_1..\acute{d}_n}(p_i)$ and executes the corresponding control. Then the system is guaranteed to be in $B_{\acute{d}_1..\acute{d}_n}(p_{i+1})$.

## 5.3 Simulation and Experiment results

### 5.3.1 Inverted pendulum

A pendulum and a pendulum on cart are classic examples to study performance of controllers. In this section we study our method with a pendulum.

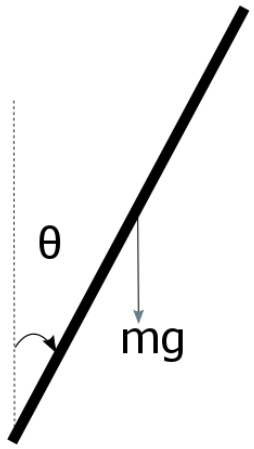


Figure 5.2. Inverted Pendulum diagram

Dynamics of the pendulum can be mapped onto a cylinder or a plain. When the dynamics are mapped onto a cylinder

The dynamics of the simple pendulum can be described by the following equations.

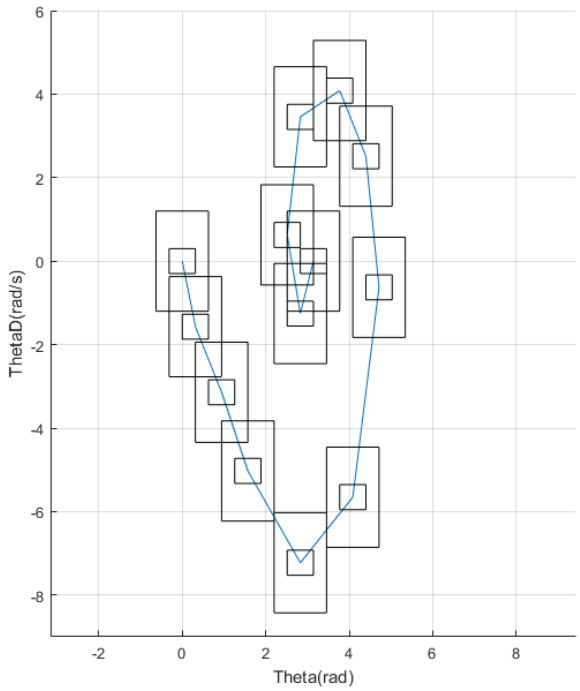


Figure 5.3. Computed path for the Inverted Pendulum from state (pi,0) to state (0,0) : Two dimensional view of State path, $Theta$ vs $thetaD(\omega)$.

$$\begin{aligned} \dot{\theta} &= \omega \\ \dot{\omega} &= -(g/l)sin(\theta) - (b/m)\omega + (12/(ml^2)) * u \end{aligned} \tag{5.1}$$

The specifications of the pendulum we tested our control was the fowling.

$m = .234kg$

$l = .638m$

$b = .02Ns/m$

$g = 9.8m/s^2$

Inverted Pendulum(IP) state space was discretized with $c_1 = 0.1571, c_2 = 0.1571$, $Q = [\mathbb{R}^4]_{c_1,c_2}$. With this discretization the set of $B_{d_1,d_2}(q)$ was defined with

$d_1 = 0.3142, d_2 = 0.3$. Each pair of these balls were tested whether to be an edge of candidate path map, $G_c \subseteq Q \times Q$. Thus $G_c$ was created for the IP.

The goal state $g \in Q$ was set to $(0, 0)$ and the start location $s \in Q$ was set to $(pi, 0)$. Then with a breath first search of the graph $G_c$ implemented in Matlab, we found a path from $s$ to $g$. This path is shown in Figure 5.3.

From each point on the path $p_i$, balls $B_{\acute{d_1},\acute{d_2}}(p_i)$ and $B_{d_1,d_2}(p_{i+1})$ and a second discretization of the state space $Q_2 = [\mathbb{R}^n]_{e_1,e_2}$ with $e_1 = 0.05, e_2 = 0.05$. Then from every point on the path, $p_i \in Q_1$ and every point $q \in B_{\acute{d_1},\acute{d_2}}(p_i)$ and $q \in Q_2$, we find a control such that output location is guaranteed to be in $B_{d_1,d_2}(p_{i+1})$. We do this process for all $i$, thus a control is found. In the execution of the control law, we first sense where the systems is in the current $i^{th}$ ball $B_{\acute{d_1},\acute{d_2}}(p_i)$ and executes the corresponding control. Then the system is guaranteed to be in $B_{\acute{d_1},\acute{d_2}}(p_{i+1})$.

