ENPM690 - Robot Learning

Final Project Report

Decentralised Multi-Robot Collision Avoidance via Deep Reinforcement Learning

Koyal Bhartia (116350990)

Date : 05/11/2020

Contents

1 Abstract

The multi-robot system is attracting attention at an exponential rate, having several advantages and applicability much beyond the scope of a single-robot system. However, with advantages comes challenges. Multi robot systems face several challenges in terms of communication, coordination and navigation. Thus developing a safe and efficient collision avoidance policy especially for a clustered or crowded environment is still a matter of concern, recent research and interesting investigation.

In this project I have focused on the problem of developing a decentralized sensor-level collision avoidance policy, for a warehouse like scenario. This directly maps raw sensor measurements to the agent's steering commands in terms of movement velocity. Centralised and decentralised methods both have their own advantages and disadvantages. The proposed approach bridges this performance gap by adapting a centralized learning, decentralized execution. This also performs better than other decentralised methods which depends on the extraction of agent-level features to plan a local collision-free action, which can be computationally prohibitive and not robust. Moreover a multi-scenario multi-stage training framework is implemented to learn the optimal policy.

The policy is trained over large number of robots in complex environments simultaneously using a policy gradient based reinforcement learning algorithm, PPO.The policy is further validated in a variety of simulated scenarios with thorough performance evaluations showing that the learned policy is able to find time efficient, collision-free paths for a large-scale robot system. The learned policy also generalizes to new scenarios. Some videos of the training and testing process can be found at

2 Introduction

Multi-robot navigation has gained a lot of interest in robotics and artificial intelligence due to its enormous real world applications including search and rescue, military and aerospace, navigation through human crowds, and autonomous warehouse. Multi robots possesses the characteristics of parallelism, redundancy, robustness, increased coverage and throughput, flexible reconfigurability and spatially diverse functionality. Amongst several other challenges, **one of the major challenges here is to develop a safe and robust collision avoidance policy for each robot navigating** from its starting position to its desired goal, enabling the robots to perform operations efficiently in terms of time and working space with advantages and applications comes challenges such as multi-robot coordination which can Some of prior works, known as centralized methods, assume that the comprehensive knowledge about all agents intents (e.g. initial states and goals) and their workspace (e.g. a 2D grid map) is given **for a central server to control the action of agents. These methods can generate the collision avoidance action by planning optimal paths.**

This problem if solved will result into great optimization in warehouse scenario and will lead to higher number of bills per hour. Also we can see its application in last mile delivery application. In this project a novel multi-scenario multi-stage training framework which exploits a robust policy gradient based reinforcement learning algorithm trained in a large-scale robot system in a set of complex environments. We demonstrate that the collision avoidance policy learned from the proposed method is able to find time efficient, collision-free paths for a large-scale nonholonomic robot system, and it can be well generalized to unseen scenarios. Its performance is also much better than previous decentralized methods, and can serve as a first step toward reducing the gap between centralized and decentralized navigation policies.

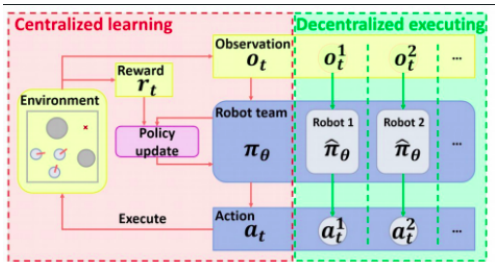


Figure 1: Centralised Learning - Decentralised Execution

3 Background/Related Work

Learning-based collision avoidance techniques have been extensively studied for one robot avoiding static obstacles. Many approaches adopt the supervised learning paradigm to train a collision avoidance policy by imitating a dataset of sensor input and motion commands.Trained a vision-based static obstacle avoidance system in supervised mode for a mobile robot by training a 6-layer convolutional network which maps raw input images to steering angles exploited a successor-feature-based deep reinforcement learning algorithm to transfer depth information.

The quad-rotor was able to successfully avoid collisions with static obstacles in the environment using only a single cheap camera. Yet only discrete movement (left/right) has to be learned and also the robot only trained within static obstacles. **Note that the aforementioned approaches only take into account the static obstacles and require a human driver to collect training data in a wide variety of environments.** Another data-driven end to-end motion planner. This model can navigate the robot through a previously unseen environment and successfully react to sudden changes. **Nonetheless, similar to the other supervised learning methods, the performance of the learned policy is seriously constrained by the quality of the labeled training sets.**

To overcome this limitation, solution of a mapless motion planner trained through a deep reinforcement learning method was proposed which is known as the Optimal Reciprocal Collision Avoidance (ORCA) framework has been popular in crowd simulation and multi-agent systems. ORCA provides a sufficient condition for multiple robots to avoid collisions with each other in a short time horizon and can easily be scaled to handle large systems with many robots. ORCA and its extensions used heuristics or first principles to construct a complex model for the collision avoidance policy, which has many parameters that are tedious and difficult to be tuned properly. Besides, these methods are sensitive to the uncertainties ubiquitous in the real world scenarios since they assume each robot to have perfect sensing about the surrounding agents' positions, velocities and shapes. To alleviate the requirement of perfect sensing, **Moreover, the original formulation of ORCA is based on holonomic robots which is less common than nonholonomic robots in the real world.** To deploy ORCA on the most common differential drive robots, several methods have been proposed to deal with the difficulty of non-holonomic robot kinematics. ORCADD enlarges the robot to twice the radius of the original size to ensure collision free and smooth paths for robots under differential constraints. However, this enlarged virtual size of the robot can result in problems in narrow passages or unstructured environments. NH-ORCA makes a differential drive robot tracking a holonomic speed vector with a certain tracking error ϵ. In this method, we focus on learning a collision-avoidance policy which can make multiple non-holonomic mobile robots navigate to their goal positions without collisions .

