# Multi-Agent Coordination and Control under Information Asymmetry with Applications to Collective Load Transport

by

Karthik Kambam

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2018 by the
Graduate Supervisory Committee:

Wenlong Zhang, Co-chair
Angelia Nedich, Co-chair
Yi Ren

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Coordination and control of *Intelligent Agents* as a team is considered in this thesis. Intelligent agents learn from experiences, and in times of uncertainty use the knowledge acquired to make decisions and accomplish their individual or team objectives. Agent objectives are defined using cost functions designed uniquely for the collective task being performed. Individual agent costs are coupled in such a way that group objective is attained while minimizing individual costs. *Information Asymmetry* refers to situations where interacting agents have no knowledge or partial knowledge of cost functions of other agents. By virtue of their intelligence, i.e., by learning from past experiences agents learn cost functions of other agents, predict their responses and act adaptively to accomplish the team's goal.

Algorithms that agents use for learning others' cost functions are called *Learning Algorithms*, and algorithms agents use for computing actuation (control) which drives them towards their goal and minimize their cost functions are called *Control Algorithms*. Typically knowledge acquired using learning algorithms is used in control algorithms for computing control signals. Learning and control algorithms are designed in such a way that the multi-agent system as a whole remains stable during learning and later at an equilibrium. An equilibrium is defined as the event/point where cost functions of all agents are optimized simultaneously. Cost functions are designed so that the equilibrium coincides with the goal state multi-agent system as a whole is trying to reach.

In collective load transport, two or more agents (robots) carry a load from point A to point B in space. Robots could have different control preferences, for example, different actuation abilities, however, are still required to coordinate and perform load transport. Control preferences for each robot are characterized using a scalar parameter $\theta_i$ unique to the robot being considered and unknown to other robots.

With the aid of state and control input observations, agents learn control preferences of other agents, optimize individual costs and drive the multi-agent system to a goal state.

Two learning and Control algorithms are presented. In the first algorithm(LCA-1), an existing work, each agent optimizes a cost function similar to 1-step receding horizon optimal control problem for control. LCA-1 uses recursive least squares as the learning algorithm and guarantees complete learning in two time steps. LCA-1 is experimentally verified as part of this thesis.

A novel learning and control algorithm (LCA-2) is proposed and verified in simulations and on hardware. In LCA-2, each agent solves an infinite horizon linear quadratic regulator (LQR) problem for computing control. LCA-2 uses a learning algorithm similar to line search methods, and guarantees learning convergence to true values asymptotically.

Simulations and hardware implementation show that the LCA-2 is stable for a variety of systems. Load transport is demonstrated using both the algorithms. Experiments running algorithm LCA-2 are able to resist disturbances and balance the assumed load better compared to LCA-1.

*For my parents who gave me a free hand in deciding what I wanted to do with my life.*

ACKNOWLEDGMENTS

I would like to thank my advisor Dr.Wenlong Zhang for being patient with me and showing confidence in me in times of self-doubt, Dr.Angelia Nedich for agreeing to be my co-chair from Electrical Engineering, Dr.Yi Ren for agreeing to be a part of my thesis committee, and Dr.Armando Rodriguez for inspiring many students at ASU to take up controls as their specialization. I would also like to thank my friends Naveen, Praveen and Chaitanya who were a constant source of encouragement throughout my program at ASU.

TABLE OF CONTENTS

Page

LIST OF FIGURES ........ viii

CHAPTER

1 INTRODUCTION ........ 1

1.1 Multi-agent Systems ........ 1

1.2 Intention Estimation ........ 1

1.3 MAS control using Cost Functions ........ 2

1.4 Problem with Information Asymmetry ........ 3

1.5 Modelling Agent Intention ........ 4

1.6 The First Learning and Control Algorithm ........ 4

1.7 The Second Learning and Control Algorithm ........ 5

1.8 Collective Load Transport and Considerations ........ 6

1.9 Problem statement and Motivation ........ 6

1.10 Contributions of the Work ........ 7

2 LEARNING AND CONTROL ALGORITHMS FOR MAS WITH INFORMATION ASYMMETRY ........ 8

2.1 NOMENCLATURE ........ 8

2.2 The Multi-agent Model ........ 8

2.3 Simultaneous Dynamic Game, Response Curves and Nash Equilibrium ........ 9

2.3.1 Complications of Simultaneous Estimation and Action in MAS ........ 11

2.4 Learning and Control Algorithm 1 ........ 14

2.5 Learning and Control Algorithm 2 ........ 16

2.5.1 Preliminaries ........ 17

CHAPTER Page


2.5.2 Process steps - Control Algorithm: ........................ 20

2.5.3 Process steps - Learning Algorithm: ........................ 21

3 EXPERIMENTAL SETUP ........................ 25

3.1 Modelling of Hercules Robot ........................ 26

3.2 Position data ........................ 26

3.3 Two Robot System ........................ 28

3.4 Choice of matrices P or Q, Initial conditions and Preferences ........................ 29

4 SIMULATION AND EXPERIMENTAL RESULTS ........................ 30

4.1 Simulation Results and Comparison ........................ 31

4.1.1 System - 1 ........................ 31

4.1.2 System - 2 ........................ 35

4.1.3 System-3 ........................ 39

4.1.4 System - 4 ........................ 43

4.1.5 Simulation results for the 2-Agent mobile robot model ........................ 45

4.1.6 Data from Experiments - LCA-2 ........................ 48

4.1.7 Data from Experiments - LCA-1 ........................ 65

4.1.8 Comparison: LCA-2 v LCA-1 ........................ 69

4.1.9 Reaction to disturbances after learning ........................ 69

4.1.10 Stability of MAS with small P or Q elements ........................ 71

4.1.11 LCA-2 v LCA-1 under learning failure ........................ 74

5 SUMMARY AND FUTURE WORK ........................ 78

5.1 Summary ........................ 78

5.2 Future work ........................ 79

CHAPTER Page

REFERENCES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81

APPENDIX

A PYTHON PROGRAMS USED FOR EXPERIMENTAL VERIFICATION 83

A.1 Python code used for running Learning and Control Algorithm 2: . . 84

A.2 Python code used for running Learning and Control Algorithm 1: . . 92

## LIST OF FIGURES

Figure | Page


2.1 Response Curves and Nash Equilibrium ..... 10
2.2 Blame Me Learning Strategy ..... 12
2.3 Blame All Learning Strategy ..... 13
2.4 LQR Gain Variation with R for a Stable System ..... 18
2.5 LQR Gain Variation with R for an Unstable System ..... 19
2.6 Learning Process - Scenario 1 ..... 22
2.7 Learning Process - Scenario 2 ..... 23
3.1 Hercules Robot ..... 25
3.2 Two Robot Team ..... 28
4.1 State Evolution Comparison LCA-1 v LCA-2 ..... 32
4.2 Control Evolution Comparison LCA-1 v LCA-2 ..... 33
4.3 System-1 Parameter Learning Evolution with LCA-2 ..... 34
4.4 State Evolution Comparison LCA-1 v LCA-2 ..... 36
4.5 System-2 Parameter Learning Evolution with LCA-2 ..... 37
4.6 Control Evolution Comparison LCA-1 v LCA-2 ..... 38
4.7 State Evolution Comparison LCA-1 v LCA-2 ..... 40
4.8 Control Evolution Comparison LCA-1 v LCA-2 ..... 41
4.9 System-3 Parameter Learning Evolution with LCA-2 ..... 42
4.10 System-4 State Evolution with LCA-2 ..... 43
4.11 System-4 Parameter Learning Evolution with LCA-2 ..... 44
4.12 System-4 Control Input Sequence with LCA-2 ..... 45
4.13 Mobile Robot Simulation - State Evolution LCA-2 ..... 46
4.14 Mobile Robot Simulation - Parameter Learning LCA-2 ..... 46
4.15 Mobile Robot Simulation - Output Evolution LCA-2 ..... 47

Figure Page


4.16 Mobile Robot Simulation - Control Evolution LCA-2 . . . . . . . . . . . . . . . . . 47

4.17 Trial-1 Mobile Robot Experiment - State Evolution . . . . . . . . . . . . . . . . . . . 49

4.18 Trial-1 Mobile Robot Experiment - Parameter Evolution . . . . . . . . . . . . . . 50

4.19 Trial-1 Mobile Robot Experiment - Output Evolution . . . . . . . . . . . . . . . . . 51

4.20 Trial-1 Mobile Robot Experiment - Control Input Evolution . . . . . . . . . . . 52

4.21 Trial-2 Mobile Robot Experiment - State Evolution . . . . . . . . . . . . . . . . . . . 53

4.22 Trial-2 Mobile Robot Experiment - Parameter Evolution . . . . . . . . . . . . . . 54

4.23 Trial-2 Mobile Robot Experiment - Output Evolution . . . . . . . . . . . . . . . . . 55

4.24 Trial-2 Mobile Robot Experiment - Control Input Evolution . . . . . . . . . . . 56

4.25 Trial-3 Mobile Robot Experiment - State Evolution . . . . . . . . . . . . . . . . . . . 57

4.26 Trial-3 Mobile Robot Experiment - Parameter Evolution . . . . . . . . . . . . . . 58

4.27 Trial-3 Mobile Robot Experiment - Output Evolution . . . . . . . . . . . . . . . . . 59

4.28 Trial-3 Mobile Robot Experiment - Control Input Evolution . . . . . . . . . . . 60

4.29 Trial-4 Mobile Robot Experiment - State Evolution . . . . . . . . . . . . . . . . . . . 61

4.30 Trial-4 Mobile Robot Experiment - Parameter Evolution . . . . . . . . . . . . . . 62

4.31 Trial-4 Mobile Robot Experiment - Output Evolution . . . . . . . . . . . . . . . . . 63

4.32 Trial-4 Mobile Robot Experiment - Control Input Evolution . . . . . . . . . . . 64

4.33 Mobile Robot Experiment - State Evolution . . . . . . . . . . . . . . . . . . . . . . . . . 66

4.34 Mobile Robot Experiment - Output Evolution . . . . . . . . . . . . . . . . . . . . . . . 67

4.35 Mobile Robot Experiment - Control Input Evolution . . . . . . . . . . . . . . . . . 68

4.36 Reaction to a Disturbance LCA-1 v LCA-2 . . . . . . . . . . . . . . . . . . . . . . . . . . 70

4.37 Example 1: Stability for Small P,Q Matrices - LCA-1 vs LCA-2 . . . . . . . 72

4.38 Example 2: Stability for Small P,Q Matrices - LCA-1 vs LCA-2 . . . . . . . 73

4.39 System-1 Learning Failure LCA-1 vs LCA-2 . . . . . . . . . . . . . . . . . . . . . . . . . 75

Figure Page

4.40 System-2 Learning Failure LCA-1 vs LCA-2 . . . . . . . . . . . . . . . . . . . . . . . . . . 76

4.41 System-4 Learning Failure LCA-1 vs LCA-2 . . . . . . . . . . . . . . . . . . . . . . . . . . 77

# Chapter 1

# INTRODUCTION

## 1.1 Multi-agent Systems

Agents in a Multi-agent system(MAS) are subsystems in their own right, with their own states, dynamics, inputs, and outputs. Agents can simply be software agents e.g. threads running simultaneously, physical engineered systems like robots, or systems occurring in nature like birds in a flock, fish in a schooling or a team of ants moving a load. Two or more such agents may decide to cooperate and perform tasks which otherwise are not possible. Dynamics of interactions between cooperating subsystems and their control is dealt with in MAS. Agent-Agent interactions affect shared states between agents or agent specific states. In some instances, a group of active agents (agents with actuation) comes together to work on a passive system (agent with no actuation) changing its state as desired. An example of such a scenario is the case of Collective load transport, where a group of robots works on moving a large mass. Agents in a MAS can be continuously interacting with each other or can have event triggered interactions. A team of ants rolling a heavy load are interacting with each other in a continuous fashion, whereas autonomous cars on road interact only for safety when one car moves into the area of influence of another.

## 1.2 Intention Estimation

Agents sense outputs of interest and act to alter shared states during the interaction. In some cases, agents know the intention of other agents, whereas in rest agents are required to estimate intentions of other agents. Knowledge of agent intention is

paramount since agents are cooperating with each other. If agents don't have knowledge of other agent intentions, they can't cooperate. In the case of a homogeneous group like bird flock, where every bird knows what the other birds are trying to do, there is little intention left to be estimated. Consider the case of an autonomous car interacting with a manually driven car. Both systems, manual and auto need to constantly estimate each other's intention and act accordingly. This is referred to as Human-Robot Interaction (HRI). The same is the case when two autonomous robots interact with each other, termed as Robot-Robot Interaction (RRI).

As opposed to a bird in a bird flock, a driver in a HRI or RRI is more "selfish" in the sense that the needs of the driver are put first and the needs of an interacting driver are accommodated only to an extent that their safety is guaranteed. In HRI and RRI, objectives of individual agents, agents they interact with, and the topology of interactions are constantly changing. Hence a fixed control law for all agents will be detrimental to system stability and performance. However, if the MAS involves "bird" like agents, where all agents are identical, have the same goal, are less "selfish", and there is no need for intention estimation, fixed decentralized control laws can be defined for all agents to make the MAS work e.g., Consensus Algorithms (Yu and Nagpal (2011),Ren *et al.* (2007), Mesbahi and Egerstedt (2010), Ren and Beard (2008), Fax and Murray (2004)).

## 1.3 MAS control using Cost Functions

For HRI and RRI interactions modeled as multi-agent interactions, it is desired that MAS is controlled by defining and solving individual cost functions (Semsar-Kazerooni and Khorasani (2009)) capturing agent preferences. Individual cost functions are designed in such a way that the desired behavior evolves out of agent interactions. Cost functions offer more flexibility in terms of adapting to changing

environment compared to fixed distributed control laws. When two agents come into contact for the first time, they may have no or only partial knowledge of cost functions of other agents. Such a scenario is termed "Information Asymmetry", a term from economics theory. Agents require knowledge of other agent cost functions to estimate agent intentions. Hence a control algorithm and a learning algorithm are employed at each agent for estimating cost functions of interacting agents and control. Control algorithms make use of cost functions defined at each agent and knowledge of other agent cost functions in computing control inputs to be applied at each agent. At each agent, the learning and control algorithms work hand in hand ensuring collective task execution, individual objective satisfaction, and stability of the MAS. Two such learning and control algorithms are presented and evaluated.

### 1.4 Problem with Information Asymmetry

Under Information Asymmetry, agents simultaneously estimate each others' intentions and act upon estimated values. Simultaneous estimation may result in scenarios where agent intention estimates don't converge to true intentions with time. This can arise due to two factors: 1. The agent's wrong estimation (wrong prediction) and 2. Other agents' wrong estimation and subsequent improper actions (Liu *et al.* (2016)). With simultaneous estimation in a MAS, all agents can be shooting in the dark all the time with their estimates, making the system useless in a transient state for perpetuity. The only way to make a MAS with Information Asymmetry work is to have all agent estimates agree upon a certain reference value at all agents, even if it isn't the true intention. Agents can compare the estimated reference value with observations and compute corrections needed for each agent intention estimation. This is achieved by having the same learning algorithm run at all agents with the same initial estimates, and with agents estimating their own intentions in addition to esti-

mating others. Agents estimate their own intentions even though they know them to understand how they are being incorrectly estimated at other agents.

## 1.5 Modelling Agent Intention

Quadratic costs are assumed in both LCAs'. All agents have the same state penalizing cost term, while their control penalizing cost terms differ. The sum of state and control penalizing costs is solved for a control input which minimizes the cost at each agent. A scalar $\theta_i$ defined for each agent $i$ differentiates agent cost functions. It is assumed that parameter $\theta_i$ for each agent models its preferences or intentions. When agents are learning other agent intentions or preferences they are essentially estimating other agent $\theta_i$ parameter values. The objective of learning algorithms is thus to estimate the values $\theta_i$ at all agents for all agents iteratively or otherwise and ensure their convergence to true values.

## 1.6 The First Learning and Control Algorithm

First LCA (Liu *et al.* (2016)) works for all control-affine systems including nonlinear systems. Control algorithm solves a one-step receding horizon optimal cost function with different $\theta_i$ values at each agent (Mattingley *et al.* (2011)) for every time step. Learning algorithm in this approach guarantees $\theta_i$ estimation convergence in two time steps of all agents, at all agents. Parameter updates at each time step can be computed all at once, with an analytic expression. Learning and control algorithms run at the same rate. Agents participate in a simultaneous cooperative dynamic game at all time steps. Individual cost functions are optimized for an estimate of other agent $\theta_i$'s. Control inputs computed at every instant are thus the best an agent do with its estimates. When all estimates converge to true $\theta_i$ values, the control inputs converge to a Nash equilibrium (Marden and Shamma (2018)), where agents cannot have a

lower cost by deviating from their control computed alone. The Nash equilibrium is designed to be stable by an appropriate choice of state weighting cost matrix.

## 1.7 The Second Learning and Control Algorithm

Second LCA requires that the MAS is Linear and Time Invariant (LTI). Control algorithm at each agent solves a cost function that resembles Infinite Horizon Discrete Linear Quadratic Regulator (IHDLQR (Lewis *et al.* (2012))) for the *whole system* with *estimates* for other *agent parameters*, however applies the control input applicable to the agent from the *whole* system control computed. Since the control computation considers all time steps to infinity as compared to only one step in the first LCA, the second LCA has better stability bounds. This makes it better suited for collective load transport. As with the first LCA, the same learning algorithm runs at all agents and for all agents with the same initial estimates. Learning algorithm guarantees asymptotic convergence to true values. The learning algorithm can be run multiple times per a run of control algorithm, i.e., per an observation of true control and state values, the learning algorithm can be run multiple times for faster convergence. There is no analytic expression for parameter estimate updates for all agents at once, agents learn parameters of each agent one by one. With the cost function considered, the system reaches Nash equilibrium, where every agent is optimal for ever other agents' control when there is parameter convergence to true values and all agents are essentially solving the same IHDLQR and generating the same outputs for all agents. As a Discrete Linear Quadratic Regulator always results in a stable closed loop state feedback system for controllable and observable open loop systems, the second LCA guarantees closed loop stability as soon as parameter estimates converge to true values at all agents.

## 1.8 Collective Load Transport and Considerations

The objective of collective load transport as a task is that the load is transported from point A to point B in space by a team of agents which can be a team of Mobile robots, Quadcopters, or natural agents like ants. A requirement of load transport is that the agents don't deviate as much from their relative starting positions that their evolving relative deviations don't make load carrying a physically impossible task. For example, in a two agent case, with just one degree of motion possible, the allowed lateral difference between two robot positions during the entire course of motion is limited by the dimensions of a load being carried. Lyapunov local stability (Slotine *et al.* (1991)) definition applies to relative deviation between individual agent positions in collective load transport. The control algorithm should guarantee that for a given load dimension, there are threshold initial relative position deviations, which when not violated will always lead to a stable system.

## 1.9 Problem statement and Motivation

With increase in Human-Robot interactions and Robot-Robot interactions foreseen in future, there is a need for learning and control algorithms which estimate intentions and compute controls facilitating agent interactions in carrying out tasks collectively. The goal of this thesis is to analyze and evaluate learning and control algorithms in the literature, propose, analyze, evaluate new learning and control algorithms, and provide a comparative analysis between existing and new algorithms. The algorithms are chosen with practical applicability (Hardware implementation) in mind. This work is motivated in large parts from the work presented in Liu *et al.* (2016).

## 1.10 Contributions of the Work

- Implementation of work presented in Liu *et al.* (2016) on a team of mobile robots. Evaluation of the work's practical implementation for the application of collective load transport.

- A new learning and control algorithm is presented and implemented on a team of mobile robots. It is found that since the new algorithm considers infinite step predictions from the current instant, it offers better stability compared to the first algorithm.

- The new algorithm required inquiry of several fundamental questions related to Linear Quadratic Regulator and its related theory. The observations presented will find use in many applications given the extensive use of LQR optimal control in industry.

- By a simple change in cost functions to track a trajectory instead of origin, the new algorithm proposed can be extended to tracking optimal multi-agent system control applications.

The rest of the document is organized as follows: Chapter 2 details upon LCA-1 and LCA-2 - theory and intuition behind each method are explained; Chapter 3 presents robot modeling and experimental setup used; Chapter 4 presents simulation and experimental results of LCA-1 and LCA-2, and their comparative analysis; Chapter 5 summarizes and concludes present work. Possible directions for improvement are also presented.

Chapter 2

# LEARNING AND CONTROL ALGORITHMS FOR MAS WITH INFORMATION ASYMMETRY

## 2.1 NOMENCLATURE

| | |
|---|---|
| $\underline{q}$ | Observed value of physical quantity q |
| $q^{o}\left(k\right)$ | Optimal value of quantity $q$ at time instant $k$ |
| $\hat{q}, \tilde{q}$ | Estimated value of quantity $q$, Error in the estimate of q |
| $\hat{q}_j^i$ | Estimated value of quantity $q$ at agent $i$ for agent $j$ |

## 2.2 The Multi-agent Model

Consider a $n$ state MAS comprised of $N$ agents, with dynamics described by the equation:

$$x(k+1) = f(x(k)) + \sum_{i=1}^{N} h_i(x(k))u_i(k), \tag{2.1}$$

where $x \in \mathbb{R}^{n\times 1}$ denotes system state, $u_i \in \mathbb{R}^{r_i\times 1}$ is the control input of agent $i$, and $h_i \in \mathbb{R}^{n\times r_i}$ control affine matrix of agent $i$. $f(x(k)) \in \mathbb{R}^{n\times n}$ is the system's characteristic function. For a linear system this reduces to a matrix.

An equivalent representation is given by the equation:

$$x(k+1) = f(x(k)) + \mathbf{B}\left(k\right)\mathbf{U}\left(k\right), \tag{2.2}$$

where $B \in \mathbb{R}^{n\times r}$ is formed by all individual control affine matrices - $h_i's$ stacked laterally, $B_i\left(k\right) = h_i\left(k\right)$, $U \in \mathbb{R}^{r\times 1}$ is all individual control inputs - $u_i's$ stacked vertically, and $r = \sum_i r_i$. Another representation when the MAS considered is linear

is given by

$$x\left(k+1\right)=A_k\left[x\left(k\right)\right]+\sum_{i=1}^{N}h_i\left[x\left(k\right)\right]u_i\left(k\right)=A_k\left[x\left(k\right)\right]+\mathbf{B}\left(k\right)\mathbf{U}\left(k\right), \tag{2.3}$$

where $A_k \in \mathbb{R}^{n\times n}$ is the system characteristic matrix. It is assumed that the system state is accessible to all agents at all instants of time. It is assumed that control input $u_i\left(k\right)$ applied at agent $i$ is visible at agent $j$ at time instant $k+1$. For a time invariant linear system, the matrices $A_k$ and $B\left(k\right)$ of 2.3 don't change with time. Agents solve cost functions defined for each agent of the generic form:

$$u_i=\underset{u_i}{\operatorname{argmin}}\; J_i(x,u_i,u_{-i},G_i), \tag{2.4}$$

where $u_{-i}$ is the control input of all agents other than agent $i$, and $G_i \in \mathbb{R}^{n\times 1}$ denotes the goal state of agent $i$. It is assumed that $G_i$ is same for all agents given by $G$. In regulation problems $G$ is assumed to be at the origin without loss of generality. For tracking problems, $G$ is time varying. Regulation problem is considered in this thesis, however it can easily be extended to tracking problem with minor changes in cost function. In cases where system states are not accessible, the MAS output as a whole $Y$ is related to the state by the static output equation:

$$Y\left(k\right)=Cx\left(k\right)+D\mathbf{U}\left(k\right), \tag{2.5}$$

where $C$ is the output matrix and $D$ is the feed forward matrix of the system as a whole.

## 2.3 Simultaneous Dynamic Game, Response Curves and Nash Equilibrium

Consider equation 2.4, it is clear that the solution $u_i\left(k\right)$ is a function of $u_{-i}\left(k\right)$ at all other agents. i.e., at all agents the control input chosen is a function of control inputs chosen at other agents. If agent $j$ decides to choose a random $u_j\left(k\right)$ to be its

control, it will impact solutions to controls (equation 2.4) at all other agents. This is a simultaneous dynamic game and the curves $u_i$, control input at agent $i$ as functions of $u_{-i}$, control inputs agent all agents except $i$, called response curves. Consider two agents' case for simplicity. Let $u_1$ and $u_2$ be the control inputs computed at each agent as solutions of equation 2.4. The plot of $u_1$, obtained by solving equation 2.4 when $u_{-1}$ (which is $u_2$) is varied from a min to a max allowed values is the response curve for agent 1. As the name indicates, response curves plot the optimal choice (control input) an agent can make as a response to control action choices made by interacting agents. Figure 2.1 shows response curves for agents 1 and 2. The point

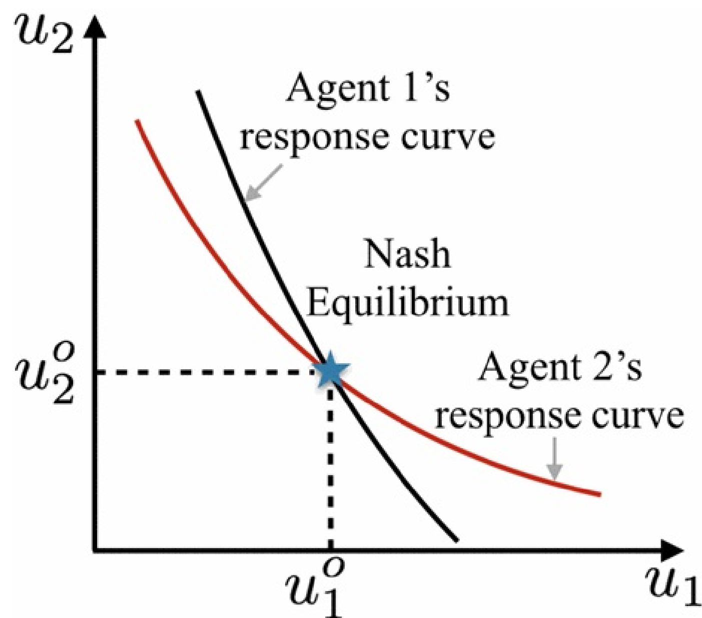



**Figure 2.1:** Response curves and Nash equilibrium Liu *et al.* (2016)

of intersection of response curves is the Nash equilibrium of the system. The point of intersection is the point where agent 1 control response ($u_1^o$) optimizes value $J_1$ for $u_2 = u_2^o$ and vice versa. i.e., $u_2^o$ is the optimizes the value $J_2$ for $u_1 = u_1^o$. Agents can't achieve a lower cost by deviating from the Nash equilibrium point alone. While Nash equilibrium gives a profile of controls that simultaneously optimizes individual agent costs, it is not guaranteed to be stable.

### 2.3.1 *Complications of Simultaneous Estimation and Action in MAS*

With estimation and subsequent control action in conventional systems, typically one of the subsystems knows or can learn the absolute truth about the world around it. Observations made at such subsystems are used for correcting estimates made at interacting subsystems which are still *figuring out* the world around them. Centralized design philosophies such as Adaptive control (Landau *et al.* (1998) and Reinforcement learning (Littman (1994) work well such scenarios (Liu *et al.* (2016)). In situations like the one described, the subsystem that is still *figuring things* assumes that the other subsystems are optimal given their observations. i.e., the observation made at *ground truth* system is their optimal value and any deviation in the estimation of *ground truth* system's quantity of interest from observations is considered to be an estimation fault of the fledgling subsystem. It (subsystem that is learning) essentially takes the blame on itself for the error in estimation. This is termed as the **BLAME ME** strategy (Liu *et al.* (2016)). With simultaneous estimation and control in MAS, learning strategies should consider the fact that all agents are imperfect and are to be blamed for errors in estimates. Convergence of Estimates to true values is possible when all sources of error in estimates are accounted for in the learning model. This approach where all interacting agents are *blamed* for estimation error is called the **BLAME ALL** (Liu *et al.* (2016)) strategy. In the case of MAS, where all agents(subsystems) are still *figuring out* the world around them BLAME ME strategy for learning will lead to instability as described in the following paragraph.

Consider Figure 2.2. The MAS considered has two agents. The agents try to be at Nash Equilibrium every time instant, where each of them is optimal given the others control input (Marden and Shamma (2018)). As opposed to a conventional system,

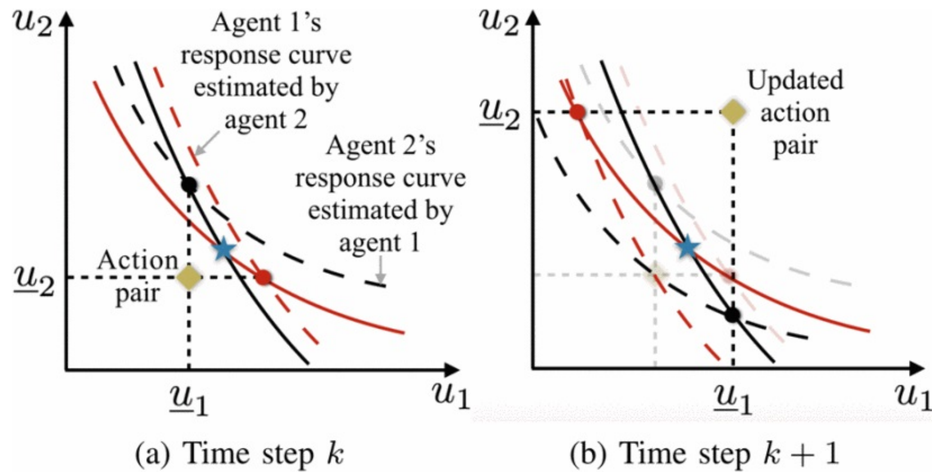



**Figure 2.2:** BLAME ME learning strategy Liu *et al.* (2016)

where one subsystem knows the absolute truth, here both the agents are required to estimate others. Solid black and red lines represent the response curves of agent 1 and agent 2 respectively. Dashed black and red lines represent the response curves estimated at agent 1 and agent 2 for agent 2 and agent 1 respectively. Agent 1 wants the MAS to get to Nash Equilibrium and the possibility it sees is the black solid dot shown in the left (a) Figure and hence chooses its control action to be $\underline{u}_1$. Agent 2 follows the same reasoning and chooses $\underline{u}_2$. The Yellow dot in part-a of Figure 2.2 represents the Action pair at time step $k$ which is an ordered pair of actions taken by all agents at the considered instant. The blue dot represents the true Nash Equilibrium the MAS would achieve if there were no information asymmetry. Since in Blame-me strategy it is assumed that an agent is optimal given its observation, agent 1 updates its estimate of agent 2's response curve to go through the action pair. This is because agent 1 observed that at it's chosen optimal value $\underline{u}_1$, the optimal value of agent 2 is $\underline{u}_2$. Hence agent 1 as per blame all strategy knows for a fact that its estimate of agent 2's response curve should go through the action pair ($\underline{u}_1$, $\underline{u}_2$). Agent 2 follows the same reasoning strategy and updates its estimate of Agent 1's response curve to go through the action pair point at time step $k$ as shown in part

b of the Figure 2.2. At time step $k+1$ Agent 1 considers the point of intersection of updated agent 2 estimated response with its own response curve to be the Nash equilibrium and decides to apply to control input $\underline{u}_1$ as shown in part b of the Figure. Agent 2 follows the same strategy. As a result, the action pair at time-step $k+1$ moved farther from Nash Equilibrium. This approach hence may drive the system away from Nash equilibrium, with response estimates never converging to their true values.