### 5.3.2 Inverted pendulum on cart

Inverted pendulum on cart is classic system that is used in testing control methods. Following equations model dynamics of the system.

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= \frac{(M+m)gsin(x_1) - (lmx_2^2 sin(x_1) + u)cos(x1)}{l(M + sin^2(x_1)m)} \\ \dot{x_3} &= x_4 \\ \dot{x_4} &= \frac{-mgsin(x_1)cos(x1) + lmx_2^2 sin(x_1) + u}{M + sin^2(x_1)m} \end{aligned} \tag{5.2}$$

The specifications of the pendulum we tested our control was the fowling.

$M = .2kg$

$m = .234kg$

$l = .638m$

$g = 9.8m/s^2$

First discretization of the state space $Q = [\mathbb{R}^4]_{c_1..c_4}$ was done with $c_1 = 0.314, c_2 = 2.8, c_3 = 0.12, c_4 = 1.2$. With this discretization the set of $B_{d_1..d_4}(q)$

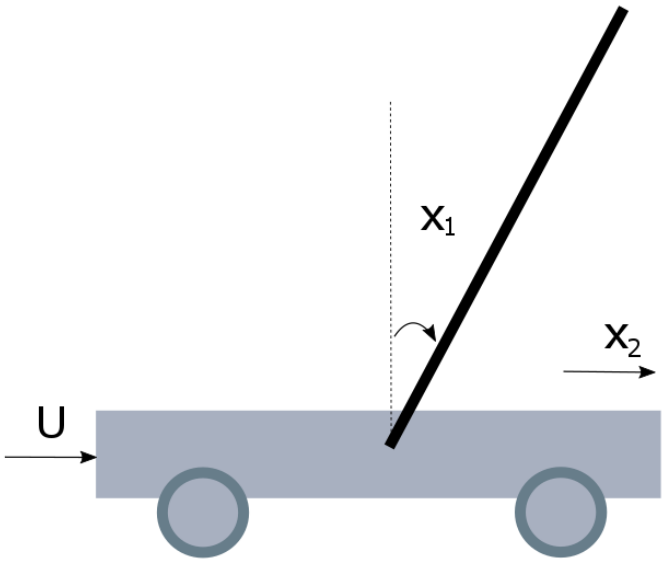


Figure 5.4. Inverted Pendulum Cart diagram

was defined with $d_1 = 0.2024, d_2 = 1.82, d_3 = 0.195, d_4 = 0.78$. Each pair of these balls were tested whether to be an edge of candidate path map, $G_c \subseteq Q \times Q$. Thus $G_c$ was created for the IPC.

Then the goal state $g \in Q$ was set to $(0, 0, 0, 0)$ and the start location $s \in Q$ was set to $(pi, 0, 0, 0)$. Then with a breath first search of the graph $G_c$ we found a path from $s$ to $g$. This path is shown in Figure 5.5 & 5.6.

From each point on the path $p_i$, balls $B_{\acute{d}_1..\acute{d}_4}(p_i)$ and $B_{d_1..d_4}(p_{i+1})$ and a second discretization of the state space $Q_2 = [\mathbb{R}^n]_{e_1..e_n}$ with
$e_1 = 0.0047, e_2 = 0.0420, e_3 = 0.0600, e_4 = 0.0180$. Then from every point on the path, $p_i \in Q_1$ and every point $q \in B_{\acute{d}_1..\acute{d}_4}(p_i)$ and $q \in Q_2$, we find a control such that output location is guaranteed to be in $B_{d_1..d_4}(p_{i+1})$. We do this process for all $i$, thus a control is found. In the execution of the control law, we first sense where the systems is in the current $i_{th}$ ball $B_{\acute{d}_1..\acute{d}_4}(p_i)$ and executes the corresponding control.

Then the system is guaranteed to be in $B_{\acute{d}_1..\acute{d}_4}(p_{i+1})$.

