SAFE POLICY OPTIMIZATION FOR MULTI-AGENT CONSTRAINED MARKOV DECISION PROCESSES

Thesis submitted by

Sparsh Chaudhri
2019MT10765

under the guidance of

Prof. Kamana Porwal

in partial fulfilment of the requirements
for the award of the degree of

Bachelor of Technology

Department Of Mathematics
INDIAN INSTITUTE OF TECHNOLOGY DELHI
May 2023

THESIS CERTIFICATE

This is to certify that the thesis titled **SAFE POLICY OPTIMIZATION FOR MULTI-AGENT CONSTRAINED MARKOV DECISION PROCESSES**, submitted by **Sparsh Chaudhri (2019MT10765)**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Kamana Porwal
Professor
Dept. of Mathematics
IIT-Delhi

Place: New Delhi

Date:

ACKNOWLEDGEMENTS

I wish to thank Prof. Kamana Porwal for giving me the opportunity to work on the intersection of Numerical Analysis, Optimization and Stochastic Processes. Her guidance has been invaluable for my journey.

ABSTRACT

KEYWORDS: Constrained Markov Process; Lagrangian Policy; Policy Convergence, Multi-Agent, Competitive Games

Safe learning is an emerging field where constraints are not to be violated during the training phase of a policy. Such processes can be model Constrained Markov Decision Processes. Lagrangian-based policy correctors have been proposed to augment actions in single agent environments to avoid constraint violation. In this thesis, we extend this to a two-agent environment playing a competitive game against each other. The behaviour of two safety models are analyzed. It is found that a safety model agnostic of the rival agent leads to high constraint violation, an oscillatory behavior and dual convergence of agents. More interestingly, it is found that a safety model semi-agnostic of the rival agent is able to absolutely dominate the learning environment and exhausts the other agent of any learning opportunities. The behaviour is non-oscillatory and the better agent is able exploit the rivals constraint model and non-convergent policy.

Contents

List of Figures

ABBREVIATIONS

RL	Reinforcement Learning
MDP	Markov Decision Process
CMDP	Constrained Markov Decision Process
DDPG	Deep Deterministic Policy Gradient
RASM	Rival Agnostic Safety Model
RGSM	Rival Gnostic Safety Model
RSGSM	Rival Semi-Gnostic Safety Model

Chapter 1

INTRODUCTION

Learning policies under constraints is an emerging problem which has a vast number of applications, especially when learning in real world instead of simulators. An example of this is learning policies to implement a battery management system, in which the safety limits are not to be exceeded. Another example is learning of motion where the angle of the joints should not exceed their limits, as it will cause irreparable damage. In such scenarios, while the final policy is expected to be constrained, we need to learn the policies under constraints as well else the agent or the environment might be damaged. Thus, the agent needs to interact with the environment through safe policies only. Such problems are formulated as Constrained Markov Decision Processes (CMDPs). Research has been done to develop closed form solutions using the Lagrangian and Lyapunov constraints which project the actions of a policy into a "safe" action space. In this study, these approaches will be extended to a multi-agent environment where along with environmental constraints, constraints exist between the states of agents as well. Our study aims to conclude with an analysis of stability, convergence and scalabilty of the extended methods.

Reinforcement Learning (RL) is a type of machine learning that involves an agent that learns by interacting with an environment to maximize a reward signal. The process can be mathematically modeled as a Markov Decision Process (MDP). A MDP is a tuple (S, A, P, R, γ), where S is a state space, A is an action space, $P : S * A * S \to [0, 1]$ is a transition kernel, $R : S * A \to \mathbb{R}$ is a reward function, and $\gamma \in (0, 1)$ is a discount factor. The goal of the agent is to find a policy $\mu : S \to A$ that maps states to actions and maximizes the expected sum of rewards over time, given by the objective function:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...)|_{s_t,a_t} \tag{1.1}$$

where $\mathbb{E}$ is the expectation operator. Let $[K]$ denote the set $1, ..., K$ and $[x]^+$ denote the operation $max\{x, 0\}$. A CMDP is a tuple (S, A, P, R, γ, C), where additionally over a MDP, $C = \{c_i : S * A \to R | i \in [K]\}$ is a set of immediate-constraint functions. Based on that, we also define a set of safety signals $\bar{C} = \{\bar{c_i} : S \to R | i \in [K]\}$. These are per-state observations of the immediate-constraint values. In the type of systems tackled in this work, P is deterministic and determines f s.t. $s' = f(s, a)$. Thus, we have $\bar{c_i}(s') \triangleq c_i(s, a)$.

Chapter 2

LITERATURE REVIEW

A common approach to solve CMDPs is to use the Lagrangian method that augments the original objective function with a penalty on constraint violation and computes the saddle-point of the constrained policy optimization via primal-dual methods [CGJP17]. Although safety is ensured when the policy converges asymptotically, a major drawback of this approach is that it makes no guarantee with regards to the safety of the policies generated during training. Thus focus of this work is on control problems with continuous state and action spaces. In this context, the comparison is limited to the literature on safe reinforcement learning (RL) that aims to ensure safety during the learning process as well.

For instance, [AHTA17] proposed a modified trust-region policy gradient algorithm that projected the policy to a safe feasibility set in each iteration to maintain safety during policy optimization. However, this algorithm is not suitable for our use cases where safety must be ensured for all visited states. Similarly, [BTSK17] identified control-theoretic conditions for safe operation in a discretized deterministic control framework, but it required knowledge about the specific system and may not be applicable to systems that are not Lipschitz continuous. [PDMT18] utilized an in-graph QP solver to ensure safety on a state-wise basis, but it is computationally expensive and requires expert knowledge to design physical constraints.

[DDV$^+$18] proposed a state-based action correction mechanism using a safety layer, which accomplishes the goal of zero constraint-violations in tasks where the agent is constrained to a confined region using Lagrange multipliers. [CNF$^+$19] proposed the θ-projection that combines policy gradient with a projection of the policy parameters onto the set of feasible solutions induced by the Lyapunov constraints.

The generalization of MDP to the multi-agent case is the stochastic game. It is a tuple $(S, A_1, ..., A_n, P, R_1, ..., R_n, \gamma)$, where S is a state space, A_i, $i = 1...n$ is the set of action spaces of agents, yielding a joint action space $\mathbf{A} = A_1 * ... * A_n$, $P : S * \mathbf{A} * S \to [0, 1]$ is a transition kernel, $R_i : S * A_i \to \mathbb{R}$, $i = 1..n$ is the set of reward functions of agents and $\gamma \in (0, 1)$ is a discount factor.