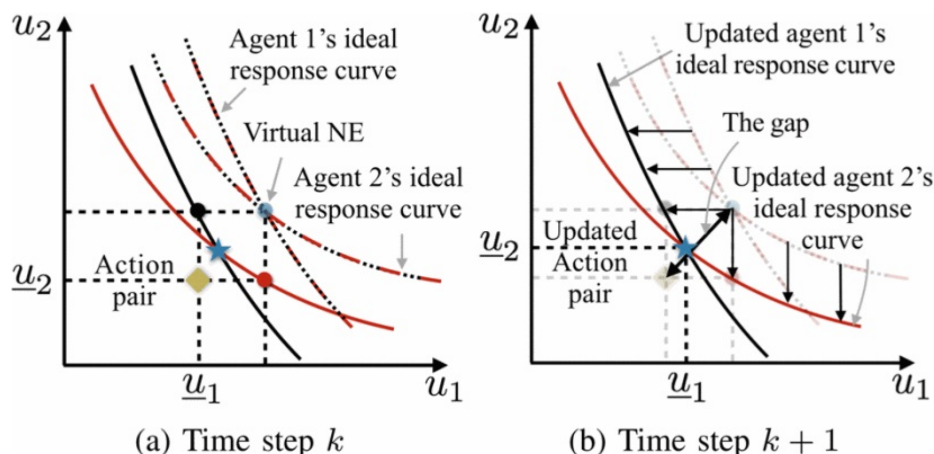



**Figure 2.3:** BLAME ALL learning strategy Liu *et al.* (2016)

Consider Figure 2.3. In BLAME ALL strategy, all agents are considered to be imperfect. Agents are optimal given their estimates of other agents. To predict the action of an agent accurately, how it estimates 'me' should be considered (Liu *et al.* (2016)). Hence agents in addition to estimating response curves of others estimates their own as well. Estimation of response curve for an agent at all agents start with the same initial value and employ the same learning algorithm. Therefore, response curve estimates for an agent at all agents will all be the same as the same observations are made at every agent. The red and black dotted lines shown in part (a) of the

Figure are estimated responses of Agent 1 and Agent 2, accessible at all agents. The point of intersection of estimated responses of all agents is called the virtual Nash equilibrium (light blue dot in part(a) of the Figure 2.3). The best action an agent can take is to assume all agents take to their respective control profiles at virtual Nash equilibrium and choose the corresponding action on its response curve. Agent 1 and Agent 2 follows this approach and choose an action pair (yellow dot) as shown in part (a) of the Figure. At time step $k$, Agent 1's belief is that $u_2$ corresponding to virtual Nash equilibrium is the optimal value for Agent 2. However, It observes that observed $\underline{u}_2$ is different from its expectation. In the BLAME ALL strategy, agent 1 knows that its estimate of agent 2 differs from true value due to two reasons: 1. Agents 1's wrong estimate of agent 2 and 2. Agent 2's wrong action based on a wrong estimate of Agent 1, which agent 1 observes (Liu *et al.* (2016)). The second factor is already taken care of when agent 1 estimated how agent 2 estimated agent 1. What is left is for agent 1 to correct its own wrong estimate. Hence at time step $k+1$ agent 1 updates the response curve of agent 2 by the difference it observed between it's estimated Nash equilibrium value and true observed value in $u_2$ direction. Agent 2 follows the same approach and the new action pair converges to true Nash equilibrium at time step $k+1$ in part (b) of the Figure. Learning and control algorithms for multi-agent systems with information asymmetry therefore, should adopt BLAME ALL strategy for guaranteed convergence to Nash Equilibrium of the system. Acronyms LAC and LCA are used exchangeably for learning and control algorithms.

## 2.4 Learning and Control Algorithm 1

Learning and control algorithm 1(LCA-1) is presented in Liu *et al.* (2016). Key equations from the reference are presented in this section. The system being consid-

ered is defined in 2.1. The cost function $J_i$ chosen at each agent is given by:

$$J_i(k) = x^T(k+1)Px(k+1) + u_i^T(k)u_i(k)\theta_i, \forall i, \tag{2.6}$$

where $\theta_i$ encodes the preference or intention of agent $i$ which is not known to others, $P \in \mathbb{R}^{n\times n}$ is a positive definite matrix chosen such that the closed loop system under Nash Equilibrium is globally asymptotically stable around the origin. Define $\mathrm{R} = diag(\theta_1 I_{r_1}, ..., \theta_N I_{r_N}) \in \mathbb{R}^{r\times r}$, where $r_1, r_2...$ are as defined in section 2.2. The optimal control law for each agent, obtained by optimizing 2.6 is given by:

$$u_i(k) = -[\theta_i I_{r_i} + B_i^T P B_i]^{-1} B_i^T P[f(\underline{x}(k)) + \sum_{j\neq i} B_j \hat{u}_j^{(i)}(k)], \tag{2.7}$$

where $\underline{x}$ and $\hat{u}_j$ are observed value of state $x$ and estimated control input of agent $j$, estimated at agent $i$ respectively. i.e., to compute control at agent $i$, for instant $k$, estimated values of $u_{-i}(k)$ are required in agreement with 2.4. The optimal control law 2.7 is thus a linear combination of a state feedback control and a predictive control law. At Nash equilibrium, agents have complete knowledge of parameters ($\theta_i's$) of other agents and hence can estimate their control inputs accurately. i.e., at Nash equilibrium, predictions $\hat{u}_j^{(i)}(k)$ at all agents converge had converged to true values. The profile of control laws at Nash equilibrium obtained from 2.7 by stacking all $u_i's$ together as given in the equation:

$$U^o(k) = \begin{bmatrix} u_1^o(k) \\ \vdots \\ u_N^o(k) \end{bmatrix} = -[R + B^T P B]^{-1} B^T P f(\underline{x}(k)). \tag{2.8}$$

The closed loop system under Nash equilibrium is thus given by:

$$\underline{x}(k+1) = [I - BK]f(\underline{x}(k)), \tag{2.9}$$

where $K = -[R + B^T P B]^{-1} B^T P$, is the system's feedback gain. Agent $i's$ feedback gain is defined as $K_i = -T_i K$, where $T_i = [0, ...0, I_{r_i}, 0, ...0] \in \mathbb{R}^{r_i \times n}$. Matrix $P$ in

2.6 is chosen such that the closed loop system in 2.9 is globally asymptotically stable about the origin as mentioned before. An equivalent form of equation 2.7 is given by the equation

$$u_i(k) = -T_i[\hat{R}^{(i)} + B^T PB]^{-1} B^T Pf(\underline{x}(k)), \tag{2.10}$$

where $\hat{R}^{(i)}$ is $diag\left(\hat{\theta}_1 I_{r1}, .., \theta_i I_{m_i}., \hat{\theta}_N I_{m_N}\right)$. In Blame all strategy, all agents start with the same initial values for $\theta_i$ estimates, use the same learning algorithm and maintain the $\hat{R}\,(k)$ matrix across all agents. The Virtual Nash equilibrium is then given by the equation:

$$\hat{U}(k) = -[\hat{R}(k) + B^T PB]^{-1} B^T Pf(\underline{x}(k)). \tag{2.11}$$

Agent $i's$ response corresponding to Virtual Nash equilibrium 2.11 is given by

$$u_i(k) = -[\theta_i I_{r_i} + B_i^T PB_i]^{-1} B_i^T P[f(\underline{x}(k)) + (B - B_i T_i)\hat{U}(k)]. \tag{2.12}$$

Learning algorithm is given by following two equations:

$$\hat{\theta}_j^{(i)}(k+1) = \hat{\theta}_j^{(i)}(k) + e_j^{(i)}(k+1)\underline{u}_j^T(k)\underline{u}_j(k)F(k) \tag{2.13}$$

$$e_j^{(i)}(k+1) = -\underline{u}_j^T(k)B_j^T P\underline{x}(k+1) - \hat{\theta}_j^{(i)}(k)\underline{u}_j^T(k)\underline{u}_j(k) \tag{2.14}$$

Choosing multiplication factor $F\,(k) = \left(\underline{u}_j^T\,(k)\,\underline{u}_j\,(k)\right)^{-2}$ if $\underline{u}_j\,(k) \neq 0$ and 0 otherwise. It can be proved easily as in (Liu *et al.* (2016)) that parameters $\theta_i's$ converge to true values in two time steps. For more details on this algorithm refer Liu *et al.* (2016).

## 2.5 Learning and Control Algorithm 2

In an implementation on a team of 2 mobile robots, it is found that LCA-1 algorithm wasn't enabling the system to satisfactorily recover from disturbances. Hence it is decided to consider multiple steps in the future for optimizing rather than one step as considered in LCA-1. Following subsection summarizes techniques, inferences and intuitions which motivated learning and control algorithm 2 (LCA-2).

### *2.5.1 Preliminaries*

Infinite Horizon Discrete Linear Quadratic Regulator (DLQR) optimizes a cost function of the form:

$$J(k) = \sum_{k=0}^{\infty} \left[ x^T(k) Qx(k) + U^T(k) RU(k) \right], \tag{2.15}$$

for a centralized system described by

$$X(k+1) = AX(k) + BU(k), \tag{2.16}$$

where $A$ is the system characteristic matrix, $B$ is the system control gain matrix, $U$ is the vector of inputs acting on the system, $Q$ is a Positive semi-definite matrix and $R$ is a Positive definite-matrix. It is called infinite horizon since the summation is to infinity in the cost function 2.15. The $U$ at instant $k$ that optimizes 2.15 is given by

$$U_k = -K_\infty x_k, K_\infty = \left( B^T S_\infty B + R \right)^{-1} B^T S_\infty A, \tag{2.17}$$

where $S_\infty$ is the steady state solution of Discrete Algebraic Riccati equation given by

$$S = A^T \left[ S - SB \left( B^T SB + R \right)^{-1} B^T S \right] A + P \tag{2.18}$$

Lewis *et al.* (2012). If the System 2.16 is controllable and observable its Discrete Algebraic Riccati equation will have a unique solution (Lewis *et al.* (2012), Kučera (1972)). The unique solution to 2.18 can be found analytically or numerically using several algorithms (Laub (1979), Arnold and Laub (1984) etc.).

In the article Wan (1991), the behavior of Riccati solution to a Linear Quadratic regulator is analyzed using a first order system. Inferences drawn from the behavior of Riccati solution for first order systems is extended to higher order systems. Using a similar argument, for a first order system, the variation of LQR gain $K$ with variation in $R$, both of which will be scalars for stable and unstable systems is analyzed. As

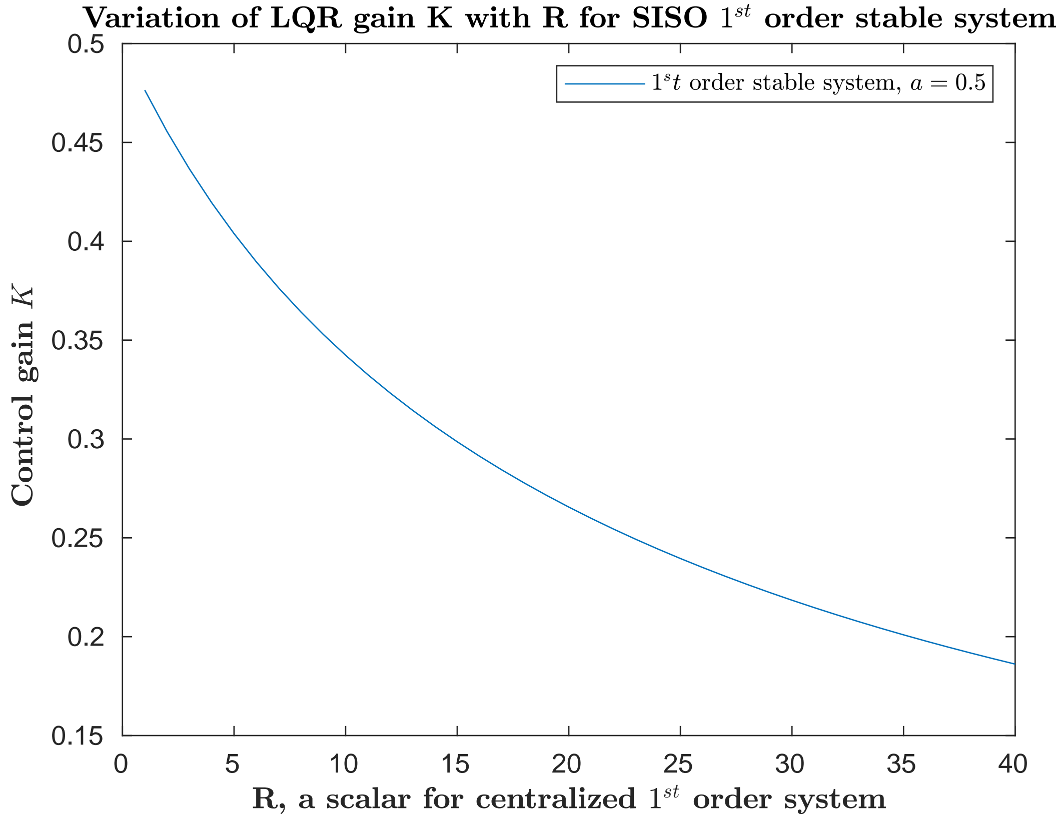



**Figure 2.4:** LQR gain variation with R for a stable system

seen in Figures 2.4, 2.5, it is inferred that, when all other interfering parameters are kept constant, the LQR gain and hence the control output decreases monotonically with increment in the value of R. This key observation will be used in devising LCA-2.

Learning and control algorithm 2 (LCA-2) is developed using ideas in Liu *et al.* (2016), presented in section 2.4. LCA-2 considers infinite steps ahead for optimization as opposed to LCA-1 which considers only one step. LCA-2 addresses issues involved with simultaneous estimation and action in an approach similar to LCA-1 by employing a learning strategy similar to BLAME ALL. Agent $i$ estimates its own preference while estimating others to know how others are performing its intention estimation. In LCA-2, the same learning algorithm with the same initial conditions runs at all

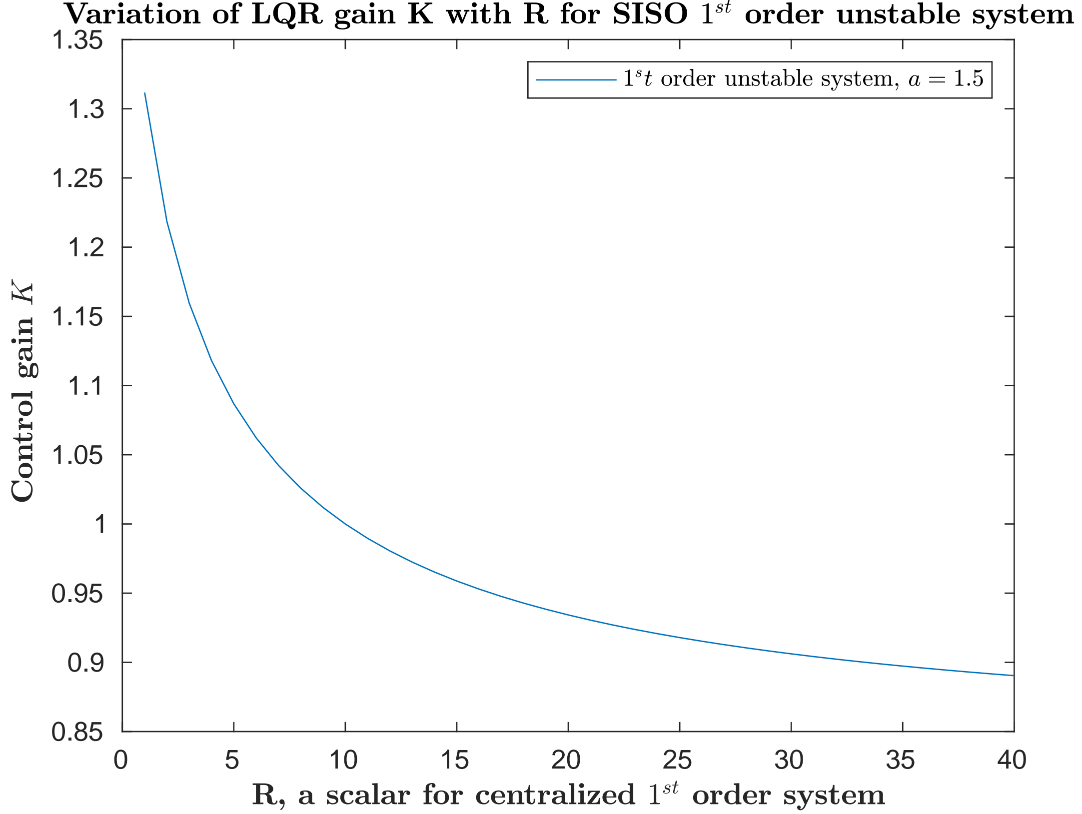



**Figure 2.5:** LQR gain variation with R for an unstable system

agents for intention estimation. However, the learning algorithm runs multiple times per an observed control input $\underline{u}_i(k)$ sample. i.e., the learning loop runs faster than control loop. LCA-2 additionally assumes that the agent parameters lie between a minimum $\theta_{min}$ and maximum $\theta_{max}$ values, known at all agents.

For a MAS comprised of agents with just one input, $R$ matrix is the diagonal matrix formed by agent parameters $\theta_i, \forall i$. For MIMO (Multi Input Multi Output) agent MAS, $R = diag\left(\theta_1 I_{r_1}, ..., \theta_N I_{r_N}\right)$ as in LCA-1. At each time step $k$, each agent maintains different $R$ matrices for learning and control. For clarity, the algorithm is detailed for a MAS with one input per system. It, however, can easily be extended to multi-input agent systems as explained later.

### 2.5.2 Process steps - Control Algorithm:

- At instant $k = 0$, agent $i$ initializes parameters $\theta_{-i}$ of all other agents to be unity. Hence the learning matrix $\widehat{R}(k)$ at $k = 0$ is an Identity matrix. $\widehat{R}(k)$, the learning matrix will be the same at all agents at all time steps. For a 3 agent system with true parameters $\theta_1, \theta_2, \theta_3$, the learning matrix at an instant will be

$$\widehat{R}(k) = \begin{pmatrix} \widehat{\theta}_1(k) & 0 & 0 \\ 0 & \widehat{\theta}_2(k) & 0 \\ 0 & 0 & \widehat{\theta}_3(k) \end{pmatrix}.$$

- At every step, for agent $i$, the control matrix is obtained by taking the learning matrix $\widehat{R}(k)$ at the instant and replacing the diagonal element corresponding to agent $i$ with its true parameter value. Agent $i$ knows its own parameter $\theta_i$. Control matrix $R_i(k)$ for agent $i$ at instant $k$ is used for $R$ in the cost function 2.19 and solved for control. The control matrix at agent 1 for a 3 agent system described before will be

$$R_1(k) = \begin{pmatrix} \theta_1 & 0 & 0 \\ 0 & \widehat{\theta}_2(k) & 0 \\ 0 & 0 & \widehat{\theta}_3(k) \end{pmatrix}.$$

- At every time step, agent $i$ solves a cost function given by

$$J_i[k] = \sum_{j=0}^{\infty} \left[ x^T(k+j) Q x(k+j) + U^T(k+j) R_i(k) U(k+j) \right], \tag{2.19}$$

where $Q$ is positive semi-definite matrix similar to $P$ in LAC-1. If the MAS is described by a $C$ matrix, the output matrix in $Y = CX + DU$ in general state space representation of a system, $Q = C^T C$ or some scalar multiplied $C^T C$. This ensures that the output state is optimized.

- Cost function 2.19 is the DLQR, described in 2.5.1. The solution 2.17 and all its related properties apply to the control sequence that minimizes cost function 2.19.
- From $U_i(k)$ Agent $i$ extracts the control $u_i(k)$ applicable to it and applies it on the MAS. MAS states evolve as a result, when all agents act on the MAS.
- control $u_i(k)$ applied on the system by agent $i$, is observed at all agents at time step $k+1$. The observed value $\underline{u}_i(k)$ is used for updating $\theta$ estimates at all agents in the learning algorithm.
- When parameters $\widehat{\theta}_i, \forall i$ in $\widehat{R}(k)$ converges to true parameters, all agents will be solving the same DLQR cost and hence generating same control outputs $u_i$ for all agents.

As mentioned before, every agent estimates every other agent parameter $\theta$, including itself.

### 2.5.3 Process steps - Learning Algorithm:

Since the same learning process is carried out at all agents for all agents, agent suffixes are dropped from symbols during explanation of the algorithm. The key idea of the learning algorithm is *approximating the variation of $u$ with $\theta$ for an agent between two known points, while all other agent parameters $\theta_{-i}$ are held constant, to straight line.* Using the line approximation, an updated $\theta$ is computed, the $u$ for which is different to observed $\underline{u}$. The updated $\theta$ and its corresponding $u$ form a new end point, with which another line is drawn and the process repeated, the steps for which are explained in detail below:

- Agent $i$ takes $\hat{R}(k)$, obtains $u_{max}$ and $u_{min}$ corresponding to $\theta_{max}$ and $\theta_{min}$ values for $\theta_i$ in $\hat{R}(k)$ using the equation 2.17.

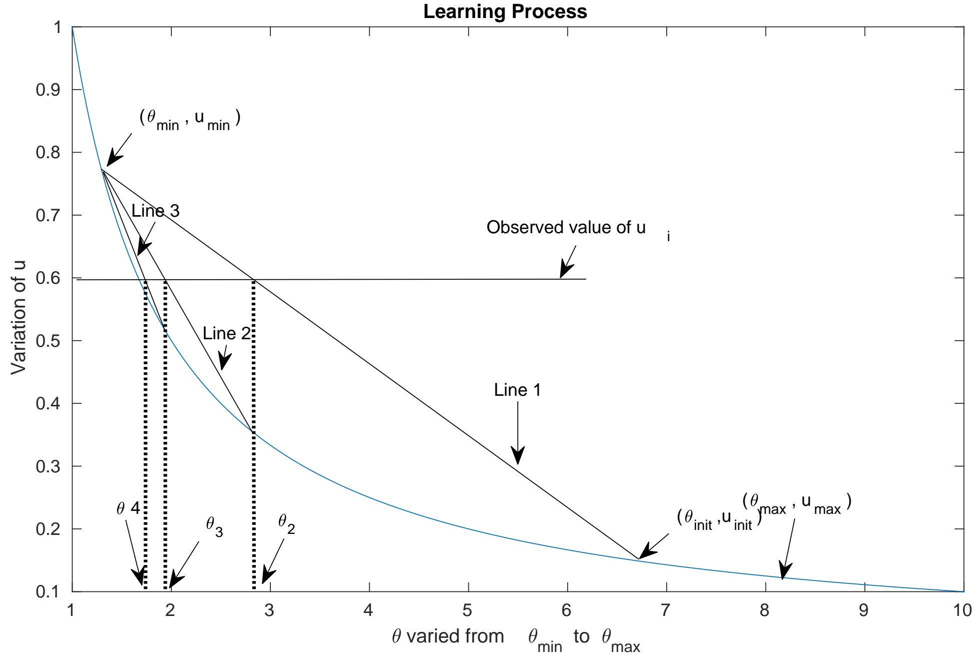



**Figure 2.6:** Learning Process - Scenario 1

- Slope between points $(u_{max}, \theta_{max})$ and $(u_{min}, \theta_{min})$ is determined to check if $u$ vs $\theta$ is a monotonically increasing or decreasing curve. As mentioned in subsection 2.5.1, slope will be negative.

- Based on the monotnocity, determine if $\theta$ for $\underline{u}$ lies between $[\theta_{min}, \theta_{est}]$ or $[\theta_{est}, \theta_{max}]$. $\theta_{est}$ is the current estimate of $\theta$. At time step $k = 0$, $\theta_{est}$ will be 1 and at other steps, it will be the final update made at a prior step, which becomes the initial value for current step. Initial value of $\theta$ for a given update cycle is denoted $\theta_{init}$ as in Figures 2.6 and 2.7. i.e.,at the beginning of an update cycle, $\theta_{est} = \theta_{init}$, in the next instant $\theta_{est} = \theta_2$, in the next instant $\theta_{est} = \theta_3$ and so on. Each time step can have multiple updates depending on computational ability available.

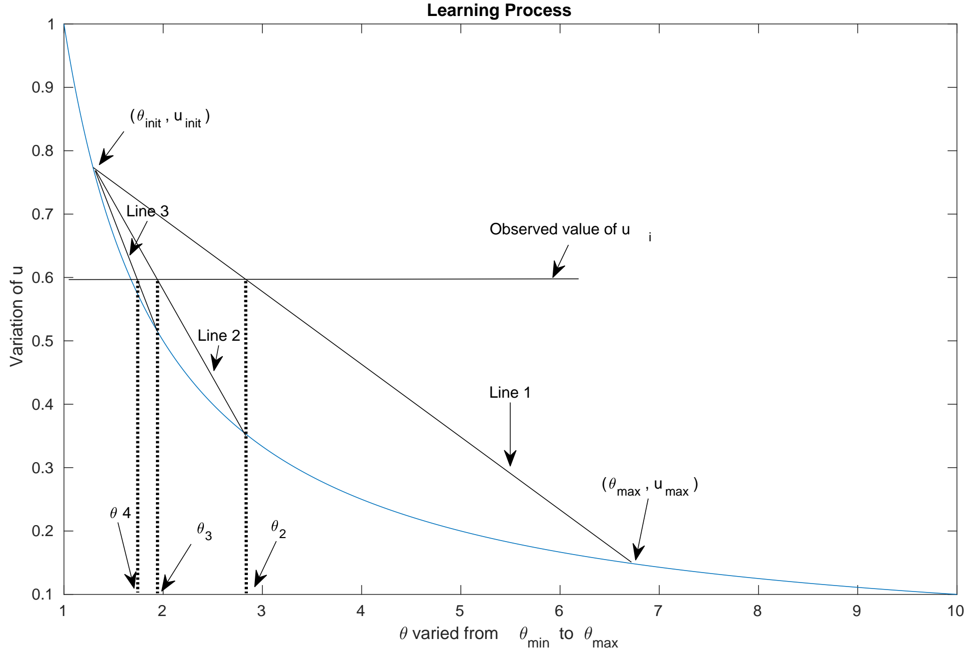



**Figure 2.7:** Learning Process - Scenario 2

- If $\theta$ for observed $\underline{u}$ lies in the interval $[\theta_{min}, \theta_{est}]$ , the events described in Figure 2.6 take place.

- The curve between endpoints $((\theta_{init}, u_{init}))$ and $((\theta_{min}, u_{min}))$ is approximated to be a straight line Line 1. $\theta_2$ corresponding to where the line cuts the observed value $\underline{u}$ is taken to be next update.

- $u$ corresponding to $\theta_2$ is obtained using the equation 2.17. Line 2 is drawn from $\theta_2$ updated point to $((\theta_{min}, u_{min}))$. $\theta_3$, $\theta_4$ etc. are obtained using the same process.

- If $\theta$ for $\underline{u}$ lies in the interval $[\theta_{est}, \theta_{max}]$ , the events described in Figure 2.7 take place.

- Figures 2.6, 2.7 are summarized: as find equation of line 1, find the $\theta_2$ where the line cuts $\underline{u}$, draw line 2, find $\theta_3$, draw line 3, find $\theta_4$ and so on. In the first

scenario the end point $((\theta_{min}, u_{min}))$ remains constant, where as in the second scenario, the initial estimate for the iteration $((\theta_{init}, u_{init}))$ remains constant.

- The limiting case in both scenarios is when the $\theta$ update coincides with the true value of $\theta$ corresponding to $\underline{u}$.
- Since there is no explicit relation available between $u$ and $\theta$, a straight line equivalent to the curve is used to close down on the value of $\theta$ being estimated. This approach has a faster convergence rate compared to methods like bisection.
- Since only one diagonal control cost weight element $\theta_i$ corresponding to the agent being estimated is varied while others are held constant in $\hat{R}$, the monotonicity described in subsection 2.5.1 holds.
- Agent $i$ repeats this process for all agents with corresponding observed values and updates $\hat{R}$ for the next instant. Updated $\hat{R}$ shall be used in determining control to be applied at each agent.
- For a MAS with multi-input agents, $u_i$ will be a vector. In the learning algorithm described, replace vector $u$ with its norm at all places. Instead of a diagonal element $\theta_i$ for agent $i$, a block matrix $\theta_i I_{r_i}$ is used in control matrix $R_i(k)$, and in learning matrix $\hat{R}(k)$.

From the Figures, 2.6, 2.7, it is seen that the estimates converge to true values asymptotically even if the updates are spread between two control iterations.

# Chapter 3

## EXPERIMENTAL SETUP

LCA-1 and LCA-2 described in chapter-2 are implemented on two mobile robots for agents. Hercules robot, shown in figure 3.1, a 4 wheel skid-steer robot designed for indoor load transport is used as an agent. Its a two input system: motors on the left and motors on the right are driven by two PWM voltage signals that vary from (-100 - 100). Robot runs on an Arduino Duemilanove w/ ATmega328 for microcontroller. PWM inputs greater than 100 in magnitude are set to 100 before applying to the robot in robot drivers. Negative control inputs are interpreted as signals for the robot to go in the reverse direction and are accordingly implemented in its driver. A robot thus can accept two signals in the range [-100, 100] without actuation saturation.