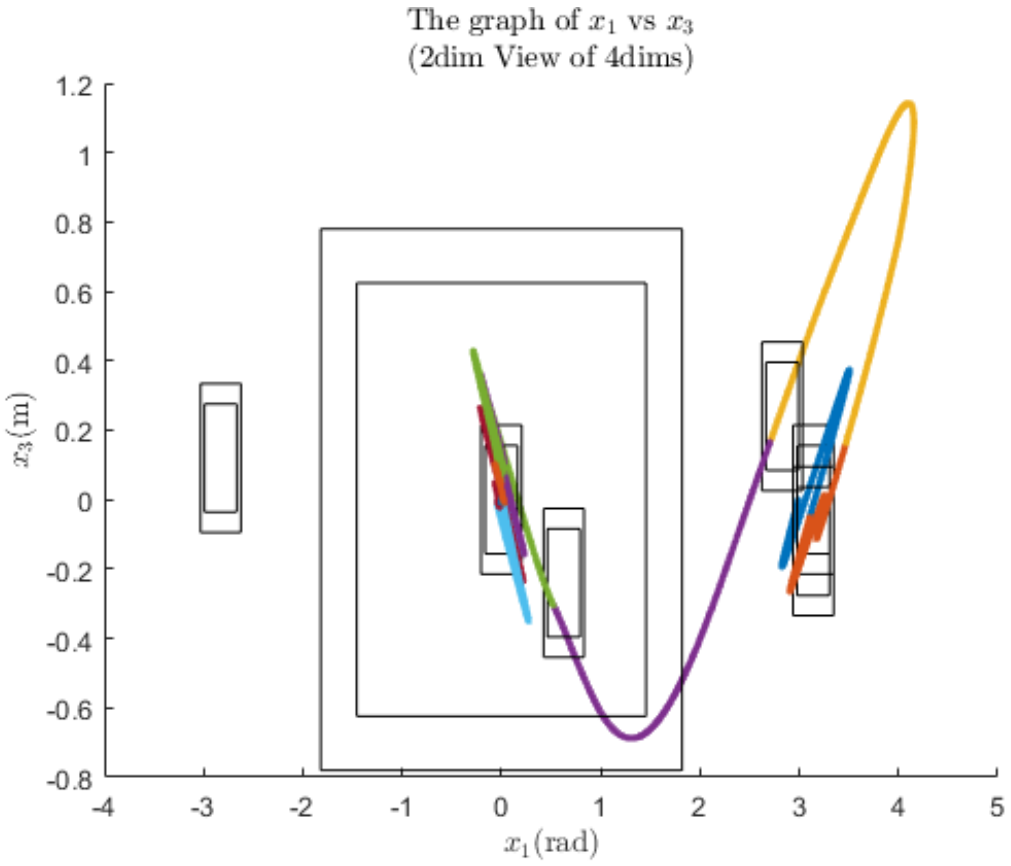


Figure 5.5. Computed path for the Inverted Pendulum Cart from state (pi,0,0,0) to state (0,0,0,0) : Two dimensional view of the four dimensional State path, $x_1$ vs $x_3$.

## 5.4 Contributions

Contributions of this chapter can be summarized as following.

- Computer can search and find a stable control by itself(Purpose and inspiration)
- The same search method can find a solution from stable equilibrium point to unstable equilibrium point

