SHEPHERDING OF MULTI-EVADER SYSTEM WITH OPTIMAL PURSUER TRAJECTORY

A Dissertation
Presented to
The Academic Faculty

By

Saqib Azim (150070031)

Advisor: Dr. Debraj Chakraborty
Department of Electrical Engineering

Indian Institute of Technology, Bombay

June 2019

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my advisor, Prof. Debraj Chakraborty, who continually and convincingly conveyed the spirit of adventure with regards to research. Without his guidance and persistent help, this dissertation would not have been possible.

I would also like to express my gratitude to Mr. Aditya Choudhary for helping me throughout the course of the project. Especially the constant help and support I got from him during the early stages of the project were of immense value to me and this project.

TABLE OF CONTENTS

Acknowledgments ii

Chapter 1: Introduction and Background 1

Chapter 2: Common Notations 3

Chapter 3: Interaction Rules of an Agent 4

3.1 Repulsion on an evader due to a pursuer 4

3.2 Attraction on an evader towards the center (centroid) of neighbours 7

3.3 Repulsion between two evaders in close proximity 7

Chapter 4: Shepherding with Optimal Pursuer Trajectory 9

4.1 Introduction 9

4.2 Problem Formulation 9

4.3 Discrete time-version of the formulated problem 10

4.4 Solving the optimization problem 11

4.4.1 Brief Overview of Global Optimization Toolbox (GOT) 12

4.4.2 Global Search Algorithm 12

4.5 Experimental Description 14

4.6 Results and Evaluation 14

4.7 Insights and Discussion 15

4.8 Dipole Intuition 15

Chapter 5: Greedy Approach 22

5.1 Introduction 22

5.2 Experiments and Results 23

Chapter 6: Learning Trajectories using Deep Learning 25

6.1 Introduction 25

6.2 Long Short-Term Memory (LSTMs) Networks 25

6.3 Machine Learning Architecture 26

6.4 Dataset and Training 27

6.5 Result Evaluation 28

6.6 Limitation and Conclusion of this approach 28

References 32

CHAPTER 1

INTRODUCTION AND BACKGROUND

Herding of sheep by shepherds or sheepdogs is a brilliant example of one individual causing many unwilling individuals to move in same direction. Shepherding is a primitive activity practised to herd the livestock animals such as sheep, cows, etc. to a particular destination or in same direction. The sheepdogs have this unique ability to herd large number of sheep to a desired destination. Despite single dogs solving the shepherding problem every day, it remains unknown which algorithm they employ or whether a general algorithm exists for shepherding. Many attempts have been made to gain an understanding of how a single agent (sheepdog) can gather and herd a group of other agents (sheep). Most research has adopted a theoretical approach and sought to model the interaction of the agents based on attraction, repulsion and alignment models that are common in studies of collective herd behaviour. A research study models and reproduces the empirical behaviour of a herd of sheep (agents) and comes up with an algorithm for the shepherd based on adaptive switching between collecting the agents when they are too dispersed and driving them when they are aggregated. The shepherd is given a different set of rules from the rest of the sheep flock, which are repelled by the shepherd. A sheep flock's response to a herding dog can be explained using selfish herd theory [1] which puts forward that aggregation among herd of sheep result from individual efforts to reduce their own predation risk or threat by moving towards the center of a group. In one class of models, the shepherds rules prescribe a side-to-side movement behind the group while herding it towards the target. Such algorithms are appropriate for herding small groups but herding of larger groups (more than 40 individuals) typically requires multiple shepherds. However, single sheepdogs can successfully herd flocks of 80 or more sheep both in their everyday work and in competitive herding trials [2, 3]. So, what are the sheepdogs doing that the agent shepherds (or the flocking agents) are not? The shepherding problem has numerous applications, for example in crowd control [4], [5], cleaning up the environment [6], herding of livestock [7], keeping animals away from sensitive areas and collecting or guiding groups of exploring robots [8]. One can add another dimension to the shepherding problem by constraining the time or the path-length of shepherd required to drive the agents to a predefined fixed destination.

(a) An example of a trained sheepdog herding the flock

(b) Drones being used for shepherding

Figure 1.1: Example of shepherding in real-life

A sheep has the tendency to evade from the shepherd or sheepdog while the shepherd has the tendency to drive (pursue) the herd of sheep towards destination. Therefore, for notational convenience, a sheep would be henceforth denoted as an evader while a shepherd or sheepdog as a pursuer. An agent would be denoted both as the pursuer and evader.

CHAPTER 2

COMMON NOTATIONS

Below are some common notations which have been used throughout this thesis.

- N_e denotes the number of evaders (≥ 1)
- N_p denotes the number of pursuers ($= 1$)
- Let $p(t) \in R^2$ denote the position of pursuer at time instant t
- Let $e_i(t) \in R^2$ denote the position of i^{th} evader at time instant t
- Let $z(t) = z \in R^2$ denote the predefined fixed destination point
- $r(a_i, a_j, t) = a_i(t) - a_j(t)$ is the line-of-sight vector pointing from agent a_j towards agent a_i at time instant t
- $d(a_i, a_j, t) = \|r(a_i, a_j, t)\|_2$ is the distance between agents a_i and a_j at time instant t
- $c_{i,n}(t)$ denotes center of n nearest evaders corresponding to i^{th} evader at time instant t
- v_{emax} denotes maximum velocity of an evader
- v_{pmax} and v_{pmin} denote maximum and minimum velocity of a pursuer respectively
- T denotes the time of completion of the task
- $u(.)$ is the unit step function
- $\dot{x}(t) = \frac{dx}{dt}$ denotes the derivative of x with respect to time
- $\hat{v} = \frac{v}{\|v\|_2}$ denotes the unit vector along the direction of v

CHAPTER 3

INTERACTION RULES OF AN AGENT

The interaction of an evader due to any other agent or a group of agents can be classified into the following categories.

3.1 Repulsion on an evader due to a pursuer

This repulsion is assumed to be along the line joining pursuer-evader, pointing towards the evader. It should depend on the pursuer-evader separation. This interaction can be modelled in terms of relation between evader repulsive velocity and pursuer-evader separation. Intuitively, this dependence of evader repulsive velocity on the pursuer-evader separation should be modelled in an inverse relation such that when the separation is large, repulsive velocity is small and vice-versa. We have compared some of the potential inverse relations which can be used to model this dependence.

1. In this type of relation, the evader velocity decreases linearly as the pursuer-evader separation increases and it vanishes once the separation attains a threshold d_{max}. This kind of relation is similar to the sheep-shepherd behaviour where a sheep starts running away from the shepherd once the shepherd is within certain distance from the sheep.

$$\|\dot{e}(t)\|_2 \propto \begin{cases} v_{emax} - d(e,p,t), & \text{if } d(e,p,t) < d_{max} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

2. Based on the inverse-power interaction rule, here the evader velocity increases as the pursuer-evader separation decreases and it explodes as the separation reaches zero. This type of interaction rule is unstable at small pursuer-evader separation as evident in Fig. 3.1. Clipping the exploding portion at some small separation threshold would make this type of interaction rule realizable.

$$\|\dot{e}(t)\|_2 \propto \frac{1}{[d(e,p,t)]^k}, \quad k \in \mathbb{Z}^+ \tag{3.2}$$

3. The exponentially decaying interaction relation, being always bounded and finite, compensates

for the instability present in the inverse-power interaction rule in Eq. 3.2. Fig. 3.1 clearly shows that the inverse-power and exponentially decaying relation have significant dissimilarities at small separation values but approximately similar behaviour at reasonable and large separation. Henceforward, this motivates us to model the relation between evader velocity and pursuer-evader separation in an exponentially decaying manner.

$$\|\dot{e}(t)\|_2 \propto e^{-kd(e,p,t)}, \quad k \in \mathbb{R}^+ \tag{3.3}$$

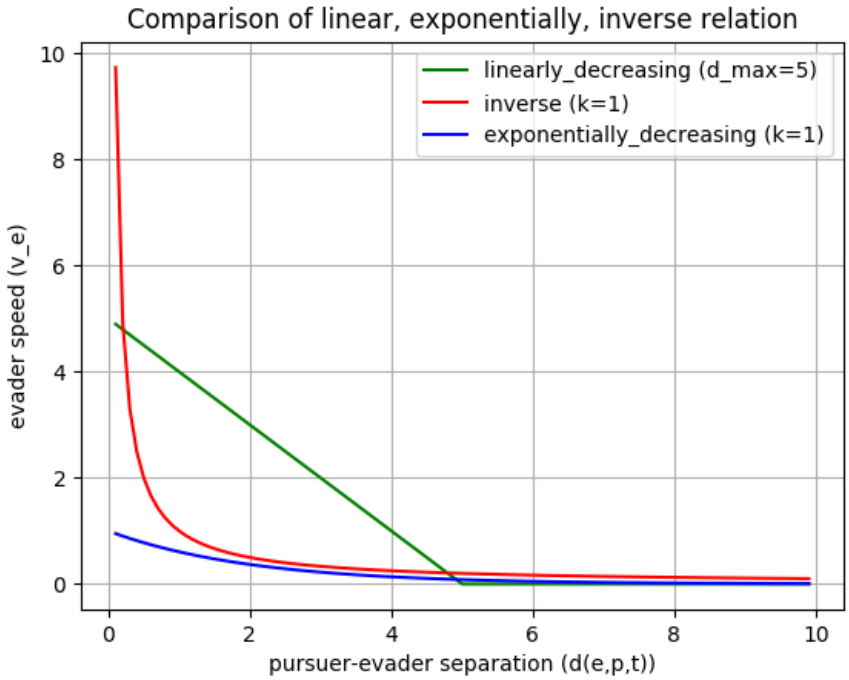


Figure 3.1: Comparison of various interaction rules by showing plots of variation between evader-velocity and pursuer-evader separation