**Figure 3.1:** Hercules Robot

The arduino running on hercules is interfaced with a powerful embedded computing device Edison from Intel, through serial communication. Arduino running on hercules robot is programmed to receive actuation signals from Edison for starting,

stopping at a speed.

## 3.1 Modelling of Hercules Robot

Skid-steer mobile robots are hard to model as they are steered using friction and since skidding is involved. Models developed for skid-steer robots in (Wang *et al.* (2009), Caracciolo *et al.* (1999)) attest to this fact. To precisely model the robot, friction coefficients of wheel surfaces, and slip characteristics, which vary from surface to surface are needed. Since the main focus of this thesis is on high level learning and control planning algorithms, the robot motion is restricted to one-dimension to obtain a linear model of the robot. The left and right motors are actuated with the same value of PWM input making the system a single input system. Experimental trials showed that the 1-D dynamics can be modeled by a first order system. Robots are subjected to step inputs in the range [0 - 100] and responses were recorded. SYSTEMID on the step response data yielded a continuous model:

$$\frac{x(s)}{pwm(s)} = \frac{0.2003}{s + 12.1}, \tag{3.1}$$

where $x$ and $pwm$ are the position in the dimension considered and pwm input respectively. The model 3.1 is converted to an equivalent State Space representation in continuous domain, and later to a discrete domain based on sampling rate used.

## 3.2 Position data

Attempts were made to have the robot estimate its own position by sensor fusion using encoders on-board the robot and an Inertial measuring unit by a Kalman filter. However, since the LCA algorithms don't take stochastic nature of measurements in to account, it is decided to not use the fused measurements while running the experiments.

Robot position is obtained in real time using VICON Motion capture system at 100 HZ. Reflection markers are placed on each robot, which the VICON system identifies and broadcasts position data. Robot Operating System (ROS) is used to obtain the streamed data from Motion capture system on to a machine running Linux. Algorithms are evaluated on the computer and control inputs computed are transmitted via Wi-Fi to two Edison computers on-board two Hercules robots. Position coordinates obtained from motion capture system are converted to experiment coordinate system before usage. A separate coordinate system for experiment is needed in view of less area of coverage available in motion capture lab and for freedom of choosing any initial condition. Occlusion can happen in some instances when there are obstructions in line of sight from Mocap cameras to markers and result in unpredictable behaviour of the algorithm and are handled in the implementation. Whenever occlusion happens, the system model is used in obtaining state evolution and hence the outputs that were occluded.

A 3rd order model captured Hercules step response dynamics better than the first order dynamics given in 3.1. However, with a third order system representing one Hercules robot, the two robot system became a sixth order system, with observations available for two outputs. The six states were internal states obtained during state space conversion and couldn't be assigned a physical meaning. While the system obtained was observable, a filter to estimate the states from observations was required. Since this would introduce additional delay in the loop, which already had delays from the network, from motion capture system and due to occlusion sometimes, it is decided to stick to a first order system representing a robot given in 3.1. With a first order model for the robots, the states of the system were readily available from outputs without the need for additional filtering. It is also to be noted that the rate at which VICON Motion capture system streams data is not constant and varied from (80 -

100) Hz. The second order and sixth order two robot systems were however simulated in MATLAB with both algorithms for verification. The 3-pole Single Input Single Output representation of the system is given by

$$\frac{x(s)}{pwm(s)} = \frac{229.1}{s^3 + 50.41s^2 + 1608s + 1.386e04} \tag{3.2}$$

## 3.3 Two Robot System

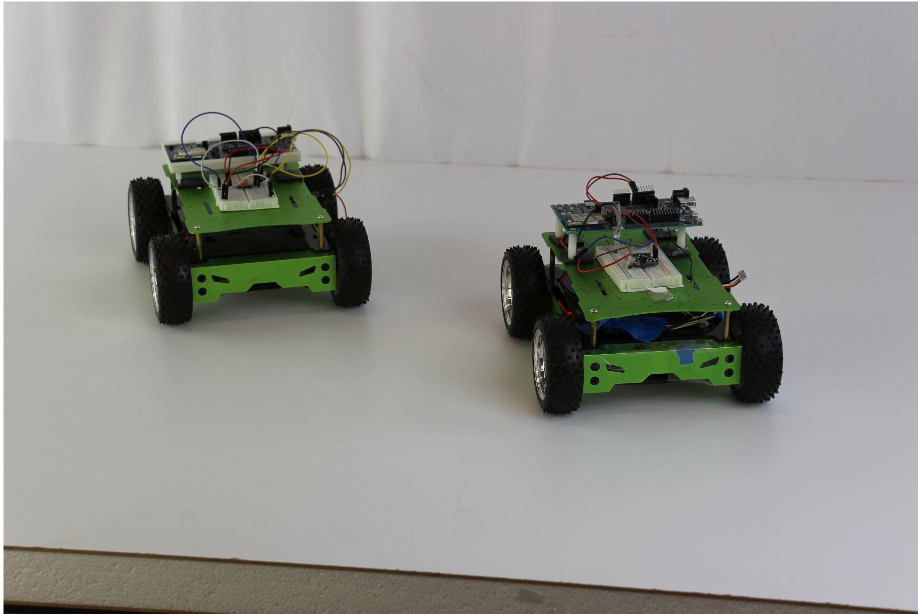


**Figure 3.2:** Two Robot team

Robots as shown in figure 3.2 are made to move in straight line parallel to each other. Instead of having the robots carry an actual joining beam, with an intelligent choice of outputs, algorithms LCA-1 and LCA-2 are implemented for collective load transport. The outputs chosen are :

- Midpoint of the line joining markers on both robots, $y_1$. This ensures that the robot team as a whole reaches the target, origin.
- Lateral distance between the robots, $y_2$. This is to ensure that the lateral distance is maintained as close to zero as possible so that the load doesn't fall

of the robots.

The discrete two robot system, in state space representation, discretized at 80 Hz sampling rate is given by:

$$A = \begin{pmatrix} 0.8597 & 0 \\ 0 & 0.8597 \end{pmatrix} B = \begin{pmatrix} 0.0116 & 0 \\ 0 & 0.0116 \end{pmatrix} C = \begin{pmatrix} 0.1001 & 0.1001 \\ -0.2003 & 0.2003 \end{pmatrix}$$

D is a null-matrix. States aren't directly accessible as outputs. States were calculated from $y = Cx$ by inverting $C$ matrix.

### 3.4 Choice of matrices P or Q, Initial conditions and Preferences

$P$ is the state weighting matrix in the cost function equation 2.6 and $Q$ is from equation 2.19. Algorithms LCA-1 and LCA-2 don't consider any constraints on either states or control inputs. However actuation in all real systems is bounded within a range, which when exceeded results in making the system unresponsive. Hercules robots can only react well to actuation with in the range [20, 80] or [-80, -20], with positive PWM corresponding to forward direction and negative PWM corresponding to reverse direction. Hence to demonstrate control algorithms, the matrices $P$ and $Q$, along with the agent preferences $\theta_1, \theta_2$, and starting position $y_{init}$ and corresponding $x_{init}$ were picked using trial and error and some commonsense in simulations to obtain values where they result in the control inputs computed lie in the ranges mentioned.

# Chapter 4

# SIMULATION AND EXPERIMENTAL RESULTS

The control $u_i(k)$ applied by an agent running LCA-2 is given by equation 2.17 for the $R_i(k)$ control matrix it forms at the considered instant $k$. In LCA-2 each agent uses a different control matrix $R_i(k)$ derived from a common learning matrix $\hat{R}(k)$ by using its true $\theta_i$ or $\theta_i I_{r_i}$ as described in Chapter 2. Thus control vector $U$ applied on the MAS is formed by stacking DLQR solutions 2.17 for different values of $R_i, \forall i$. For a 3 agent MAS, the control $U(k)$ is given by

$$\begin{bmatrix} u_1(k) \\ u_2(k) \\ u_3(k) \end{bmatrix} = \begin{bmatrix} -T_1\left(B^T S_{\infty_1}(k) B + R_1(k)\right)^{-1} B^T S_{\infty_1}(k) A x_k \\ -T_2\left(B^T S_{\infty_2}(k) B + R_2(k)\right)^{-1} B^T S_{\infty_2}(k) A x_k \\ -T_3\left(B^T S_{\infty_3}(k) B + R_3(k)\right)^{-1} B^T S_{\infty_3}(k) A x_k \end{bmatrix}, \tag{4.1}$$

where $S_{\infty_1}(k), S_{\infty_2}(k), S_{\infty_3}(k)$ are steady state solutions of riccati equations formed by matrices $R_1(k), R_2(k), R_3(k)$ respectively, and $T_i = [0, ...0, I_{r_i}, 0, ...0] \in \mathbb{R}^{r_i \times n}$. $T_i$ matrices extract $u_i$ from $U$. Given the complexity of equation 4.1, for a N-agent MAS, the stability of LCA-2, as it's learning algorithm makes $\hat{R}$ converge to $R$ with time, could not be proved mathematically.

To prove the effectiveness of LCA-2, it is applied to different kinds of systems, system responses obtained are analyzed and compared to responses with LCA-1. In addition, the worst case scenarios possible with LCA-1 and LCA-2 are compared, which is when the learning algorithm fails altogether and don't change initial estimates of $\hat{R}$ at each agent. Stability of multi-agent systems running LCA-1 and LCA-2 for a different $P/Q$ - the state weighting cost matrices is checked. Experimental implementation results for both LCA-1 and LCA-2 are presented at the end.

LCA-1 considers one step ahead for optimization, whereas LCA-2 considers infinite steps. LCA-1 is thus *Shortsighted* compared to LCA-2. Accordingly, it is found, once learning is complete that any disturbances applied to the MAS are absorbed gradually with LCA-2 compared to LCA-1. LCA-1 tries to compensate for the disturbance in just one step, whereas LCA-2 compensates in more steps. It is found in general that LCA-2 performs better compared to LCA-1 in reacting to disturbances once learning is complete. During the learning process, however, LCA-1 is found to have a better response as far as variation in states and control inputs is concerned.

## 4.1 Simulation Results and Comparison

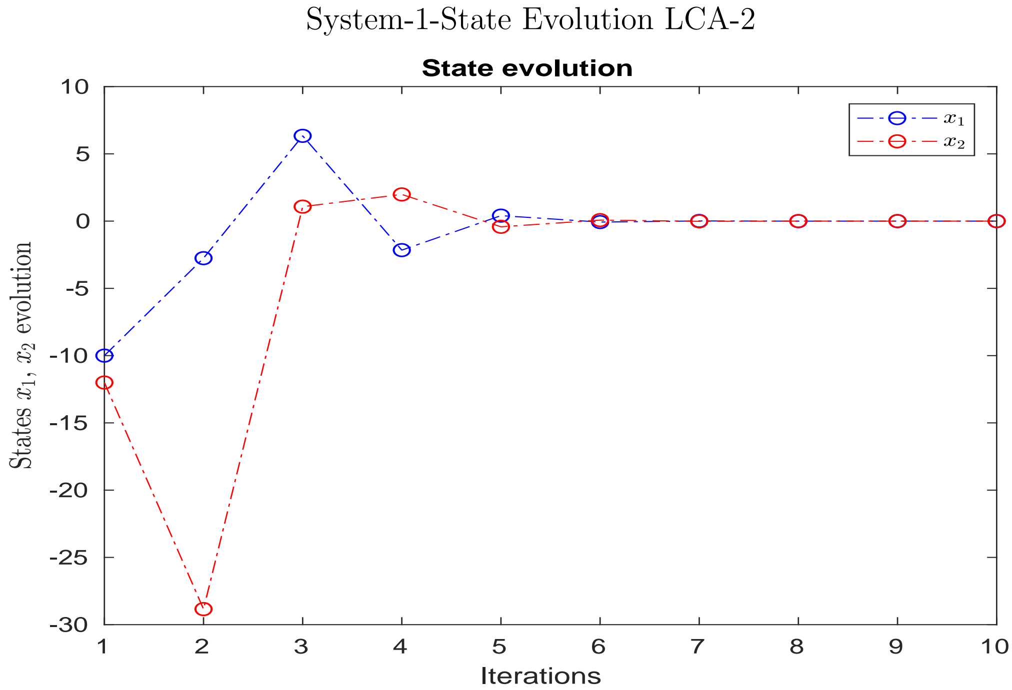
### *4.1.1 System - 1*

The system considered is an open loop unstable discrete time system. It is controllable and observable. It has cross-coupling among states via the characteristic matrix $A$ and there is weight cross coupling among states in the cost function via matrix $Q/P$. Matrices A, B of the system state space representation and state weighting matrix Q in the cost function are given below. The parameters chosen are $\theta_1 = 5$, $\theta_2 = 15$, $\theta_{max} = 100$ and $\theta_{min} = 0.1$

$$A = \begin{pmatrix} 0.5 & 0.1 \\ 3 & 0.9 \end{pmatrix} B = \begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix} Q = \begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix}$$

Some observations from the plots for System - 1:

- As seen in Figure 4.1 the system goes to the goal state within 6 iterations. Overshoot in states can be altered by tuning weights in $Q$ matrix. The MAS is therefore stable.
- Parameters $\theta_1, \theta_2$ converged to their true values by third control iteration as



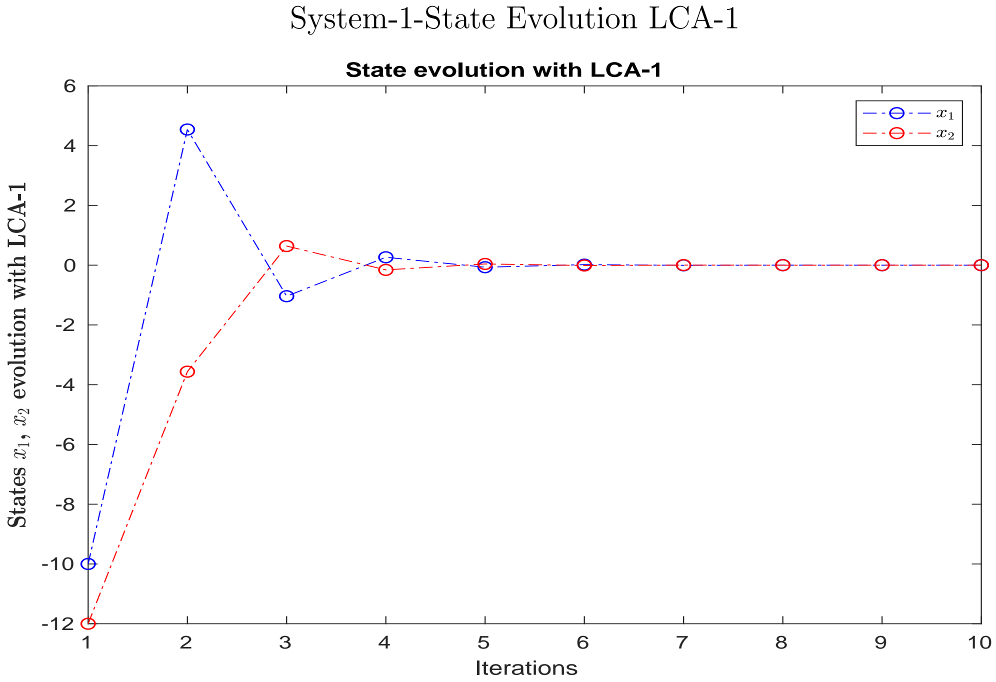
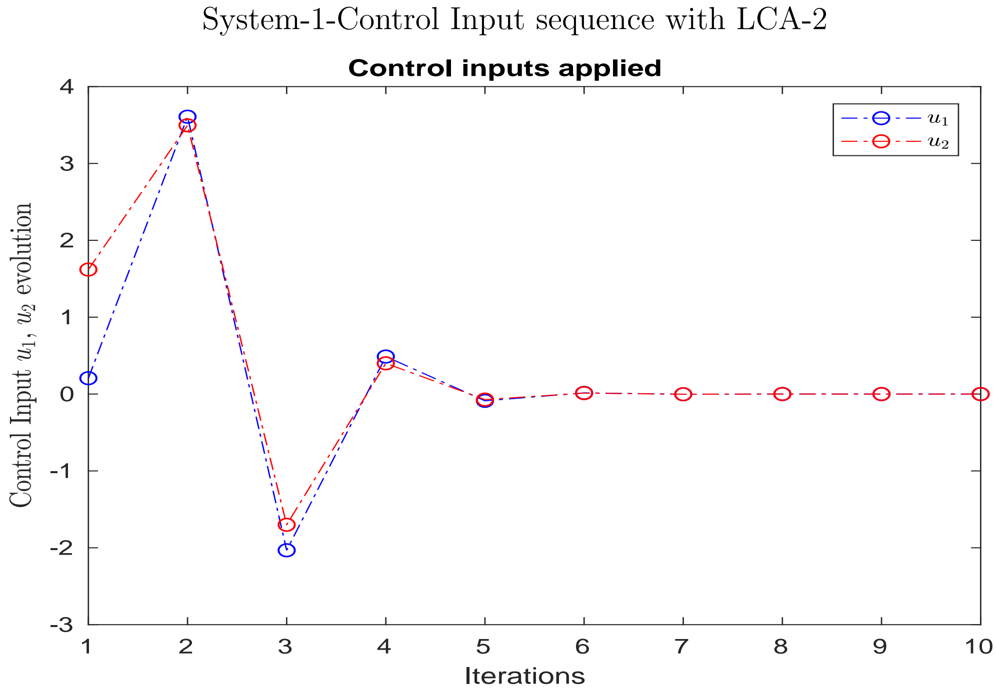




**Figure 4.1:** State Evolution Comparison LCA-1 v LCA-2

shown in Figure 4.3. It should however be noted that the learning algorithm is run ten times per observation for faster convergence.



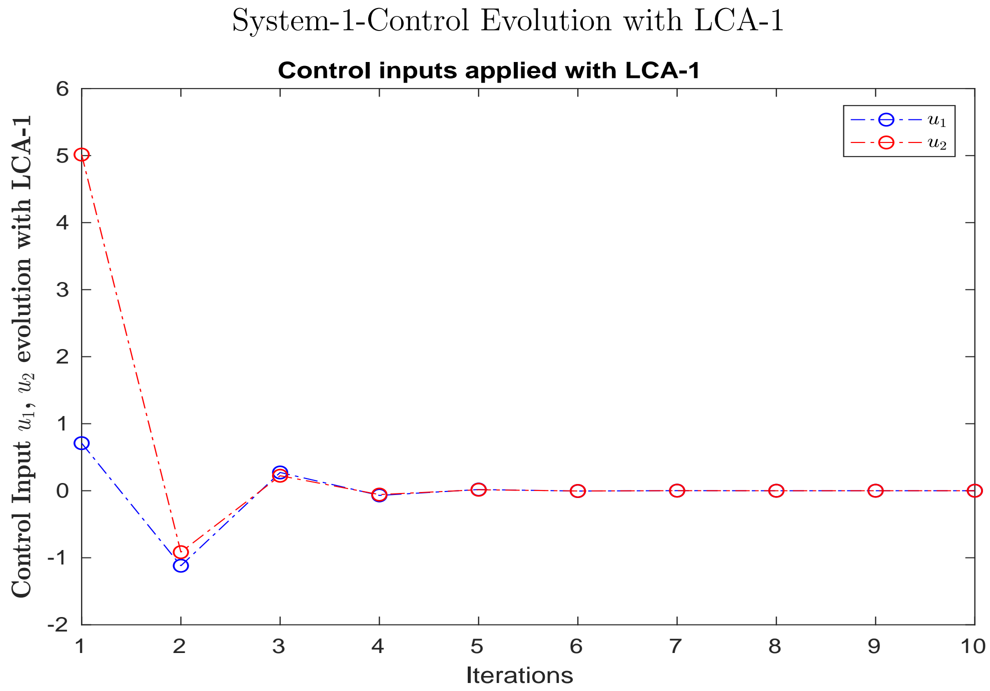




**Figure 4.2:** Control Evolution Comparison LCA-1 v LCA-2

- Control inputs shown in Figure 4.2 became zeros along with states as expected by $6^{th}$ iteration.

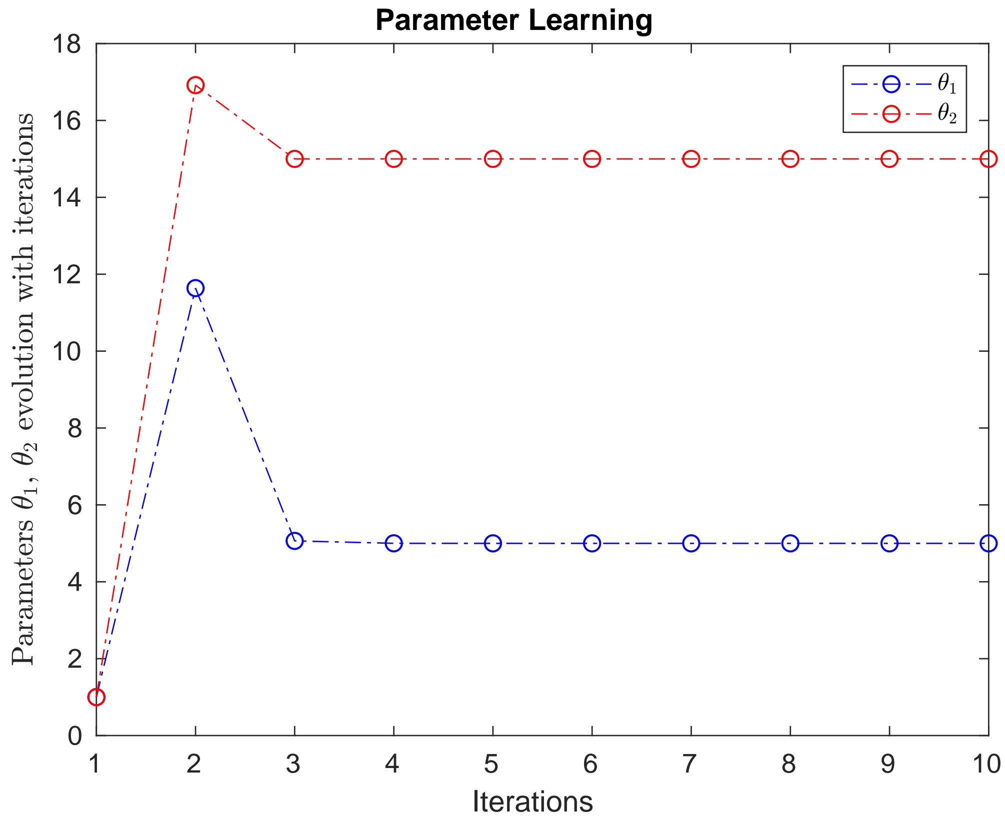



**Figure 4.3:** System-1 Parameter Learning Evolution with LCA-2

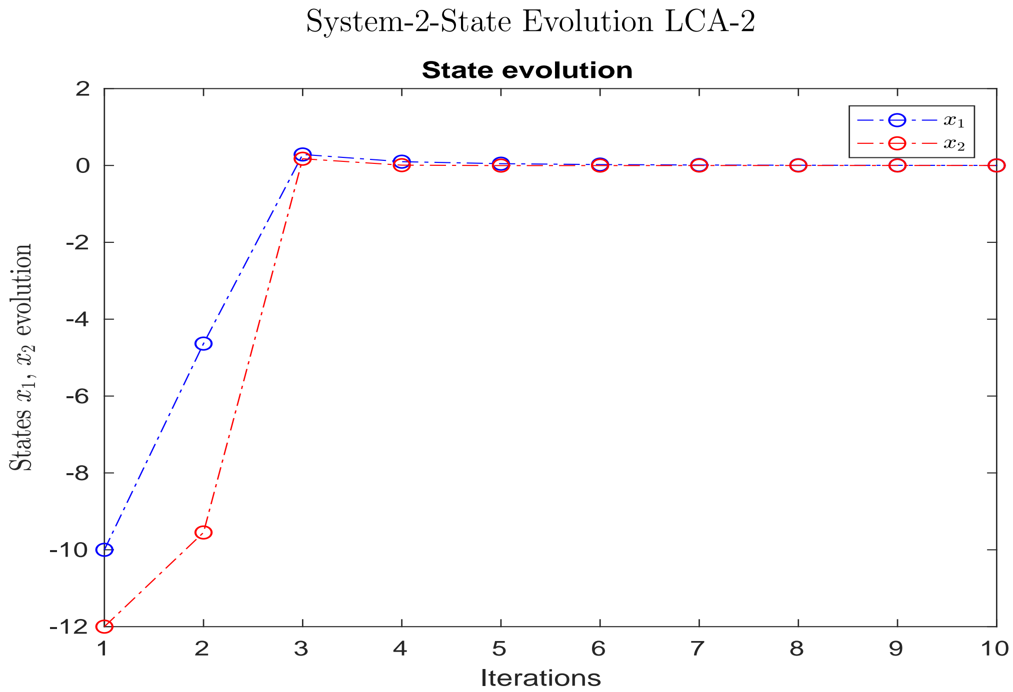
- Compared to LCA-2, the variation of applied controls $u_1, u_2$ is more in LCA-1 as shown in the Figure 4.2. State variation is about the same in both cases.

### *4.1.2 System - 2*

The system considered is an open loop stable discrete time system. It is controllable and observable. It has no cross-coupling among states via the characteristic matrix $A$ and no weight cross coupling among states in the cost function via matrix $Q$. Matrices A, B of the system state space representation and state weighting matrix in the cost function Q are given below. It is a two state system. The parameters chosen are $\theta_1 = 25$, $\theta_2 = 40$, $\theta_{max} = 100$ and $\theta_{min} = 0.1$

$$A = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.9 \end{pmatrix} B = \begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix} Q = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}.$$

Plots and observations are given below:



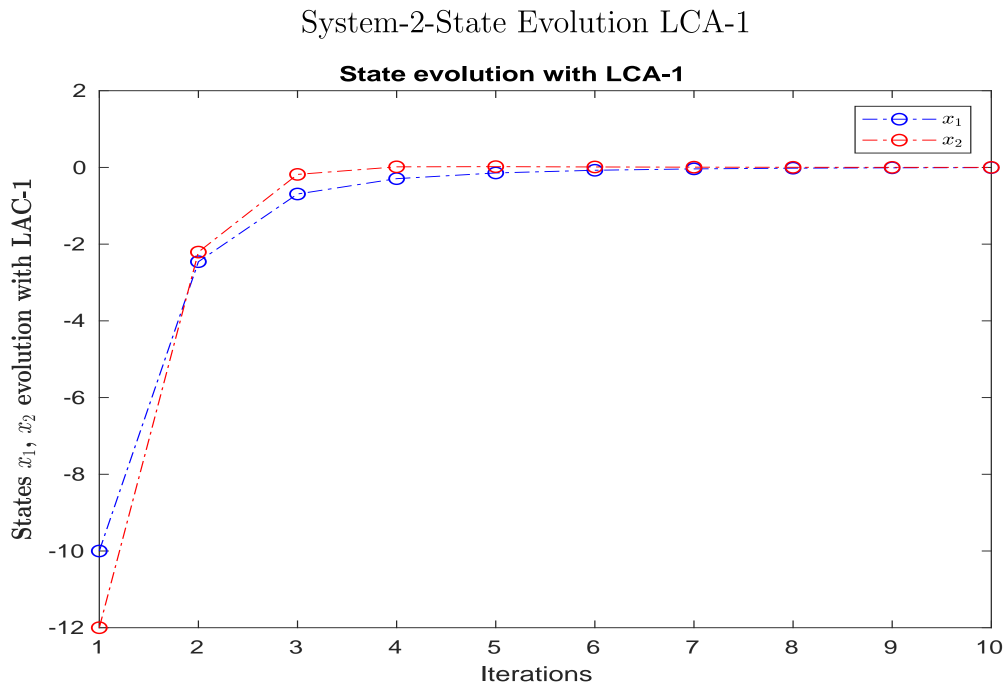




**Figure 4.4:** State Evolution Comparison LCA-1 v LCA-2

Some observations from System -2 's plots:

- As seen in Figure 4.4 the system goes to the goal state within 4 iterations.