[BBDS10] document several challenges that MARL faces, such as exponential growth of joint action space in number of agents, goal formulation, nonstationarity and instability due to exploration of one agent into another agent's behaviour. Thus it is worthwhile to extend how safety-constraints behave in different settings with multiple agents, particularly whether the safety constraint is exploitable by other agents.

2.1 Safe Exploration in Continuous Action Space

The problem of safe exploration in context of policy optimization is expressed as:

$$\max_{\theta} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, \mu_\theta(s_t, x_T))] \tag{2.1}$$

such that

$$\bar{c_i}(s_t) \leq C_i \forall i \in [K] \tag{2.2}$$

where μ_θ is a parameterized policy and x_T is the goal.

2.1.1 Linear Safety-Signal Model

Without prior knowledge on its environment, an RL agent initialized with a random policy cannot ensure per-state constraint satisfaction during the initial training stages. Thus some basic form of prior knowledge needs to be incorporated, based on single-step dynamics. To simplify learning of the immediate-constraint functions, Dalal linearizes the safety signal as:

$$\bar{c_i}(s') \triangleq c_i(s, a) \approx \bar{c_i}(s) + g(s; w_i)^T a \tag{2.3}$$

where w_i are weights of a neural network g that takes s as input and outputs a vector of the same dimension as a. Given a policy-oblivious set of tuples $D = \{(s_j, a_j, s'_j)\}$, $g(s; w_i)$ is trained by minimizing the loss:

$$L = \sum_{(s,a,s') \in D} (\bar{c_i}(s') - (\bar{c_i}(s) + g(s; w_i)^T a))^2 \tag{2.4}$$

2.1.2 Safety Layer via Analytical Optimization

On top of the policy network, Dalal composes a safety layer which solves:

$$\underset{a}{argmin} \frac{1}{2} ||a - \mu_\theta(s, x_T)||^2 \tag{2.5}$$

such that

$$c_i(s, a) \leq C_i \forall i \in [K] \tag{2.6}$$

On substituting in the linear model, we get the constraint formulation to optimize 2.5 such that

$$\bar{c_i}(s) + g(s; w_i)^T a \leq C_i \forall i \in [K] \tag{2.7}$$

As the objective is positive-definite and quadratic, and the constraints are linear, a global solution can be found to this complex problem. Dalal simplifies this further under the assumption that only a single constraint is active at a time.

Theorem 1 Assume there exists a feasible solution to 2.5 under inequality 2.7 denoted by $(a^\star, \{\lambda_i^\star\}_{i=1}^K)$, where $\{\lambda_i^\star\}$ is the optimal Lagrange multilier associated with the i-th constraint. Also, assume $|\{i|\lambda_i^\star > 0\}| \leq 1$, i.e. at most one constraint is active at a time. Then under these assumptions:

$$\{\lambda_i^\star\} = [\frac{g(s;w_i)^T\mu_\theta(s,x_T) + \bar{c_i}(s) - C_i}{g(s;w_i)^Tg(s;w_i)}]^+ \tag{2.8}$$

and

$$a^\star = \mu_\theta(s,x_T) - \lambda_{i^\star}^\star g(s;w_i), \tag{2.9}$$

where $i^\star = argmax_i\lambda_i^*$.

Proof As the objective function and constraints are convex, a sufficient condition for optimality of a feasible solution $(a^\star, \{\lambda_i^\star\}_{i=1}^K)$ is for it to satisfy the KKT conditions. The Lagrangian of the system is

$$L(a,\lambda) = \frac{1}{2}||a - \mu_\theta(s,x_T)||^2 + \sum_{i=1}^K \lambda_i(\bar{c_i}(s) + g(s;w_i)^Ta - C_i);$$

and hence the KKT conditions at $(a^\star, \{\lambda_i^\star\}_{i=1}^K)$ are

$$\nabla_a L = a^\star - \mu_\theta(s,x_T) + \sum_{i=1}^K \lambda_i^\star g(s;w_i) = 0, \tag{2.10}$$

$$\lambda_i^\star(\bar{c_i}(s) + g(s;w_i)^Ta^\star - C_i) = 0\forall i \in [K] \tag{2.11}$$

First, we consider the case where $|\{i|\lambda_i^\star > 0\}| = 1$, i.e. $\lambda_{i^\star}^\star > 0$. Then from 2.11 $\bar{c_i}(s) + g(s;w_i)^Ta^\star - C_i = 0$. Substituting 2.9 gives $\lambda_{i^\star}^\star g(s;w_{i^\star})^Tg(s;w_{i^\star}) = g(s;w_{i^\star})^T\mu_\theta(S) + \bar{c_{i^\star}}(s) - C_{i^\star}$. This gives 2.8 when $i = i^\star$.

As rest of the constraints are inactive, $\bar{c_i}(s) + g(s;w_i)^Ta^\star - C_i < 0$ and hence $\lambda_i^\star = 0$ due to $[.]^+$ operator in 2.8. Finally, the proof for $\lambda_i^\star = 0\forall i \in [K]$ follows similarly as all constraints are inactive and there is no correction.

The solution 2.9 is essentially a linear projection of the original action $\mu_\theta(s)$ to the "safe" hyperplane with slope $g(s;w_{i^\star}$ and intercept $\bar{c_{i^\star}}(s) - C_{i^\star}$

2.2 A Lyapunov-based Approach to Safe Reinforcement Learning

Lyapunov functions are a mathematical tool used in control theory to analyze the stability of dynamical systems. In reinforcement learning, Lyapunov functions have been used to improve the safety and stability of policy optimization algorithms.

A Lyapunov function is a scalar function $V(x)$ that measures the energy of the system, where x is the state of the system. The Lyapunov function is designed to be positive and continuously differentiable. The derivative of the Lyapunov function along the trajectory of the system is denoted by $\dot{V}(x,u)$, where u is the control input. The Lyapunov function satisfies the following properties:

1. $V(x) > 0, \forall x \neq 0$
2. $V(0) = 0$
3. $\dot{V}(x,u) \leq 0, \forall x, u$, except at the equilibrium point(s)

The third property ensures that the system is stable, as the Lyapunov function decreases along the trajectory of the system except at the equilibrium point(s), where $\dot{V}(x,u) = 0$.