We compared different types of relation between repulsive velocity of evader and the pursuer-evader separation and we established that exponentially decaying relation is better at characterizing the sheep-shepherd behaviour among other relations. From now on, we will model the evader repulsive velocity in terms of exponentially decaying form of pursuer-evader separation. We have formulated few such possible relations which are presented and explained below.

inputencIn a real-life shepherding task, a flock of sheep generally grazes a field when the shepherd is far from them but when the shepherd approaches the herd and starts getting more and more closer

to the herd, then only the flock starts driving away. The relation in Eq. 3.4 draws its inspiration from real-life instances. We have modeled below the repulsive velocity such that it starts acting only when the pursuer reaches within d_{max} distance of that evader.

$$\dot{e}(t) = \begin{cases} v_{emax\_repl} e^{-kd(e,p,t)} \hat{r}(e,p,t), & \text{if } d(e,p,t) < d_{max} \\ 0, & \text{otherwise} \end{cases}, \quad k \in \mathbb{R}^+ \tag{3.4}$$

where v_{emax_repl} denotes maximum evader repulsive velocity due to pursuer, d_{max} denotes threshold pursuer-evader separation beyond which repulsive evader velocity vanishes.

A plausible variation of the interaction rule in Eq. 3.4 is presented below. The only difference is the exclusion of threshold separation distance d_{max}. This signifies that the repulsive evader velocity is always some positive quantity. At first, this might seem unreasonable but due to the exponential decaying term and appropriate k value, the velocity approximates to zero and becomes negligible as the separation distance becomes large.

$$\dot{e}(t) = v_{emax\_repl} e^{-kd(e,p,t)} \hat{r}(e,p,t), \quad k \in \mathbb{R}^+ \tag{3.5}$$

Compared to Eq. 3.5, the equation below consists of an additional factor $[\frac{1+cos\theta}{2}]$ which always lies in the range $[0, 1]$. Here, $\theta(t)$ denotes the angle between pursuer-evader line-of-sight vector $\hat{r}(e,p,t)$ and pursuer velocity $\dot{p}(t)$ at time t.

$$\dot{e}^{repl\_p}(t) = v_{emax\_repl} e^{-kd(e,p,t)} \left[\frac{1 + cos\theta(t)}{2}\right] \hat{r}(e,p,t), \quad k \in \mathbb{R}^+ \tag{3.6}$$

$$cos\theta(t) = \frac{\dot{p}(t).\hat{r}(e,p,t)}{\|\dot{p}(t)\|_2} \tag{3.7}$$

The motivation behind including this factor can be explained based on logical and empirical reasoning. When chased by a shepherd, a sheep would have much higher tendency to run away if the shepherd is moving directly towards it compared to when the shepherd is moving in some other direction. Mathematically, this means that when the angle between pursuer velocity and pursuer-evader line-of-sight vector is small, the evader repulsive velocity is large and vice-versa.

3.2 Attraction on an evader towards the center (centroid) of neighbours

This interaction force on an evader directed towards the center of n nearest evaders (where $n < N_e$ is a hyperparameter) should depend on two factors:

1. distance between an evader and center of n nearest neighbouring evaders $d(c_{i,n}, e_i, t)$
2. separation between an evader and pursuer $d(e_i, p, t)$

The following argumentative example has been provided in order to support this dependence. In a typical herd, a sheep has much higher tendency to not be very far from the center of the herd in presence of a threat rather than in absence of any threat. More the separation between a sheep and its herd center $d(c_{i,n}, e_i, t)$, more the tendency of the sheep to move towards its herd center and vice-versa. Similarly, larger the distance between a sheep and shepherd $d(e_i, p, t)$, the smaller the threat to the sheep, which implies lesser tendency to move towards its herd center and vice-versa (smaller the distance between a sheep and shepherd, the larger the threat to the sheep, implying much higher tendency to move towards its herd center). Based on these arguments, we formulate this attractive interaction in terms of dependence of attractive component of evader velocity on the pursuer-evader and evader-centroid separation.

$$\|\dot{e}_i^{attr}(t)\|_2 \propto \frac{d(c_{i,n}, e_i, t)}{d(e_i, p, t)}, \quad \forall \quad i = 1, \ldots, N_e \tag{3.8}$$

The right hand side in eq. 3.8 becomes unstable as the pursuer-evader separation approaches zero. As explained in the previous section, we modify Eq. 3.8 by including exponentials in a way such that the attractive behaviour remains unchanged.

$$\dot{e}_i^{attr}(t) = v_{emax\_attr} \frac{e^{k_1 d(c_{i,n}, e_i, t)}}{e^{k_2 d(e_i, p, t)}} \hat{r}(c_{i,n}, e_i, t), \quad \forall \quad i = 1, \ldots, N_e, \quad k_1, k_2 \in \mathbb{R}^+ \tag{3.9}$$

where v_{emax_attr} denotes the maximum evader velocity due to attraction towards center.

3.3 Repulsion between two evaders in close proximity

This kind of repulsive interaction between any two evaders has been modelled such that it is active only when they are in close proximity to each other. Regardless of the distance to the pursuer, intuitively any

two evaders should be repelled by each other at very small distances of less than r_a (hyperparameter). The role of this type of interaction is to ensure that none of any pair of evaders are occupying same or very nearby positions. For any two evaders, this repulsive velocity is equal in magnitude and directed along the line joining them and away from each other. It can be modelled as constant velocity or decreasing with increasing separation upto r_a.

$$\dot{e}_i^{repl\_e}(t) = \begin{cases} v_{emax} e^{-kd(e_i,e_j,t)} \hat{r}(e_i, e_j, t), & \text{if } d(e_i, e_j, t) < r_a \\ 0, & \text{otherwise} \end{cases}, \quad k \in \mathbb{R}^+ \tag{3.10}$$

In order to model the real-life scenario, all of the three above mentioned interactions should be included. Therefore, evader velocity would then be the sum of the velocity component due to repulsive interaction from pursuer ($\dot{e}_i^{repl\_p}(t)$), velocity component due to attractive interaction towards the center of n nearest evaders ($\dot{e}_i^{attr}(t)$) and velocity component due to repulsive interaction due to other evaders in close proximity ($\dot{e}_i^{repl\_e}(t)$). Mathematically, the final evader velocity will be given by the following expression.

$$\dot{e}_i(t) = \dot{e}_i^{repl\_p}(t) + \dot{e}_i^{attr}(t) + \dot{e}_i^{repl\_e}(t), \quad \forall \quad i = 1, \ldots, N_e \tag{3.11}$$

In our work, we have primarily modelled the evader velocity with only the repulsive component due to pursuer unless otherwise stated. Multiple reasons account for not including the velocity component due to attraction and velocity component due to repulsion from other evaders. Firstly, in this work we would be dealing with small N_e (2,3,4, or 5) and in order to understand the effect of repulsive interaction due to pursuer, it is favourable to exclude other interaction components. Secondly, with the attractive force towards the center playing its role, the aggregation and driving becomes easier as well as the pursuer's task is reduced because of strong cohesion. In fact, even if the pursuer is very far away or practically absent, the evaders due to attraction would aggregate in some finite time and that would certainly reduce the desired complexity and spirit of this problem.

CHAPTER 4

SHEPHERDING WITH OPTIMAL PURSUER TRAJECTORY

4.1 Introduction

In the previous chapter, the objective for the controlling agent (pursuer) was to drive the evaders to a fixed destination point with the aid of only repulsive interaction on the evader. There were no constraints on the time required for task completion as well as on the trajectory of any agent. Various scenarios such as shepherding competitions, robotic manoeuvring, controlling herd of animals or robots, etc, demand time or trajectory based optimality. Many researchers have tried to model the trajectories of the agents or device an algorithm with single or multi-pursuer system for the shepherding task with various interaction rules ([9], [10], [11]). redBut hardly anything concrete or solid can be found in literature regarding shepherding which includes time-optimal or trajectory-optimal problem. This inclusion certainly brings a new dimension to the problem and makes the problem much more challenging as well as interesting. Therefore, there are two major ways in which the shepherding problem can be modified.

1. Minimize the trajectory length of the pursuer
2. Minimize the time required (T) by pursuer to complete the task

The following work deals with the inclusion of pursuer's trajectory length minimization to the shepherding problem.

4.2 Problem Formulation

Consider a system with N_e evaders and a single pursuer. The task of the pursuer is to drive the evaders to within ϵ radius of fixed destination point and simultaneously minimize its own path-length with only pursuer-evader repulsive interaction. This optimally-minimizing behaviour can be achieved by visualizing the above task as a constrained optimization problem as formulated below.