4 Approach

4.1 Overview

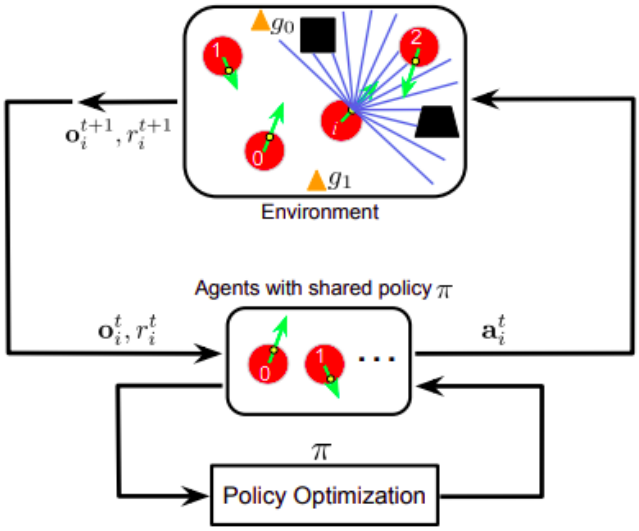


Figure 2: Overview of approach

The above image displays an overview of the complete approach. As illustrated, at each timestep t, each robot receives its **observation** o_i^t and **reward** r_i^t from the environment, and generates an **action** a_i^t following the **policy** π**.** The policy π is shared across all robots and updated by the on-policy **policy gradient** based reinforcement learning algorithm used i.e PPO.

This is similar to how a Reinforcement Learning setup works by having an AI agent interact with our environment. The agent observes the current **state** of our environment, and based on some **policy** makes the decision to take a particular **action**. This action is then relayed back to the environment which moves forward by one **step**. This generates a **reward** which indicates whether the action taken was positive or negative in the context of the game being played. Using this reward as a feedback, the agent tries to figure out how to modify its existing policy in order to obtain better rewards in the future.

4.2 Reinforcement Learning Setup

In the real world, majority of the mobile robots are nonholonomic differential drive; thus I have defined the multi robot collision avoidance problem primarily in the context of a nonholonomic differential drive robot moving on the Euclidean plane with obstacles and other decision-making robots. The above problem can be defined as partially observable sequential decision making which can be formulated formulated as a Partially Observable Markov Decision Process (POMDP). A POMDP can be described as a 6-tuple (S, A,P, R, ω, O), where S is the state space, A is the action space, P is the state-transition model, R is the reward function, Ω is the observation space ($o \in \Omega$) and O is the observation probability distribution given the system state ($o = O(s)$).

In the project each robot has access to the observation sampled from the underlying system states. **Note:** Since each robot plans its motions in a fully decentralized manner, a multirobot state-transition model P determined by the robots' kinematics and dynamics is not needed.

4.2.1 Observation Space : $o^t = [o_z^t, o_g^t, o_g^t]$

The observation space consists of the following readings:

1. o_z^t : This consists of the readings of the 2D laser range finder. It includes the measurements of the last three consecutive frames from a 180-degree laser scanner which has a maximum range of 4 meters and provides 512 distance values per scanning (i.e. $o_z^t \in R3512$). The scanner is mounted on the forepart of the robot.
2. o_g^t : This is a 2D vector which represents the relative goal in polar coordinate (distance and angle) with respect to the robot's current position.
3. o_g^t : This observation includes the current translational and rotational velocity of the differential-driven robot. The observations are normalized by subtracting the mean and dividing by the standard deviation using the statistics aggregated over the course of the entire training.

4.2.2 Action space : $a^t = [v^t, w^t]$

The action space is a set of permissible velocities in continuous space. The action of differential robot includes the translational and rotational velocity, i.e. $at = [v^t, w^t]$. Considering the real robot's kinematics and the real world applications, the following range has been set:

- Translational Velocity: $v \in (0.0, 1.0)$.
- Rotational Velocity: $v \in (-1.0, 1.0)$.

Note: Backward moving (i.e. $v < 0.0$) is not allowed since the laser range finder can not cover the back area of the robot.

4.2.3 Reward design

As the objective of the project is multi-robot collision avoidance during navigation and minimize the mean arrival time of all robots, the reward function is designed to to achieve the same.

$$r_i^t = (g_r)_i^t + (c_r)_i^t + (w_r)_i^t \tag{1}$$

The reward r received by robot i at timestep t is a sum of three terms in Eqn[1]. The robot is awarded by ($g_r)_i^t$ for reaching its goal, When the robot collides with other robots or obstacles in the environment, it is penalized by $g_r)_i^t$. To encourage the robot to move smoothly, a small penalty $(w_r)_i^t$ is introduced to punish the large rotational velocities. Following is an illustration of the reward functions.