[CNDGG18] made advances in using Lyapunov functions to satisfy constraints. They model it as a cost-minimization problem and define the MDP as a tuple $(\mathcal{X}, \mathcal{A}, \gamma, c, P, x_0)$, where $\mathcal{X}$ and $\mathcal{A}$ are the state and action spaces; $\gamma \in [0,1)$ is a discounting factor; $c(x,a) \in [0, C_{\max}]$ is the immediate cost function; $P(\cdot \mid x, a)$ is the transition probability distribution; and $x_0 \in \mathcal{X}$ is the initial state. The CMDP model extends MDP by introducing additional costs and the associated constraints, and is defined by $(\mathcal{X}, \mathcal{A}, \gamma, c, P, x_0, d, d_0)$, where the first six components are the same as in the unconstrained MDP; $d(x) \in [0, D_{\max}]$ is the (state-dependent) immediate constraint cost; and $d_0 \in \mathbb{R}_{\geq 0}$ is an upper-bound on the expected cumulative constraint cost.

At each state $x \in \mathcal{X}$, the generic Bellman operator is defined w.r.t. a policy $\pi \in \Delta$ and a cost function h as

$$T_{\pi,h}[V](x) = \sum_{a} \pi(a \mid x) \left[h(x,a) + \gamma \sum_{x' \in \mathcal{X}} P\left(x' \mid x, a\right) V\left(x'\right) \right] \tag{2.12}$$

Given a policy π, the expected cumulative cost and the expected cumulative constraint cost are defined as $\mathcal{C}_\pi\left(x_0\right) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c\left(x_t, a_t\right) \mid \pi, x_0\right]$ and $\mathcal{D}_\pi\left(x_0\right) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t d\left(x_t\right) \mid \pi, x_0\right]$. The goal in CMDPs is to solve the constrained optimization problem

$$\pi^* \in \min_{\pi \in \Delta} \left\{ \mathcal{C}_\pi\left(x_0\right) : \mathcal{D}_\pi\left(x_0\right) \leq d_0 \right\}. \tag{2.13}$$

They defines a set of Lyapunov functions w.r.t. a baseline policy π_B , initial state $x_0 \in \mathcal{X}$ and constraint threshold d_0 as

$$\mathcal{L}_{\pi_B}\left(x_0, d_0\right)=\left\{L: \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}: T_{\pi_B, d}[L](x) \leq L(x), \forall x \in \mathcal{X} ; L\left(x_0\right) \leq d_0\right\}, \tag{2.14}$$

and call the constraints in this feasibility set the Lyapunov constraints. For any arbitrary Lyapunov function $L \in \mathcal{L}_{\pi_B}\left(x_0, d_0\right)$, the set of L-induced Markov stationary policies is denoted by

$$\mathcal{F}_L(x)=\left\{\pi(\cdot \mid x) \in \Delta: T_{\pi, d}[L](x) \leq L(x)\right\} \tag{2.15}$$

While any L-induced policy is a feasible policy of Equation 2.13, in general, the set $\mathcal{F}_L(x)$ does not necessarily contain an optimal policy. To construct a Lyapunov function $L \in \mathcal{L}_{\pi_B}\left(x_0, d_0\right)$ such that

$$L(x) \geq T_{\pi^*, d}[L](x), \quad L\left(x_0\right) \leq d_0 \tag{2.16}$$

[CNDGG18] show in their work that without loss of optimality, the Lyapunov function can be expressed as

$$L_\epsilon(x):=\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t\left(d\left(x_t\right)+\epsilon\left(x_t\right)\right) \mid \pi_B, x\right], \tag{2.17}$$

where $\epsilon(x) \geq 0$ is some auxiliary constraint cost uniformly upper-bounded by

$$\epsilon^*(x):=2 D_{\max } D_{TV}\left(\pi^* \| \pi_B\right)(x) /(1-\gamma), \tag{2.18}$$

They also show that by further restricting $\widetilde{\epsilon}(x)$ to be a constant function, the maximizer is given by

$$\widetilde{\epsilon}(x)=(1-\gamma) \cdot\left(d_0-\mathcal{D}_{\pi_B}\left(x_0\right)\right), \forall x \in \mathcal{X} \tag{2.19}$$

Using the construction of the Lyapunov function $L_{\widetilde{\epsilon}}$, the Safe Policy Iteration (SPI) algorithm can be used to optimize the policy.

Algorithm 1 Safe Policy Iteration (SPI)

```
Input Initial feasible policy $\pi_0$
for $k = 0, 1, 2, \ldots$ do
    Step 0: With $\pi_b = \pi_k$, evaluate the Lyapunov function $L_{\epsilon_k}$
    Step 1: Evaluate the cost value function $V_{\pi_k}(x) = \mathcal{C}_{\pi_k}(x)$
    Then update the policy by solving the following problem: $\pi_{k+1}(\cdot \mid x) \in$
    argmin $\pi_{\pi \in \mathcal{F}_{L_{\epsilon_k}}(x)} T_{\pi,c}\left[V_{\pi_k}\right](x), \forall x \in \mathcal{X}$
end for
Return Final policy $\pi_{k^*}$
```

Chapter 3

METHODOLOGY

In this section, we first examine the environment and its corresponding task. We then present a two-agent formulation of the safety signal and its corresponding solution. It is followed by details about the policy network.

3.1 Environment and Task

The task is modelled in a custom OpenAI Gym environment developed by us which has a PyBullet physics simulation engine running in the background. While the task in this study is simple, more complex problem formulations can be addressed using the same environment. PyBullet also enables easy visualization of the training stage by directly rendering the environment as a GUI.

3.1.1 Single Agent Environment

The environment consists of a 2-D square $B_{[a,b]} = \{x | a \leq x_i \leq b, i = 1, 2, 3\}$. The agent is a cube of side 0.1 units which is allowed to be in $B_{[0,1]}$. There exists a target which spawns in $B_{[0.2,0.8]}$. The goal is to bring the agent as close as possible to a changing target location, by directly setting the velocity of the agent. A slack is introduced to limit the feasible region of the agent to $B_{[0.1,0.9]}$ so that the safety layer can correct it. Let agent position, agent velocity and agent position be x_B, v_B, x_T, each $\in \mathbb{R}^2$. The state space $S = (x_B, v_B, x_T)$ while the action space $A = (v_B)$ vector. The reward function $R = max(1 - 10 * ||x_B - x_T||_2^2, 0)$ and $\gamma = 0.99$. The constraint C is $x_B \in B_{[0.1,0.9]}$, which is the feasible square region.