Speed of pursuer at any time instant $t = v_p(t) = \|\dot{p}(t)\|_2 = \sqrt{\dot{p}(t).\dot{p}(t)}$

Path-length traversed by pursuer in infinitesimal small time interval $dt = \sqrt{\dot{p}(t).\dot{p}(t)}dt$

Trajectory length traversed by pursuer in time-interval $T = \int_0^T \sqrt{\dot{p}(t).\dot{p}(t)}dt$

Objective Function:

$$\underset{p(t)}{\text{minimize}} \int_0^T \sqrt{\dot{p}(t)\dot{p}(t)}dt \tag{4.1}$$

Constraints:

1. This constraint relates the evader velocity with only the velocity component due to repulsive interaction from pursuer.

$$\dot{e}_i(t) = \dot{e}_i^{repl\_p}(t) \tag{4.2}$$

2. The final position of all evaders at completion time T should be within an ϵ radius of destination point z.

$$\|e_i(T) - z\|_2 \leq \epsilon, \quad i = 1, \ldots, N_e \tag{4.3}$$

3. Realistically, any agent (pursuer or evader) should have certain limit on its velocity (for example shepherd, sheep or robots). Eq. 4.2 already limits the maximum velocity of an evader to be v_{emax_repl}. The constraint below puts restriction on the maximum as well as minimum velocity of the pursuer.

$$v_{pmin} \leq \|\dot{p}(t)\|_2 \leq v_{pmax}, \quad 0 \leq t \leq T \tag{4.4}$$

4.3 Discrete time-version of the formulated problem

Any numerical optimization tool requires time to be discrete rather than continuous for numerical simulation. Hence we discretize time from 0 to T (where T is an unknown quantity) into N equal time intervals each of time-length t_e. The position of all agents at each time step (except at $t = 0$) are the variables in the optimization problem. Let t_0, t_1, t_2, …, t_N denote the discrete time steps. At each time step t_n ($0 \leq n \leq N$), $p(t_n)$, $e_1(t_n)$, $e_2(t_n)$, …, $e_{N_e}(t_n)$ denote the agent positions in R^2. $n = 0$ corresponds to the initial position of agents which is provided as input.

Number of optimization variables at each time step $= (2N_e + 2N_p)$

Total variables in optimization space $= (2N_e + 2N_p)N$

The discretized version of the optimization problem is mathematically given below:

Objective Function:

$$\underset{p(t)}{\text{minimize}} \sum_{n=0}^{N-1} \sqrt{\dot{p}(t_n)\dot{p}(t_n)} \tag{4.5}$$

where $\dot{p}(t_n) = \frac{p(t_n)-p(t_{n-1})}{t_n - t_{n-1}}$

Constraints:

$$\dot{e}_i^{repl\_p}(t_n) = v_{emax\_repl} e^{-kd(e_i,p,t_n)} \left[\frac{1+cos\theta_i(t_n)}{2}\right] \hat{r}(e_i,p,t_n), \quad \forall \quad i = 1,\ldots,N_e, \quad n = 0,\ldots,N \tag{4.6}$$

$$\dot{e}_i(t_n) = \dot{e}_i^{repl\_p}(t_n), \quad \forall \quad i = 1,\ldots,N_e, \quad n = 0,\ldots,N \tag{4.7}$$

$$\|e_i(t_N) - z\|_2 \leq \epsilon, \quad i = 1,\ldots,N_e \tag{4.8}$$

$$v_{pmin} \leq \|\dot{p}(t_n)\|_2 \leq v_{pmax}, \quad n = 0,\ldots,N \tag{4.9}$$

4.4 Solving the optimization problem

There are various optimization libraries used for solving constrained and unconstrained optimization problems. For example, NLOPT, IPOPT, GPOPT, etc, are built for python whereas in MATLAB numerous powerful optimization solvers such as fmincon (for constrained problems), fminunc (for unconstrained problems), etc, are available. The global search algorithm present in global optimization toolbox and which uses fmincon solver has been used for solving the above optimization problem. fmincon is a non-linear programming solver which finds minimum of constrained non-linear multivariable function. This solver itself consists of five algorithms listed below.

- interior-point
- sequential quadratic programming (sqp)
- sqp-legacy
- active-set
- trust-region-reflective

4.4.1 Brief Overview of Global Optimization Toolbox (GOT)

Global Optimization Toolbox provides functions that search for global solutions to problems that contain multiple maxima or minima. The toolbox solvers include global search, multistart, surrogate, pattern search, genetic algorithm, particle swarm and simulated annealing. These solvers can be used for optimization problems where the objective or constraint function is continuous, discontinuous, stochastic, does not possess derivatives, or black-box functions. GOT solvers repeatedly attempt to locate a global solution, however, no solver employs an algorithm that can certify a solution as global.

4.4.2 Global Search Algorithm

GlobalSearch (GS) and MultiStart have similar approaches to finding global or multiple minima. Both algorithms start a local solver (such as fmincon) from multiple start points. Let $f(x)$ denote the objective function and $g_1(x)$, $g_2(x)$, ..., $g_n(x)$ denote the constraint equations. We have denoted the starting point given as input in the problem structure as x_0 and the fmincon solution point from any starting point x as s_x. The entire algorithm has been described in detail in the following steps.

1. *Run fmincon from* x_0: GS runs fmincon from the start point (x_0) given as input in the problem structure. If this run converges, GS records the start (x_0) and end point (s_{x_0}) for an initial estimate on the radius of basin of attraction [12] as well as the final objective function value $\big(f(s_{x_0})\big)$ for use in the score function. The score function is the sum of the objective function value at a point and a multiple (λ) of the sum of constraint violations wherein the multiple is updated during the run.

2. *Generate Trial Points*: GS uses the scatter search algorithm [13] to generate a set of ntp (short for number of trial points) trial points within finite bounds given as input (lb and ub) and these trial points are potential start points. For unbounded components, $lb = -10^4 + 1$ and $ub = 10^4 + 1$ are the default bounds.

3. *Obtain Stage 1 Start Point*: GS evaluates the score function of a set of $nsop$ (short for number of stage 1 point) trial points. It then takes the point with the minimum score (y) and runs fmincon from that point. GS removes the set of $nsop$ trial points from its list of points to examine.

4. *Initialize Basins, Counters, Threshold*: The lst (stands for local solver threshold) is initially the

smaller of the two objective function values at the fmincon solution points starting from x_0 and stage 1 start point (y). $lst = min(f(s_{x_0}), f(s_y))$. If both of these solution points do not exist or are infeasible, lst is initially the penalty function value of the point y. GS heuristically assumes basins of attraction to be spherical. The initial estimate of basins of attraction for the solution point from x_0 and y are spheres centered at the solution points. The radius of each sphere is the distance from the initial point to the solution point. These estimated basins can overlap. There are two sets of counters (initialized to 0) associated with the algorithm. Each counter is the number of consecutive trial points that:

- Lie within a basin of attraction. There is one counter for each basin.
- Have score function greater than lst.

5. *Begin Main Loop*: GS repeatedly examines a remaining trial point from the list and performs the following steps.

6. *Examine Stage 2 Trial point*: GS runs fmincon from trial point p if the following conditions hold:

   - p is not in any existing basin. The criterion for every basin i is $|p - center(i)| > dtf \times radius(i)$ where dtf refers to distance threshold factor
   - score function at p should be smaller than lst

7. *When fmincon runs*: Set the counters for basins and threshold to 0. If fmincon runs starting from p, it can yield a positive exit flag which indicates convergence. In that case, GS updates the vector of global optim solution (gos) objects. Following two cases arise:

   - For every other solution point s_q with objective function value $f(s_q)$, $|s_q - s_p| > x_{tol} \times max(1, |s_p|)$ or $|f(s_q) - f(s_p)| > f_{tol} \times max(1, f(s_p))$. In this case, GS creates new element in the vector of gos objects
   - For some other solution point s_q with objective function value $f(s_q)$, $|s_q - s_p| \leq x_{tol} \times max(1, |s_p|)$ and $|f(s_q) - f(s_p)| \leq f_{tol} \times max(1, |f(s_p)|)$. In this case, GS regards s_p equivalent to s_q. The GS modifies the gos of s_q by adding p to the cell array of x_0 points.

   If the exitflag of current fmincon run is positive then,

(a) Set threshold to the score value at start point p

(b) Set basin radius for s_p equal to maximum of the existing radius and the distance between p and s_p.

8. *When fmincon does not run*: Increment counter for every basin containing p. Reset the counter of every other basin to 0. Increment the threshold counter if $score(p) \geq lst$. Otherwise reset counter to 0. For each basin with counter equal to mwc (max wait cycle), multiply basin radius by $(1 - r_{basin})$. Reset counter to 0. If the threshold counter equals mwc, increase the threshold: $th_{new} = th_{current} + ptf \times (1 + |th|)$.

   After reaching t_{max} seconds or running out of trial points, GS creates a vector of gos objects. GS orders the vector by objective function value from lowest to highest and this concludes the algorithm.

4.5 Experimental Description

As mentioned earlier, majority of the experiments has been performed for $N_e = 2$. The movement of agents has been confined to only 2-dimensional plane but it can be comfortably generalized to N-dimension. In all the experiments with $N_e = 2$, initial evader coordinates were chosen to be $(-0.5, 0)$ and $(0.5, 0)$ whereas the initial positions of pursuer and destination were varied across simulations in order to observe the trajectory of agents. The reasoning behind the fixed position of evaders across all simulations is to cover subset of all initial conditions such that it can give an approximate idea about the trajectories corresponding to the remaining conditions. Table 4.1 lists out the hyperparameters and their typical values used in the experiments.