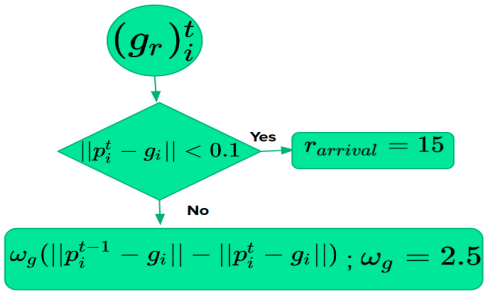
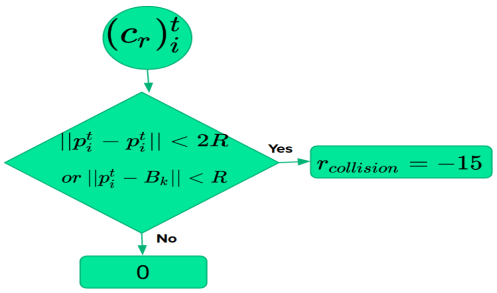
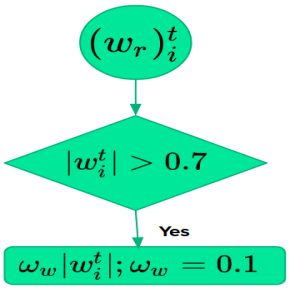


Figure 3: Reward for reaching goal

Here p_i is the location of the robot; and g_i is the goal location. Thus the robot gets a negative reward if it goes away from the goal, a positive reward for going towards the goal, and a comparatively



(a) Penalization for collision



(b) Penalty on large rotations

Figure 4: Penalties on collision and rotation

Considering the robot radius as R, the robot is penalised for collision if it either collides with another robot i.e p_i^t or collides with an obstacle. The robot is punished with a penalty of -1 to punish for large rotational velocities.

5 Implementation

5.1 Integration with ROS

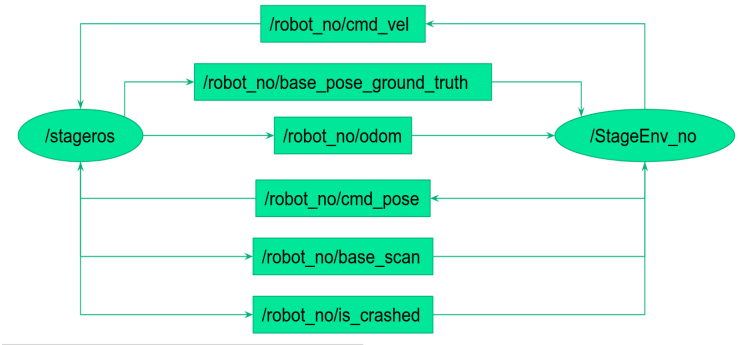


Figure 5: Integration with ROS

The implementation has been done on Stage robot simulator which is done using ROS topics and messages to send command velocities to the robots and receive the robot's current status.

5.2 Network Architecture

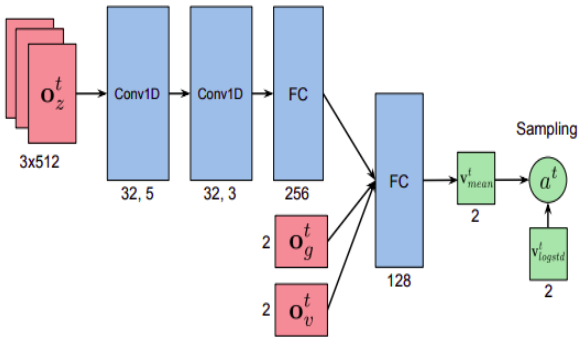


Figure 6: Neural Network Architecture

A 4-hidden-layer neural network is designed as a non-linear function approximator to the policy π_θ. The given input observation as explained earlier is o_i^t and the output action is v_i^t . The architecture has been shown above and details are below.

- The first three hidden layers is emploed to process the laser measurements o_z^t effectively.
- The first hidden layer convolves 32 one-dimensional filters with kernel size = 5, stride = 2 over the three input scans and applies ReLU nonlinearities.
- The second hidden layer convolves 32 one-dimensional filters with kernel size = 3, stride = 2, again followed by ReLU nonlinearities.
- The third hidden layer is a fully-connected layer with 256 rectifier units.
- The output of the third layer is concatenated with the other two inputs ($o_g^t and o_v^t$), and then are fed into the last hidden layer, a fully-connected layer with 128 rectifier units.
- The output layer is a fully-connected layer with two different activations: a sigmoid function is used to constrained the mean of translational velocity v^t in (0.0, 1.0) and the mean of rotational velocity w^t in (1.0, 1.0) through a hyperbolic tangent function (tanh).

Overall, the neural network maps the input observation vector o^t to a vector v^t mean. The final action a t is sampled from a Gaussian distribution N (v^t mean, v_{logstd}^t), where v_t mean serves as the mean and v_{logstd}^t) refers to a log standard deviation which will be updated solely during training.

5.3 The PPO algorithm used for training

5.3.1 My understanding of Proximal Policy Optimization (PPO)

PPO also known as "on-policy learning" approach is where typically the experience samples collected are only useful for updating the current policy once.

PPO basically involves the following steps:

- Collecting a small batch of experiences interacting with the environment.
- Using this batch to update its decision-making policy.
- Once the policy is updated, the experiences are thrown away and a newer batch is collected with the newly updated policy.
- PPO mainly takes care that a new update of the policy does not change it too much from the previous policy.

The above behavior of PPO leads to less variance in training at the cost of some bias, but ensures smoother training and also makes sure the agent does not go down an unrecoverable path of taking senseless actions.

Further Actor Critic has been used which changes the policy update feature. The Actor model performs the task of learning what action to take under a particular observed state of the environment. The critic model takes the reward to evaluate if the action taken by the Actor led the environment to be in a better state or not and give its feedback to the Actor. It outputs a number indicating a rating of the action taken in the previous state. By comparing this rating obtained from the Critic, the Actor can compare its current policy with a new policy and decide how it wants to improve itself to take better actions.