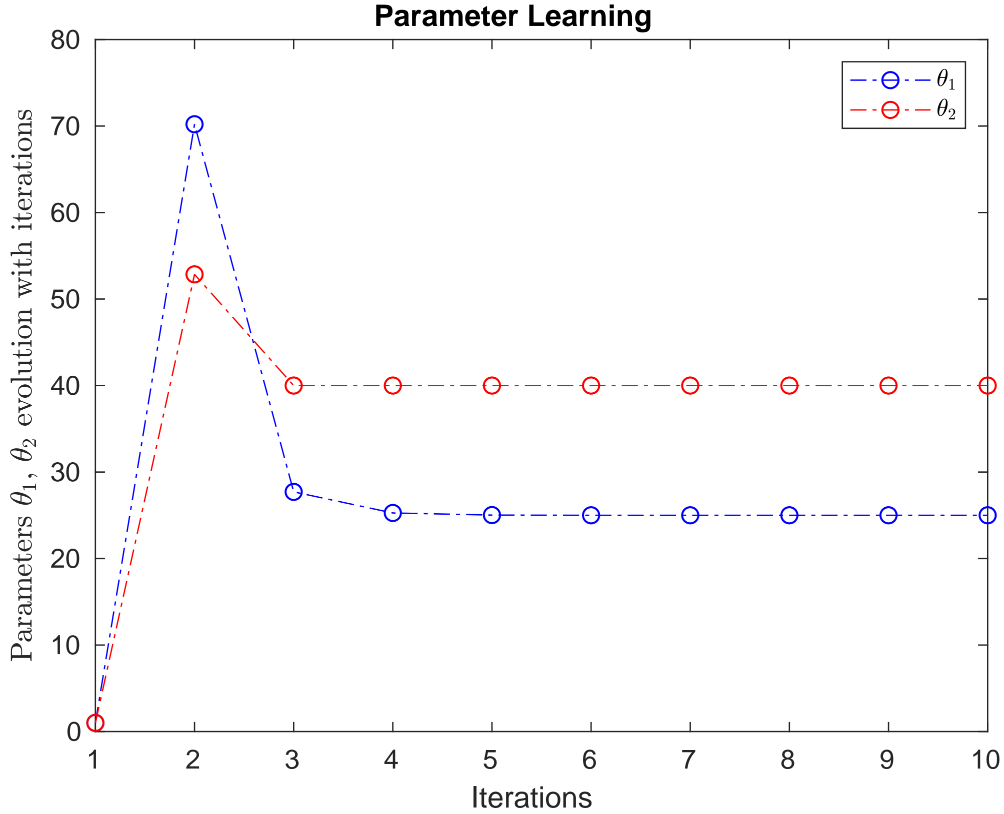


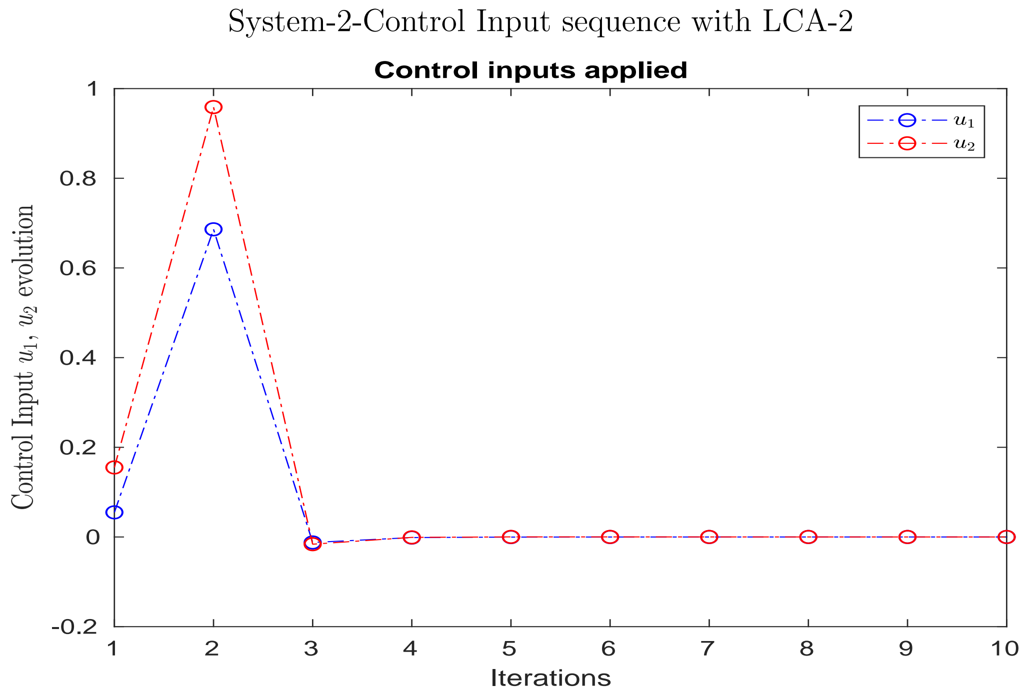

**Figure 4.5:** System-2 Parameter Learning Evolution with LCA-2

Overshoot in states can be altered by tuning weights in $Q$ matrix. The MAS is therefore stable.

- Parameters $\theta_1, \theta_2$ converged to their true values by fifth control iteration as shown in Figure 4.5. This is because the the parameters chosen for the system $\theta_1, \theta_2$ are distant from their minimum and maximum values compared to system-1. Learning loop is run 10 times per observation.

- Control inputs as in Figure 4.6 became zeros along with states as expected by $4^{th}$ iteration.

- It is seen from Figure 4.4, and Figure 4.6 that with LCA-1 evolution of state and control are gradual compared to LCA-2 even though not by a large magnitude



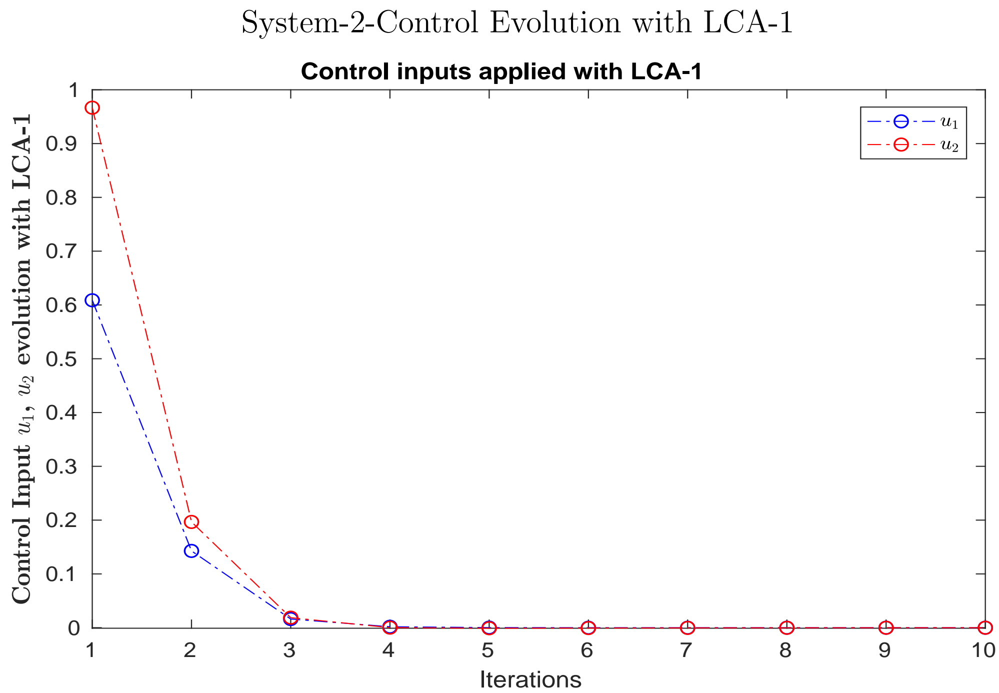




**Figure 4.6:** Control Evolution Comparison LCA-1 v LCA-2

### 4.1.3 System-3

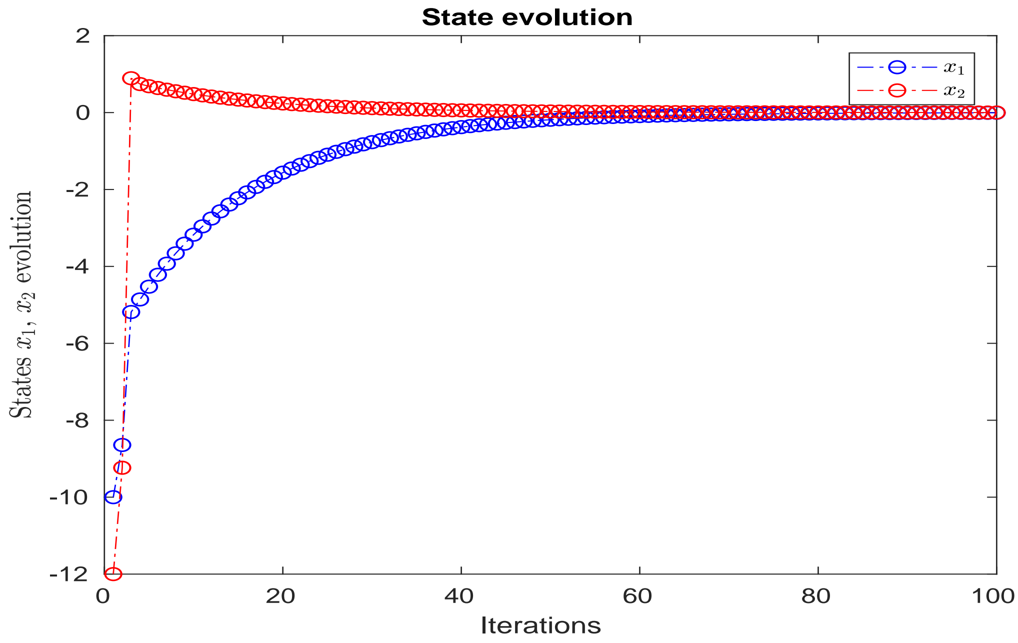
The system considered is an open loop marginally stable discrete time system. Matrices A, B of the system state space representation and the state weighting matrix in the cost function Q are given below. The parameters chosen are $\theta_1 = 5$, $\theta_2 = 15$, $\theta_{max} = 100$ and $\theta_{min} = 0.1$.

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} B = \begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix} Q = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}$$

Some observations on plots of System - 3:

- As seen in Figure 4.7, with LCA-2, the system considered is sluggish since the open loop system is marginally unstable. Convergence to goal state is at almost 70 steps.
- Parameter evolution as seen in Figure 4.9 is complete in 3 steps. Learning loop is run 10 times per observation.
- As expected, shown in Figure 4.8 control inputs don't get to zeros until about iteration 70, as with the state convergence.
- For system-3, as seen in Figure 4.7, state evolution is even more sluggish than with LCA-2. This reflects the *Shortsightedness* of LCA-1 compared to LCA-2. A similar behavior with control evolution is observed as shown in Figure 4.8.
- For system-3 it can be concluded that LCA-2 has better performance compared to LCA-1.

System-3-State Evolution with LCA-2




System-3-State Evolution with LCA-1

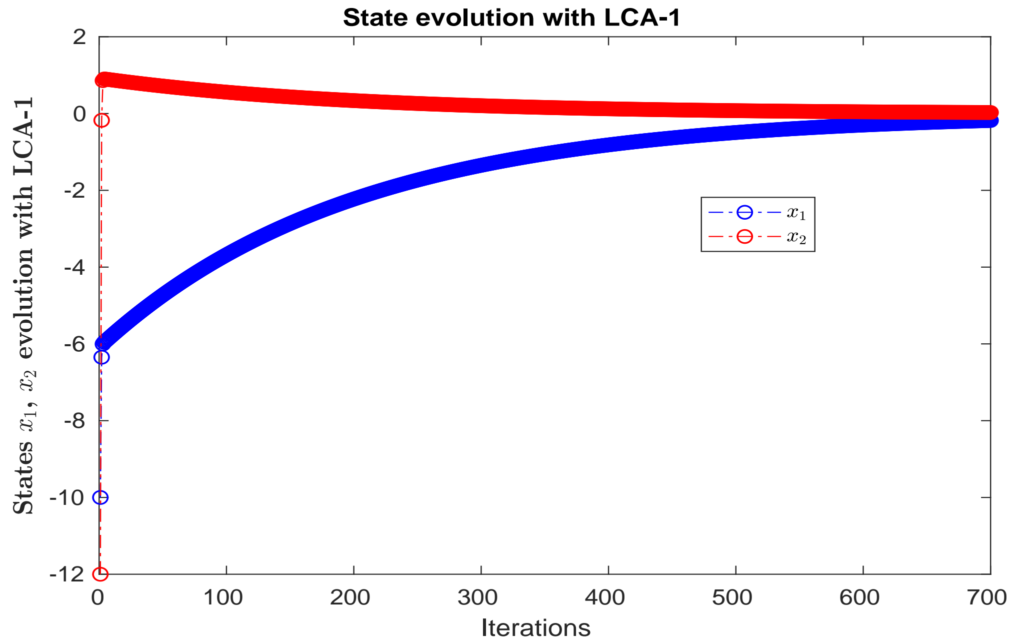
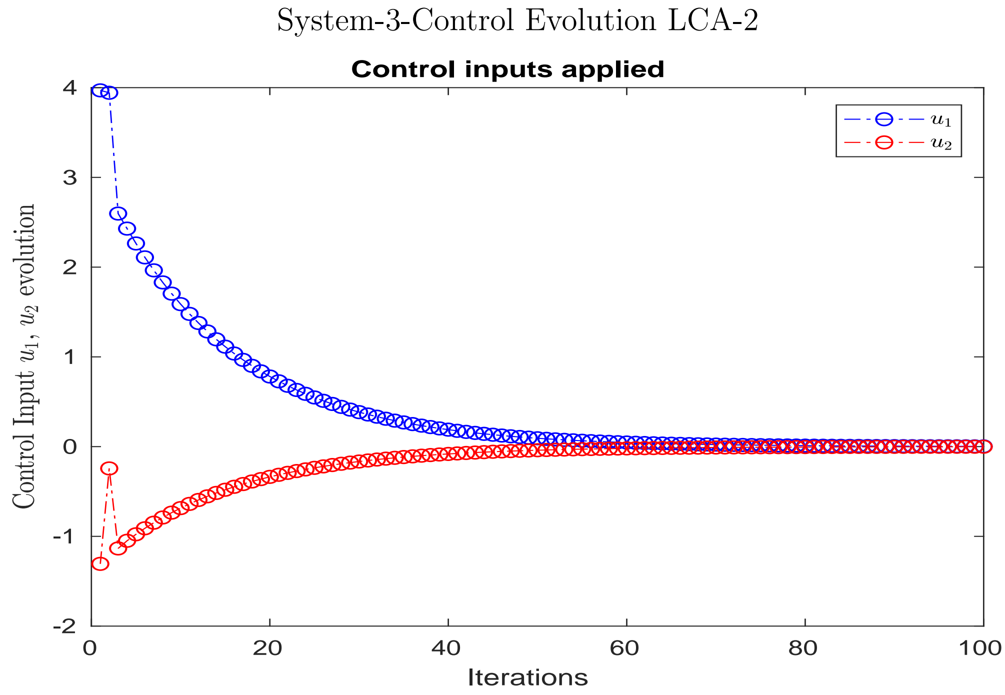



**Figure 4.7:** State Evolution Comparison LCA-1 v LCA-2



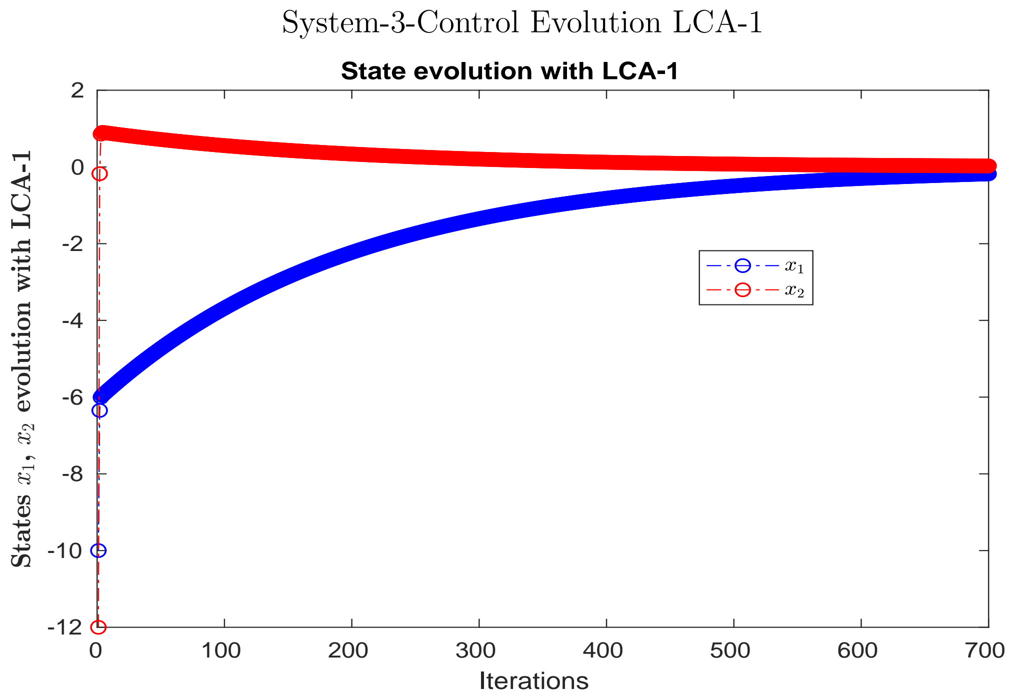




**Figure 4.8:** Control Evolution Comparison LCA-1 v LCA-2

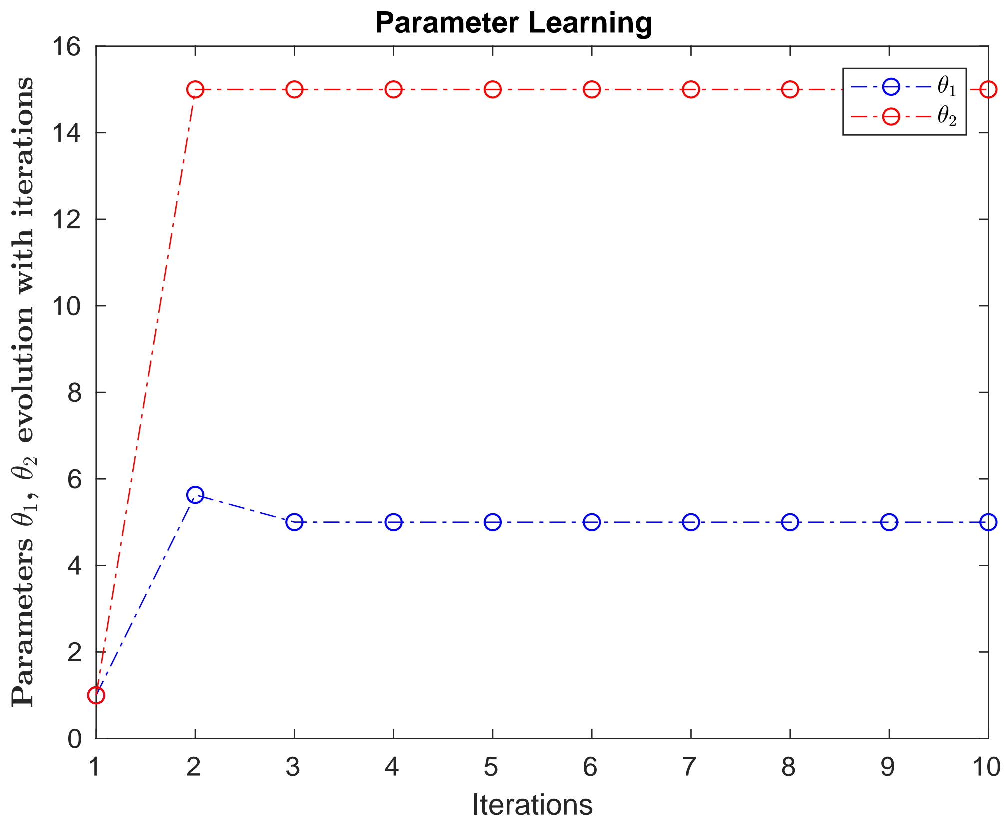



**Figure 4.9:** System-3 Parameter Learning Evolution with LCA-2

### *4.1.4 System - 4*

The system considered is an open loop unstable 3rd order system with 2 inputs per agent and 2 Agents. Matrices A, B of the system state space representation and state weighting matrix in the cost function Q are given below. The parameters chosen are $\theta_1 = 50$, $\theta_2 = 90$, $\theta_{max} = 100$ and $\theta_{min} = 0.1$

$$A = \begin{pmatrix} 1 & 3 & 2 \\ 3 & 1 & 5 \\ 2 & 9 & 7 \end{pmatrix} B = \begin{pmatrix} 1 & 4 & 2 & 5 \\ 3 & 5 & 7 & 3 \\ 2 & 7 & 1 & 2 \end{pmatrix} Q = \begin{pmatrix} 20 & 0 & 0 \\ 0 & 30 & 0 \\ 0 & 0 & 50 \end{pmatrix}$$

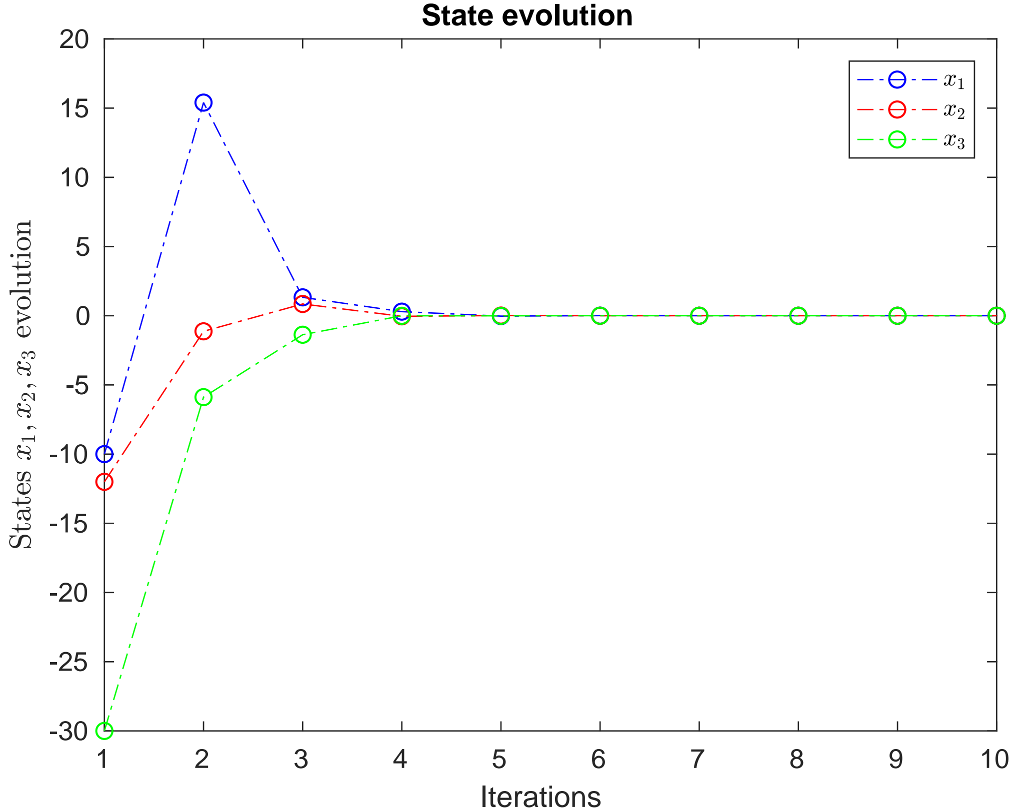



**Figure 4.10:** System-4 State Evolution with LCA-2

Some observations on plots of System-4:

- As seen in Figure 4.10, states are driven to the origin in 5 time-steps. The

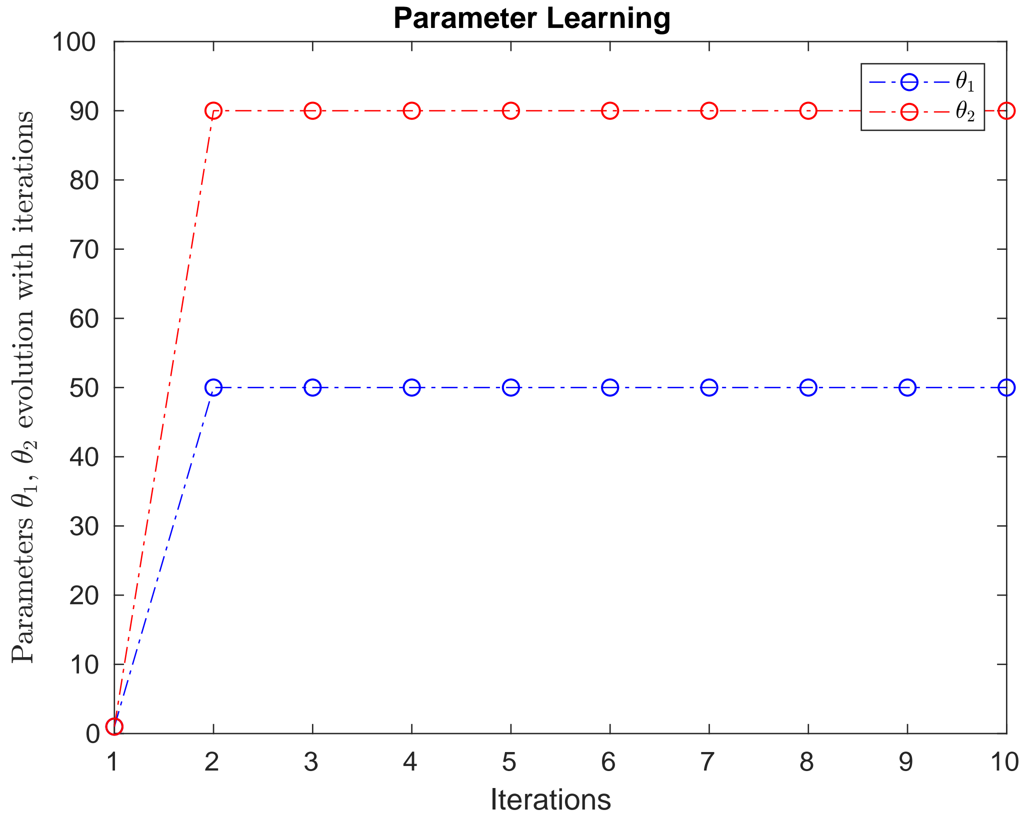



**Figure 4.11:** System-4 Parameter Learning Evolution with LCA-2

algorithm LCA-2 thus works well even for multi-input agent systems.

- It is seen that the learning algorithm converges faster to true parameter $\theta$ values with multi-input agent systems. As seen in Figure 4.11, convergence happened in just two steps. The learning algorithm is run 5 times per observation.

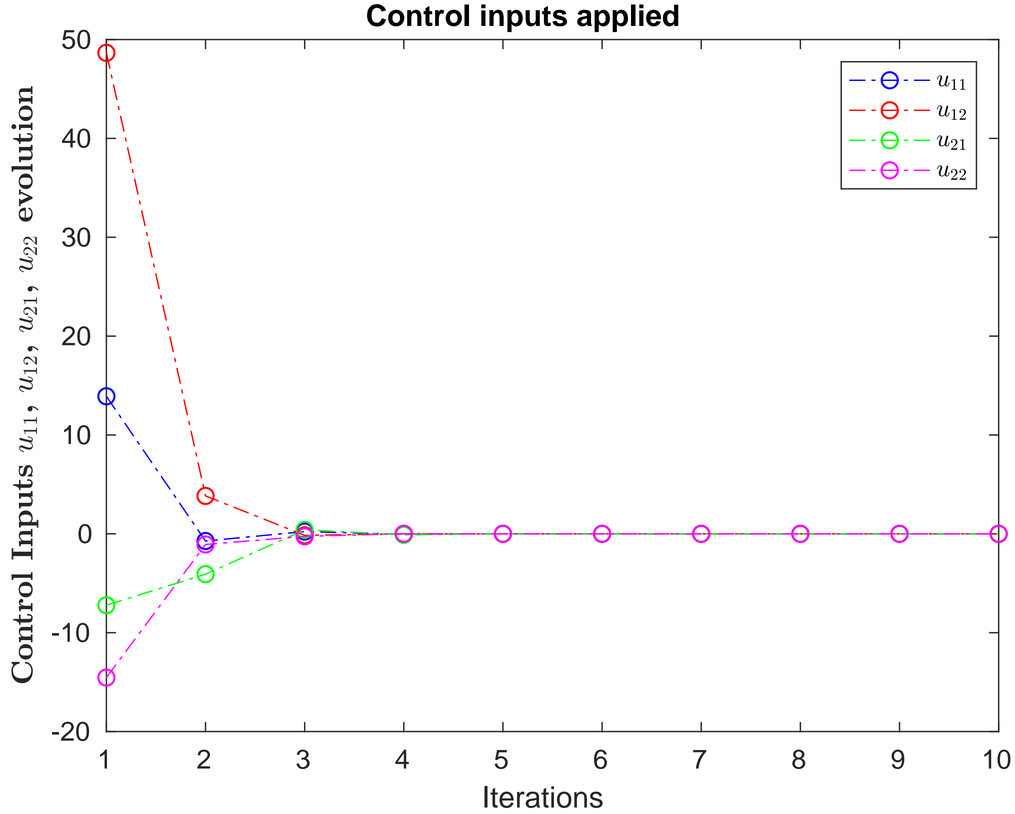



**Figure 4.12:** System-4 Control Input Sequence with LCA-2

### *4.1.5 Simulation results for the 2-Agent mobile robot model*

The system matrices are already presented in Chapter 3. Plots for the system are as follows. Agent specific parameters $\theta_1 = 0.6, \theta_2 = 0.3$, the state weighting matrix $Q$ in cost function is given by

$$Q = \begin{pmatrix} 5.0138 & -3.0083 \\ -3.0083 & 5.0138 \end{pmatrix}$$

Some observations from the plots:

- As seen in Figure 4.13 both states start at -300, -300 and reach origin in about 40 iterations. -300, -300 are the initial states set so that the system has control

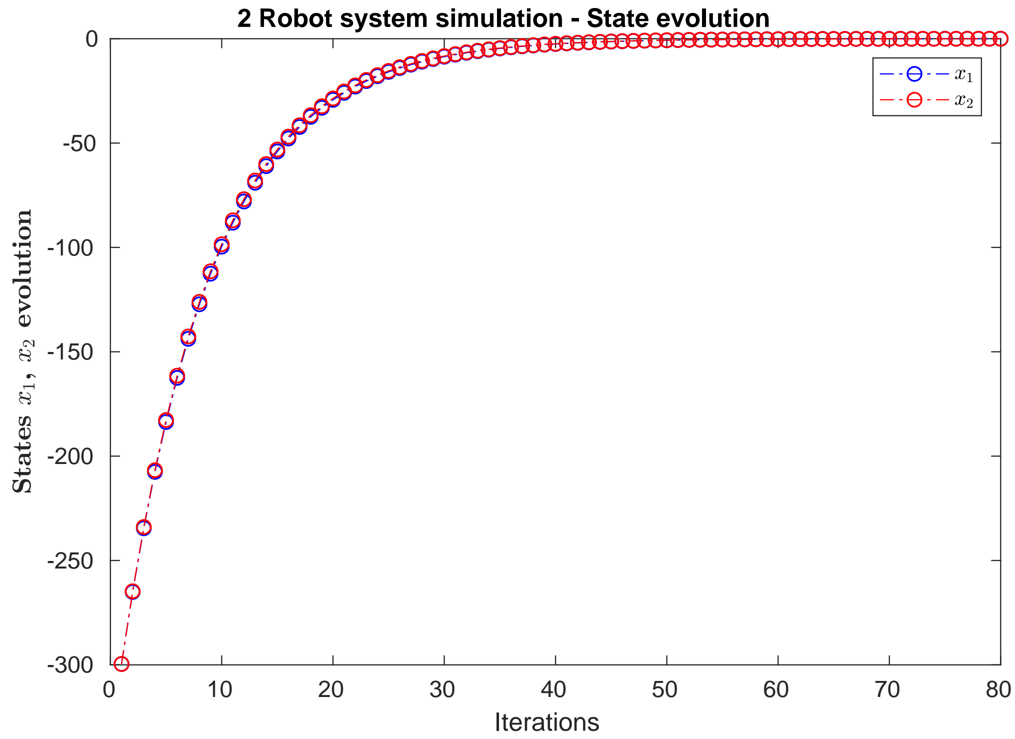



**Figure 4.13:** Mobile Robot Simulation - State Evolution LCA-2

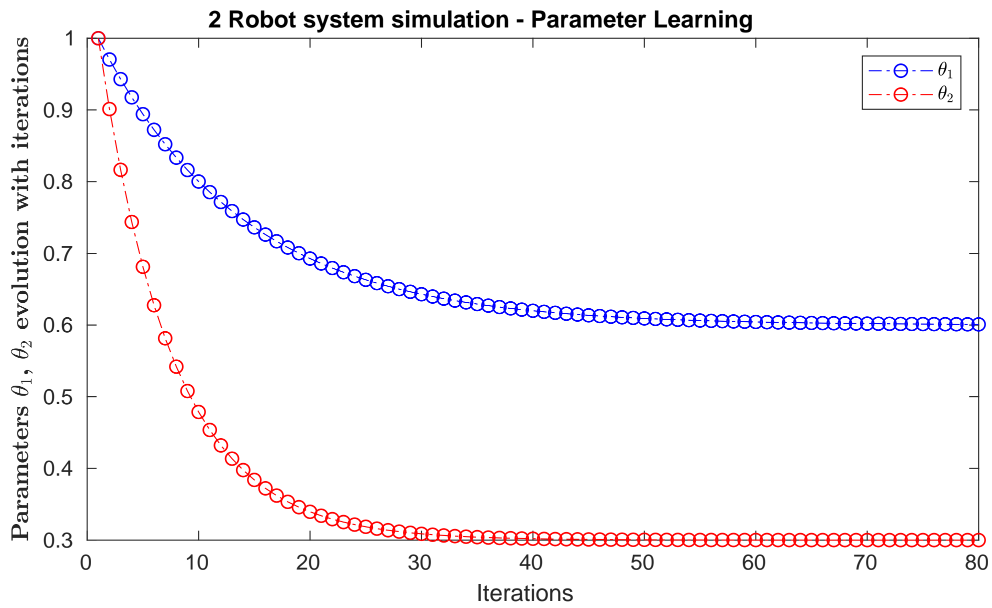



**Figure 4.14:** Mobile Robot Simulation - Parameter Learning LCA-2

inputs in the range of actuation as described in chapter 3.