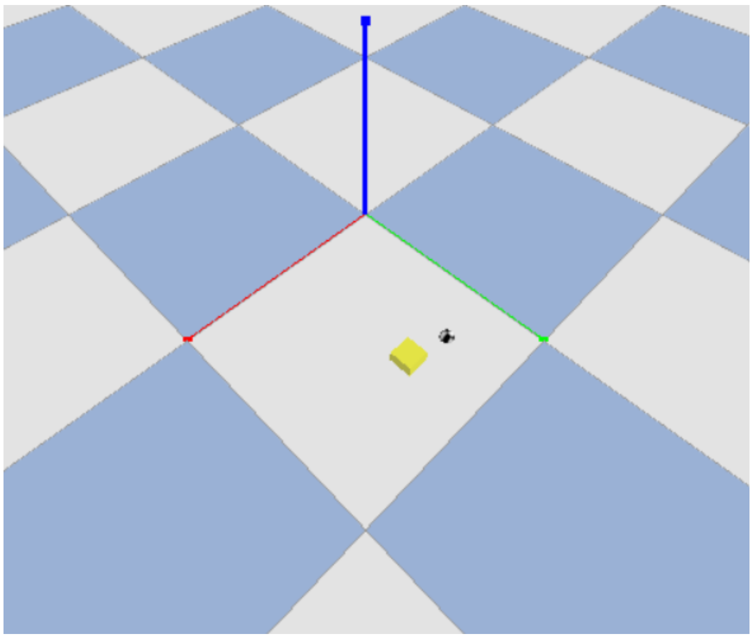

Figure 3.1: PyBullet environment for single agent (yellow cube) and target (checkered sphere).

3.1.2 Two Agent Environment

The environment consists of a 2-D square $B_{[a,b]} = \{x | a \leq x_i \leq b, i = 1, 2, 3\}$. There are now two agents which are each allowed to be in $B_{[0,1]}$, each is a cube of side 0.1 units. There exists a target which spawns in $B_{[0.2,0.8]}$. The goal for each agent is to reach the target as quickly as possible by directly setting its velocity. However, as there are two agents and a single target, only one of them can succeed for each run of the task. A slack is introduced to limit the feasible region of the agent to $B_{[0.1,0.9]}$ so that the safety layer can correct it. Let an agent's position and velocity be x_B, v_B, each $\in \mathbb{R}^2$. Then the rival agent's position and velocity is x_B^R, v_B^R, each $\in \mathbb{R}^2$. The target position is $x_T \in \mathbb{R}^2$. The state space for each agent is $S = (x_B, v_B, x_B^R, v_B^R, x_T)$ while the action space $A = (v_B)$ vector. The reward function $R = max(1 - 10 * ||x_B - x_T||_2^2, 0)$ and $\gamma = 0.99$. The constraint C consists of:

1. $x_B \in B_{[0.1,0.9]}$, which is the feasible square region,
2. $|x_{Bi} - x_{Bi}^R| > 0.1 \forall i \in \{1, 2\}$, which is the separation between the agents along each axis.

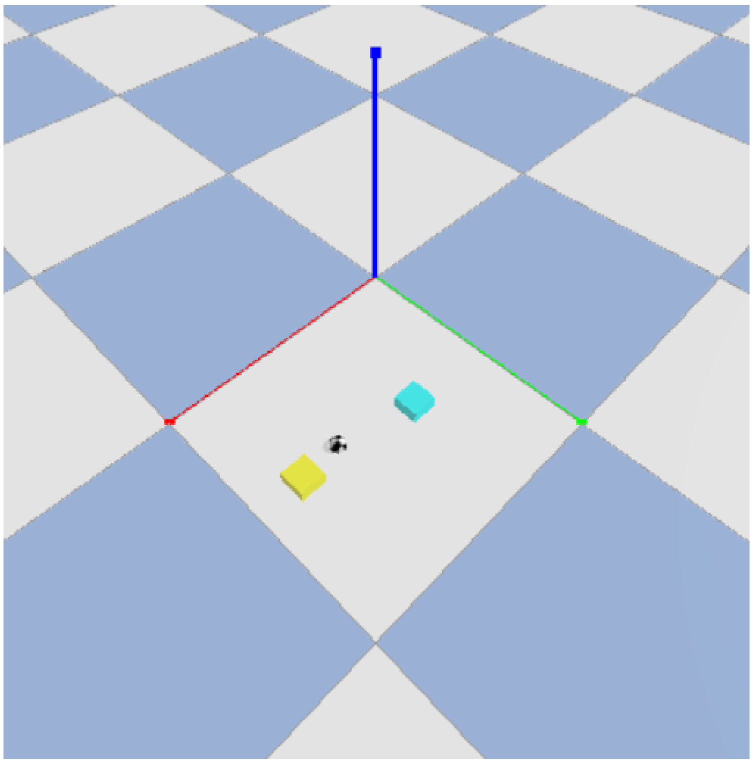
Figure 3.2: PyBullet environment for two agents (Agent 1 is the yellow cube and Rival Agent 2 is cyan) and target (checkered sphere).

3.2 Two Agent Safety Signals

To accommodate for an increase in number of agents, we define three types of safety models.

3.2.1 Rival Agnostic

Let $s = \{x_B, v_B\}$. Then the Rival Agnostic safety model (RASM) does not account for the rivals state $s^R = \{x_B^R, v_B^R\}$. Thus the safety signal remains the same.

3.2.2 Rival Semi-Gnostic

As the new constraint is dependent on the state (specifially the position) of both the agents, we incorporate s^R, and and even a^R into our model.

$$\bar{c}_i(s', s'^R) \triangleq c_i(s, s^R, a, a^R) \approx \bar{c}_i(s, s^R) + g(s, s^R; w_i)^T a + g^R(s, s^R; w_i)^T a^R \tag{3.1}$$

However, directly applying the Lagrangian to the above equation might introduce instabilities to the solution as a^R can be an unsafe action. We can simplify the above equation by neglecting the a^R term and approximating $\bar{c}_i(s', s'^R)$ as follows:

$$\bar{c}_i(s', s'^R) \triangleq c_i(s, s^R, a) \approx \bar{c}_i(s, s^R) + g(s, s^R; w_i)^T a \tag{3.2}$$

The resulting Rival Semi-Gnostic safety model (RSGSM) has a closed form solution following Equations 2.8 and 2.9 as:

$$\{\lambda_i^\star\} = [\frac{g(s, s^R; w_i)^T \mu_\theta(s, s^R, x_T) + \bar{c}_i(s, s^R) - C_i}{g(s, s^R; w_i)^T g(s, s^R; w_i)}]^+ \tag{3.3}$$

and

$$a^\star = \mu_\theta(s, s^R, x_T) - \lambda_{i^\star}^\star g(s, s^R; w_i), \tag{3.4}$$

where $i^\star = argmax_i \lambda_i^*$.