Moving forward is the calculation of Advantage using GAE.

5.3.2 Generalised Advantage Estimation - GAE

Advantage is a way to measure how much better off we can be by taking a particular action when we are in a particular state. Advantage is calculated by using the rewards that is collected at each time step for the action that was taken. Thus we calculate how much better off we are by taking any action, not only in the short run but also over a longer period of time. This way, even if we do not immediately get reward in the next time step, we still look at few time steps after that action into the longer future to see if we can get rewarded.

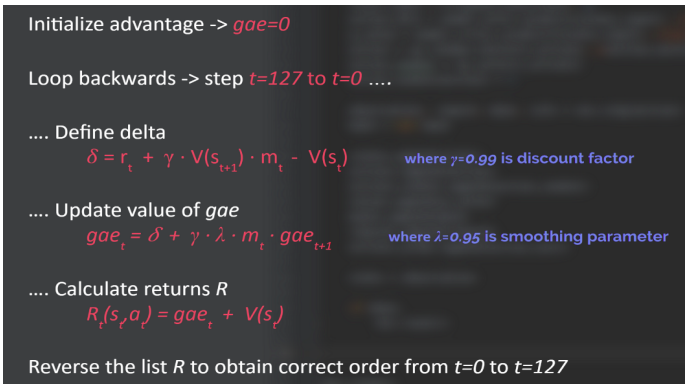


Figure 7: Implementation of GAE

The above is a representation of how I have updated the value of GAE and calculated the expected reward using these parameters:

Gamma (γ): Gamma is basically the **discount factor** in order to reduce the value of the future state since we want to emphasize more on the current state than a future state. The value of γ **used in the project is 0.99**

Lambda (λ): is a **smoothing parameter** used for reducing the variance in training which makes it more stable. The value of λ used in the project is 0.95. Hence, this gives us the advantage of taking an action both in the short term and in the long term.

```
new_value, new_logprob, dist_entropy = policy.evaluate_actions(sampled_obs, sampled_goals, sampled_speeds, sampled_actions)

sampled_logprobs = sampled_logprobs.view(-1, 1)
ratio = torch.exp(new_logprob - sampled_logprobs)

sampled_advs = sampled_advs.view(-1, 1)
surrogate1 = ratio * sampled_advs
surrogate2 = torch.clamp(ratio, 1 - clip_value, 1 + clip_value) * sampled_advs
policy_loss = -torch.min(surrogate1, surrogate2).mean()

sampled_targets = sampled_targets.view(-1, 1)
value_loss = F.mse_loss(new_value, sampled_targets)

loss = policy_loss + 20 * value_loss - coeff_entropy * dist_entropy
optimizer.zero_grad()
loss.backward()
optimizer.step()
info_p_loss, info_v_loss, info_entropy = float(policy_loss.detach().cpu().numpy()), \
                                         float(value_loss.detach().cpu().numpy()), float(
                                        dist_entropy.detach().cpu().numpy())
logger_ppo.info('{}, {}, {}'.format(info_p_loss, info_v_loss, info_entropy))
```

Figure 8: Calculation of PPO Loss

Above is the code snippet of the calculation of the loss. The Actor loss in Fig[8] refers to the policy loss in the code. On similar lines, the Critic loss in Fig[**??**] refers to the value loss in the code.

The above code is a part of the ppo update stage code present in the file /model/ppo.py which can be found in teh code folder provided as the github link.

The following points can be further noted:

- The values of the coefficient entropy and the clip value taken are 0.2 each.
- As seen PPO uses a ratio between the newly updated policy and old policy in the update step, representated in the log from.
- This ratio lets us decide how much of a change in policy we are willing to tolerate. Hence, we use a clipping parameter epsilon to ensure we only make the maximum of % change to our policy at a time. Thus the value of epsilon used is 0.2
- The critic loss or teh value loss is the usual mean squared error loss with the Returns.
- The actor (policy loss) and critic (value loss) is combined using a discount factor using the basic equation as shown below:

$$loss = policy_loss + value_loss - entropy \tag{2}$$

The entropy term is optional, but it encourages the actor model to explore different policies and the

degree to which we want to experiment can be controlled by an entropy beta parameter. The entropy factor used in the code is 0.2

5.3.3 KL Divergence

In my understanding KL Divergence basically helps us to measure just how much information we lose when we choose an approximation. It is a way of measuring the matching between two distributions. In this project we measure the divergence between the old policy and the new updated policy. Below is the general formula of finding the divergence between 2 distributions, explanation of which is above the scope of the project.

$$D_{KL}(p||q) = \Sigma_{i=1}^{N} p(x_i) \log(\frac{p(x_i)}{q(x_i)}) \tag{3}$$

Here q(x) is the approximation and p(x) is the true distribution, that we wish to match to q(x). Thus this measures how much a given arbitrary distribution is away from the true distribution. Thus **Lower the KL divergence value, the better the true distribution is matched to the approximation.**

The algorithm and steps has been shown as part of the appendix. The explanation is also included there. The parallel PPO algorithm can be easily scaled to a largescale multi-robot system with hundred robots in a decentralized fashion since each robot in the team is an independent worker collecting data. The decentralized execution not only dramatically reduce the time of sample collection, also make the algorithm suitable for training many robots in various scenarios.