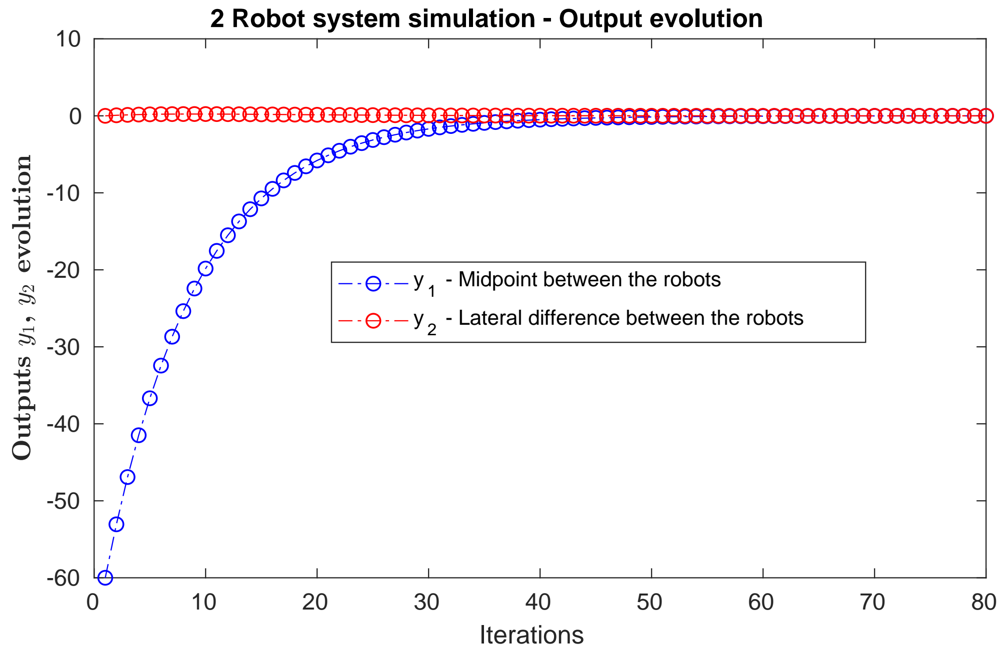



**Figure 4.15:** Mobile Robot Simulation - Output Evolution LCA-2

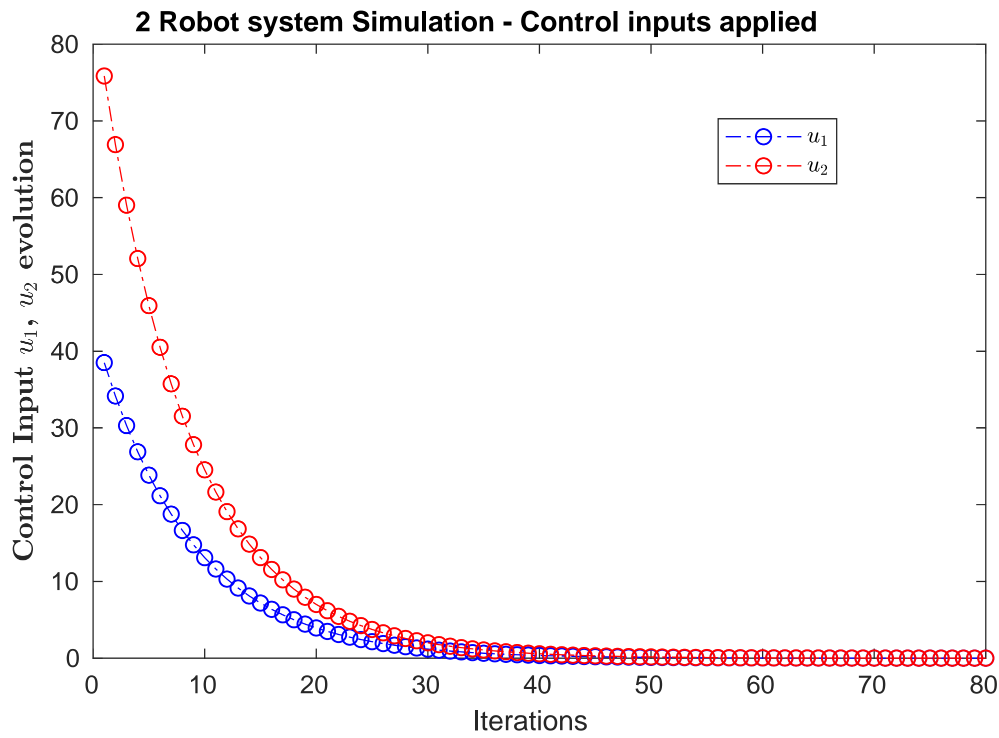



**Figure 4.16:** Mobile Robot Simulation - Control Evolution LCA-2

- The main plot of interest is Figure 4.15. As it is seen, both robots start with zero lateral difference and maintain it throughout their journey, regardless of the fact that they are solving different cost functions and learning each other parameters while doing so.

- From the Figure 4.16, it is seen that the control inputs to robots in simulation start and end in the range [0,70], perfect range for Hercules robot actuation

#### *4.1.6 Data from Experiments - LCA-2*

The two robot system, using the experimental setup described in Chapter-3 and the values for $Q$, $\theta_1, \theta_2$ and $x_{init}, y_{init}$ chosen as described in the section *Simulation results for the 2-Agent mobile robot model*, are put through experiments to check if LCA-2 can be implemented in real systems. Multiple trials are conducted. Occlusions of the robots in Mocap system were a source of external unpredictable disturbance. Trial-1 is unaffected by occlusions. Trials - 2,3 and 4 are all affected by occlusion. In addition, in trial-4 the system is subjected to an intentional disturbance and its behavior observed.

Plots obtained and observations are given below for all trials:

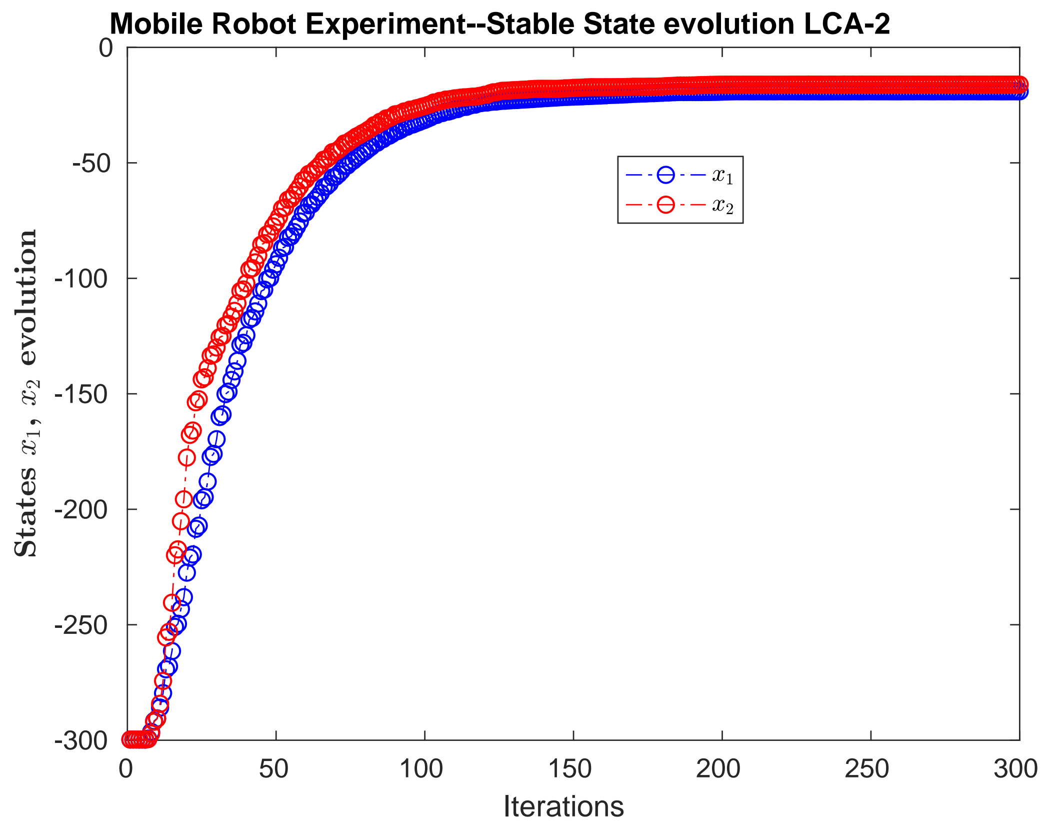



**Figure 4.17:** Trial-1 Mobile Robot Experiment - State Evolution

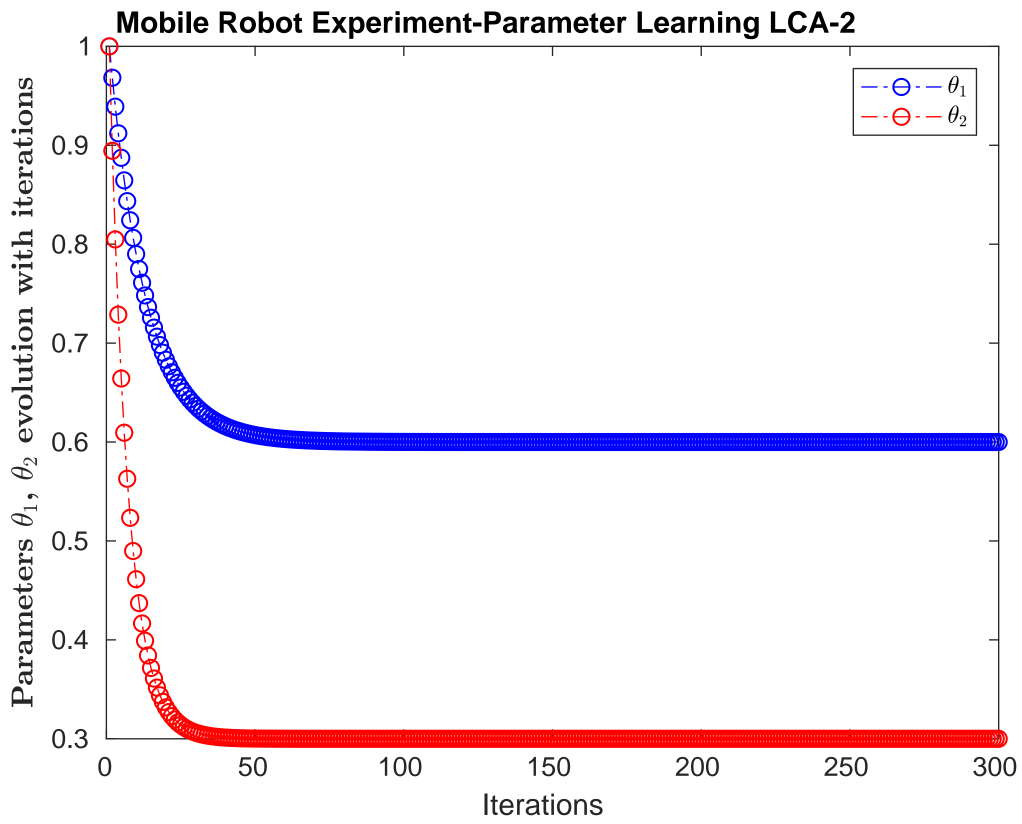



**Figure 4.18:** Trial-1 Mobile Robot Experiment - Parameter Evolution

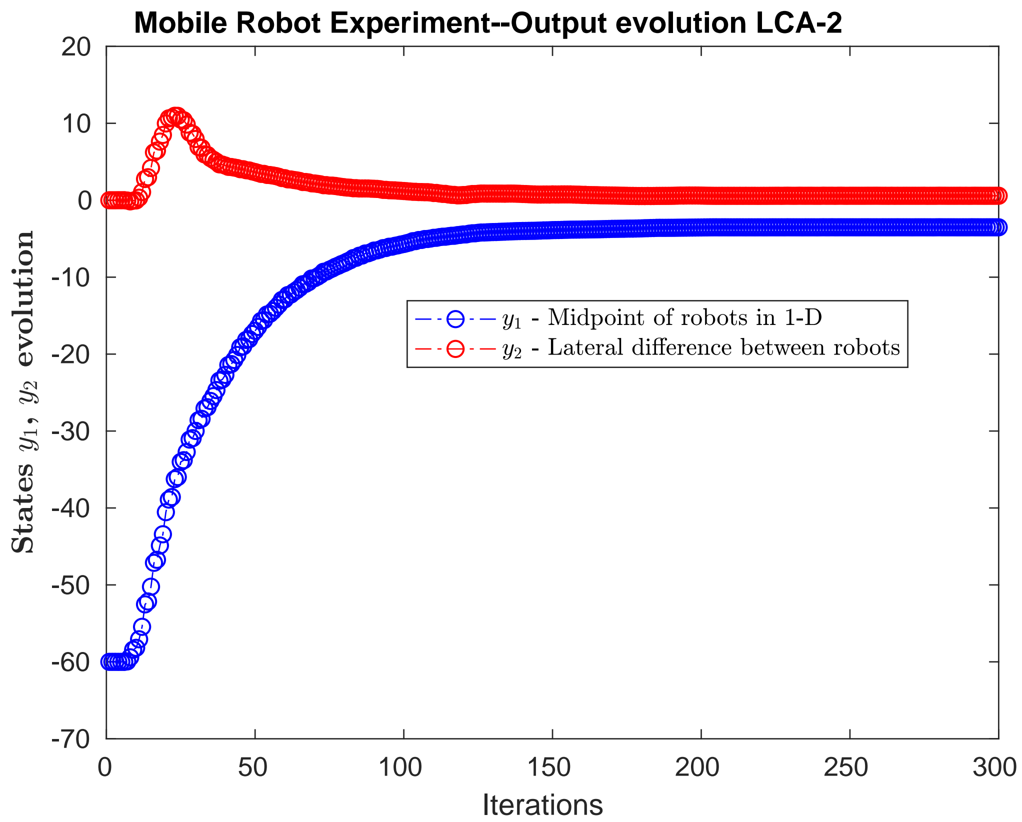



**Figure 4.19:** Trial-1 Mobile Robot Experiment - Output Evolution

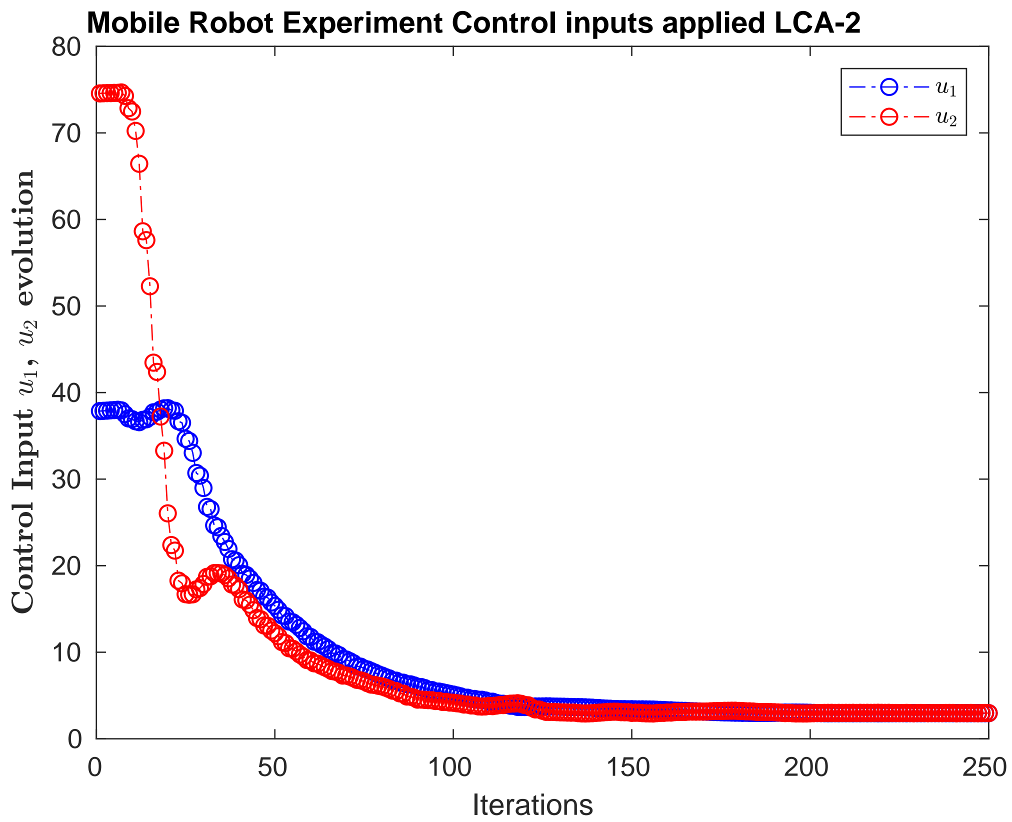



**Figure 4.20:** Trial-1 Mobile Robot Experiment - Control Input Evolution

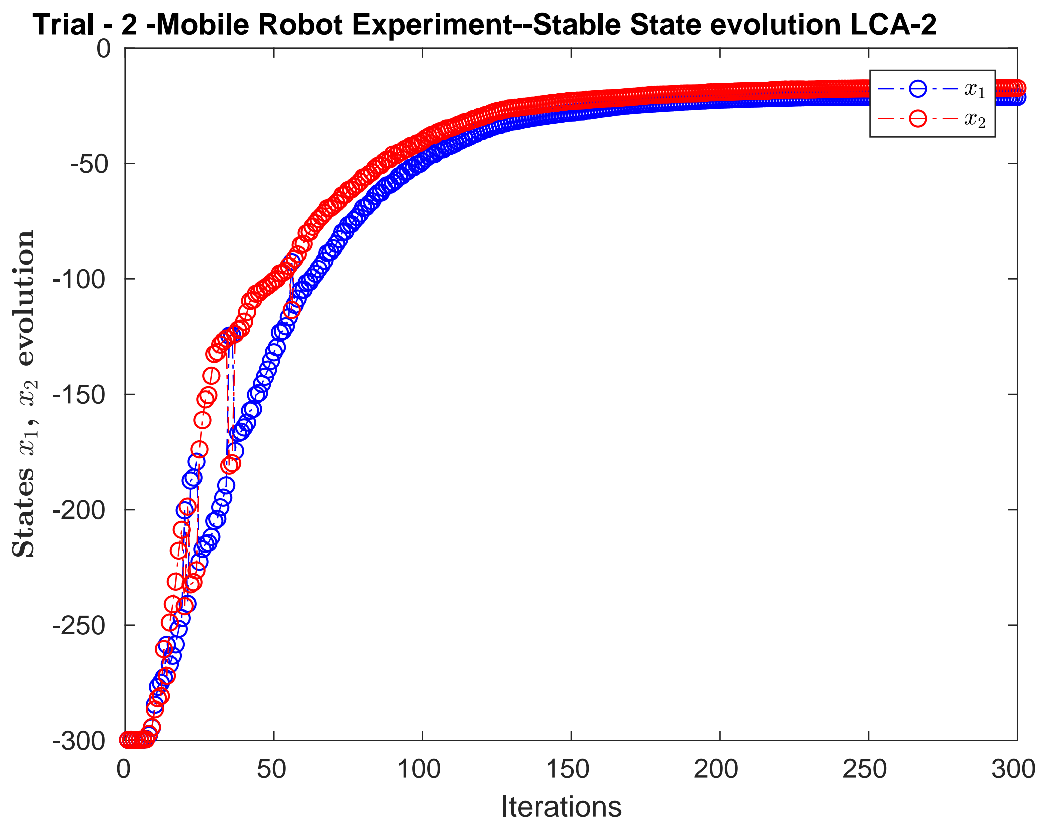


**Figure 4.21:** Trial-2 Mobile Robot Experiment - State Evolution

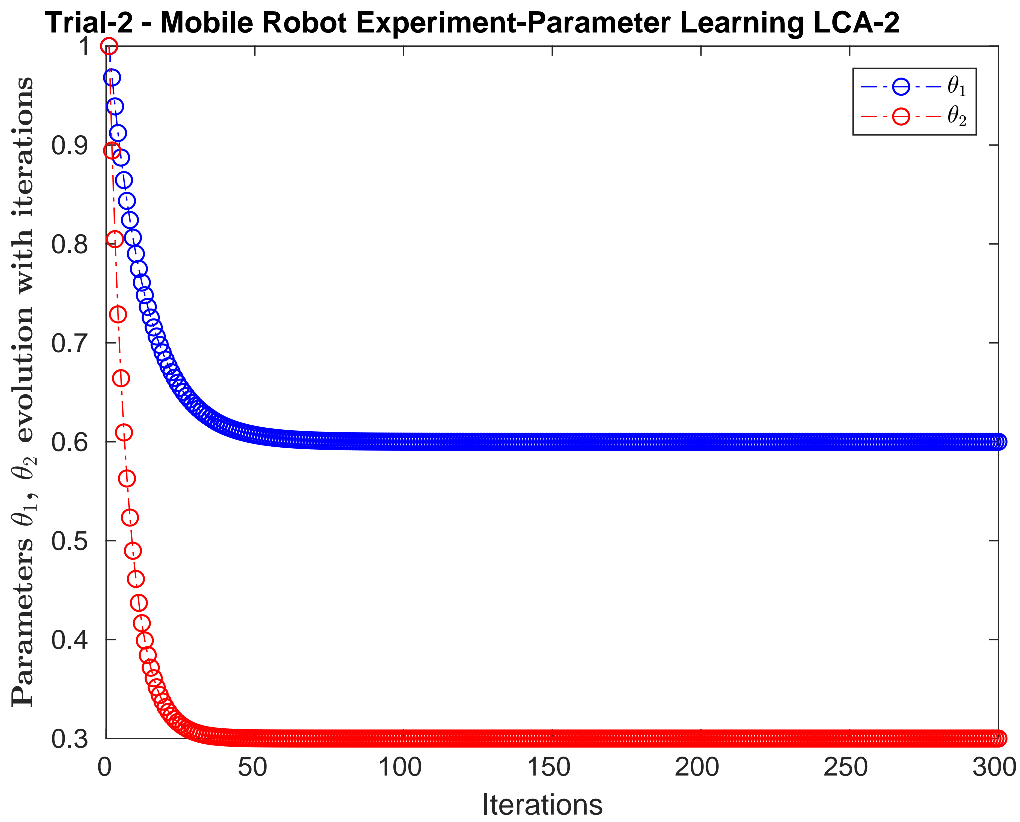



**Figure 4.22:** Trial-2 Mobile Robot Experiment - Parameter Evolution

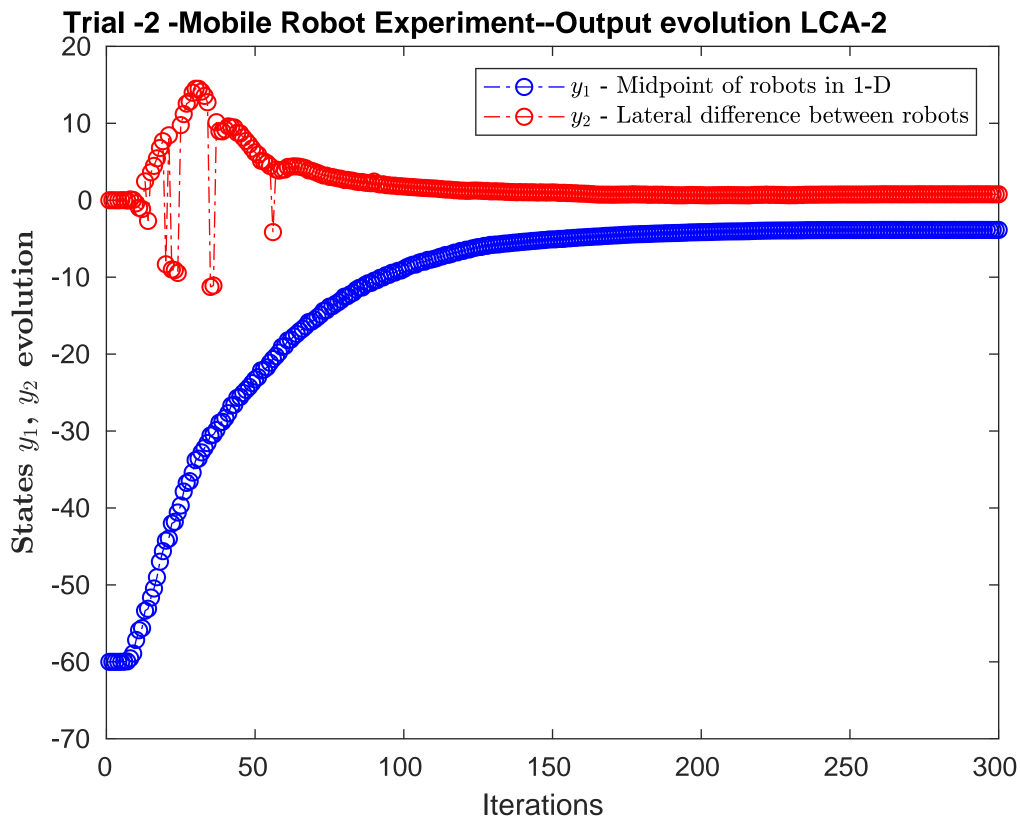



**Figure 4.23:** Trial-2 Mobile Robot Experiment - Output Evolution

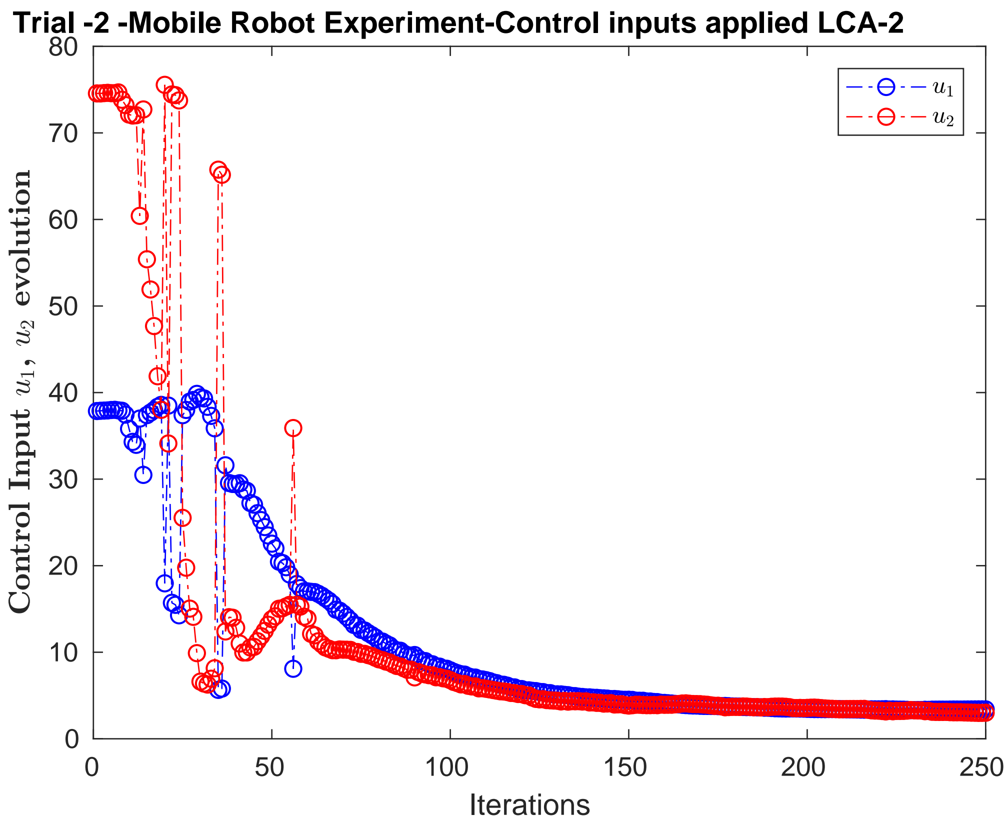



**Figure 4.24:** Trial-2 Mobile Robot Experiment - Control Input Evolution

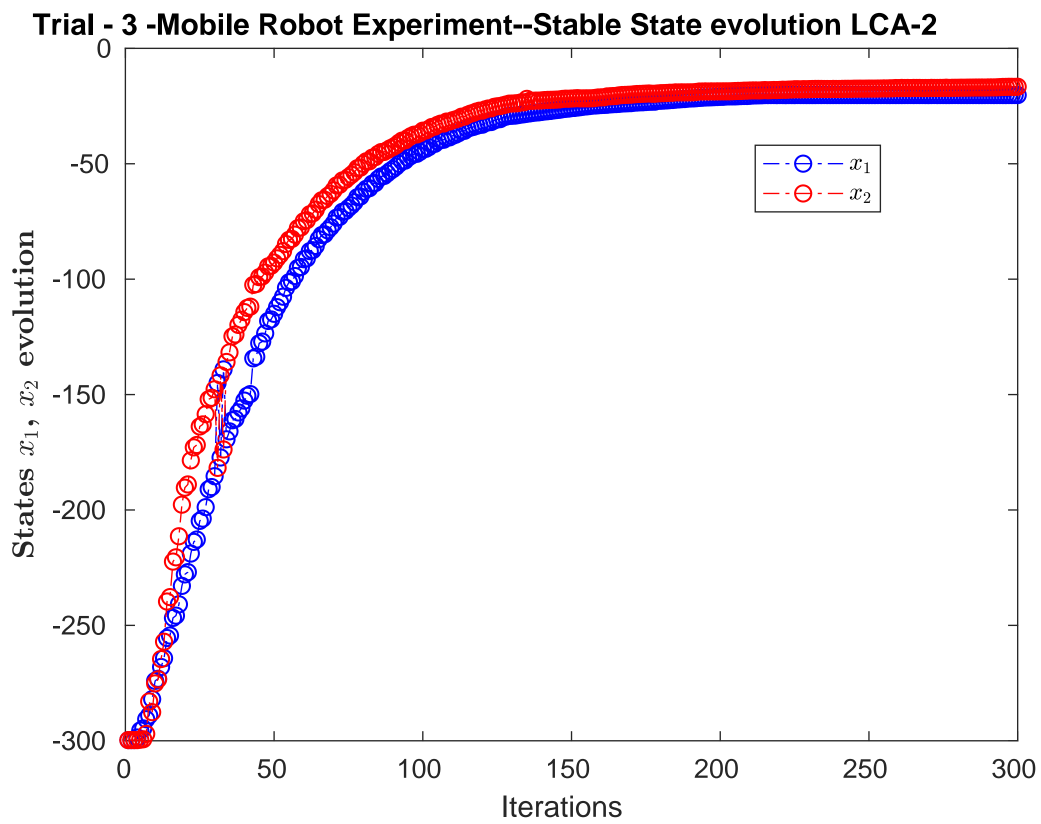



**Figure 4.25:** Trial-3 Mobile Robot Experiment - State Evolution

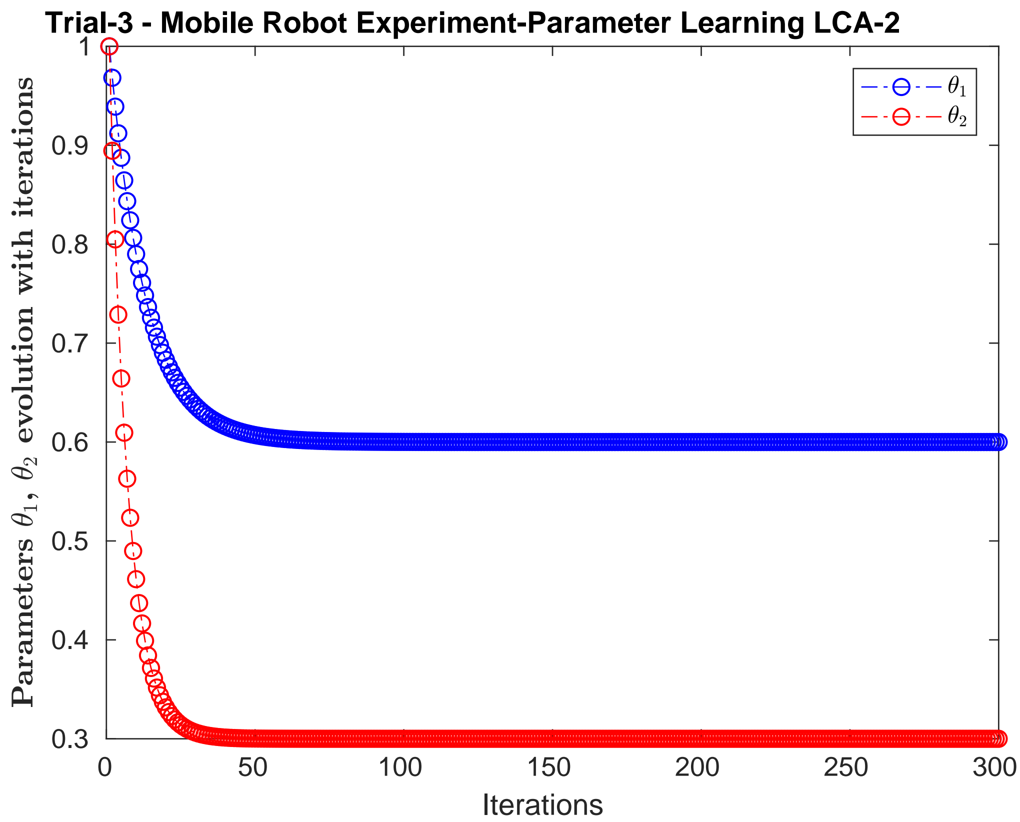



**Figure 4.26:** Trial-3 Mobile Robot Experiment - Parameter Evolution

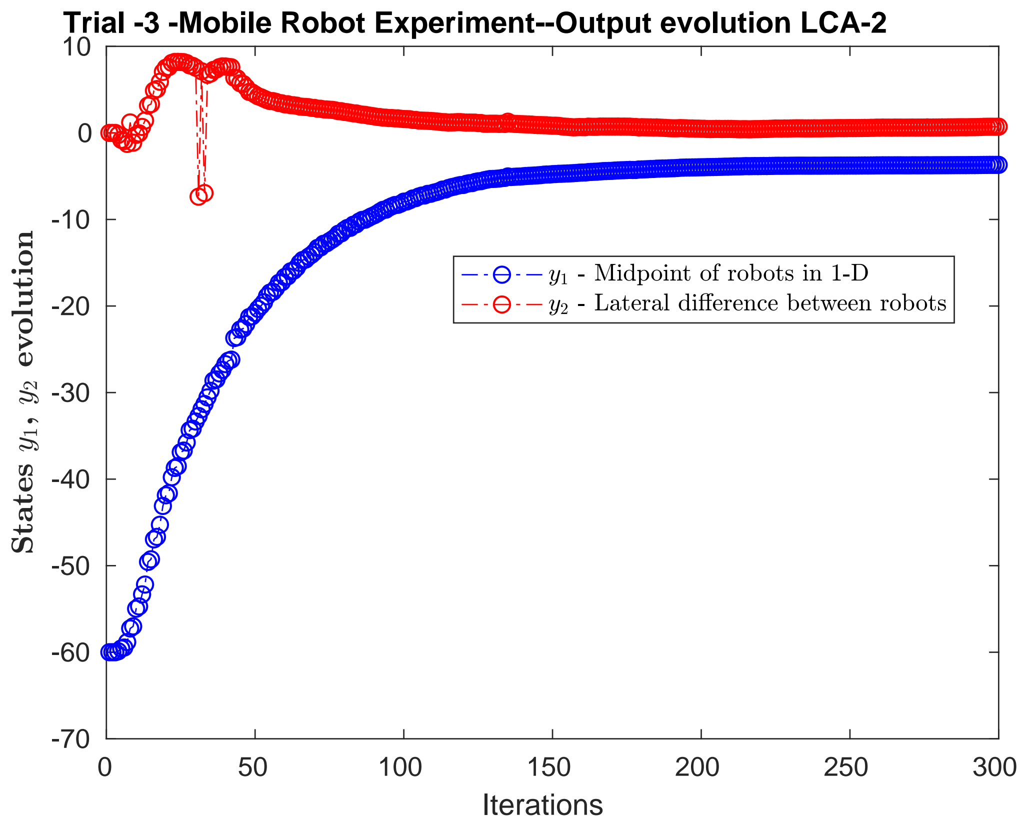



**Figure 4.27:** Trial-3 Mobile Robot Experiment - Output Evolution

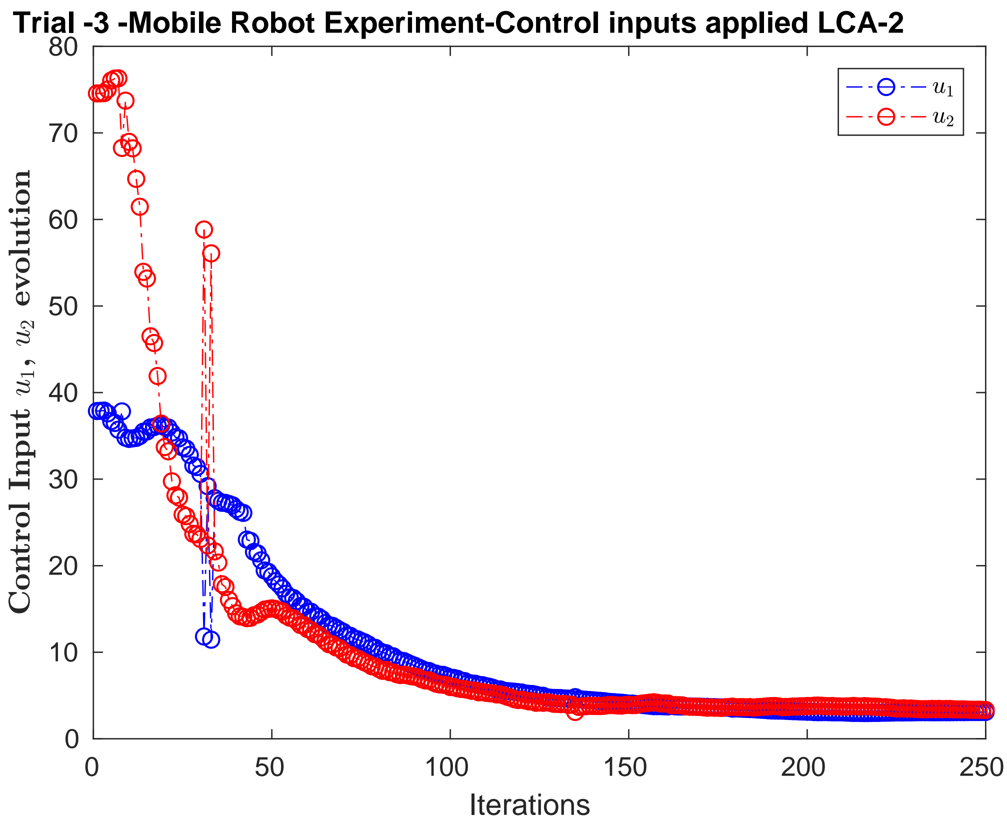



**Figure 4.28:** Trial-3 Mobile Robot Experiment - Control Input Evolution

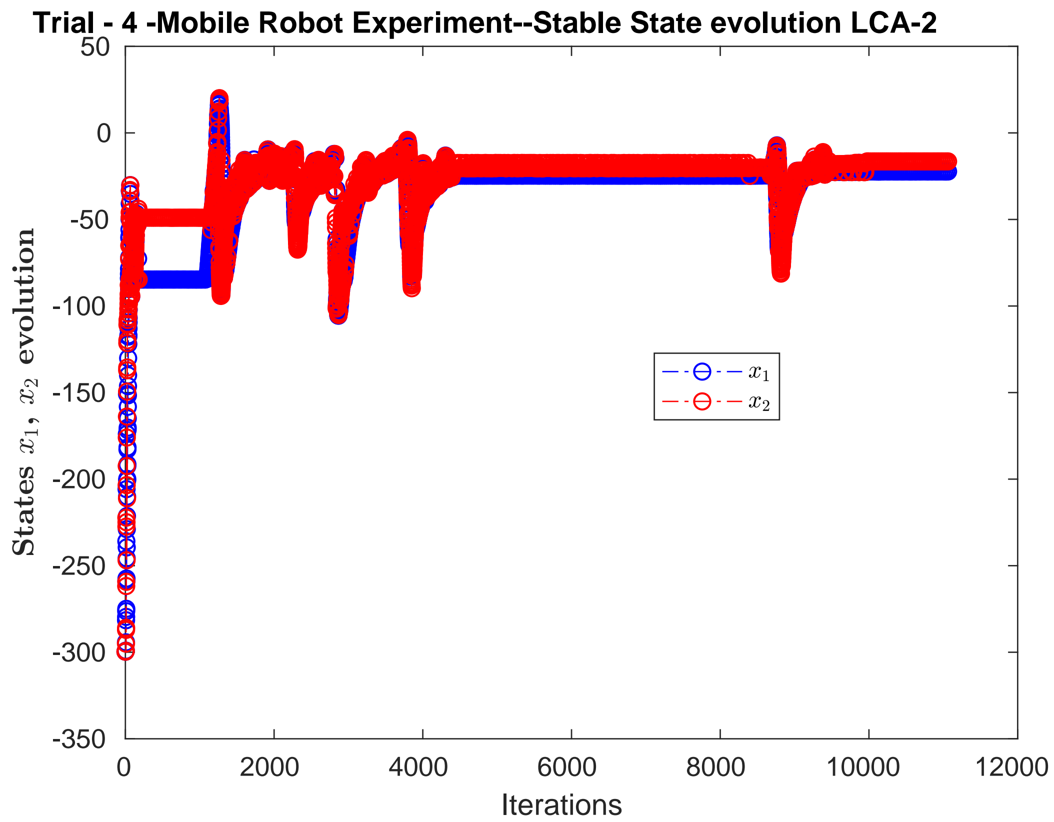



**Figure 4.29:** Trial-4 Mobile Robot Experiment - State Evolution

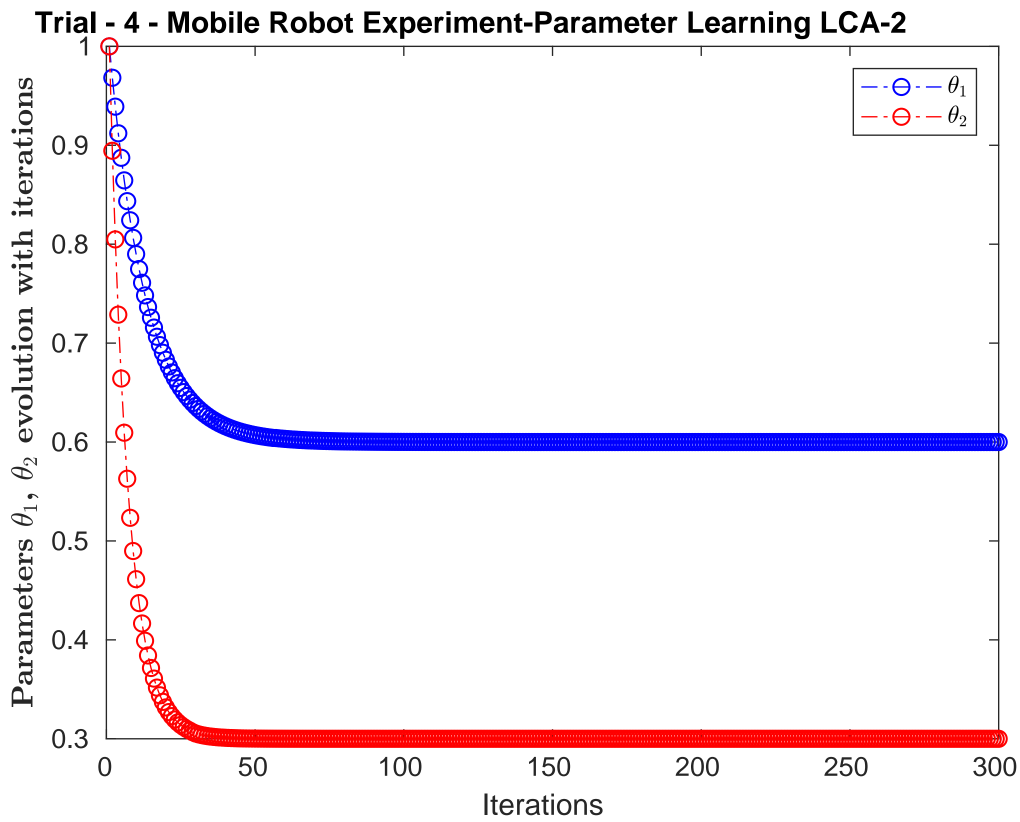



**Figure 4.30:** Trial-4 Mobile Robot Experiment - Parameter Evolution

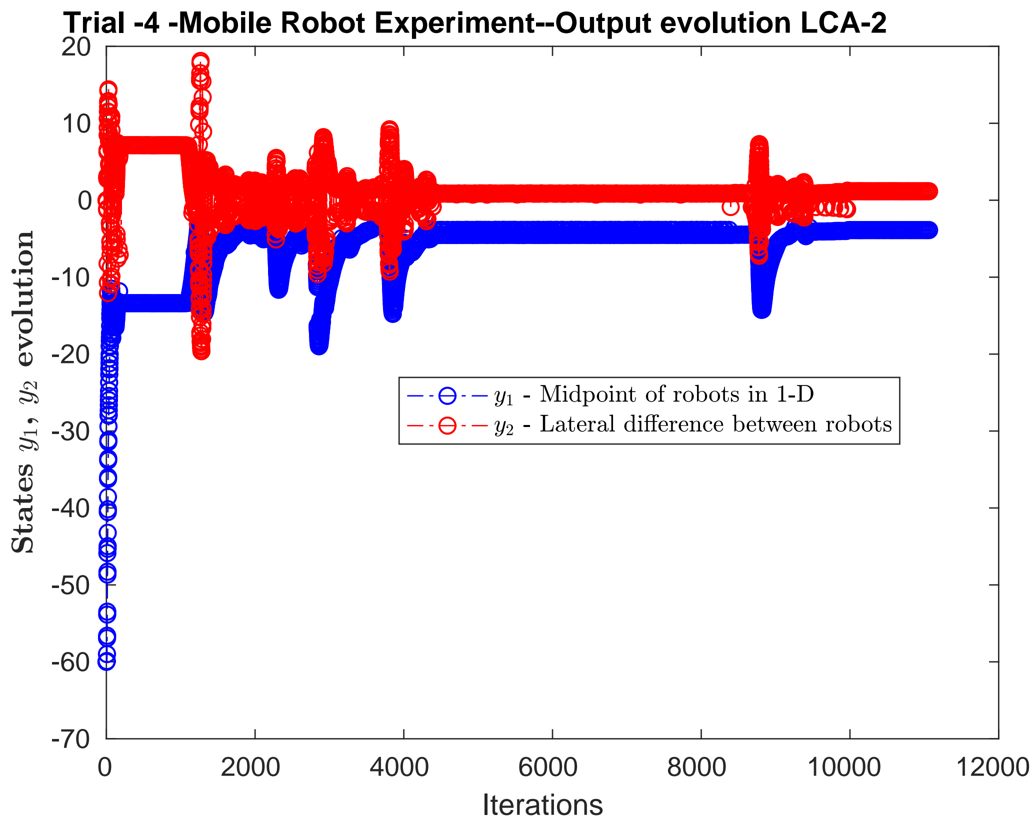



**Figure 4.31:** Trial-4 Mobile Robot Experiment - Output Evolution

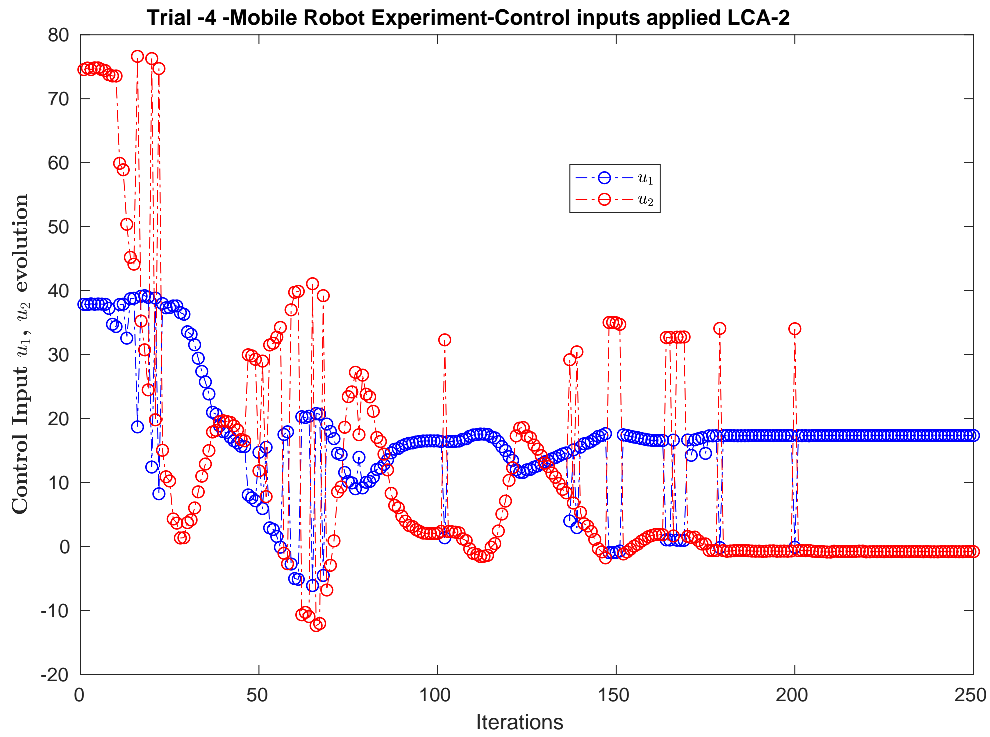



**Figure 4.32:** Trial-4 Mobile Robot Experiment - Control Input Evolution

Experimental results obtained resemble delayed simulation results presented in previous section. This can be attributed to delays introduced in the system during data transmission. As seen in trial -2 and trial-3, the system was able to recover from disturbances due to occlusions. Trial-4 demonstrates the robustness of the algorithm to external disturbances. The system is subjected to disturbances due to occlusion, which are random, as well as a deterministic disturbance applied to the system. The last deviation from origin in Figure 4.29 is due to a deterministic disturbance applied, while others are due to occlusion. As evident, the system was able to resist disturbances and get back to goal state as discussed. It is, therefore, concluded that LCA-2 is suitable for real-time applications.

#### *4.1.7 Data from Experiments - LCA-1*

Experimental data obtained by running LCA-1 is presented in this section. Agent preference parameters $\theta_1, \theta_2$, and the state weighting matrix $P$ in the cost function 2.6 chosen are different to experiment trials conducted for LCA-2. This is to ensure that the control outputs generated are with in actuation limits applicable for Hercules robots. Values chosen are $\theta_1 = 0.2, \theta_2 = 0.4$,

$$P = \begin{pmatrix} 15.04203 & -9.030024 \\ -9.030024 & 15.04203 \end{pmatrix}.$$

Results are given below:

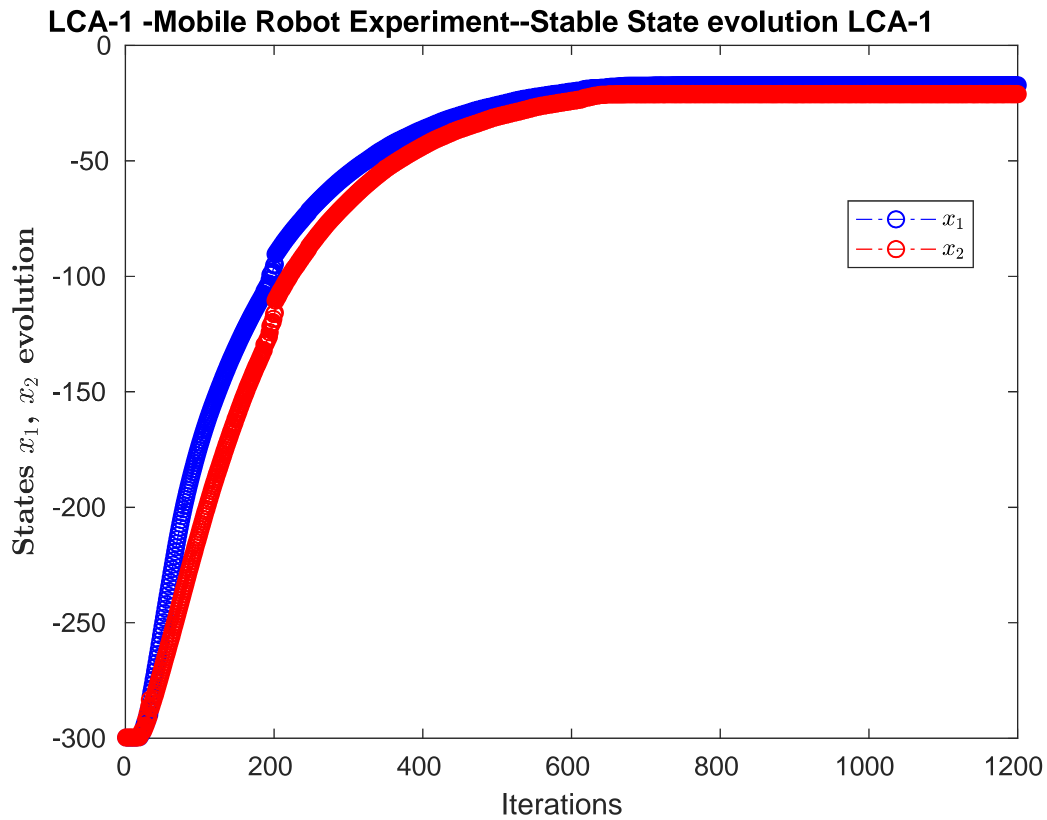



**Figure 4.33:** Mobile Robot Experiment - State Evolution

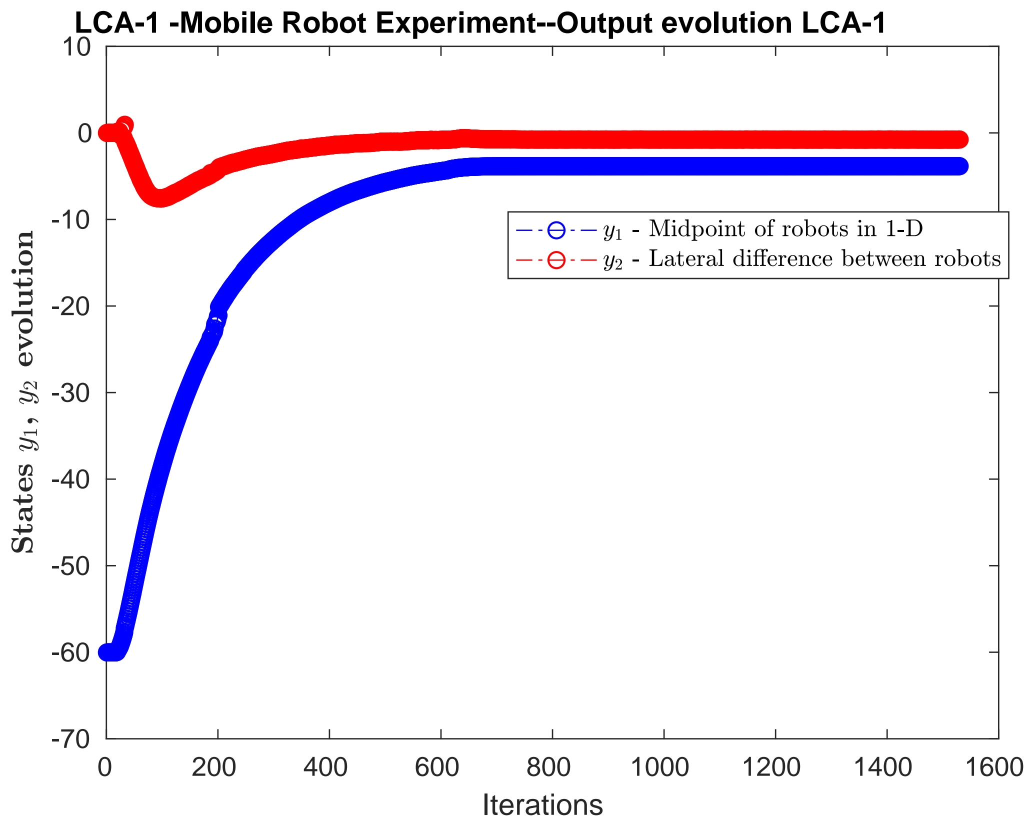



**Figure 4.34:** Mobile Robot Experiment - Output Evolution

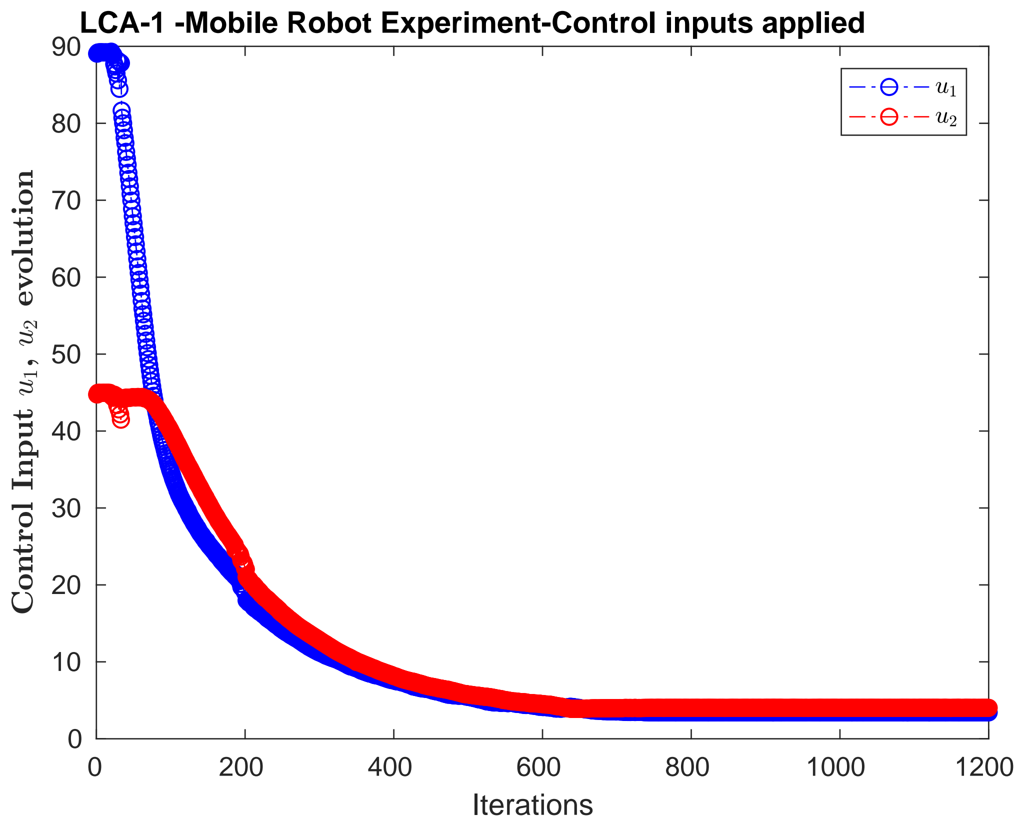



**Figure 4.35:** Mobile Robot Experiment - Control Input Evolution

As seen, without any occlusions, LCA-1 experimental evolution is smoother compared to LCA-2. However, LCA-1 failed to resist disturbances and the system became unstable with occlusion as well as with added disturbances.