3.2.3 Rival Gnostic

On top of the RSGSM, an even more complete picture of the constraints is provided by predicting the constraint value using the actions of both the agent and its rival. The third linearization called the Rival Gnostic safety model of n^{th} order (RGSM^n) is formulated as follows with c_i^j being the ith constraint after the j^{th} iteration and a^j, a^{Rj} being the agent and rival actions after the j^{th} correction.

$$\bar{c_i}^j(s', s'^R) \triangleq c_i^j(s, s^R, a^j, a^{Rj}) \approx \bar{c_i}^j(s, s^R) + g(s, s^R; w_i)^T a^j + g^R(s, s^R; w_i)^T a^{Rj} \tag{3.5}$$

Thus, instead of obtaining a closed form solution to Equation 3.1, we can iteratively correct the solution n times.

3.3 Policy Algorithm

The Deep Deterministic Policy Gradient (DDPG) algorithm [LHP+15] is used to train the policy network. DDPG is a popular algorithm for continuous control problems. There are two main entities within the DDPG. The first is the Actor network, which gives a raw policy (devoid of noise). The second is the Critic network, which is used to estimate the Q-value (Equation 1.1) of the current state-action pair. The agent interacts with the environment to receive the state, and based on this state, the actor network generates an action. The environment then returns the reward and the next state to the agent, and this information is used to update the actor and critic networks. Specifically, the critic is directly used to update the actor network. There are target networks, which are copy of the two networks which lag behind in updates to provide stability. They are periodically updated by copying the weights of the local networks.