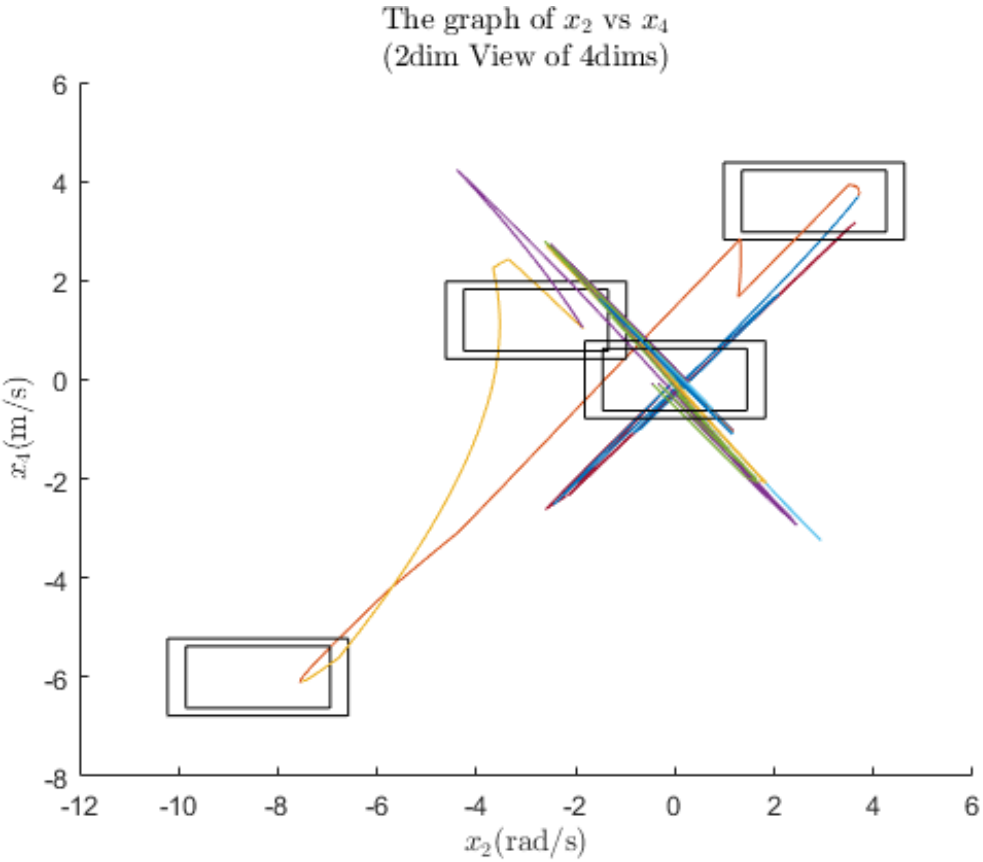


Figure 5.6. Computed path for the Inverted Pendulum Cart from state (pi,0,0,0) to state (0,0,0,0) : Two dimensional view of the four dimensional State path, $x_2$ vs $x_4$.

  - Existing methods run an potential/kinetic energy increasing control and then switch to a linear control.

- Handling the separatrices of dynamical systems with transition systems using feedback control.

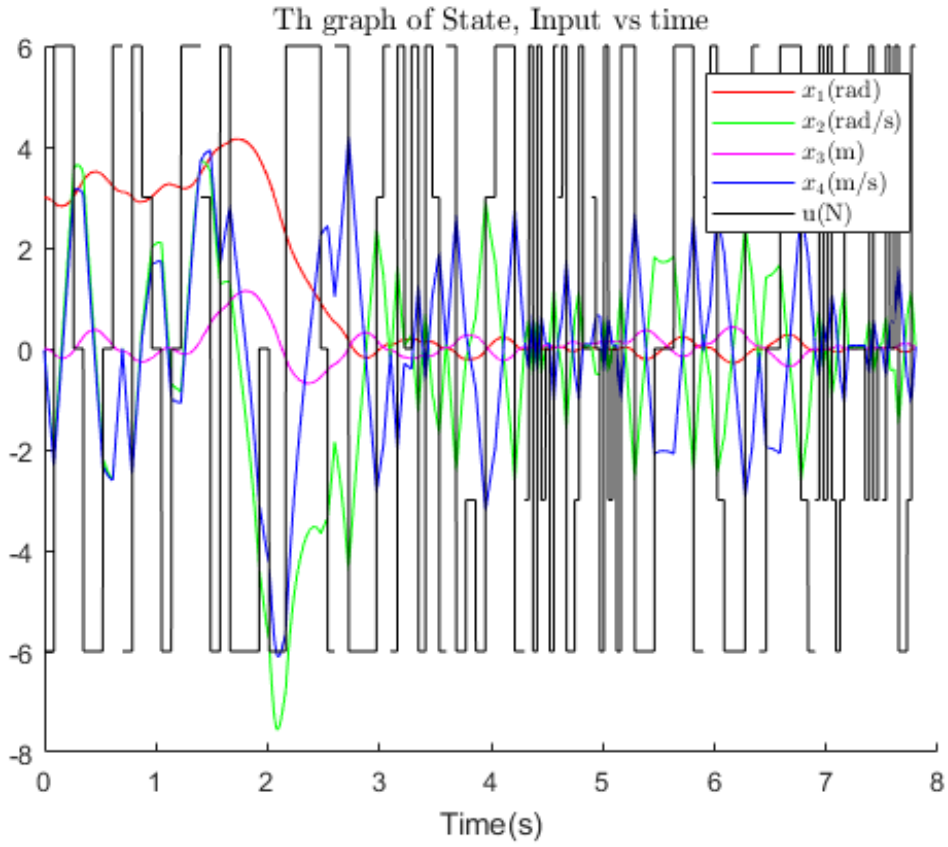


Figure 5.7. Simulation results for Inverted Pendulum Cart: The graph of States and Input vs time

# CHAPTER 6
# FUTURE WORK

There are several ways this work can be improved. Briefly they are the following.