### *4.1.8 Comparison: LCA-2 v LCA-1*

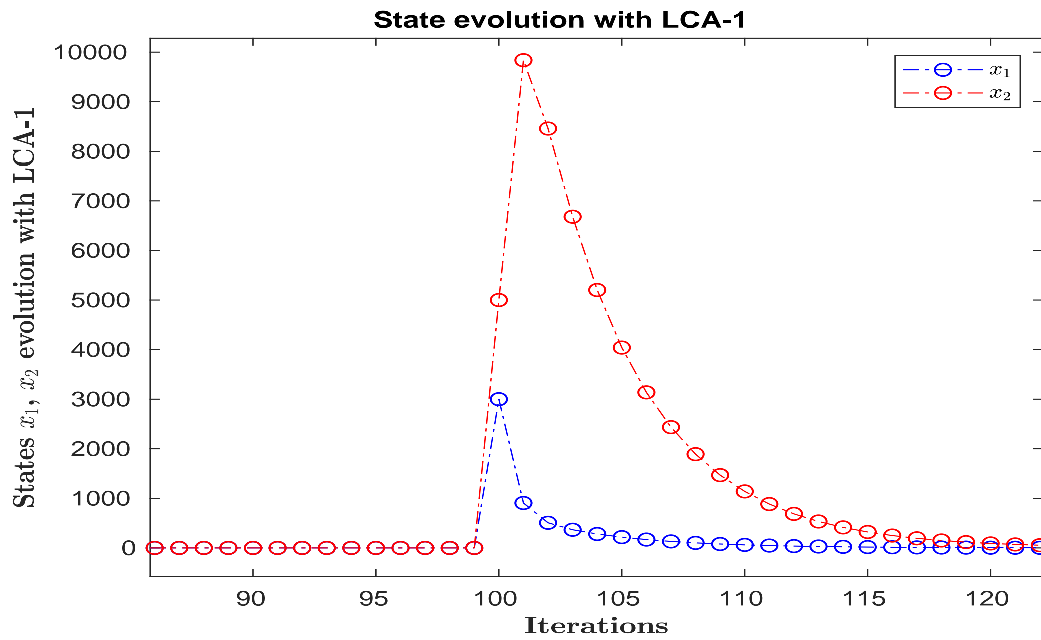
### *4.1.9 Reaction to disturbances after learning*

Since LCA-2, as mentioned is *farsighted* compared to LCA-1, when subjected to a disturbance after learning, MAS running LCA-2 reacts better compared to MAS running LCA-1. Consider a 2 agent 2 state system with parameters $\theta_1 = 40$, $\theta_2 = 80$, $\theta_{max} = 100$ and $\theta_{min} = 0.1$, at initial state [-100; -120] and matrices

$$A = \begin{pmatrix} 0.5 & 0.1 \\ 3 & 0.9 \end{pmatrix} B = \begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix} Q = \begin{pmatrix} 0.2 & 0.1 \\ 0.1 & 0.4 \end{pmatrix}$$

. After the initial learning phase, when the agents are at Nash equilibrium, a disturbance is applied and the state of the system is moved from origin to [3000;5000] at time step 100. Reaction of a MAS running LCA-1 and LCA-2 are shown in Figure 4.36. As seen, MAS with LCA-1 overreacts compared to LCA-2. A similar behavior is observed with multiple system examples. It is observed that during initial learning phase, LCA-1 provides gradual change in states and controls compared to LCA-2, whereas after learning LCA-2 provides a gradual change in states and controls in response to disturbances. During learning, however, the rate of change in states and controls obtained with LCA-2 are not prohibitively bad for load transport.

Reaction to a disturbance - LCA-1




Reaction to a disturbance - LCA-2

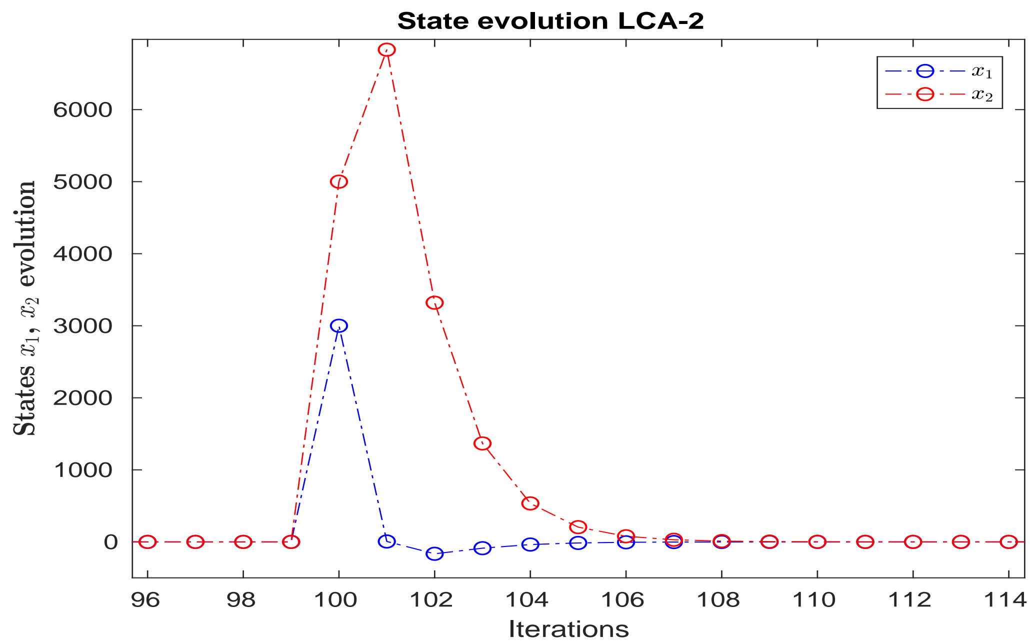



**Figure 4.36:** Reaction to a Disturbance LCA-1 v LCA-2

### *4.1.10 Stability of MAS with small P or Q elements*

Closed-loop dynamics of MAS running LCA-1 are given by equation 2.9. The characteristic matrix of the closed-loop system tends to an Identity matrix when elements of P or Q matrices are made smaller. This makes the closed-loop system unstable/marginally stable. The same systems, however, can be stabilized when using LCA-2. Following examples demonstrate this behavior. Consider the system presented in the subsection *Reaction to disturbances after learning* with P or Q state cost weighting matrix as

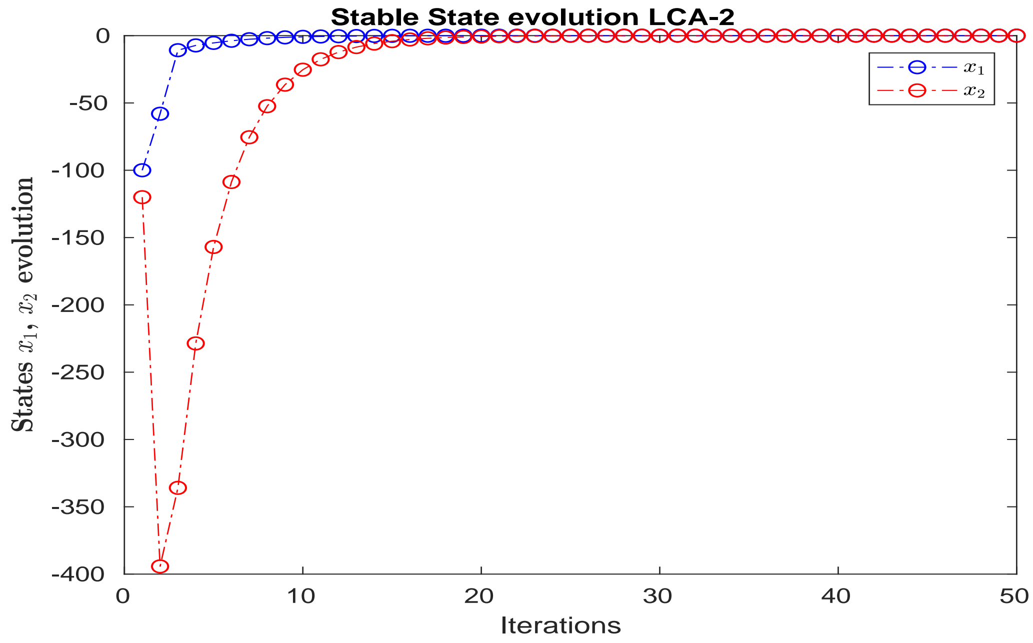
$$P = \begin{pmatrix} 0.02 & 0.01 \\ 0.01 & 0.04 \end{pmatrix}.$$

Figure 4.37 compares state evolution with LCA-1 and LCA-2. Consider another MAS with following parameters and matrices: $\theta_1 = 50$, $\theta_2 = 90$, $\theta_{max} = 100$ and $\theta_{min} = 0.1$,

$$A = \begin{pmatrix} 1 & 3 & 2 \\ 3 & 1 & 5 \\ 2 & 9 & 7 \end{pmatrix} B = \begin{pmatrix} 1 & 4 & 2 & 5 \\ 3 & 5 & 7 & 3 \\ 2 & 7 & 1 & 2 \end{pmatrix} Q = \begin{pmatrix} 0.02 & 0 & 0 \\ 0 & 0.03 & 0 \\ 0 & 0 & 0.05 \end{pmatrix}.$$

State dynamics with LCA-2 and LCA-1 are shown in Figure 4.38. MAS which are unstable with LCA-1 are asymptotically stable with LCA-2. This is as a result of *farsightedness v shortsightedness* of LCA-2 and LCA-1 respectively.

E.g.1-Stable system with small P matrix-LCA-2




E.g.1-Unstable system with small P matrix-LCA-1

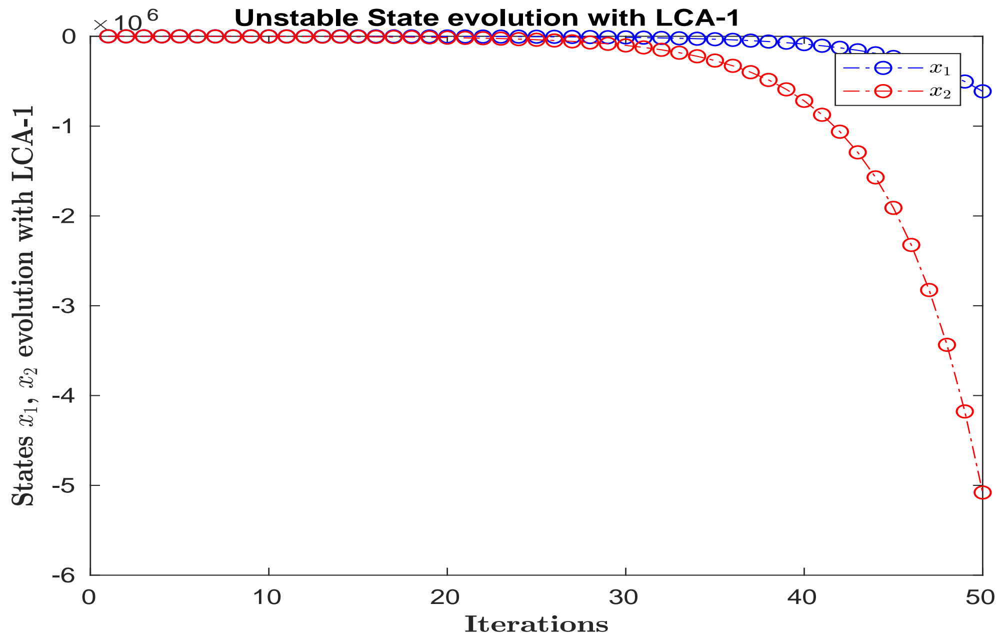
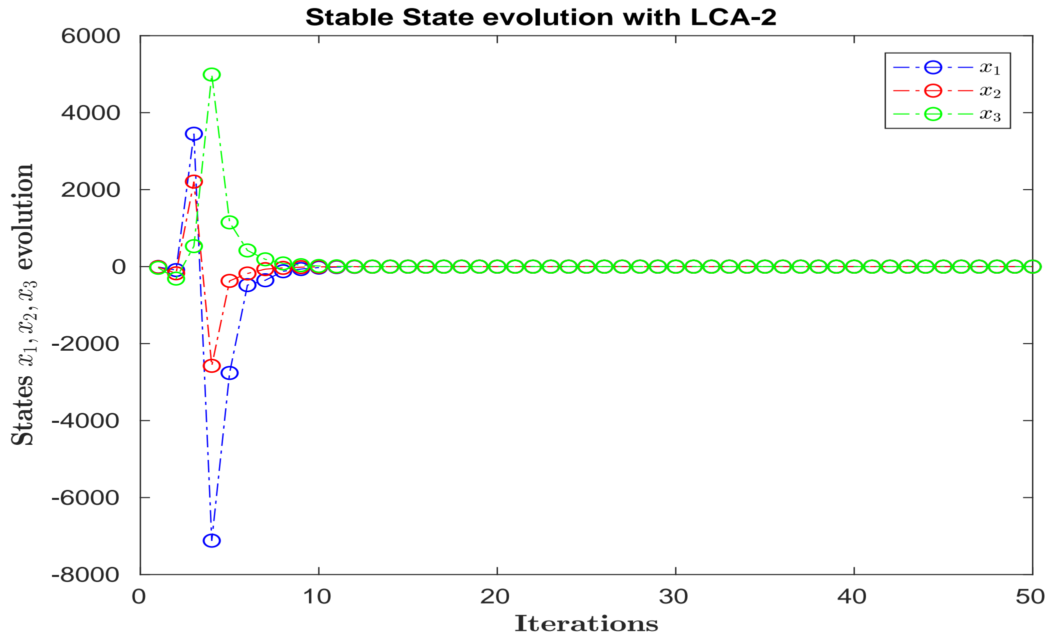



**Figure 4.37:** Example 1: Stability for Small P,Q Matrices - LCA-1 vs LCA-2

E.g.2-Stable system - small P matrix-LCA-2




E.g.2-Unstable system - small P matrix-LCA-1

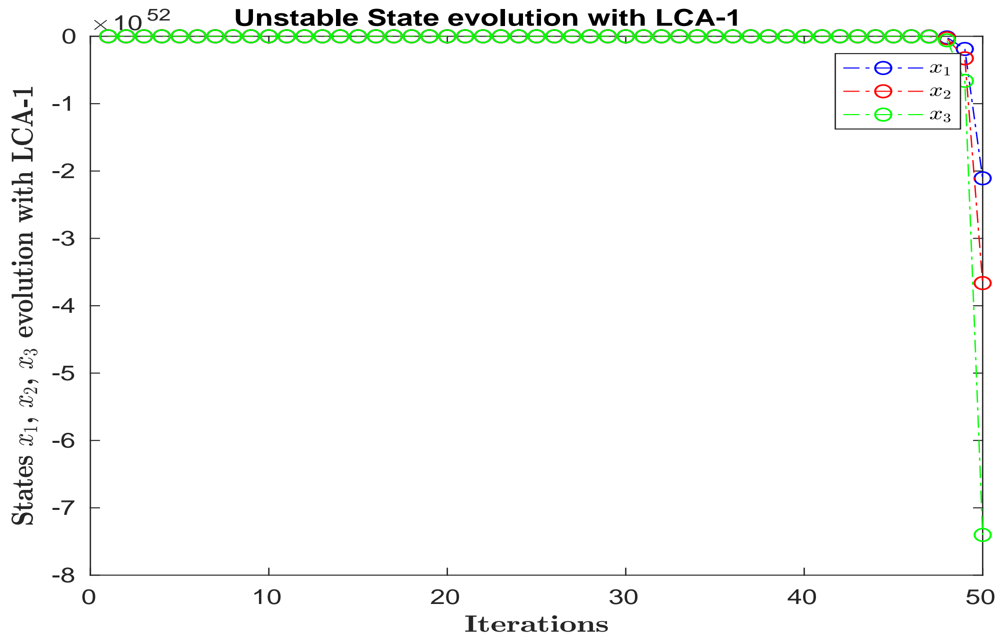
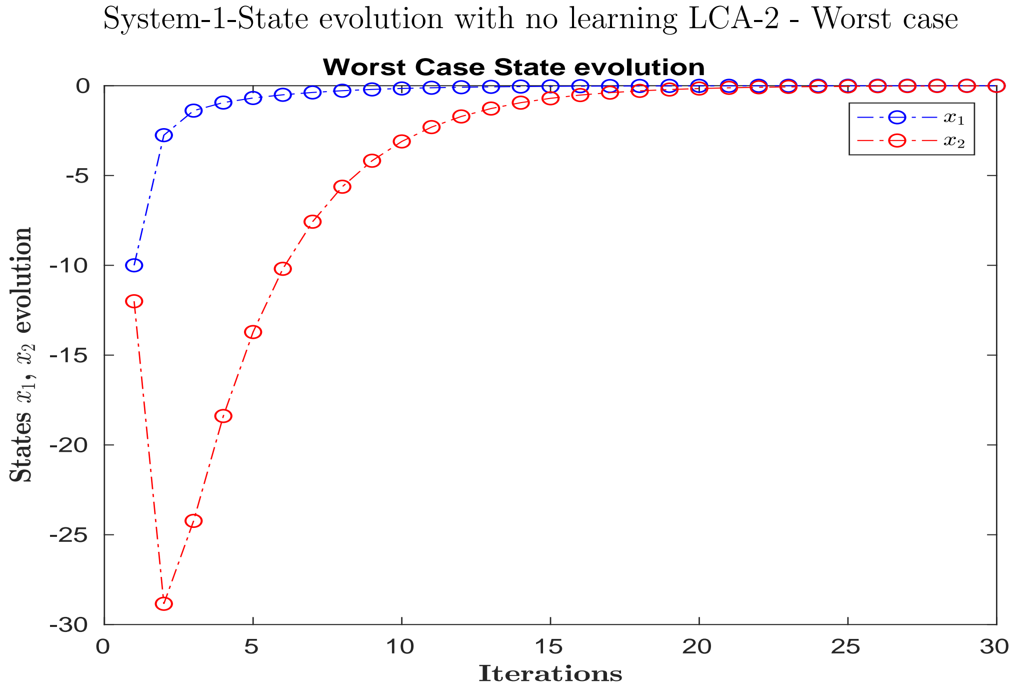



**Figure 4.38:** Example 2: Stability for Small P,Q Matrices - LCA-1 vs LCA-2

#### *4.1.11 LCA-2 v LCA-1 under learning failure*

LCA-1 and LCA-2 behaviors when learning algorithm fails altogether is analyzed in this section. Figures 4.39, 4.40, and 4.41 present state evolution of multi-agent systems System - 1, System - 2 and System - 4 presented in *Simulation Results and Comparison* subsection with LCA-2 and LCA-1 algorithms. As seen in Figure 4.39 and Figure 4.40 , Systems 1, 2 running LCA-2 asymptotically converge to goal state even without any learning, whereas the same systems running LCA-1 are unstable. However for System - 4, as shown in 4.41, both LCA-1 and LCA-2 are unstable without learning. Thus even with no learning some systems running LCA-2 are stable, whereas with learning failure all systems running LCA-1 are becoming unstable. Reason as to why System-4 with learning failure became unstable with LCA-2 needs to be analyzed.



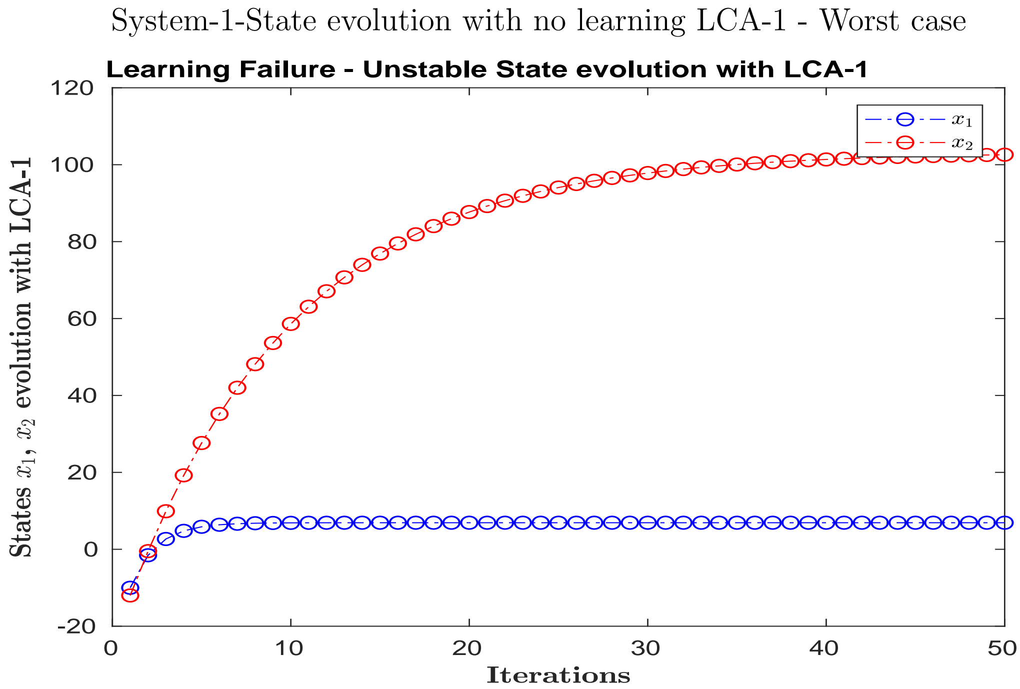
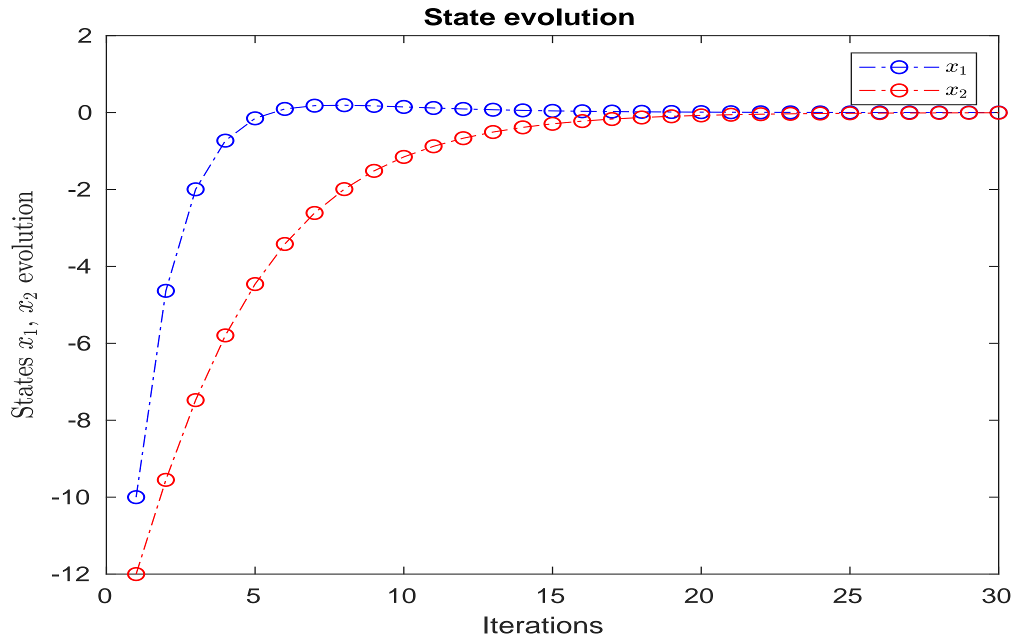




**Figure 4.39:** System-1 Learning Failure LCA-1 vs LCA-2

System-2-State evolution with no learning LCA-2 - Worst case




System-2-State evolution with no learning LCA-1 - Worst case

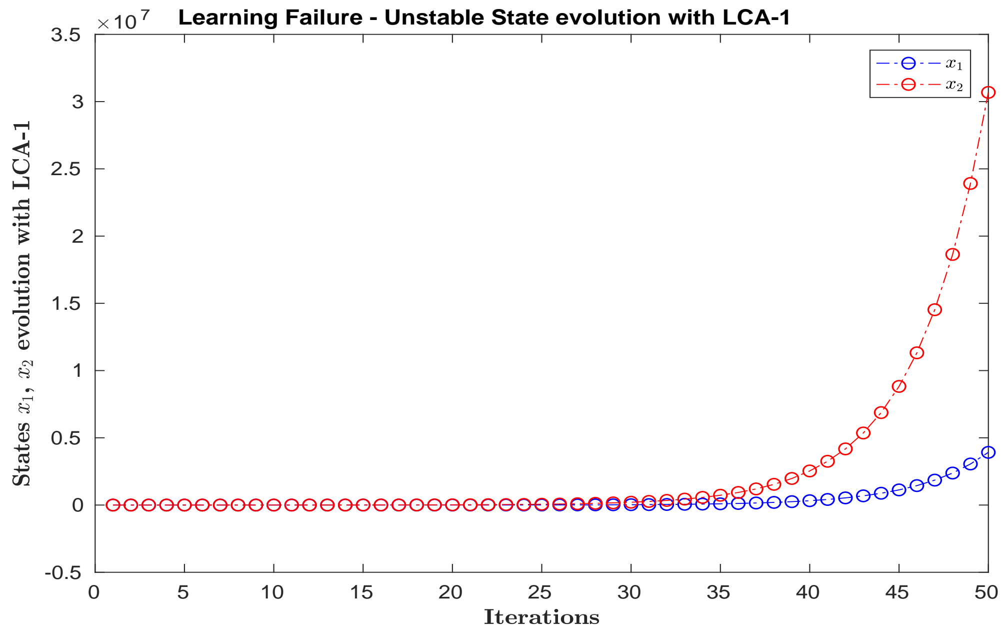
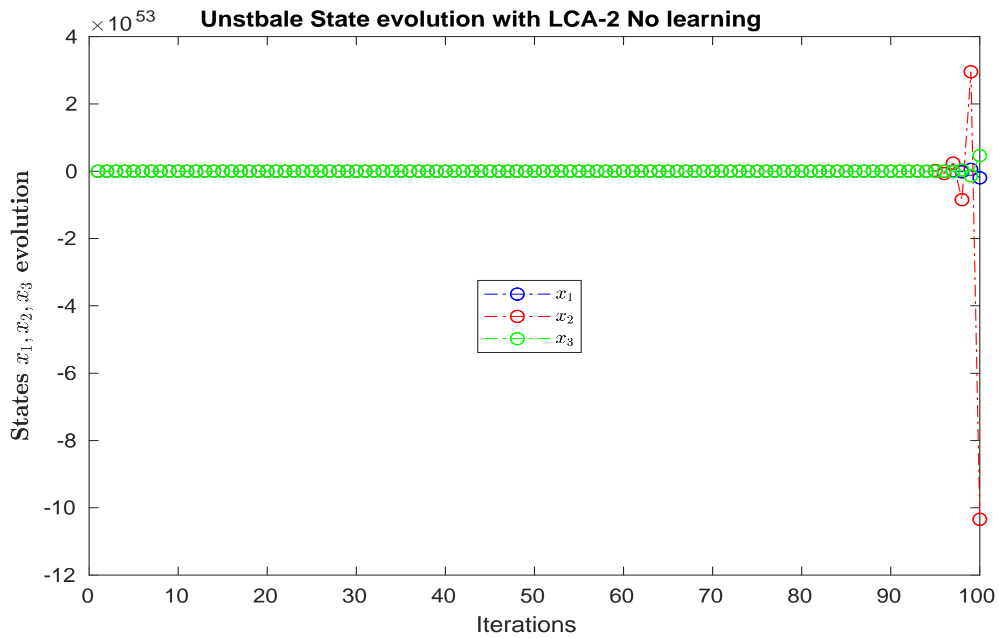



**Figure 4.40:** System-2 Learning Failure LCA-1 vs LCA-2

System-4-State evolution with no learning LCA-2 - Worst case




System-4-State evolution with no learning LCA-1 - Worst case

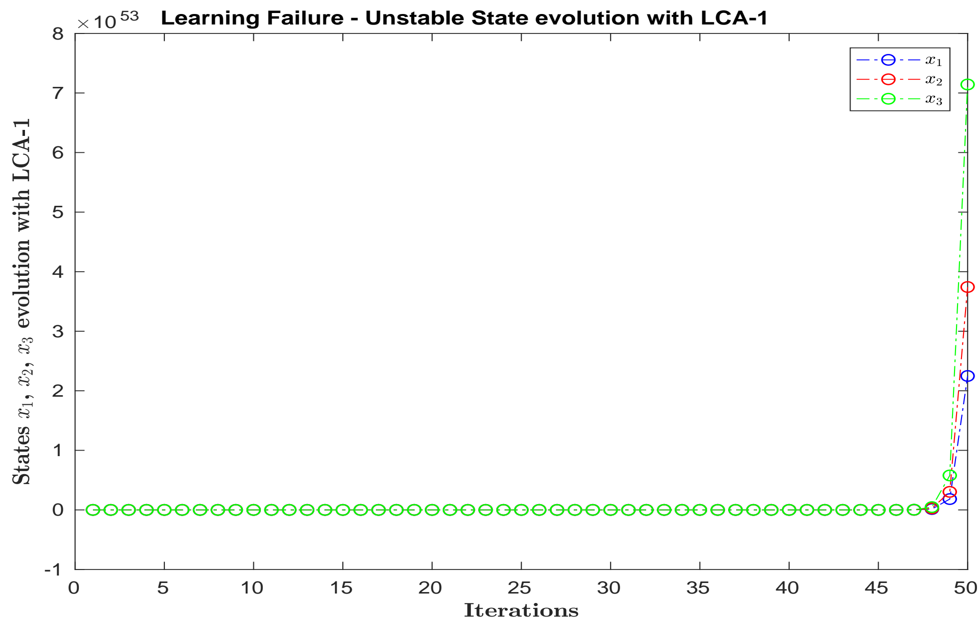



**Figure 4.41:** System-4 Learning Failure LCA-1 vs LCA-2

Chapter 5

# SUMMARY AND FUTURE WORK

## 5.1 Summary

This thesis work accomplished the following tasks:

- A new algorithm LCA-2 for Multi Agent Systems with information asymmetry is proposed and tested out in simulations and on a real system.
- The algorithm LCA-2 is found to be asymptotically stable for all systems considered.
- The new algorithm used BLAME-ALL methodology for successful learning.
- Hardware implementation of the algorithm is successfully carried out on two mobile robots. Results are encouraging. Real-time experimental data matched closely with simulated data.
- Discrete Linear Quadratic regulator output as a function of control cost weighting parameters ($\theta_i$ - diagonal elements of $R$ matrix in standard LQR cost function (2.15)) of each agent is analyzed. In all the trials carried out it is found that, with all other except one parameter constant, the corresponding agent's control output decreases monotonically with increase in parameter value $\theta_i$.
- Algorithm LCA-2 is compared with LCA-1. LCA-2 outperforms LCA-1 in the following aspects. It is to be noted that this is a exploratory comparison analysis between LCA-1 and LCA-2.