4.6 Results and Evaluation

The figures below are the optimization result obtained using global search algorithm. In all these plots, the blue curve with dots corresponds to the pursuer trajectory while the red line represents line-joining the two evaders at any time step. The destination point (z) has been shown with a black circle of radius ϵ.

Hyperparameters	Typical Values
v_{emax_repl}	0.3-0.4
v_{emax_attr}	0 (in most cases)
v_{pmax}	0.3-0.5
v_{pmin}	0.05
Number of Intervals (N)	30-80 (depending on initial conditions)
time interval	1.0
epsilon (ϵ)	0.05
k	1.0
fmincon solver	interior-point or sqp

Table 4.1: List of all hyperparameters and their typical values

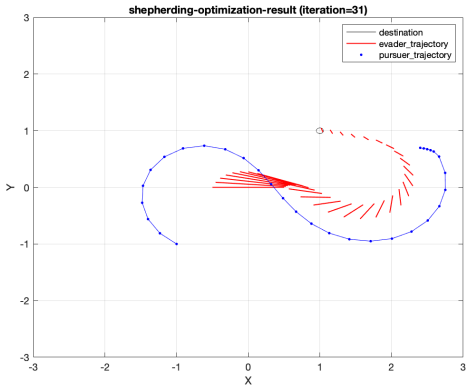
4.7 Insights and Discussion

The objective function and the constraints equation were designed to estimate a function $p(t)$ which minimizes the objective function as well as satisfies the constraints. But there are certain limitations of this process. Firstly, the optimization results are always subjected to certain tolerance on the constraints and objective function. Thus the results may not completely satisfy the constraints. Secondly, the optimization library does not certify the solution obtained as global optimum. Finally, for each initial position of agents, the optimization needs to be run separately without a guarantee that it would converge to some local optimum. Therefore, it is a time-consuming process without any upper bound on time.

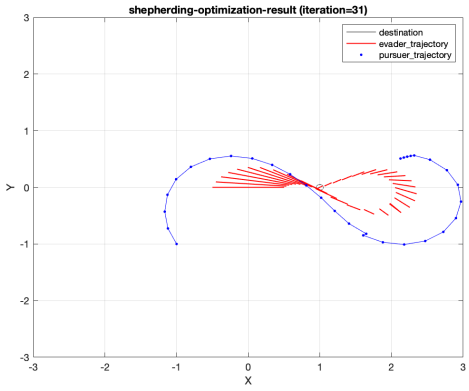
As shown in Fig 4.4, we have already dealt with the optimization problem setup and used optimization solvers to estimate optimal trajectories of agents for numerous initial conditions. A potential forward approach is to design a feedback control based iterative algorithm to predict the pursuer trajectory given initial position of agents using the obtained set of results and this step has been shown in Fig 4.4 with a red indicator. The overview of the desired feedback-based system which needs to be designed has been presented in Fig 4.5.

4.8 Dipole Intuition

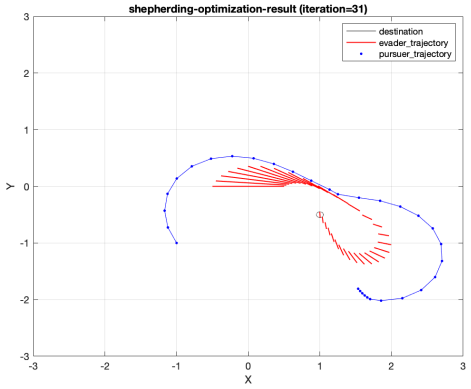
Based on the results obtained, one can visualize the two-evader system as a dipole where strength of the dipole can be modelled as the separation between them and direction can be assumed to be parallel or perpendicular to the line joining the two evaders. The dipole is experiencing repulsive force due to



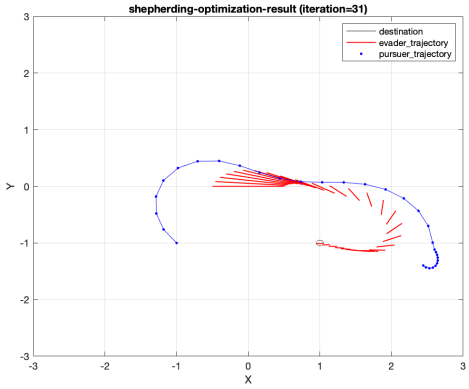
(a) $z = (1, 1)$



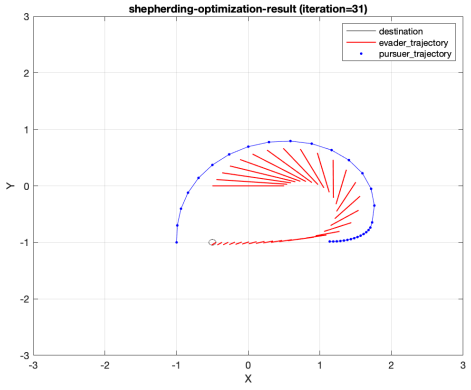
(b) $z = (1, 0.5)$



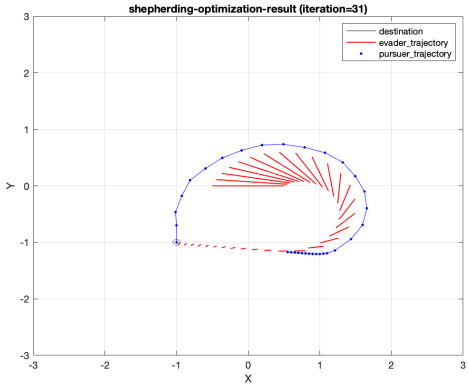
(c) $z = (1, -0.5)$



(d) $z = (1, -1)$

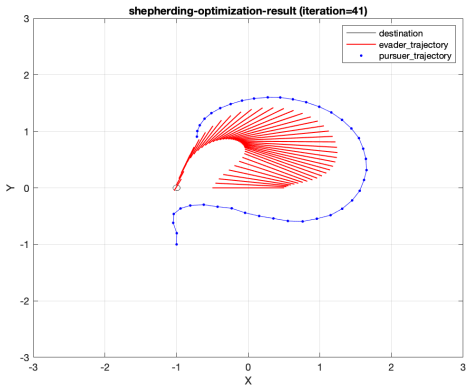


(e) $z = (-0.5, -1)$

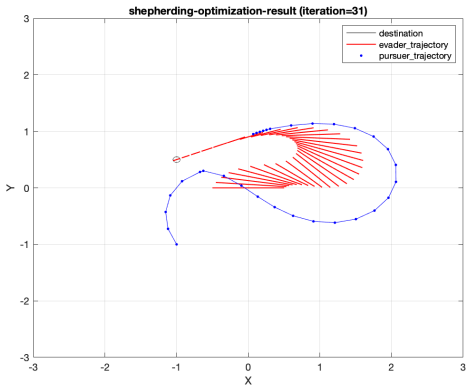


(f) $z = (-1, -1)$

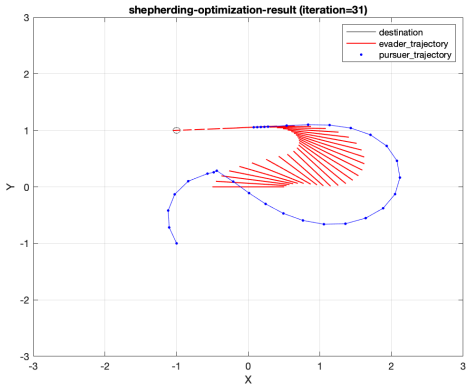
the pursuer. From the figures, it can be observed that the pursuer tries to rotate the dipole in a way such that the dipole, the pursuer and the destination are aligned in a straight line. More intuition about the feedback input based algorithm can be concluded by analyzing the evader trajectory in terms of linear and alignment motion of dipole.