- Noise is assumed to be bounded. No discretization is done on noise. A more detailed noise study can be done. However that will increase the cost of complexity of algorithms.
- The Packet sniffer's detection problem rejects some data packets and presents a possible state path. However the rejected data can be further analyzed. For and example we can check whether the rejected data is just random or belongs to another possible trajectory of the system. That information could be very useful in the attack detection process.
- The optimization problem can be implemented with various algorithms. The complexity of those alorithms haven't been analyzed in this work. To get the results we have been using Answer set prolog and Matlab codes.
- One of the major motivations for this work is making machines take more critical intelligent decision of dynamical systems in it's surrounding. However, in this analysis we start with a model. This makes the analysis highly depend on the given model. Making the machine learn the model from the dynamical system is very interesting.
- Handling the separatrices with continuous feedback control and switch between attractor regions with transition systems

# CHAPTER 7
# CONCLUSIONS

In this text mainly, we considered two problems/applications. They are the problem of attack detection of cyber physical systems and controller synthesis. Consider the problem of attack detection. In the literature there are studies [14] that present theoretical limitations of the attack detection problem. They prove some attacks are theoretically undetectable. However if we use related but outside knowledge there is a better chance of solving even theoretically unsolvable problems considering only the physical dynamical system. That's one reason why putting humans in the process of attack detection is very important. In fact in the todays world a lot of humans are put in this job of monitoring ICS for safety and security purposes. Our attempt is to give some of these human intelligent capabilities regarding reasoning and fusion of information of dynamical systems, to machines.

Consider the second problem, controller synthesis. We humans are intelligent. We study the dynamical system and the control problem. Depending on the problem we may design a controller or a filter etc. It will perform very well and achieve it's purpose. However it does so little to give machines freedom and flexibility to explore, understand the behavior and come up with it's own control to achieve control needs.

The main downside of these methods is extensive computational power demand. There wasn't machines capable of providing the required computational power at the controller synthesis level and implementation level. However recent advancements of parallel processors has changed the environment. For an example modern commercially available GPUs(Second quarter of 2017) has computational power in the range of 10TFLOPS for reasonable prices. On the other hand implementation of controls can be assisted with FPGAs. They provide dedicated hardware to implement fast and constant time control cycles. Some research communities have identified the power of parallel processing and hence considerable advancements can be seen. As the control engineering community we need to identify the power of the combo GPU & FPGA.

In conclusion, I would like to emphasize the fact, today's world need machines to

be able to take decisions regarding critical physical dynamical systems. To achieve this target we need a lot more research done on this field. I hope the work on this text would serve as one fundamental step towards this area of research.

# BIBLIOGRAPHY

[1] Rafael Ramos Regis Barbosa and Aiko Pras. Intrusion detection in scada networks. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 163–166. Springer, 2010.

[2] Andrea Carcano, Alessio Coletta, Michele Guglielmi, Marcelo Masera, I Nai Fovino, and Alberto Trombetta. A multidimensional critical state analysis for detecting intrusions in scada systems. *IEEE Transactions on Industrial Informatics*, 7(2):179–186, 2011.

[3] Hamza Fawzi, Paulo Tabuada, and Suhas Diggavi. Secure state-estimation for dynamical systems under active adversaries. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 337–344. IEEE, 2011.

[4] Antoine Girard. Approximately bisimilar finite abstractions of stable linear systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 231–244. Springer, 2007.

[5] Antoine Girard and George J Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.

[6] Antoine Girard, Giordano Pola, and Paulo Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control*, 55(1):116–126, 2010.

[7] Niv Goldenberg and Avishai Wool. Accurate modeling of modbus/tcp for intrusion detection in scada systems. *International Journal of Critical Infrastructure Protection*, 6(2):63–75, 2013.

[8] Thomas A Henzinger, Rupak Majumdar, and Vinayak S Prabhu. Quantifying similarities between timed systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 226–241. Springer, 2005.

[9] Nazrul Hoque, Monowar H Bhuyan, Ram Charan Baishya, DK Bhattacharyya, and Jugal K Kalita. Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, 40:307–324, 2014.

[10] Robert Mitchell and Ing-Ray Chen. A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)*, 46(4):55, 2014.

[11] Il Moon. Modeling programmable logic controllers for logic verification. *IEEE Control Systems*, 14(2):53–59, 1994.

[12] Thomas H Morris, Bryan A Jones, Rayford B Vaughn, and Yoginder S Dandass. Deterministic intrusion detection rules for modbus protocols. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 1773–1781. IEEE, 2013.

[13] George J Pappas. Bisimilar linear systems. *Automatica*, 39(12):2035–2047, 2003.

[14] Fabio Pasqualetti, Florian Dörfler, and Francesco Bullo. Attack detection and identification in cyber-physical systems. *IEEE Transactions on Automatic Control*, 58(11):2715–2729, 2013.

[15] Giordano Pola, Antoine Girard, and Paulo Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.