  - After the completion of learning process, when the Multi-Agent system is in Nash equilibrium, LCA-2 is found to react better to disturbances compared to LCA-1
  - Multi-Agent systems with state cost weighting matrix (P/Q) elements close to zero are found to exhibit instability with LCA-1. The same systems are stabilized with LCA-2. Thus the choice of state cost weighting matrix (P/Q) elements is limited with LCA-1 compared to LCA-2.
  - Under learning failure it is seen that a Multi Agent system running LCA-2 can still be asymptotically stable. However, with learning failure a Multi Agent system running LCA-1 is always unstable.
  - It can be concluded that both algorithms LCA-1 and LCA-2 have their own advantages and disadvantages over the other and are suited for different applications.

- Load transport using two mobile robots is successfully demonstrated albeit indirectly, when it is found that the lateral difference between robots running LCA-2 don't change as much. Load transport is demonstrated using LCA-1 as well.

## 5.2 Future work

The following mathematical proofs are required to be able to confirm the usability of LCA-2

- Monotonicity of $u_i$ with $\theta_i$ variation needs to be proved.
- A hard guarantee on the learning algorithm needs to be obtained, in the form of number of time steps needed to get learning error down to a threshold.

- The stability of the MAS running LCA-2 needs to be proved.
- The algorithm can be extended to trajectory tracking by changing the cost functions used at each agent.
- Algorithm can be extended to work for Human-Robot interaction where one robot operated manually and the other autonomously.
- Several assumptions are made in algorithms LCA-1 and LCA-2, the most general case of these algorithms would be when agents have no clue of what other other agent cost functions are. Work can be carried out in this direction.

# REFERENCES


Arnold, W. F. and A. J. Laub, “Generalized eigenproblem algorithms and software for algebraic riccati equations”, Proceedings of the IEEE **72**, 12, 1746–1754 (1984).

Caracciolo, L., A. De Luca and S. Iannitti, “Trajectory tracking control of a four-wheel differentially driven mobile robot”, in “ICRA”, vol. 4, pp. 2632–2638 (1999).

Fax, J. A. and R. M. Murray, “Information flow and cooperative control of vehicle formations”, IEEE transactions on automatic control **49**, 9, 1465–1476 (2004).

Kučera, V., “The discrete riccati equation of optimal control”, Kybernetika **8**, 5, 430–447 (1972).

Landau, I., R. Lozano, M. M’Saad and A. Karimi, “Adaptive control vol. 51”, (1998).

Laub, A., “A schur method for solving algebraic riccati equations”, IEEE Transactions on automatic control **24**, 6, 913–921 (1979).

Lewis, F. L., D. Vrabie and V. L. Syrmos, *Optimal control* (John Wiley & Sons, 2012).

Littman, M. L., “Markov games as a framework for multi-agent reinforcement learning”, in “Machine Learning Proceedings 1994”, pp. 157–163 (Elsevier, 1994).

Liu, C., W. Zhang and M. Tomizuka, “Who to blame? learning and control strategies with information asymmetry”, in “American Control Conference (ACC), 2016”, pp. 4859–4864 (IEEE, 2016).

Marden, J. R. and J. S. Shamma, “Game theory and control”, Annual Review of Control, Robotics, and Autonomous Systems , 0 (2018).

Mattingley, J., Y. Wang and S. Boyd, “Receding horizon control”, IEEE Control Systems **31**, 3, 52–65 (2011).

Mesbahi, M. and M. Egerstedt, *Graph theoretic methods in multiagent networks*, vol. 33 (Princeton University Press, 2010).

Ren, W. and R. W. Beard, *Distributed consensus in multi-vehicle cooperative control* (Springer, 2008).

Ren, W., R. W. Beard and E. M. Atkins, “Information consensus in multivehicle cooperative control”, IEEE Control Systems **27**, 2, 71–82 (2007).

Semsar-Kazerooni, E. and K. Khorasani, “Multi-agent team cooperation: A game theory approach”, Automatica **45**, 10, 2205–2213 (2009).

Slotine, J.-J. E., W. Li *et al.*, *Applied nonlinear control*, vol. 199 (Prentice hall Englewood Cliffs, NJ, 1991).

Wan, Y.-H., "The hyperbolic map and applications to the linear quadratic regulator (bj daiuto, tt hartley, and sp chicatelli)", SIAM Review **33**, 2, 338 (1991).

Wang, H., J. Zhang, J. Yi, D. Song, S. Jayasuriya and J. Liu, "Modeling and motion stability analysis of skid-steered mobile robots", in "Robotics and Automation, 2009. ICRA'09. IEEE International Conference on", pp. 4112–4117 (IEEE, 2009).

Yu, C.-H. and R. Nagpal, "A self-adaptive framework for modular robots in a dynamic environment: theory and applications", The International Journal of Robotics Research **30**, 8, 1015–1036 (2011).

# APPENDIX A

# PYTHON PROGRAMS USED FOR EXPERIMENTAL VERIFICATION

A.1 Python code used for running Learning and Control Algorithm 2:

```
#!/usr/bin/env python2.7
from __future__ import division
from control.matlab import dare
from numpy import dot, sum, tile, linalg
from numpy.linalg import inv
from numpy import linalg as LA
import numpy as np
from time import gmtime, strftime
from scipy.linalg import block_diag
import time, sys, signal, atexit
import math
import pdb
import scipy.io as sio
from numpy.linalg import multi_dot
import zmq
import rospy
import os
from vicon_bridge.msg import Markers
from geometry_msgs.msg import Point


moment=strftime("%Y-%b-%d__%H_%M_%S",time.localtime())
# to write states to a matlab file:
matfile1 = 'states'+moment+'.mat'

# to write learnt R as it evolves to a matrix:
matfile2 = 'LearntR'+moment+'.mat'

# to write control inputs actually applied to the robots:
matfile3 = 'AppliedControl'+moment+'.mat'

# to write outputs to a matlab file:
matfile4 = 'Outputs'+moment+'.mat'

messageProcessingIndex = 0
trueIndex = 0
port = "5556"
context = zmq.Context()
socket = context.socket(zmq.PUB)

#needs to be the same as current IP
socket.bind("tcp://192.168.1.4:%s" % port)

A = np.array([[0.8597,0.0],[0.0,0.8597]])
B = np.array([[0.0116, 0.0],[0.0, 0.0116]])
```

```
C = np.array([[0.1001,  0.1001],[ -0.2003, 0.2003]])
D = np.array([[0.0,0.0],[0.0,0.0]])
Q = 100*dot(C.T,C)

Cinv = inv(C) # will be used in extracting X from Y, the observations


# variable to store learnt R matrices:
Learnt_Rstorage = [np.array([[1.0,1.0]])]
X = np.array([[0.0],[0.0]])
#Y = dot(C,X)
u1_observed = 0
u2_observed = 0

X_iter = [] # to store states iteratively as a list
Y_iter = [] # to store states iteratively as a list
U_applied = [] # to store the sequence of U's applied to the system as a list



traMat = np.eye(2)

# to count the times when occlusion happened because of MOCAP system
Occlusioncounter = 0


def findTransfromationMatrix(robot1MocapCoordinates, robot2MocapCoordinates):
# multiply all incoming position coordinates with the transformaton matrix
    global traMat
    global messageProcessingIndex

    r1M = robot1MocapCoordinates # assume r1M to be a 2x1 vector
    r2M = robot2MocapCoordinates

    try:

        tempMat = inv(np.array([[r1M[0,0], r1M[1,0]],
                                [r2M[0,0], r2M[1,0]]]))
        dInt = LA.norm(r1M - r2M) # distance between two points
        ab1 = dot(tempMat, np.array([[-60],[-60]]))
        ab2 = dot(tempMat, np.array([[dInt/2],[-dInt/2]]))
        transfromationMatrix = np.array([[ab1[0,0], ab1[1,0]],
                                [ab2[0,0], ab2[1,0]]])

    except IndexError as ie:
        print ('transformation matrix cant be found since \
            input data is empty')
```

```
        messageProcessingIndex = 0
        raise ie

    print('transfromationMatrix:')
    print(transfromationMatrix)
    return transfromationMatrix

def findTransfromationMatrixdiff(robot1MocapCoordinates\
    , robot2MocapCoordinates):
# multiply all incoming position coordinates with the transformaton matrix
    global traMat
    global messageProcessingIndex

    r1M = robot1MocapCoordinates # assume r1M to be a 2x1 vector
    r2M = robot2MocapCoordinates

    try:

        tempMat = inv(np.array([[r1M[0,0], r1M[1,0]],
                               [r2M[0,0], r2M[1,0]]]))
#         dInt = LA.norm(r1M - r2M) # distance between two points
        dInt = abs(r1M[0,0] - r2M[0,0]) # distance between two points
        ab1 = dot(tempMat, np.array([[-65],[-55]]))
        ab2 = dot(tempMat, np.array([[dInt/2],[-dInt/2]]))
        transfromationMatrix = np.array([[ab1[0,0], ab1[1,0]],
                               [ab2[0,0], ab2[1,0]]])

    except IndexError as ie:
        print ('transformation matrix cant be found since input \
            data is empty')
        messageProcessingIndex = 0
        raise ie

    print('transfromationMatrix:')
    print(transfromationMatrix)
    return transfromationMatrix

def agentlearning(A,B,Q,R_est,u_obs,X,theta_max,theta_min,ForAgent):

    # for now considering only one input at each agent
    # this function provides an update for learnt theta's for each agent
    # this is the crux of learning algorithm
    # ForAgent - for whom learning is taking place
    # estimation for u1 is done at agent 1 as well,
    # it doesn't matter where the
    # learning is being carried out.
    # running the learning loop 10 times per one iteration of control loop
```

```
# u_obs - observed value of (ForAgent) control by (atAgent).
# Observation will be same everywhere

# It starts from 0.
# (N_iter+1) times learning loop runs per a run of control loop
N_iter = 10

# Est_R is the R matrix used for estimation in the function:
Est_R = np.copy(R_est, order='k')
[Sinf_thetaEst,L_thetaEst,G_thetaEst] = dare(A,B,Q,Est_R,S=None, E=None)
U_est = - dot(G_thetaEst , X)
u_est = U_est[ForAgent,0]

if (abs(u_est - u_obs) < 1e-5): # set precision for estimation
    return Est_R


Est_R_MAX = np.copy(R_est, order='k')
# agent indexing should start from 0
Est_R_MAX[ForAgent,ForAgent] = theta_max
[Sinf_thetaMax,L_thetaMax,G_thetaMax] = dare(A,B,Q,Est_R_MAX\
    ,S=None, E=None)

Est_R_MIN = np.copy(R_est, order='k')
Est_R_MIN[ForAgent,ForAgent] = theta_min
[Sinf_thetaMin,L_thetaMin,G_thetaMin] = dare(A,B,Q,Est_R_MIN\
    ,S=None, E=None)

# finding exteremes and intial slope



U_max = - dot(G_thetaMax , X)
u_max = U_max[ForAgent,0]
U_min = - dot(G_thetaMin , X)
u_min = U_min[ForAgent,0]

# multiplication factor, takes values 1 or -1 dep on some rules
mulfac = 1
theta_est = R_est[ForAgent,ForAgent]



slope = (u_max - u_min)/(theta_max - theta_min)
endpoint1 = np.array([[theta_est],[u_est]])

if (slope > 0) and (u_obs < u_est):
```

```
            mulfac = -1

        if (slope < 0) and (u_obs > u_est):
            mulfac = -1

        if (mulfac == 1):
            endpoint2 = np.array([[theta_max],[u_max]])

        if (mulfac == -1):
            endpoint2 = np.array([[theta_min],[u_min]])

        for i in range(0, N_iter):
            Invslope =  (endpoint2[0,0] - endpoint1[0,0])\
            /(endpoint2[1,0] - endpoint1[1,0])
            theta_est_new = endpoint1[0,0] +  Invslope\
             * (u_obs - endpoint1[1,0])
            Est_R[ForAgent,ForAgent] = theta_est_new
            [Sinf_thetaEst,L_thetaEst,G_thetaEst] = \
            dare(A,B,Q,Est_R,S=None, E=None)
            U_est_new = - dot(G_thetaEst , X)
            u_est_new = U_est_new[ForAgent,0]

            if (((u_est_new) - (u_obs)) \
                * ((u_obs) - (endpoint2[1,0]))) > 0 :
                endpoint1 = np.array([[theta_est_new],[u_est_new]])
            else:
                endpoint2 = np.array([[theta_est_new], [u_est_new]])

            theta_est = theta_est_new   # not being used anywhere

        return Est_R


    def callback(Data):
        global messageProcessingIndex
        global Occlusioncounter
        global X_iter
        global Y_iter
        global Learnt_Rstorage
        global U_applied
        global traMat
        global matfile1
        global matfile2
        global matfile3
        global X
        global Y
        global u1_observed
```

```
global u2_observed
global trueIndex
occlusionflag = 0

theta_max = 100
theta_min = 0.001;
theta1 = 0.6
theta2 = 0.3
R1_control = block_diag(theta1, 1) # for the control matrix at agent 1
R2_control = block_diag(1, theta2) # for the control matrix at agent 2

#todo ensure that marker 0 is always one robot and marker 1 is the other one

try:
    robot1Pose = np.array([[Data.markers[0].translation.x]\
        ,[Data.markers[0].translation.y]])
    robot2Pose = np.array([[Data.markers[1].translation.x]\
        ,[Data.markers[1].translation.y]])



    robot1Pose = robot1Pose*0.001 #to convert mocap coordinates to meters
    robot2Pose = robot2Pose*0.001
    messageProcessingIndex = messageProcessingIndex + 1



except IndexError:
    print('Occlusion happened. Ensure markers are in \
        the  field of view of VICON')
    occlusionflag = 1
    Occlusioncounter = Occlusioncounter + 1



try:
    if messageProcessingIndex == 1:
        # find the transformation matrix
        traMat = findTransfromationMatrix(robot1Pose, robot2Pose)

    if messageProcessingIndex > 1:  # perform learning as soon as you observe
        R_hat = block_diag(Learnt_Rstorage[(trueIndex -1)][0,0]\
            ,Learnt_Rstorage[(trueIndex -1)][0,1])
        # Theta 1 learning, agents start at 0. So agent 0
        Rhat_theta1_update = agentlearning(A,B,Q,R_hat,u1_observed,X\
            ,theta_max,theta_min, 0)
        # Theta 2 learning, agents start at 0. So agent 1
```

```
        Rhat_theta2_update = agentlearning(A,B,Q,R_hat,u2_observed,X\
            ,theta_max,theta_min, 1)
        Learnt_Rstorage.append(np.array([[Rhat_theta1_update[0,0]\
            ,Rhat_theta2_update[1,1]]]))
        #X_iter = np.concatenate((X_iter, X))
        X_iter.append(X.T)
        #Y_iter = np.concatenate((Y_iter, Y))
        Y_iter.append(Y.T)
        # form correct control matrices
        R1_control[1,1] = Rhat_theta2_update[1,1]
        R2_control[0,0] = Rhat_theta1_update[0,0]

    if occlusionflag == 1 and messageProcessingIndex > 1:
        # using predicted value for the state
        X = dot(A,X) + dot(B,np.array([[u1_observed],[u2_observed]]))
        Y = dot(C,X)

    if occlusionflag == 0:
        # robot one pose in Experiment coordinate system:
        ro1ExpPos = dot(traMat, robot1Pose)
        # robot two pose in Experiment coordinate system
        ro2ExpPos = dot(traMat, robot2Pose)
        Y = np.array([[0.5*(ro1ExpPos[0,0] + ro2ExpPos[0,0])],
                      [(ro2ExpPos[0,0] - ro1ExpPos[0,0])]])
        X = dot(Cinv, Y)


    if messageProcessingIndex > 0:
        [Sinf_ag1,L_ag1,G_ag1] = dare(A,B,Q,R1_control,S=None, E=None)
        [Sinf_ag2,L_ag2,G_ag2] = dare(A,B,Q,R2_control,S=None, E=None)
        U1_observed = - dot(G_ag1 , X) # 2 x 1 vector
        U2_observed = - dot(G_ag2 ,X) # 2 x 1 vector
        u1_observed = U1_observed[0,0] # control to be applied at agent 1
        u2_observed = U2_observed[1,0] # control to be applied at agent 2
        # sending control to robots:
        socket.send_string("%4.3f %4.3f" % (u1_observed, u2_observed))
        U_applied.append(np.array([[u1_observed, u2_observed]]))
        trueIndex = trueIndex + 1

    if trueIndex == 400:
        # sending control to robots:
        socket.send_string("%4.3f %4.3f" % (-30, 0))


except KeyboardInterrupt:
    print ('Interrupted due to abnormal robot behaviour or \
        intentionally')
```

```
        sio.savemat(matfile1, mdict={'states' : X_iter}, oned_as = 'row')
        sio.savemat(matfile4, mdict={'Outputs' : Y_iter}, oned_as = 'row')
        sio.savemat(matfile2, mdict={'LearntR' : Learnt_Rstorage}\
            , oned_as = 'row')
        sio.savemat(matfile3, mdict={'AppliedControl' : U_applied}\
            , oned_as = 'row')
        for dummy in range(50):
            socket.send_string("%4.3f %4.3f " % (0, 0))
            socket.send_string("0.0 0.0")
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)


def main():

    # to start camera and motion capture system for recording the trial
    time.sleep(90)

    try:
        rospy.init_node('subscriber', anonymous =True)
        sub = rospy.Subscriber('/vicon/markers',Markers\
            ,callback,queue_size=1)
        rospy.spin()
        print ('Interrupted due to abnormal robot behaviour')
        for dummy in range(50):
            socket.send_string("%4.3f %4.3f " % (0.0, 0.0))
            socket.send_string("0.0 0.0")

        print ('Occlusioncounter:')
        print (Occlusioncounter)
        sio.savemat(matfile1, mdict={'states' : X_iter}\
            , oned_as = 'row')
        sio.savemat(matfile2, mdict={'LearntR' : Learnt_Rstorage}\
            , oned_as = 'row')
        sio.savemat(matfile3, mdict={'AppliedControl' : U_applied}\
            , oned_as = 'row')
        sio.savemat(matfile4, mdict={'Outputs' : Y_iter}\
            , oned_as = 'row')


        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

```
    except rospy.ROSInterruptException:
        print('scrwed')
        pass

if __name__ == '__main__':
    main()
```

A.2 Python code used for running Learning and Control Algorithm 1:

```
#!/usr/bin/env python2.7
from __future__ import division
from numpy import dot, sum, tile, linalg
from numpy.linalg import inv
from numpy import linalg as LA
import numpy as np
from time import gmtime, strftime
from scipy.linalg import block_diag
import time, sys, signal, atexit
import math
import pdb
import scipy.io
from numpy.linalg import multi_dot
import zmq
import sys
import time
import rospy
import os
from vicon_bridge.msg import Markers
from geometry_msgs.msg import Point

moment=strftime("%Y-%b-%d__%H_%M_%S",time.localtime())
matfile1 = 'states'+moment+'.mat' # to write states to a matlab file
matfile2 = 'LearntR'+moment+'.mat'  # to write learnt R as
                                    #it evolves to a matrix
matfile3 = 'AppliedControl'+moment+'.mat' # to write control inputs actually
                                          # applied to the robots
matfile4 = 'Outputs'+moment+'.mat' # to write outputs to a matlab file



messageProcessingIndex = 0
port = "5556"
context = zmq.Context()
socket = context.socket(zmq.PUB)
socket.bind("tcp://192.168.1.4:%s" % port)
A = np.array([[0.8597,0.0],[0.0,0.8597]])
B = np.array([[0.0116, 0.0],[0.0, 0.0116]])
B1 = np.array([[0.0116],[0.0]])
```

```
B2 = np.array([[0.0],[0.0116]])

C = np.array([[0.1001,  0.1001],[ -0.2003, 0.2003]])
D = np.array([[0.0,0.0],[0.0,0.0]])
P = 300*dot(C.T,C)
Cinv = inv(C) # will be used in extracting X from Y, the observations
T1 = np.array([[1.0,0.0]])
T2 =  np.array([[0.0,1.0]])
X_iter = [] # to store states iteratively
U_iter = [] # to store control input, in this case velocity
U_hat_iter = [] # to store virtual nash equilibrium values
u1_hat_iter = [] # to store virtual nash equilibrium values
u2_hat_iter = [] # to store virtual nash equilibrium values
# Parameters
theta1 = .2
theta2 = .4

theta=np.array([[theta1],[theta2]])
theta1_hat = 1
theta2_hat = 1

R = block_diag(theta1 * np.eye(1) , theta2 * np.eye(1))
R_hat = block_diag(theta1_hat * np.eye(1) , theta2_hat * np.eye(1))


G1 = dot(B.T, P)
G2 = dot(G1, B)
G3 = dot(B1.T, P)
G4 = dot(G3, B1)
G5 = dot(B2.T, P)
G6 = dot(G5, B2)
G7 = B - dot(B1, T1)
G8 = B - dot(B2, T2)
#traMat = np.eye(1)
Occlusioncounter = 0




def findTransfromationMatrix(robot1MocapCoordinates\
    ,robot2MocapCoordinates):
# multiply all incoming position coordinates with the transformaton matrix
    global traMat
    global messageProcessingIndex

    r1M = robot1MocapCoordinates # assume r1M to be a 2x1 vector
    r2M = robot2MocapCoordinates
```

```
    try:

        tempMat = inv(np.array([[r1M[0,0], r1M[1,0]],
                              [r2M[0,0], r2M[1,0]]]))
        dInt = LA.norm(r1M - r2M)
        ab1 = dot(tempMat, np.array([[-60],[-60]]))
        ab2 = dot(tempMat, np.array([[dInt/2],[-dInt/2]]))
        transfromationMatrix = np.array([[ab1[0,0], ab1[1,0]],
                              [ab2[0,0], ab2[1,0]]])

    except IndexError as ie:
        print ('transformation matrix cant be found since input data\
            is empty')
        messageProcessingIndex = 0
        raise ie

    print('transfromationMatrix:')
    print(transfromationMatrix)
    return transfromationMatrix

def findTransfromationMatrixdiff(robot1MocapCoordinates\
    , robot2MocapCoordinates):
    # multiply all incoming position coordinates with the transformaton matrix
    global traMat
    global messageProcessingIndex

    r1M = robot1MocapCoordinates # assume r1M to be a 2x1 vector
    r2M = robot2MocapCoordinates

    try:

        tempMat = inv(np.array([[r1M[0,0], r1M[1,0]],
                              [r2M[0,0], r2M[1,0]]]))
#         dInt = LA.norm(r1M - r2M) # distance between two points
        dInt = abs(r1M[0,0] - r2M[0,0]) # distance between two points
        ab1 = dot(tempMat, np.array([[-65],[-55]]))
        ab2 = dot(tempMat, np.array([[dInt/2],[-dInt/2]]))
        transfromationMatrix = np.array([[ab1[0,0], ab1[1,0]],
                              [ab2[0,0], ab2[1,0]]])

    except IndexError as ie:
        print ('transformation matrix cant be found since\
         input data is empty')
        messageProcessingIndex = 0
        raise ie
```

```
    print('transfromationMatrix:')
    print(transfromationMatrix)
    return transfromationMatrix


def callback(Data):
    global messageProcessingIndex
    global Occlusioncounter
    global X_iter
    global U_iter
    global U_hat_iter
    global R_hat
    global theta1_hat
    global theta2_hat
    global traMat
    global u1_hat_iter
    global u2_hat_iter
    global matfile1
    global matfile2
    global matfile3
    global matfile4

    print('theta1_hat:')
    print(theta1_hat)
    print('theta2_hat:')
    print(theta2_hat)

    try:
        robot1Pose = np.array([[Data.markers[0].translation.x]\
            ,[Data.markers[0].translation.y]])
        robot2Pose = np.array([[Data.markers[1].translation.x]\
            ,[Data.markers[1].translation.y]])

        robot1Pose = robot1Pose*0.001 #to convert mocap coordinates to meters
        robot2Pose = robot2Pose*0.001 #to convert mocap coordinates to meters
        messageProcessingIndex = messageProcessingIndex + 1
        if messageProcessingIndex == 1:
            # find the transformation matrix
#             traMat = findTransfromationMatrix(robot1Pose, robot2Pose)
            traMat = findTransfromationMatrixdiff(robot1Pose, robot2Pose)
#             findTransfromationMatrixdiff
#Karthik
#         if messageProcessingIndex == 30:
#             raise KeyboardInterrupt

#         if messageProcessingIndex == 15:
#             raise IndexError
```

```
    # robot one pose in Experiment coordinate system:
    ro1ExpPos = dot(traMat, robot1Pose)

    # robot one pose in Experiment coordinate system:
    ro2ExpPos = dot(traMat, robot2Pose)

    Y = np.array([[0.5*(ro1ExpPos[0,0] + ro2ExpPos[0,0])],
                  [(ro2ExpPos[0,0] - ro1ExpPos[0,0])]])
    X = dot(Cinv, Y)

    X_iter.append(X.T)

    f_X = dot(A,X)   # f(X(k)) defined in the paper

    #  virtual nash equilibrium calculation
    U_hat = -1 * multi_dot([inv(R_hat + G2) , G1, f_X])
    U_hat_iter.append(U_hat.T)
    local1 = (f_X + dot(G7,U_hat))
    local2 = (f_X + dot(G8,U_hat))

    # calculating corresponding control inputs for agent1 & 2
    u1 = -1 * multi_dot([inv((theta1 * np.eye(1)) + G4), G3,local1])
    u1_hat_iter.append(u1)


    u2 = -1 * multi_dot([inv((theta2 * np.eye(1)) + G6), G5,local2])
#    print('u2:')
#    print(u2)
    u2_hat_iter.append(u2)


    socket.send_string("%4.3f %4.3f" % (u1[0,0], u2[0,0]))


    e1  = (-1 * multi_dot([u1.T, G3, (local1 + dot(B1,u1))]))\
     - theta1_hat * dot(u1.T, u1)
    theta1_hat = theta1_hat + e1 * pow(dot(u1.T, u1), -1)

    # update of paramter estimates in robot 1
    e2 = (-1 * multi_dot([u2.T, G5, (local1 + dot(B2,u2))]))\
     - theta2_hat * dot(u2.T, u2)
    theta2_hat = theta2_hat + e2 * pow(dot(u2.T, u2), -1)

    # updated parameter block diagonal matrix
    R_hat = block_diag(theta1_hat * np.eye(1) , theta2_hat\
     * np.eye(1))
```

```
    except IndexError:
        print('Occlusion happened')
        Occlusioncounter = Occlusioncounter + 1
    # below piece of code can be removed. It doesn't serve any purpose

    except KeyboardInterrupt:
        print ('Interrupted due to abnormal robot behaviour')
        scipy.io.savemat(matfile1, mdict={'states' : X_iter}\
            , oned_as = 'row')
        scipy.io.savemat(matfile2, mdict={'u1hat' : u1_hat_iter}\
            , oned_as = 'row')
        scipy.io.savemat(matfile4, mdict={'u2hat' : u2_hat_iter}\
            , oned_as = 'row')
        scipy.io.savemat(matfile3, mdict={'UhatIter' : U_hat_iter}\
            , oned_as = 'row')
        for dummy in range(50):
            socket.send_string("%4.3f %4.3f" % (0.0, 0.0))
            socket.send_string("0.0 0.0")
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)


#     X_iter.append(X.T)



def main():

    time.sleep(90)

    try:
        rospy.init_node('subscriber', anonymous =True)
        sub = rospy.Subscriber('/vicon/markers',Markers,callback\
            ,queue_size=1)
        rospy.spin()
        print ('Interrupted due to abnormal robot behaviour')
        for dummy in range(50):
            socket.send_string("%4.3f %4.3f" % (0, 0))
            socket.send_string("0.0 0.0")
        scipy.io.savemat(matfile1, mdict={'states' : X_iter}, \
            oned_as = 'row')
        scipy.io.savemat(matfile2, mdict={'u1hat' : u1_hat_iter}\
            , oned_as = 'row')
        scipy.io.savemat(matfile4, mdict={'u2hat' : u2_hat_iter}\
```

```
            , oned_as = 'row')
        scipy.io.savemat(matfile3, mdict={'UhatIter' : U_hat_iter}\
            , oned_as = 'row')
        print ('Occlusioncounter:')
        print (Occlusioncounter)

        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)

    except rospy.ROSInterruptException:
        print('scrwed')
        pass

if __name__ == '__main__':
    main()
```