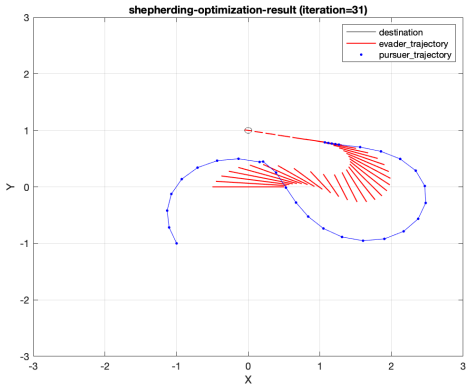
(g) $z = (-1, 0)$



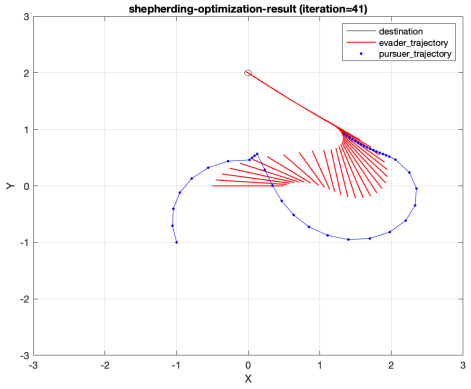
(h) $z = (-1, 0.5)$



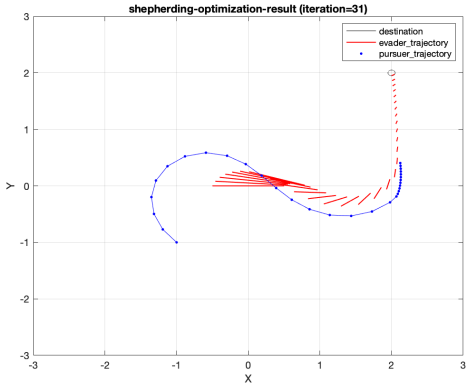
(i) $z = (-1, 1)$



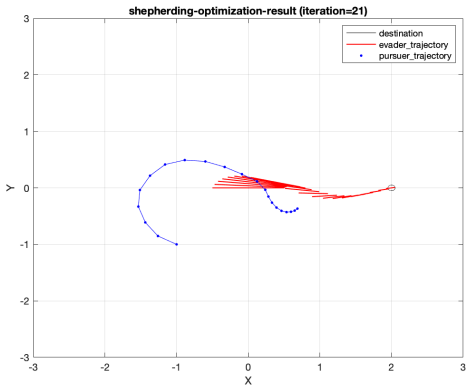
(j) $z = (0, 1)$



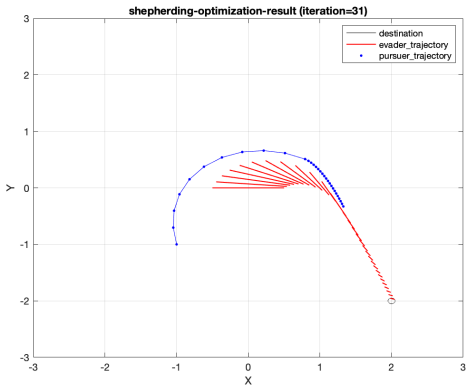
(k) $z = (0, 2)$



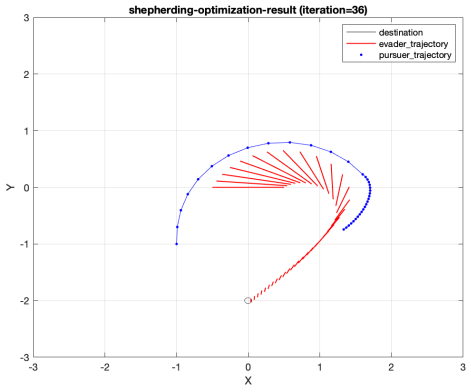
(l) $z = (2, 2)$



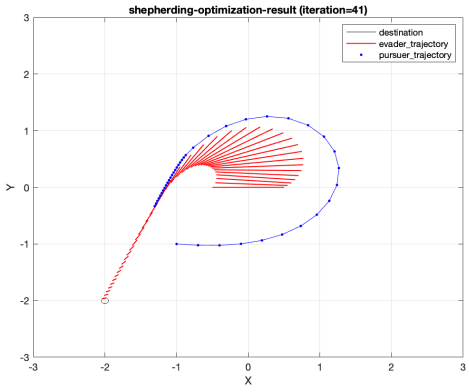
(m) $z = (2, 0)$



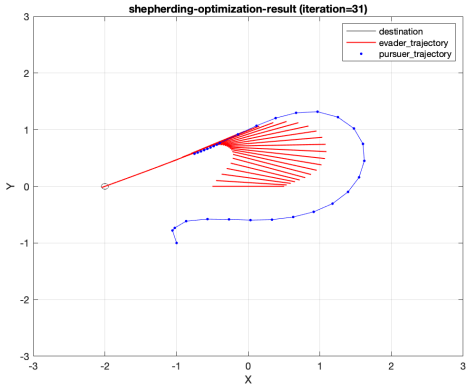
(n) $z = (2, -2)$



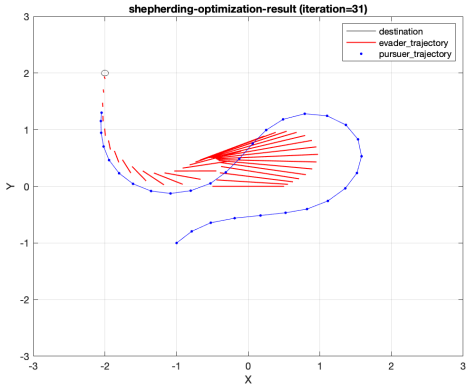
(o) $z = (0, -2)$



(p) $z = (-2, -2)$

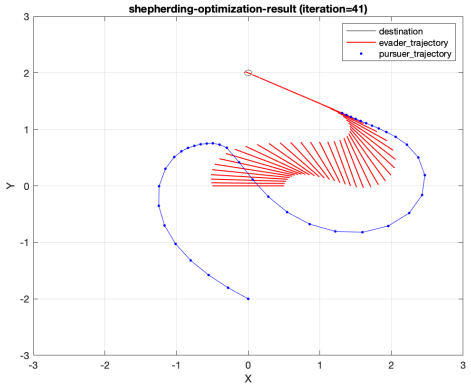


(q) $z = (-2, 0)$

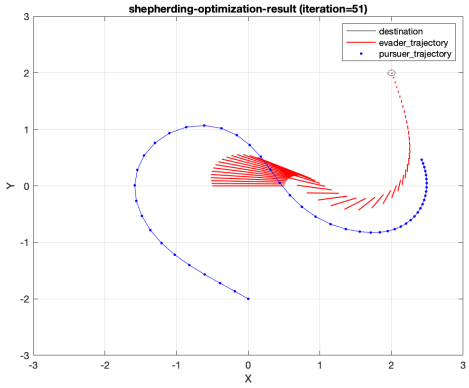


(r) $z = (-2, 2)$

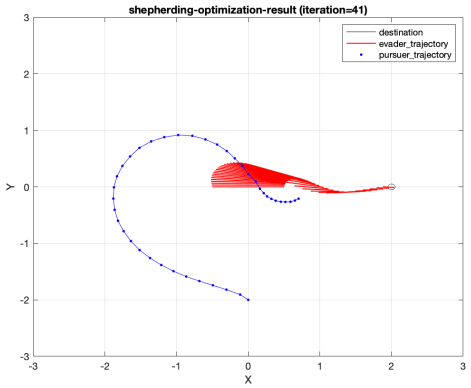
Figure 4.1: In these figures, initial pursuer position is $(-1, -1)$. In figures, (a) through (j), destination points are varied along 1×1 square arena centered at origin whereas in figures (k) to (r), the destination points are positioned around 2×2 square arena centered at origin



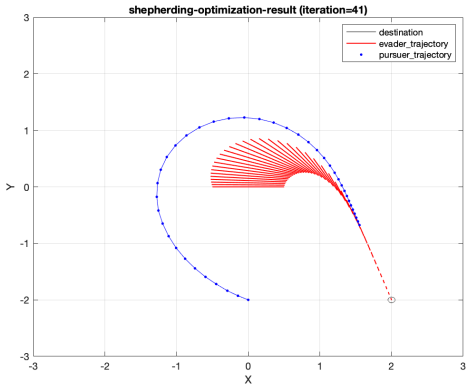
(a) $z = (0, 2)$



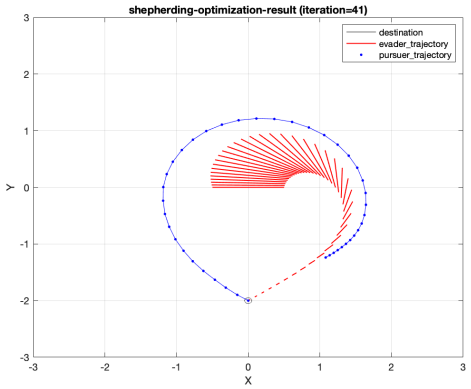
(b) $z = (2, 2)$



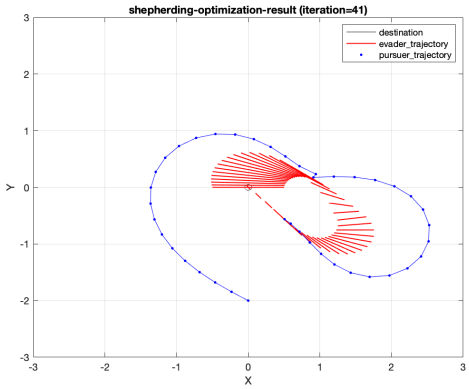
(c) $z = (2, 0)$



(d) $z = (2, -2)$

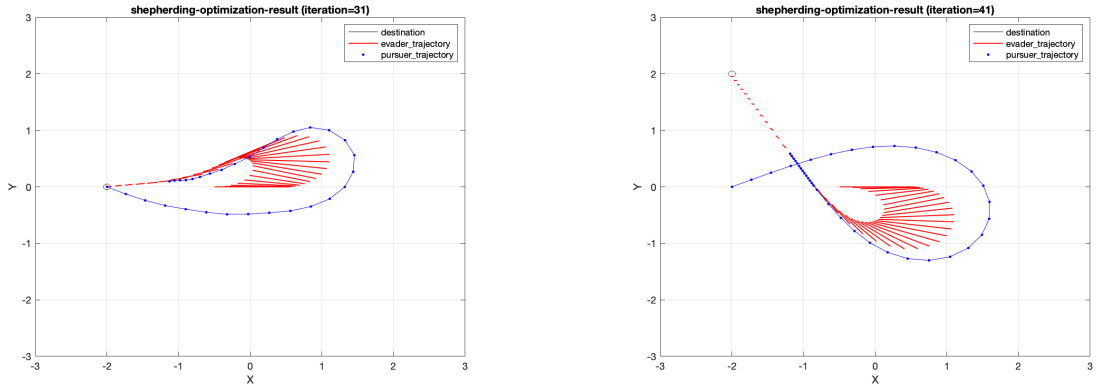


(e) $z = (0, -2)$



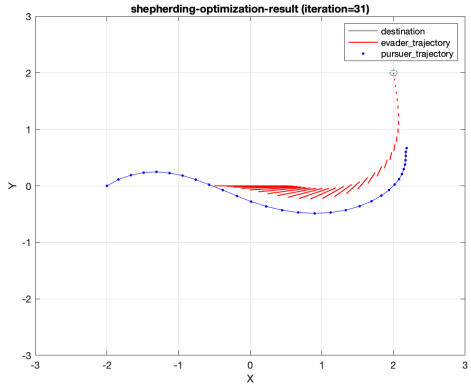
(f) $z = (0, 0)$

Figure 4.2: In these plots, initial pursuer position is $(0, -2)$ and destination points are varied along 2×2 square arena centered at origin