[16] Giordano Pola and Paulo Tabuada. Symbolic models for linear control systems with disturbances. In *Decision and Control, 2007 46th IEEE Conference on*, pages 432–437. IEEE, 2007.

[17] Giordano Pola and Paulo Tabuada. Symbolic models for nonlinear control systems: Alternating approximate bisimulations. *SIAM Journal on Control and Optimization*, 48(2):719–733, 2009.

[18] Kalana Pothuwila and Jordan M Berg. Qualitative behavioral analyzer for fault detection and cyber security of control networks. In *ASME 2014 Dynamic Systems and Control Conference*, pages V002T26A005–V002T26A005. American Society of Mechanical Engineers, 2014.

[19] Jan-David Quesel, Stefan Mitsch, Sarah Loos, Nikos Aréchiga, and André Platzer. How to model and prove hybrid systems with keymaera: A tutorial on safety. *International Journal on Software Tools for Technology Transfer*, 18(1):67–91, 2016.

[20] Paulo Tabuada. Approximate simulation relations and finite abstractions of quantized control systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 529–542. Springer, 2007.

[21] Mingsheng Ying and Martin Wirsing. Approximate bisimilarity. *Algebraic Methodology and Software Technology*, pages 309–322, 2000.

[22] Majid Zamani and Rupak Majumdar. A lyapunov approach in incremental stability. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 302–307. IEEE, 2011.

# APPENDIX: CUDA CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 16
# define M_PI           3.14159265358979323846
__global__ void add1( float * pi, float c )
{
    *pi += c;
}

__global__ void add2( float * v1, const float * v2 )
{
    int idx = threadIdx.x;
    v1[idx] += v2[idx];
}


__global__ void myODE( float  out[512][6561][2][4],
float * v1 , float * v2, float * v3 , float * v4,
float* U1 , float*  U2  ,float* U3 , float*  U4,
float* U5 , float*  U6  ,float* U7 , float*  U8  ,
float * dtArray,float * inputAmplitudeArray,
float * g, float * m, float * l,float * b,float * I,
float *CM,float * ftheta, float * h,
float * sparseGrid_MaxPossibleTheta,float * sparseGrid_MaxPossibleVelo,
float * sparseGrid_MaxPossibleCartX,float * sparseGrid_MaxPossibleCartV)


{

   int idx= threadIdx.x;
   int dTi=  blockIdx.x;
   int Ui= blockIdx.y;
```

```
int stepI=1;
float x1=v1[idx];
float x2=v2[idx];
float x3=v3[idx];
float x4=v4[idx];


float T= dtArray[dTi];
int  stepNum =  (int)(T/(*h));
float u=0;
float t=0;

float sin_x1=0,cos_x1=0,denominator=0;
float np1x1=0,np1x2=0,np1x3=0,np1x4=0;



while(stepI< stepNum){

t=stepI*(*h);


if(t<(T*4/8)){
if (t<(T*2/8)){
if (t<(T/8)){


u=inputAmplitudeArray[0]*U1[Ui];


}else{

u=inputAmplitudeArray[0]*U2[Ui];

}

}else{
if (t<(T*3/8)){

u=inputAmplitudeArray[0]*U3[Ui];
```

```
}else{

u=inputAmplitudeArray[0]*U4[Ui];

}


}

}else{

if (t<(T*6/8)){
if (t<(T*5/8)){


u=inputAmplitudeArray[0]*U5[Ui];


}else{

u=inputAmplitudeArray[0]*U6[Ui];

}

}else{
if (t<(T*7/8)){

u=inputAmplitudeArray[0]*U7[Ui];



}else{

u=inputAmplitudeArray[0]*U8[Ui];

}
```

```
}

}


sin_x1=sin(x1);
cos_x1=cos(x1);
denominator=(*CM+(1-cos_x1*cos_x1)**m);

np1x1 = x1+(*h)*x2;
np1x2 = x2+(*h)*(((*CM+*m)*(*g*sin_x1-*ftheta*x2)
-(*l**m*x2*x2*sin_x1+u-*b*x4)*cos_x1)/(*l*denominator));
np1x3 = x3+(*h)*x4;
np1x4 = x4+(*h)*((-(*m)*(*g)*sin_x1*cos_x1+*m*(*l)*x2*x2*sin_x1
+*ftheta*(*m)*x2*cos_x1+u-*b*x4)/denominator);

x1=np1x1;
x2=np1x2;
x3=np1x3;
x4=np1x4;

if (abs(x3)>*sparseGrid_MaxPossibleCartX)
{
stepNum=0;

x2=15;
x3=1;
x4=20;
}
stepI+=1;
}

out[idx][Ui][dTi][0]=x1;
out[idx][Ui][dTi][1]=x2;
out[idx][Ui][dTi][2]=x3;
out[idx][Ui][dTi][3]=x4;
```