6 Results

6.1 Hyperparamters

Following are the hyper parameters that have been used for sampling of the trah6+99

- **MAXEPISODES = 5000** A large no to have a successful training process to get good rewards and loss functions.
- **LASERHIST = 3** Three consecutive frames of laser data taken by the robots is feeded into the neural network.
- **HORIZON = 128** These are the number of robot trajectories that are sampled brefore the next policy update can be made.
- **GAMMA = 0.99**
- **LAMDA = 0.95**
- **BATCHSIZE = 512** As mentioned earlier the 1st parameter in the observation space: otz includes the measurements of the last three consecutive frames from a 180-degree laser scanner which has a maximum range of 4 meters and provides 512 distance values per scanning (i.e. $o_z^t \in R^{3*512}$)
- **EPOCH = 4** The policy is updated foe 4 epochs during the policy update stage.
- **COEFFENTROPY = 5e-4**

- **CLIPVALUE = 0.1**
- **NUMENV = 44** These are the number of robots which are used for either training or testing. Hence a different object is made for each robot to aid in parallel processing.
- **LEARNINGRATE = 5e-5**

6.2 Multi-scenario Multi-stage training

6.2.1 Multi Scenario Training

To expose the robots to diverse environments, different scenarios have been created with a variety of obstacles using the Stage mobile robot simulator and all the robots move concurrently.