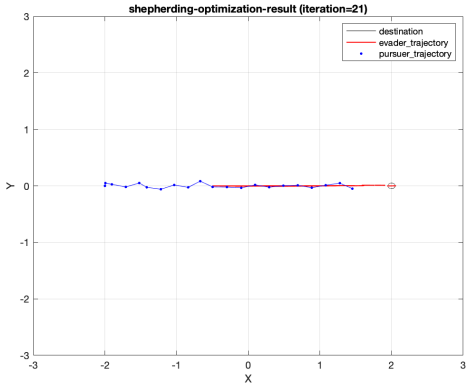


(a) $z = (-2, 0)$

(b) $z = (-2, 2)$



(c) $z = (2, 2)$



(d) $z = (2, 0)$

Figure 4.3: In these plots, initial pursuer position is $(-2, 0)$ and destination points are varied along 2×2 square arena centered at origin

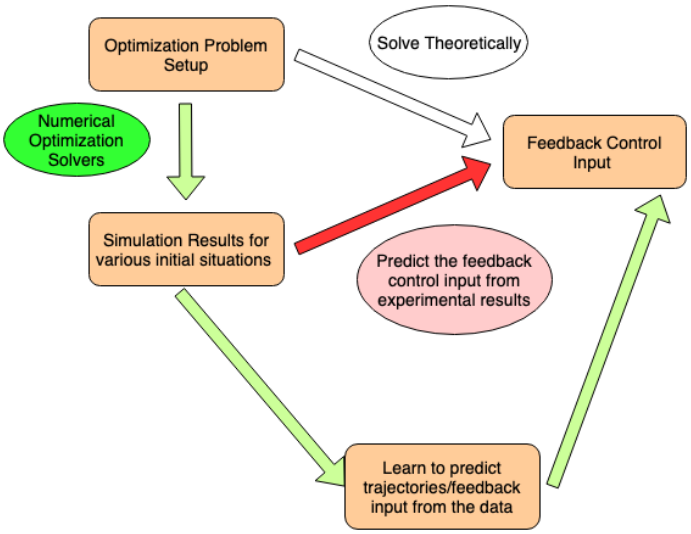


Figure 4.4: Problem Flowgraph

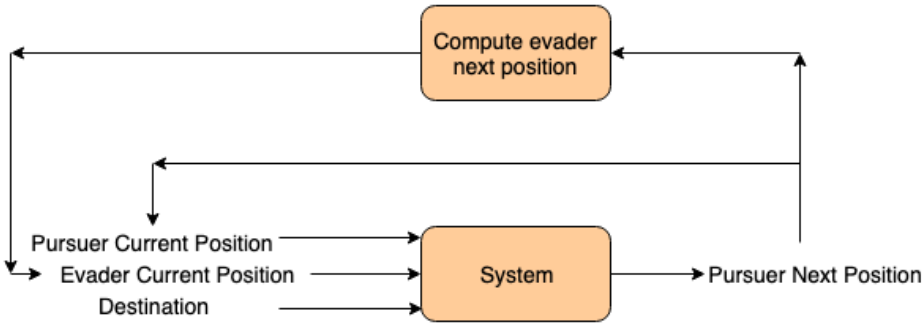


Figure 4.5: System level feedback design

CHAPTER 5

GREEDY APPROACH

5.1 Introduction

This approach is based on the greedy algorithm which is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum. It is based on the ideology of taking a decision which is best for present situation without considering the future and hoping that all the present-based decisions would shape the best future as well. In our problem, the task is to drive N_e evaders from their initial position to within an ϵ radius of destination point along with minimizing pursuer trajectory length. It is an iterative algorithm where at each step the pursuer decides to move in a direction which minimizes a combination of evader-destination distance and evader-evader distance. The pursuer speed is assumed to be constant throughout and at each iteration, only pursuer velocity direction is estimated. This intuition has been formulated in the cost function below.

$$J(t) = \alpha \sum_{i=1}^{N_e} d(e_i, z, t+1) + \beta \sum_{i=1}^{N_e-1} \sum_{j=i+1}^{N_e} d(e_i, e_j, t+1) \tag{5.1}$$

where α and β are weights in order to decide the trade-off or importance of the two set of summation terms.

$$\underset{\hat{p}(t)}{\text{minimize}}\ J(t) \tag{5.2}$$

The entire algorithm can be described in below mentioned steps:

1. The pursuer estimates the direction $\theta \in [0, 2\pi]$ in which it should move with fixed speed such that the cost function $J(t)$ is minimum among all possible directions.
   (a) Among all possible directions at each step i, pursuer and evaders position at next time step $(i+1)$ is computed.
   (b) The cost function value is then computed using updated position of evaders.
   (c) The above two steps are repeated for all directions $\theta \in [0, 2\pi]$.

(d) $\hat{\theta}$ denotes the angle (direction) which corresponds to the minimum cost function.

2. The pursuer moves in the estimated direction $\hat{\theta}$ and based on that evaders next position is updated.

3. Repeat step 1 to 3 until all the evaders are within an ϵ radius of destination point z or the number of iterations exceeds maximum number of iterations.

$$d(e_i, z, t) \leq \epsilon, \quad \forall \quad i = 1, \ldots, N_e, \quad \text{for some } t$$

5.2 Experiments and Results

The experiments have been performed for $N_e = 2$. In these experiments, the interaction of an evader is only repulsive due to the pursuer and the repulsive evader velocity relation is the same as given in Eq 3.6 and 3.7.

$$\dot{e}_i(t) = \dot{e}_i^{repl\_p}(t), \quad \forall \quad i = 1, \ldots, N_e \tag{5.3}$$

The greedy approach requires a lot of tweaking of parameters such as α, β, etc. on case-to-case basis depending on initial conditions. Empirically, it has been observed that the overall cost function decreases till certain number of iterations and then saturates to some finite non-zero value instead of zero. The cost function has been designed to provide the desired result such that all the evaders are near destination as well as the inter-evader separation is as small as possible. In the initial stage of the trajectory, the path taken by the pursuer behaves in a similar manner to the optimization results shown in Fig 4.1, 4.2, and 4.3, but after certain iterations the pursuer deviates completely from its path. This kind of behaviour has been observed in most of the cases and therefore it concludes that this approach fails in terms of results.

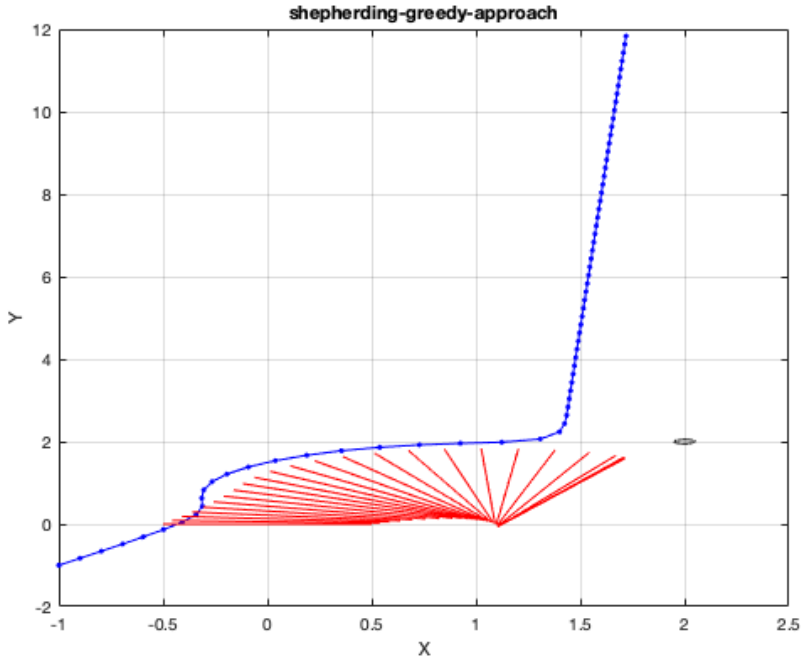


Figure 5.1: The blue curve represents pursuer trajectory while the red line represents line joining two evaders at its endpoint. Initial positions of pursuer is $(-1, -1)$ while that of evaders is $(-0.5, 0)$ and $(0.5, 0)$ with destination being at $(2, 2)$.