Algorithm 2 DDPG

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
            $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$
        Update the target networks:
            $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
            $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
    end for
end for
```

Chapter 4

EXPERIMENTAL RESULTS AND DISCUSSION

In this section, two sets of experiments are presented. The settings for the experiments are described below.

The DDPG models are trained for 10 epochs, with 6000 training steps and 1500 evaluation steps per epoch. The batch size is 256. Episodes terminate when 300 time-steps have passed or the agent violates constraints. The size of the replay buffer is 1000000. The discount and the polyak factors are 0.99 and 0.995 respectively. The actor and critic learning rate is 0.001. A Gaussian noise with mean 0.01 is added to the action. The agent randomly explores for the first 500 steps after which it starts acting under the DDPG policy. The Actor network has hidden layers [128, 64]. The Critic network has hidden layers [64, 128, 32]. Both the networks are optimized using the Adam optimizer.

Each constraint model of the Safety layer has hidden layers [10, 10]. It is trained with a batch size of 256 with 0.0001 learning rate for 10 epochs. There are 6000 training and 1500 evaluation steps per epoch. The replay buffer size is 1000000. Each model is optimized using the Adam optimizer. The plots in the following sections have an EMA smoothing of 0.6.

4.1 Convergence of Safe and Unsafe Policies for Single Agent Environment

The preliminary experiment consisted of observing the policy convergence in a single agent environment. We study the Mean Episodic Reward and Length as indicators of policy behavior. A higher average reward means that the policy is able to successfully reach the target on average. A higher average episode length indicates less number of constraint violations.

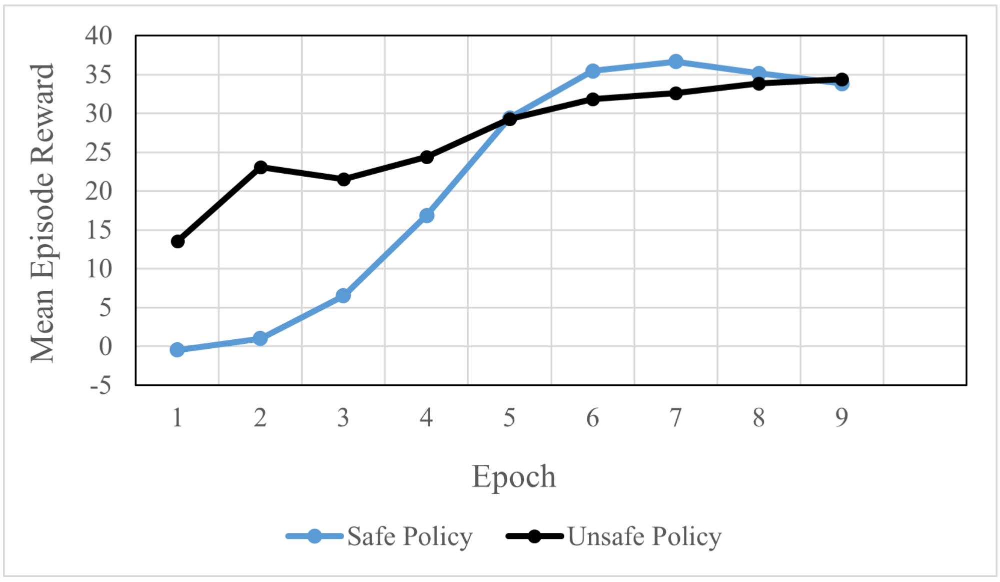

Figure 4.1: Mean Episodic Reward for Single Agent Environment

For the rest of the discussion, we will refer to the policy with safety-layer as the "Safe" policy and to the policy without the safety-layer as the "Unsafe" policy. From Figure 4.1, it is observed that the initial performance of the Safe policy is lower than the Unsafe policy. However, it soon overtakes the latter. From Figure 4.2, it can be seen that the Unsafe policy violates the constraints during multiple epochs and thus has a lower Mean Episode Length. However, even during the initial stages of the training, the Safe policy did not violate any constraints. This is potentially why it did not rapidly learn initially.

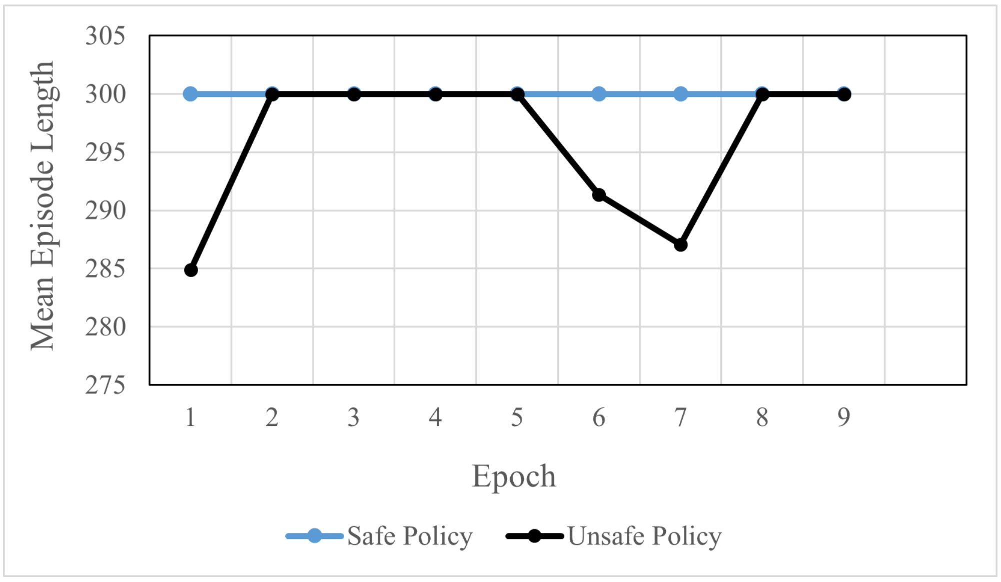

Figure 4.2: Mean Episodic Length for Single Agent Environment

4.2 Convergence of Safe and Unsafe Policies for Two Agent Environment

Much like the previous section, we look at various metrics to deduce the convergence and performance of our policies. In the following figures, the Rival policy is indicated by an "R" ahead of the policy name. Upon examining the Mean Episodic Length for the policies in Figure 4.3, it is found that the Unsafe policy has the highest length (tending to full towards the later epochs) followed by the RSGSM policy and finally the RASM policy has the lowest length. This appears to be surprising at first, as it does reflect the findings of the previous section. However, what we can infer from these findings is that the agents of the Unsafe policy have the least amount of constraint violations while the RASM policy's have the most.