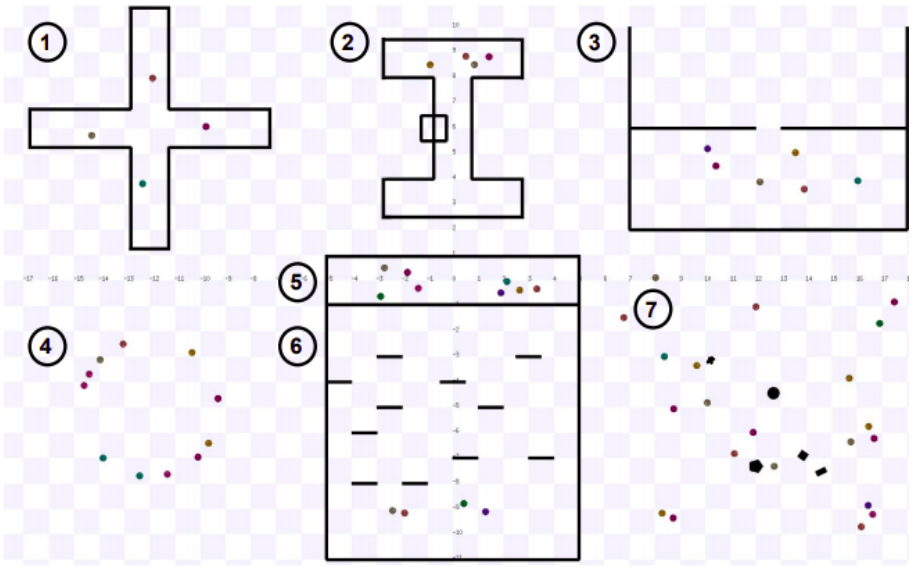


Figure 9: 7 training scenarios

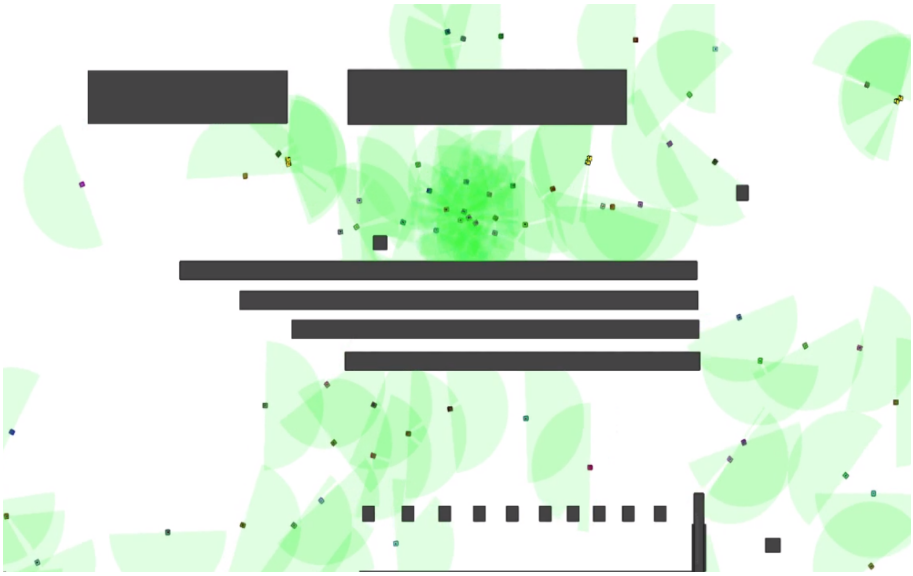


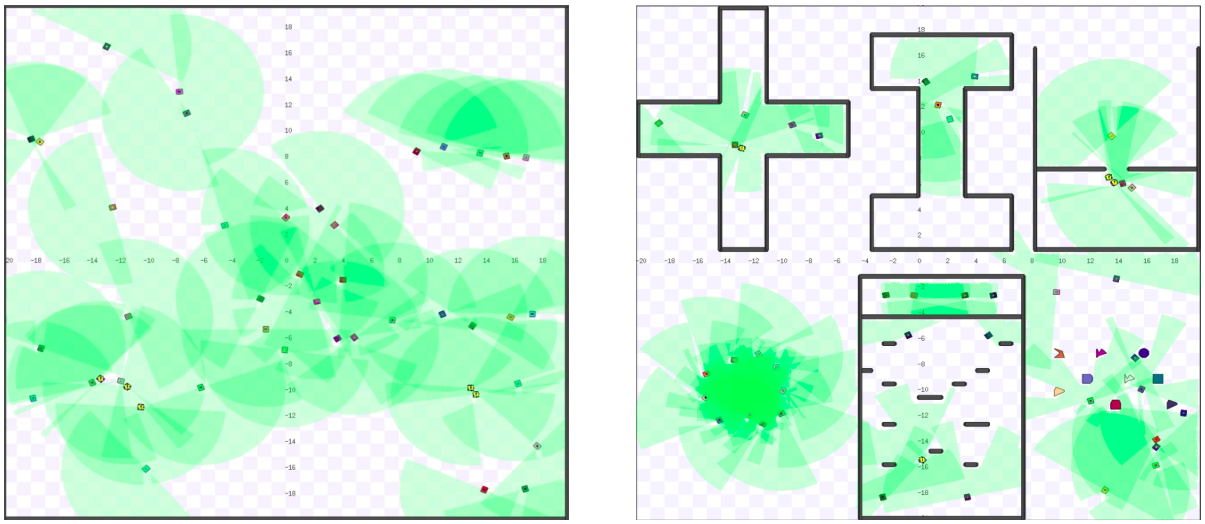
Figure 10: Large Scale Warehouse Like Scenario

In figure 9 (black solid lines are obstacles), in scenario 1, 2, 3, 5, and 6 reasonable starting and arrival areas from the available workspace is selected, and then the start and goal positions is randonly sampled for each robot. Robots in scenario 4 are randomly initialized in a circle with a varied radius, and they aim to reach their antipodal positions by crossing the central area. As for scenario 7, random positions is generated for both robots and obstacles at the beginning of each episode; and the target positions of robots are also randomly selected.

For better scalability I also trained and tested the robots on huge warehouse like scenarios as shown in Fig[10]. They showed positive test and training results as can be viewed in the videos as well as some in some images in the appendix. These rich, complex training scenarios enable robots to explore their high-dimensional observation space and are likely to improve the quality and robustness of the learned policy.

6.2.2 Multi Stage Training

Although training on multiple environments simultaneously brings robust performance over different test cases, it makes the training process harder. Here a two-stage training process has been proposed which accelerates the policy to converge to a satisfying solution, and gets higher rewards than the policy trained from scratch with the same number of epoch. In the first stage, robots are trained on the random scenarios without any obstacles, this allows our robots learn fast on relatively simple collision avoidance tasks. Once the robots achieve reliable performance, we stop the Stage 1 and save the trained policy. The policy continues to get updated in the Stage2.



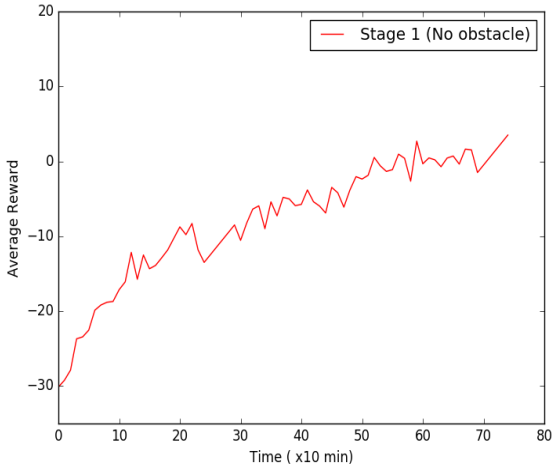
(a) Stage 1 training (without obstacle)

(b) Stage 2 training (with obstacle)

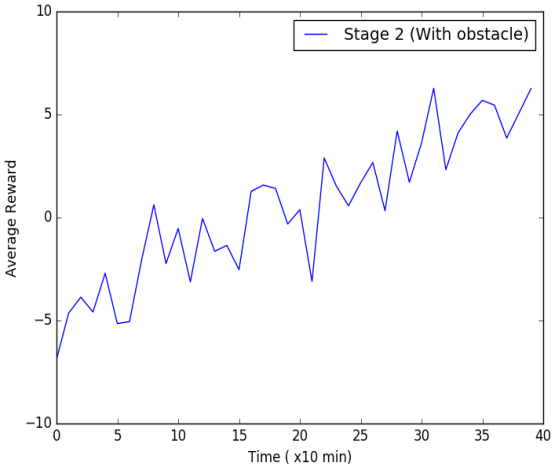
Figure 11: Multi Stage Learning

Following are the graphs interpretations that were obtained as after the several training on the multi scenario multi stage.

Following are the loss and reward graphs obtained after each Stage of training.

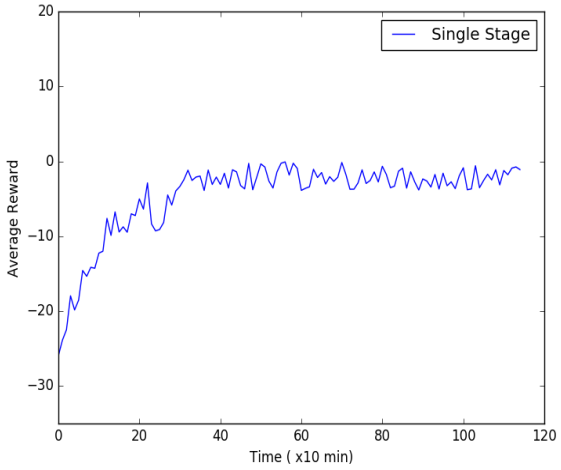


(a) Reward Graph - Stage 1(without obstacle)