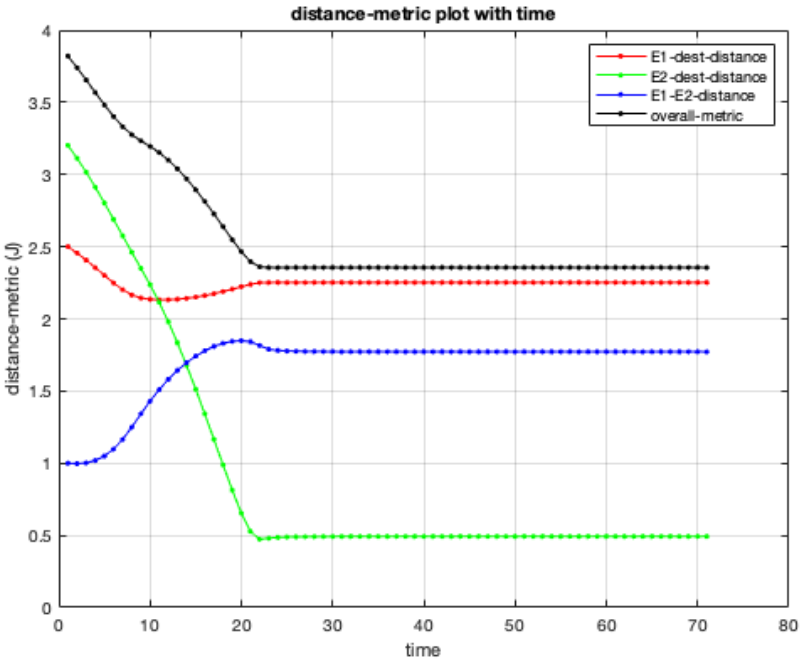


Figure 5.2: Figure showing the variation of evader-destination distance and evader-evader distance with time. The black curve represents the cost function which decreases and then saturates at some non-zero value

CHAPTER 6
LEARNING TRAJECTORIES USING DEEP LEARNING

6.1 Introduction

We tried solving the optimization problem proposed in chapter 4 in a theoretical manner but because of differential and complicated coupled constraints as well as large number of variables in optimization space, it has not been possible for us till now. We obtained the optimization results by solving the constrained optimization problem using numerical solvers. As shown in Fig 4.4, the next and final step in this problem is to predict system's feedback input or some sort of iterative algorithm which can predict future position of agents given current and past positions. Earlier, we tried to develop various intuitions such as dipole alignment, plotted various parameters in order to find some pattern in the optimization results. But this approach did not worked out. Another possible way is to learn pursuer trajectory from the results obtained (using the results as data) and then predict the feedback control input. Several machine learning techniques have been used in the past for trajectory tracking. The objective is to learn to predict pursuer's next position based on current and past trajectory information iteratively until the objective is achieved.

The first step in this process is to determine the model which can learn spatial and temporal correlations present in any trajectory. Recently, deep learning has been successfully applied to solve a wide spectrum of challenging problems. Deep learning models are able to extract complex features from the training data. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs) have performed excellently in numerous engineering and research problems. We use LSTMs followed by fully connected layers (FCNs) for predicting trajectory as they have an inherent ability to learn spatial as well as temporal patterns making them suitable for this task. We give a brief overview of LSTMs below.

6.2 Long Short-Term Memory (LSTMs) Networks

LSTMs are special kind of RNNs designed to remember and learn the long-term time dependency of data. Here we briefly describe LSTM architecture [14]. Key to their structure is the cell state to which

information can be added by the cells through gates. Gates are sigmoid neural network layers which optionally let the information pass through and are of three types:

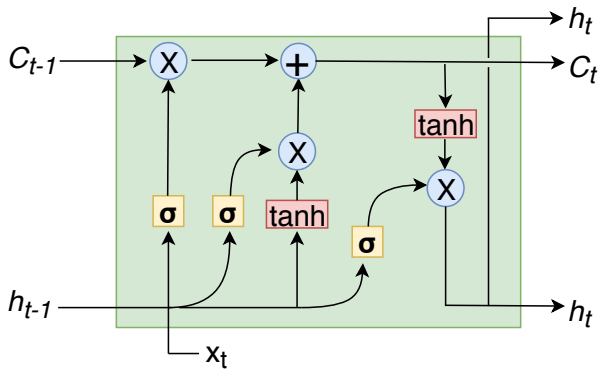


Figure 6.1: Representation of an LSTM Cell [14]. C_t represents the cell state at time step t, h_t represents the cell output at time t and σ represents sigmoid layer.

1. *Forget Gate*: A sigmoid layer is used to decide what information we are going to throw away from the current cell state. The output of this layer is multiplied with cell state to delete the information we do not want to proceed further.

2. *Input Gate*: This consists of a tanh layer which creates a new vector to be added to the current state and a sigmoid layer to select which values to add in particular. The tanh and sigmoid layers are multiplied and then added to the current cell state.

3. *Output Gate*: The current cell state is run through the tanh layer and multiplied with a sigmoid layer to output only the desired parts.

6.3 Machine Learning Architecture

We assume that the trajectory of any agent does not undergo rapid changes and thus has a smooth trajectory. At any time t, the next position of any agent should depend on the present and past positions. Therefore, the trajectory of any agent can be modelled as time-series sequence. We use LSTMs for time series modelling of trajectory as LSTMs have performed well, especially with the classification of time-sequence and text sequence [15, 16]. LSTMs are known to learn the long-term dependency as well as temporal and spatial features of input data.

We formulate our problem as a regression task. As shown in Fig 6.2, our deep learning architecture consists of a single layer of LSTM each with cell state size of 64. At the end of LSTM layer, we get

a 64-dimensional features. Since our desired output (next pursuer position) is 2-dimensional, we compress gradually from 64-dimensions to 2-dimensions using FCNs with hidden layers of sizes 32 and 8. Rectified Linear Unit (ReLU) activation function has been used in hidden layers to introduce non-linearity in the model.

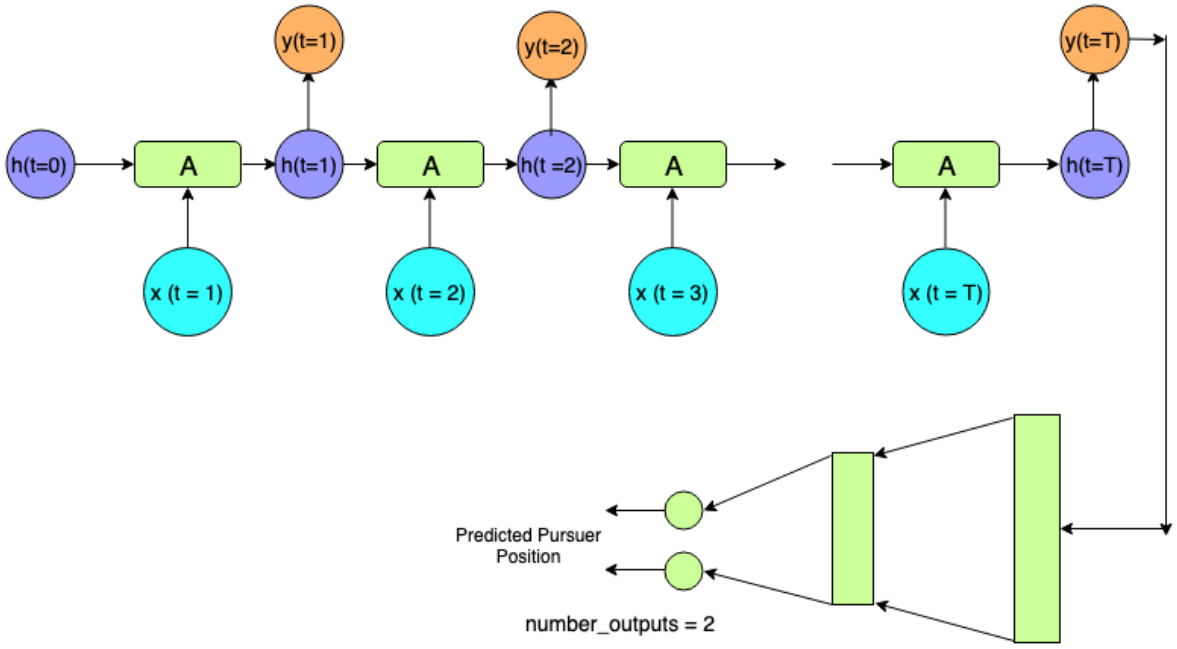


Figure 6.2: Model Architecture

6.4 Dataset and Training

The results obtained from the optimization setup for numerous initial conditions served as data for training the model. The dataset has been divided into $4 : 1$ ratio for training and testing purpose respectively. Each instance of data sample is an 8-dimensional feature vector with each element in the following order : $[p(t), e_1(t), e_2(t), z]$ where each of the entity is in R^2. For each simulation result with number of time steps N, it produced $(N - W)$ data samples with each one of size $W \times 8$ where W represents LSTM time window size.

The entire neural network architecture has been implemented in tensorflow. The input to the LSTM is of dimension $[N - W \times W \times 8]$. The network parameters are initialized using Xavier's initialization [17]. LSTM time window size represents the past time steps of the trajectory which the model would use to learn the past dependency. It was empirically chosen to be 5 steps. Mean-squared

loss function and Adam Optimizer [18] is used to train the model based on computed loss. Network gradients were clipped to a maximum of 10 as it led to better and stable training. For completeness, we provide a list of all hyperparameter values after final tuning in Table 6.1. We trained the model on google colab (Tesla K80 GPU, 12 GB RAM) which takes around 5 hours for 5000 training epochs.