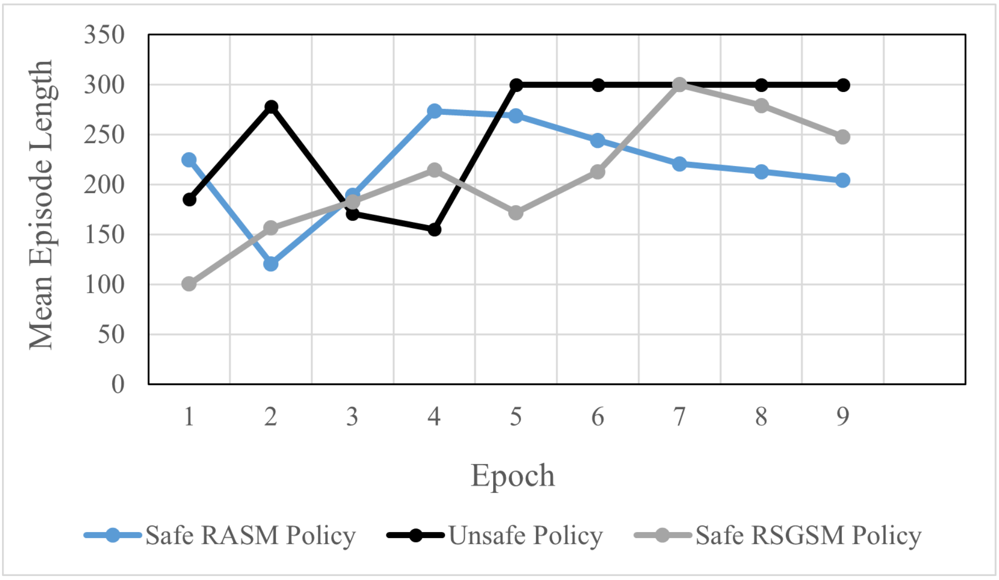

Figure 4.3: Mean Episodic Length for Two Agent Environment

Next, we examine the Mean Episodic Reward for the policies. Figure 4.4 contains the Mean Episodic Reward for RASM and Unsafe policies, while Figure 4.5 contains the same for RSGSM and Unsafe policies. The common finding of Safe policies having lower rewards during the initial epochs holds true.

Further, it is found that the Unsafe policy is oscillatory for both the agents but converges to a similar value, indicating similar chances of success for both. For the RASM policy, it is found that the magnitude of oscillations is higher. The oscillations indicates that the "better" and "worse" agents switch positions, mostly due to some "bad" episodes of the "better" agent during which the "worse" agent rapidly improves. One of the agents converges to a better policy than the other, although both have positive rewards. Much interestingly, unlike the previous two policies, one of RSGSM's agents completely dominates the other and has a much higher mean reward than its counterpart. The behaviour is not oscillatory and the "better" agents stays better throughout.

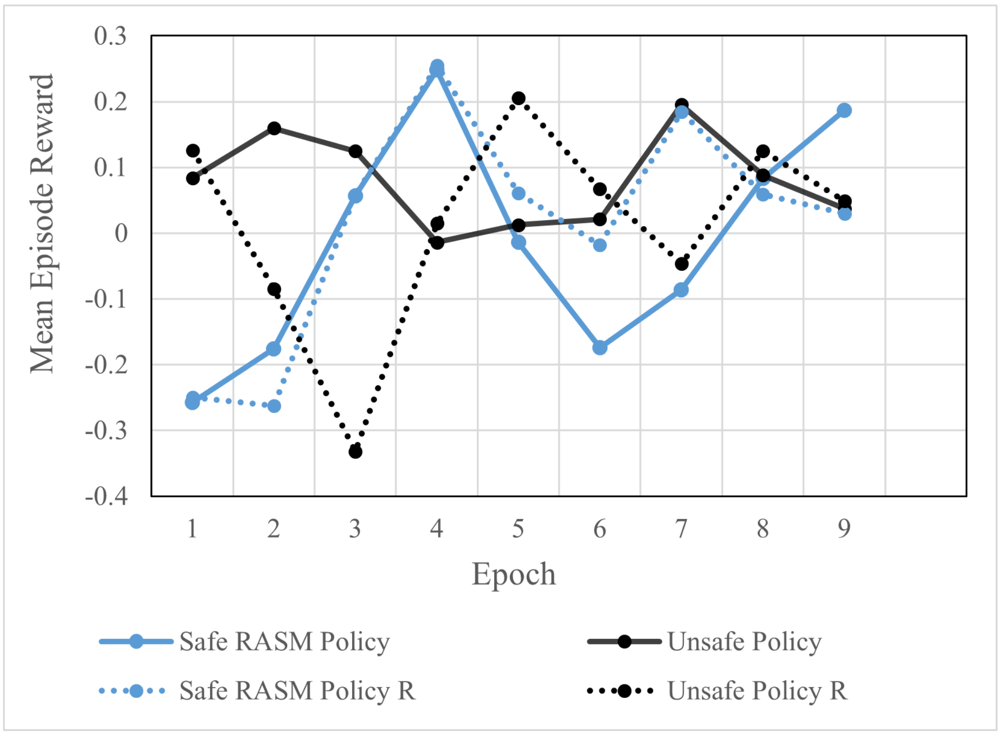

Figure 4.4: Mean Episodic Reward for Two Agent Environment for Safe RASM and Unsafe Policies

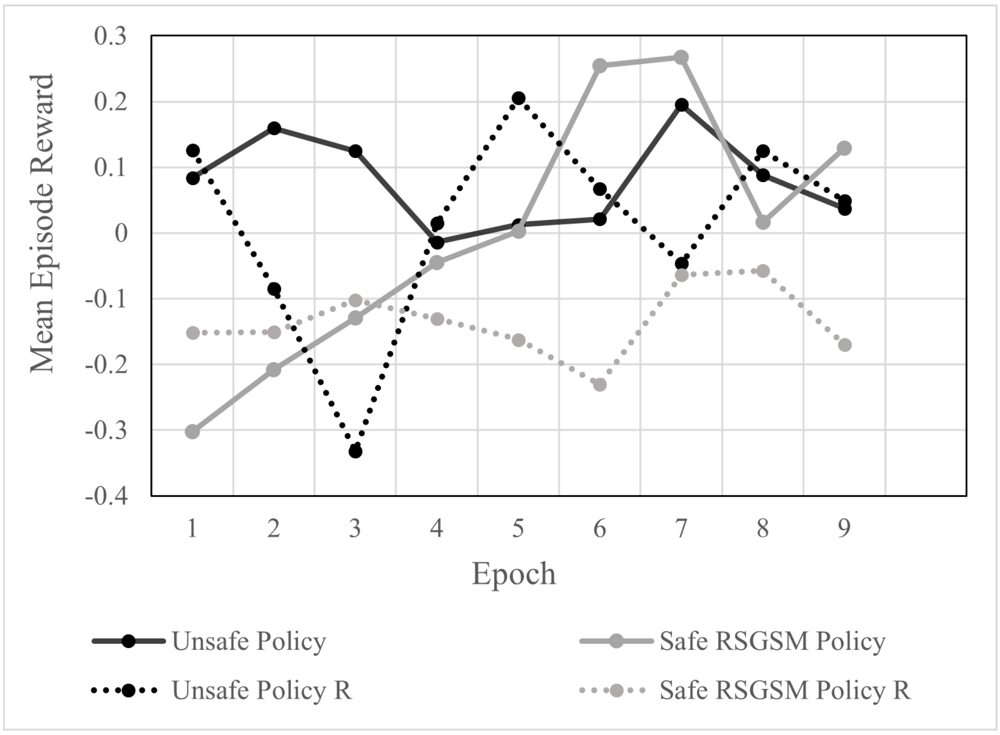

Figure 4.5: Mean Episodic Reward for Two Agent Environment for Safe RSGSM and Unsafe Policies

We now examine the action magnitudes for the three policies plotted in Figure 4.6. The three policies converge to very different average action magnitudes. The RASM agents have the highest magnitudes while the Unsafe agents have the lowest. This can be attributed to the fact that the Unsafe agents don't have a safety layer to correct their actions and thus the magnitudes are optimized to be low to avoid constraint violation over a timestep. The safety layer augments the unsafe actions and allows the Safe agents to take higher magnitude actions. However, as the RASM's safety layer is blind to the other agent, we can deduce that this failure leads to lower episodic lengths in Figure 4.3.

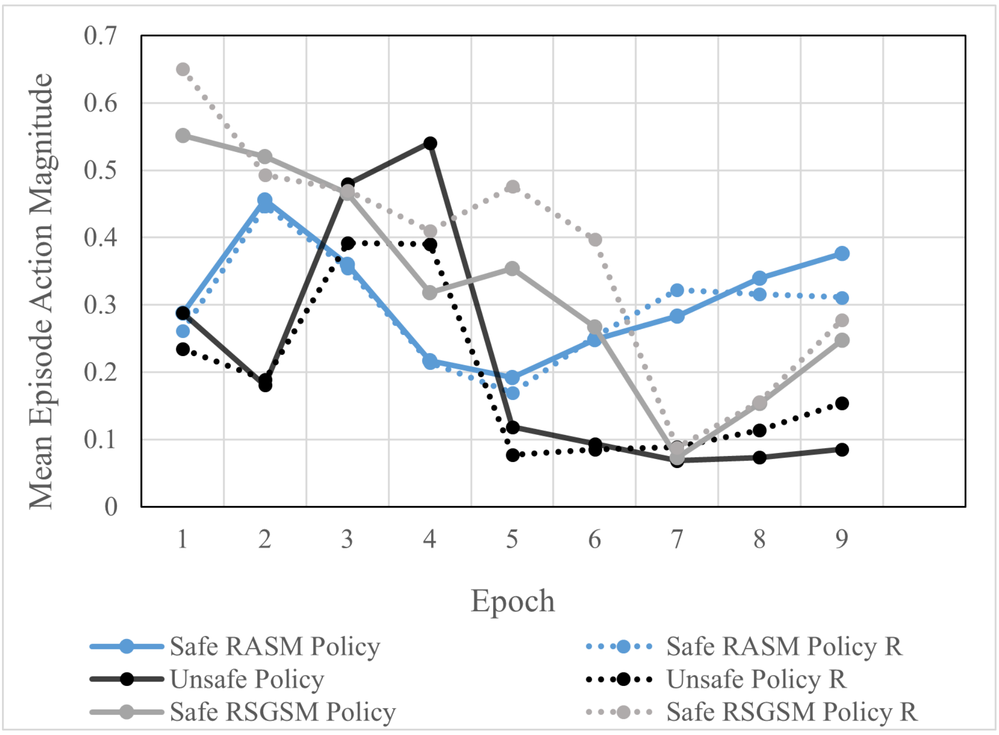

Figure 4.6: Mean Episodic Action Magnitude for Two Agent Environment for Safe and Unsafe Policies

Finally, we examine the Actor Loss itself for the three policies presented in Figures 4.7 and 4.8. From the first figure, it can be examined that the three policies exhibit similar trends for the Actor loss of Agent 1. All three of them have a monotonically decreasing loss over varying periods. Thus, this indicates convergence of Agent 1 for all three. However, a striking difference is observed for the Rival Agent 2. While the RASM and the Unsafe policies have a similar trend of decrease and convergence for Rival Agent, the RSGSM's Rival Agent instead has a loss asymptotically converging to 0.1. This means that over the steps the Rival Agent does not learn, or simply put, Agent 1 does not allow the Rival agent to learn at all after it gets a significant lead. This resulting garbage policy for the Rival agent is possibly the reason for lower episodic lengths in Figure 4.3. Furthermore, as there is no oscillatory behaviour at all unlike the other systems, it can be deduced that Agent 1 learns to exploit the behavior of the Rival Agent, i.e. it learns to exploit the constraints.