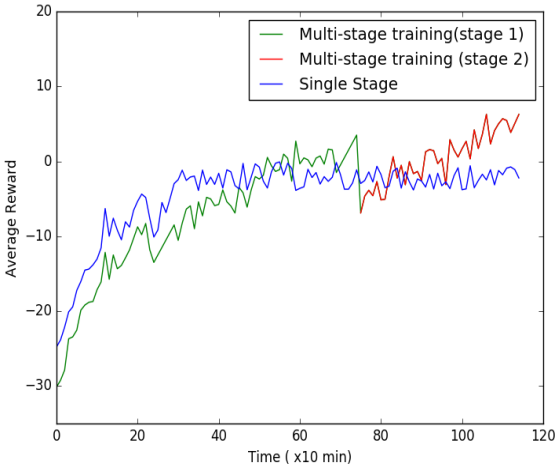


(b) Reward Graph - Stage 2(with obstacle)

Figure 12: Reward Graph - Multi Stage Training

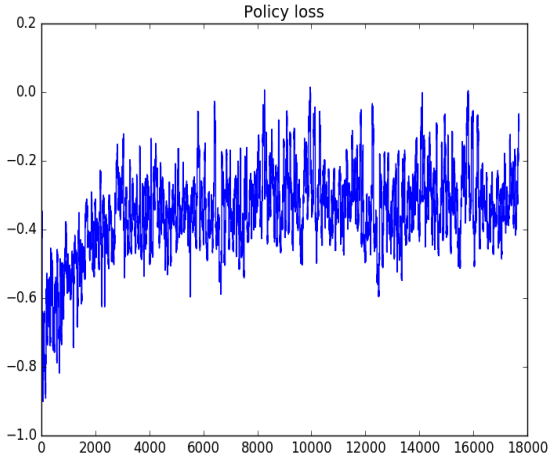


(a) Reward Graph - Single Stage Training

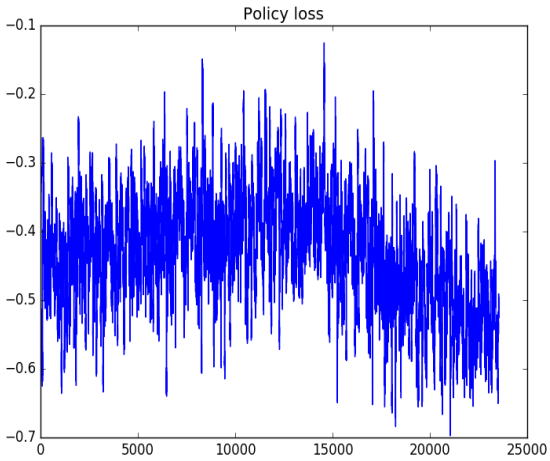


(b) Comparison of Single vs Multi-stage Training

Figure 13: Comparison of Reward

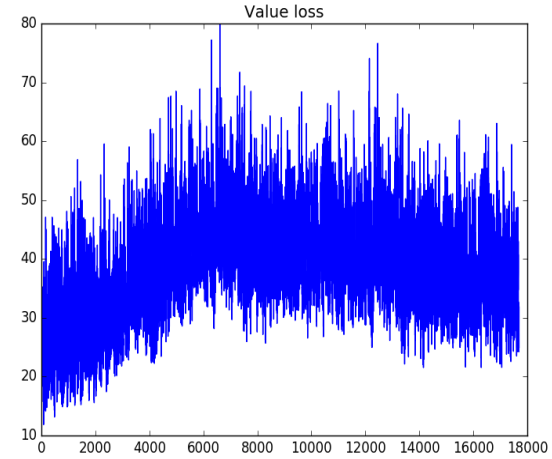


(a) Policy Loss Graph - Stage 1

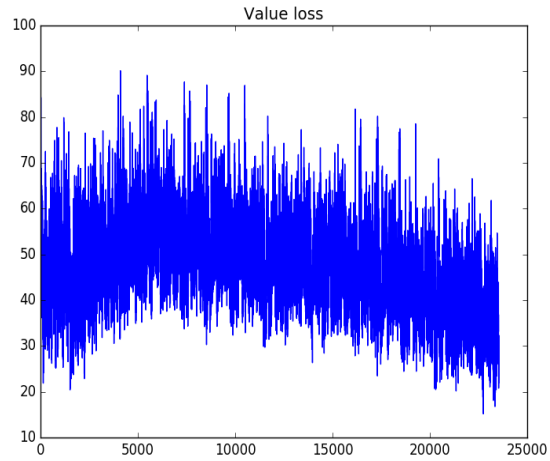


(b) Policy Loss Graph - Stage 2

Figure 14: Policy Loss

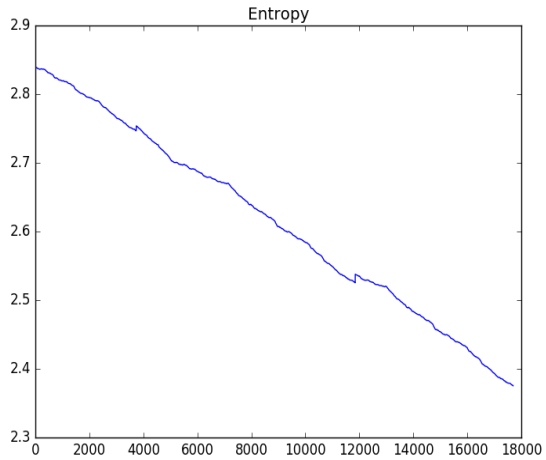


(a) Value Loss Graph - Stage 1

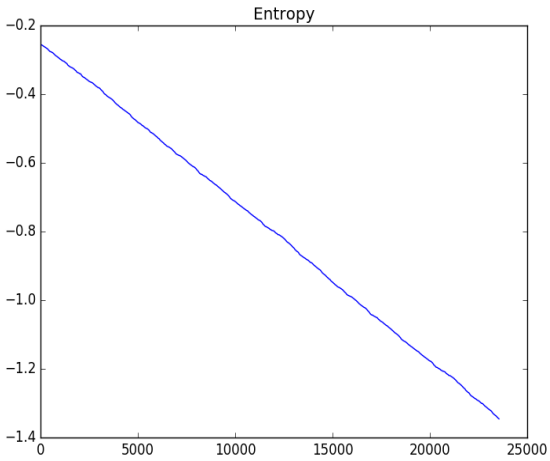


(b) Value Loss Graph - Stage 2

Figure 15: Value Loss



(a) Entropy - Stage 1



(b) Entropy - Stage 2

Figure 16: Entropy

1. **Cumulative Reward** — The mean cumulative episode reward over all agents. As can be evaluated these have increased during a successful training. Several small ups and downs can also be noticed in the graphs. As seen the average reward is better with a multi stage learning than single stage.

2. **Entropy** — The entropy signifies how random the decisions of the model are. As can be seen these have slowly decreased after a successful training process. Due to a good value of beta hyperparameter, the entropy hasn't shown signs of any sudden decrease. Hence there has been a consistent decrease. This decreases as the exploration decreases.

3. **Policy Loss** — The mean magnitude of policy loss function. Correlates to how much the policy (process for deciding actions) is changing. The magnitude of this should decrease during a successful training session. These values will oscillate during training. Generally they should be less than 1.0.

4. **Value Loss** — The mean loss of the value function update. Correlates to how well the model is able to predict the value of each state. This should increase while the agent is learning, and then decrease once the reward stabilizes. These values will increase as the reward increases, and then should decrease once reward becomes stable.

7 Conclusions

In this project, we present a multi-scenario multi-stage training framework to optimize a fully decentralized sensor level collision avoidance policy with a robust policy gradient algorithm. Also PPO has served a great paradigm for learning which uses Neural network in actor-critic sense. This project can serve as a first step towards reducing the navigation performance gap between the centralized and decentralized methods, though we are fully aware that the learned policy focusing on local collision avoidance cannot replace a global path planner when scheduling many robots to navigate through complex environments with dense obstacles.

So an Hybrid merge of classic method and decentralized scenario will always provide a better result in terms of unknown environment and difficult environment.