Hyperparameter	**Typical Values**
LSTM time window size (W)	5
Maximum gradient norm	10
Learning Rate	10^{-3}
LSTM cell state size	64
Dropout parameter	0.8

Table 6.1: List of network hyperparameter values

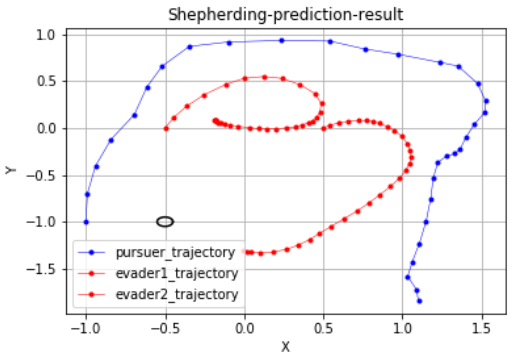
6.5 Result Evaluation

The learned network outputs the pursuer trajectory in an iterative manner and based on that the evader trajectory can be computed using Eq 3.6 and 3.7. As previously mentioned, the evader positions in the training as well as test data were kept fixed at $(-0.5, 0)$ and $(0.5, 0)$. Some of the test results are shown below in Fig 6.3.

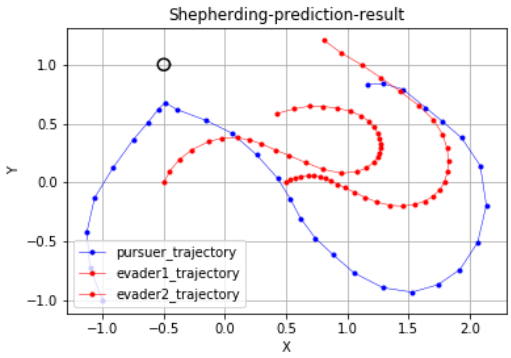
6.6 Limitation and Conclusion of this approach

The ML-based approach to predict the feedback control input (pursuer's velocity or future position) gave promising results, if not the best we would have desired. Certain reasons account for the limitation of this approach. One possible reason might be because of insufficient data as the number of plots used for creating dataset was around 150 which created approximately 6000 sample points. Given the number of parameters and complexity of the model, the data might have been insufficient for training. Secondly, the training dataset does not cover all possible set of initial conditions due to the vast nature of plots one can generate as well as due to the time-consuming nature of optimization simulations. Thirdly, the model might need to be tuned properly with respect to certain hyperparameters such as cell state size, number and units of hidden layers, dropout parameter, etc, for better performance.

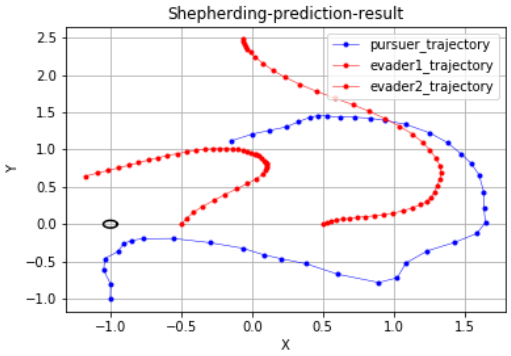
Despite of all the limitations, this approach certainly provides some crucial insights and positivity. Even with limited and small data, the model is able to learn the fact that the trajectories of agents are



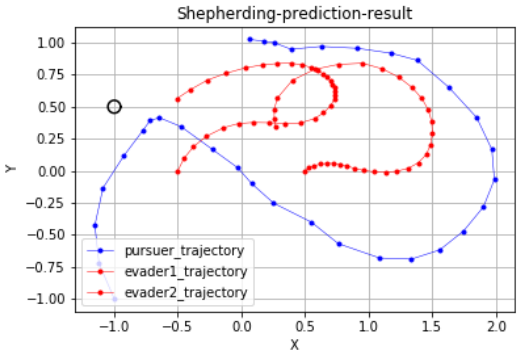
(a) $z = (-0.5, -1)$



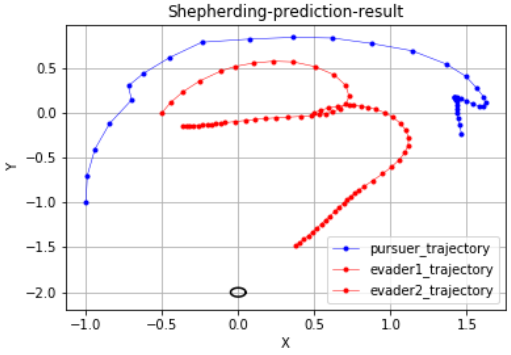
(b) $z = (-0.5, 1)$



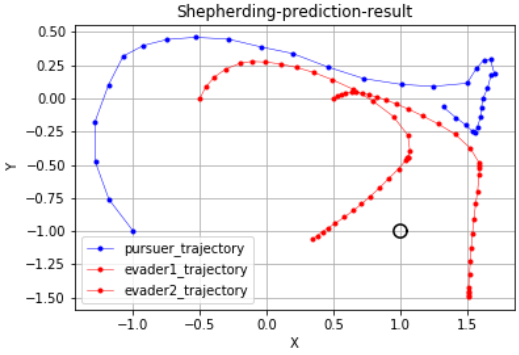
(c) $z = (-1, 0)$



(d) $z = (-1, 0.5)$



(e) $z = (-0.5, -1)$



(f) $z = (1, -1)$

Figure 6.3: Some prediction results from learned model. In these plots, initial evader position is $(-0.5, 0)$ and $(0.5, 0)$. Initial pursuer position is $(-1, -1)$ while destination points are varied

smooth, which evader the pursuer should approach first and pursuer speed range. Though, it failed to learn that the evaders have to be driven towards the destination as there was no explicit way in which the concept of destination and its information could have been fed to the network.

SUMMARY

In this report, we proposed an interaction rule between an evader and a pursuer and our objective was to try to find an optimal feedback control for the pursuer to drive the evaders to destination. With this regard, we first formulated our problem as a constrained optimization problem and solved using global search algorithm available in global optimization toolbox of matlab. The result from these experiments were then used to predict a feedback control algorithm but unfortunately this could not be made possible. Then we shifted from predicting ourselves to let the machine learn from the data and predict the trajectory for us. We used LSTM-based model with fully connected layers and posed the problem as a regression task to produce pursuer next position given current and past trajectory information of all the agents. The experimental results from the optimization task was used as dataset for this approach. After training, the trajectories were estimated iteratively for numerous initial conditions but we could not get the desired result. This approach requires modifications in order for it to work.

REFERENCES

[1] W. D. Hamilton, "Geometry for the selfish herd.," *Journal of theoretical biology*, vol. 31 2, pp. 295–311, 1971.

[2] L Coppinger and R Coppinger, *Dogs for herding and guarding livestock*. In Livestock handling and transport, 2000, pp. 235–254.

[3] F. Darling, *A herd of red deer*. Oxford University Press, 1937.

[4] D. Zhao, B. Yegenmammedov, P. Liu, and M. Zhang, "Comparative study on occupant evacuation with building exodus and a cellular automaton model," *Open Journal of Safety Science and Technology*, vol. 7, no. 1, 2017.

[5] R. L. Hughes, "The flow of human crowds," *Annual Review of Fluid Mechanics*, vol. 35, no. 1, pp. 169–182, 2003. eprint: `https://doi.org/10.1146/annurev.fluid.35.101101.161136`.

[6] M. Fingas, *The basics of oil-spill cleanup*. Lewis, 2001.

[7] A. C. Schultz and W. Adams, "Continuous localization using evidence grids," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 4, 1998, 2833–2839 vol.4.

[8] A. E. Turgut, H. Çelikkanat, F. Gökçe, and E. Şahin, "Self-organized flocking in mobile robot swarms," *Swarm Intelligence*, vol. 2, no. 2, pp. 97–120, 2008.

[9] A. Özdemir, M. Gauci, and R. Groß, "Shepherding with robots that do not compute," in *ECAL*, 2017.

[10] D. Strombom, R. Mann, A. Wilson, S. Hailes, J. Morton, D. Sumpter, and A. King, "Solving the shepherding problem: Heuristics for herding autonomous, interacting agents," in *Artificial Evolution*, J. R. Soc. Interface, 1998.

[11] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents part i: Dynamic topology," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, vol. 2, 2003, 2016–2021 Vol.2.

[12] *Basin of attraction*. eprint: `https://in.mathworks.com/help/gads/what-is-global-optimization.html#bsbalkx-1`.

[13] F. Glover, "A template for scatter search and path relinking," in *Artificial Evolution*, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Eds., Springer Berlin Heidelberg, 1998, pp. 1–51, ISBN: 978-3-540-69698-8.

[14] C. Olah, *Understanding LSTM networks*, 2015.

[15] Y. Zhang, Q. Liu, and L. Song, "Sentence-state LSTM for text representation," *CoRR*, vol. abs/1805.02474, 2018. arXiv: 1805.02474.

[16] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," *CoRR*, vol. abs/1605.05101, 2016. arXiv: 1605.05101.

[17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the* 13^{th} *International Conference on Artificial Intelligence and Statistics*, vol. 9, PMLR, 2010, pp. 249–256.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. arXiv: 1412.6980.