```
//v1[idx]=x1;
//v2[idx]=x2;

//}
}


__global__ void distTheta(float theta1,float theta2, float *out)
{

float pi =3.14159265358979323846;

float d=theta1-theta2;

int doCon = 1;

float nPi=-1*pi;

if(d<nPi)
{
    while (doCon)
{
       d=d+2*pi;
       if (nPi<=d )
       {
doCon=0;
       }
    }
}
else if(pi<d)
{
    while (doCon)
{
       d=d-2*pi;
       if (d<=pi )
   {
        doCon=0;
       }
    }
}
```

```
d=abs(d);

//return abs(d);
}


#define gpuPBsize 512
#define numTimeSteps 4
#define gpuOutsizeP4 3
#define numInputs 160000



__global__ void pathFeedbackODE2dist( float  outDist[gpuPBsize]
[numInputs][numTimeSteps],float * ps1s,float * testPoint,
float * pS2negSize,

 float * v1 , float * v2,float * v3 , float * v4,
 float * U1 , float * U2,float * U3 , float * U4,
 float * dtArray,float * inputAmplitudeArray,
 float * g, float * m, float * l,float * b,float * bs,
float * I,float *CM,float * ftheta, float * h,
 int numberofStates, float inputLength,float numberofDT,
int computingBlocksizeGPU, int numXthreads)


 {

   int idx= threadIdx.x;//+(numXthreads*blockIdx.z);

   int dTi=  blockIdx.y;
   int Ui= blockIdx.x+65535*blockIdx.z;


   if ((idx<computingBlocksizeGPU) && (Ui<numInputs))
   {


float sin_x1=0,cos_x1=0,denominator=0;//,staticFriction=0;
```

```
float np1x1=0,np1x2=0,np1x3=0,np1x4=0;
float nd2=0,nd3=0,nd4=0;
int stepI=1;
float x1=v1[idx];
float x2=v2[idx];
float x3=v3[idx];
float x4=v4[idx];

float T= dtArray[dTi];
int  stepNum =  (int)(T/(*h));
float u=0;
float t=0;

float interPointcheck=0;

while(stepI< stepNum){

t=stepI*(*h);

u=inputAmplitudeArray[0]*(U1[Ui]*sin(M_PI*(t/T))
+U2[Ui]*sin(2*M_PI*(t/T))  +U3[Ui]*cos(.5*M_PI*(t/T))
+U4[Ui]*sin(4*M_PI*(t/T)));

sin_x1=sin(x1);
cos_x1=cos(x1);
denominator=(*CM+(1-cos_x1*cos_x1)**m);

np1x1 = x1+(*h)*x2;
np1x2 = x2+(*h)*(((*CM+*m)*(*g*sin_x1-*ftheta*x2)
-(*l**m*x2*x2*sin_x1+u-*b*x4)*cos_x1)/(*l*denominator));
np1x3 = x3+(*h)*x4;
np1x4 = x4+(*h)*((-(*m)*(*g)*sin_x1*cos_x1
+*m*(*l)*x2*x2*sin_x1+*ftheta*(*m)*x2*cos_x1+u-*b*x4)/denominator);

x1=np1x1;
x2=np1x2;
x3=np1x3;
x4=np1x4;

interPointcheck=ps1s[0]-x1;
```

```
if (abs(interPointcheck)>5)
{
stepNum=0;
x2=100;
x3=100;
x4=100;
}


stepI+=1;
}

float tmp2=testPoint[1]-x2;
float tmp3=testPoint[2]-x3;
float tmp4=testPoint[3]-x4;

//////////////////////////////////////////////////


float pi =3.14159265358979323846;

float d=testPoint[0]-x1;

int doCon = 1;

float nPi=-1*pi;

if(d<nPi)
{
    while (doCon)
{
       d=d+2*pi;
       if (nPi<=d )
       {
doCon=0;
       }
    }
}
else if(pi<d)
```

```
{
    while (doCon)
{
       d=d-2*pi;
       if (d<=pi )
   {
        doCon=0;
       }
    }
}


d=(d/pS2negSize[0]);
nd2=(tmp2/pS2negSize[1]);
nd3=(tmp3/pS2negSize[2]);
nd4=(tmp4/pS2negSize[3]);

outDist[idx][Ui][dTi]=(sqrt((d)*(d)+(nd2)*(nd2)+(nd3)*(nd3)+(nd4)*(nd4)));