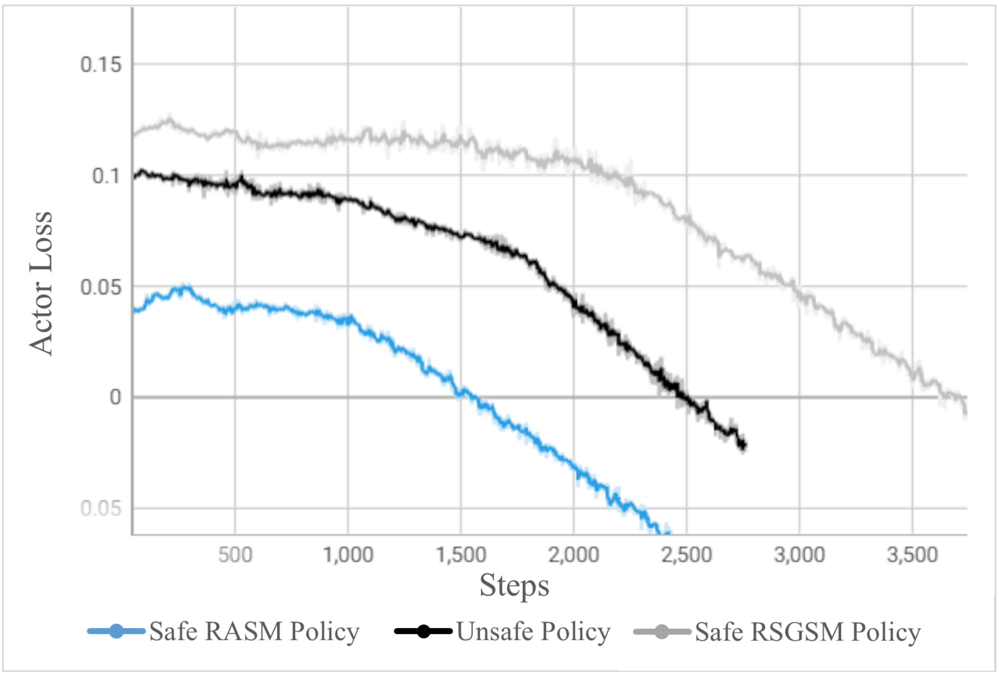

Figure 4.7: Actor Train Loss for Agent 1 of the Two Agent Environment

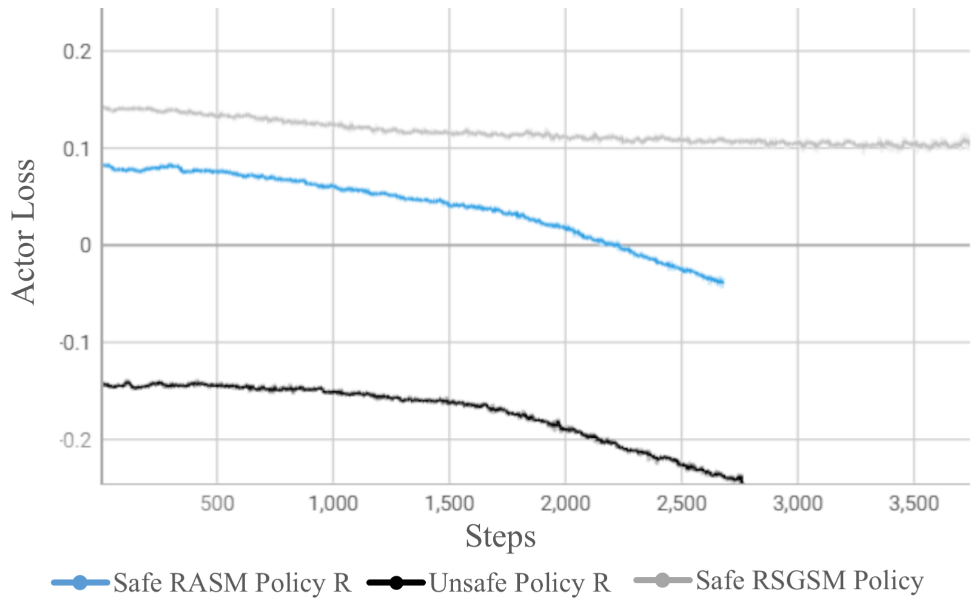

Figure 4.8: Actor Train Loss for Rival Agent 2 of the Two Agent Environment

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this study, we have formulated and studied multi-agent safety models. The primary contribution of the study is the insight into learnability of the policy of one agent by another. Here, the constraints imposed by the additional safety layer are exploited under the RSGSM, which leads to one agent emerging with a landslide victory over the other. The RASM and Unsafe models had some degree of convergence for policies of both agents unlike the RSGSM. While predicting which agent will have a higher reward is a coinflip task, under RSGSM the better agent will retain its performance. Thus, this opens multiple avenues of work into multi-agent ensembles, and has applications in competitive games like Chess and DOTA 2. It can also benefit cooperative systems such as a fleet of self-driving cars, which need to impose contraints while reaching targets.

5.2 Suggestions for Future Work

While the study has interesting results under the time constraints, more work can be done on this probelm. Some of the suggested future topics are:

1. **Scalability Analysis:** A scalability analysis of the behaviour of the system with the number of agents and targets can provide insights into complex ensemble behaviours.
2. **Game complexity:** While this study was concerned with a simple reach task, more complex games with more constraints can reveal more insights into policy convergence.
3. **Pre-trained Safety Models:** A pre-trained safety model in complex environments can facilitate learning of constraints through fine tuning instead learning from scratch.
4. **Non-Linear Constraints:** The analysis of Lagrangian constraints on non-linear constraints with size of constraint network can provide insights into constraint complexity.
5. **Convergence of RGSM:** While we have formulated the RGSM, the mathematical convergence of RGSM needs to be proven.
6. **Lyapunov Constraints:** Instead of the Lagrangian constraint model, the richer Lyapunov constraints can provide richer performance.

Appendix A

CODE AVAILABILITY

The code for the custom PyBullet-Gym environment and policies covered is available on `https://github.com/SChazal/safe-policy-multi-agent`.

Bibliography

[AHTA17] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[BBDS10] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221, 2010.

[BTSK17] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. *Advances in neural information processing systems*, 30, 2017.

[CGJP17] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120, 2017.

[CNDGG18] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[CNF$^+$19] Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.

[DDV$^+$18] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[LHP$^+$15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[PDMT18] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243. IEEE, 2018.