}

}


__global__ void myMinCopy222(float  mydestination[2][3],int  I1[2][3] ,
int I2[gpuPBsize])

 {int isP1= threadIdx.x+((gpuPBsize/gpuOutsizeP4)*blockIdx.x);
 int tmp=I2[isP1];

mydestination[blockIdx.x][threadIdx.x]=I1[blockIdx.x][threadIdx.x];



 }


 __global__ void myMinCopy(int  mydestination[gpuPBsize],
int  I1[gpuPBsize][numTimeSteps] , int I2[gpuPBsize],int computingBlocksizeGPU)
```

```
 {//int isP1= threadIdx.x+((computingBlocksizeGPU)*blockIdx.x);
 int isP1= threadIdx.x;//+((computingBlocksizeGPU)*blockIdx.x);
 int tmp=I2[isP1];

mydestination[isP1]=I1[isP1][tmp-1];

 }


__global__ void inputRefineFnc(float  bestDist[512],
 int  bestDistInputID[512] ,float * testPoint,float * pS2negSize,

 float * v1 , float * v2,float * v3 , float * v4,
 float * U1 , float * U2,float * U3 , float * U4,float * U5 ,
 float * U6,float * U7 , float * U8,
 int * inputStartLocationforState,int * inputEndLocationforState,
float * selectedDTiforState,
 float * inputAmplitudeArray,
 float * g, float * m, float * l,float * b,float * I,float *CM, float * h)
 {

 float  outDist=0;
 float tmpbestDist=100;
 int tmpbestDistInputID=0;

int idx= blockIdx.x;


int stepI=0;


float T= selectedDTiforState[idx];
int  stepNum =  (int)(T/(*h));
float u=0;
float t=0;

int Ui=0;
if (inputStartLocationforState[idx]!=0){
for(Ui=inputStartLocationforState[idx]-1;
Ui<=inputEndLocationforState[idx]-1;Ui++){
```

```
//for(Ui=1;Ui<=25;Ui++){

stepI=1;
float x1=v1[idx];
float x2=v2[idx];
float x3=v3[idx];
float x4=v4[idx];

while(stepI< stepNum){

t=stepI*(*h);

// Check here if Us are different to previous Us..
// check amplitude multiplication


if(t<(T*4/8)){
if (t<(T*2/8)){
if (t<(T/8)){


u=inputAmplitudeArray[0]*U1[Ui];


}else{

u=inputAmplitudeArray[0]*U2[Ui];

}

}else{
if (t<(T*3/8)){

u=inputAmplitudeArray[0]*U3[Ui];



}else{

u=inputAmplitudeArray[0]*U4[Ui];
```

```
}


}

}else{

if (t<(T*6/8)){
if (t<(T*5/8)){


u=inputAmplitudeArray[0]*U5[Ui];


}else{

u=inputAmplitudeArray[0]*U6[Ui];

}

}else{
if (t<(T*7/8)){

u=inputAmplitudeArray[0]*U7[Ui];



}else{

u=inputAmplitudeArray[0]*U8[Ui];

}


}

}
```

```
x1 = x1+(*h)*x2;
x2 = x2+(*h)*(((*CM+*m)*(*g*sin(x1))-(*l**m*x2*x2*sin(x1)+u)*cos(x1))
/(*l*(*CM+(1-cos(x1)*cos(x1))**m)));
x3 = x3+(*h)*x4;
x4 = x4+(*h)*((-(*m)*(*g)*sin(x1)*cos(x1)+*m*(*l)*x2*x2*sin(x1)+u)
/(*CM+(1-cos(x1)*cos(x1))**m));

stepI+=1;
}

float tmp2=testPoint[1]-x2;
float tmp3=testPoint[2]-x3;
float tmp4=testPoint[3]-x4;

//////////////////////////////////////////////////

float pi =3.14159265358979323846;

float d=testPoint[0]-x1;

int doCon = 1;

float nPi=-1*pi;

if(d<nPi)
{
    while (doCon)
{
       d=d+2*pi;
       if (nPi<=d )
       {
doCon=0;
       }
    }
}
else if(pi<d)
{
    while (doCon)
{
       d=d-2*pi;
```

```
        if (d<=pi )
   {
        doCon=0;
        }
    }
}

d=abs(d);


if (tmpbestDist>outDist){
tmpbestDist=outDist;
tmpbestDistInputID=Ui;}

}

bestDist[idx]=tmpbestDist;
bestDistInputID[idx]=tmpbestDistInputID+1;

//// The end of code!

}
}
```