8 Future Work

The problem of Scalable Multi-agent path planning model is a very interesting and crucial problem and it is possible and necessary to obtain better results. There are several areas of challenges that I faced in the project which I wish to solve .

1. Better RL techniques such as PPO2 and TRPO can be used to generate better policy update criteria.
2. The problem of robot freezing in corner cases needs work and analysis.
3. A turn around solution for robot deadlock situation where any 2 or more robots wait for the other to move, hence they all permanently freeze.
4. The environment can be made more crowded with robots in the range of hundreds.
5. The implemented policy can be compared to other policies such as ORCA, DWA etc.
6. The trajectories of the robots can be traced out for a better analysis of the path taken.

9 Bibliography

[1] https://nervanasystems.github.io/coach/components/agents/policyoptimization/ppo.html

[2] https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf

[3] https://www.freecodecamp.org/news/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d/

[4] https://medium.com/aureliantactics/ppo-hyperparameters-and-ranges-6fc2d29bccbe

[5] https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-understanding-kl-divergence-2b382ca2b2a8

[6] https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6

[7] https://link.springer.com/article/10.1007/s10514-018-9726-5

[8] https://web.stanford.edu/ schwager/MyPapers/AlonsoMoraEtAlAURO18DistOptimizationFormationControl.pd

[9] https://www.ntu.edu.sg/home/yangliu/publications/journal/tsmc.pdf

[10] https://arxiv.org/pdf/1709.10082.pdf

10 Appendix

The github repo containing the robot controller codes and the videos can be found in the below link.

`https://github.com/koyalbhartia/Decentralised-Multi-Robot-Collision-Avoidance`

10.1 PPO Policy Update Algorithm

for iteration $= 1, 2, ...,$ **do**
  // Collect data in parallel
  for robot $i = 1, 2, ...N$ **do**
    Run policy π_θ for T_i timesteps, collecting $\{\mathbf{o}_i^t, r_i^t, \mathbf{a}_i^t\}$, where $t \in [0, T_i]$
    Estimate advantages using GAE [26] $\hat{A}_i^t = \sum_{l=0}^{T_i} (\gamma\lambda)^l \delta_i^t$, where $\delta_i^t = r_i^t + \gamma V_\phi(s_i^{t+1}) - V_\phi(s_i^t)$
    break, if $\sum_{i=1}^{N} T_i > T_{max}$
  end for
  $\pi_{old} \leftarrow \pi_\theta$
  // Update policy
  for $j = 1, ..., E_\pi$ **do**
    $L^{PPO}(\theta) = \sum_{t=1}^{T_{max}} \frac{\pi_\theta(a_i^t|o_i^t)}{\pi_{old}(a_i^t|o_i^t)} \hat{A}_i^t - \beta \mathrm{KL}[\pi_{old} \mid \pi_\theta] + \xi \max(0, \mathrm{KL}[\pi_{old} \mid \pi_\theta] - 2\mathrm{KL}_{target})^2$
    if $\mathrm{KL}[\pi_{old} \mid \pi_\theta] > 4\mathrm{KL}_{target}$ **then**
      break and continue with next iteration $i + 1$
    end if
    Update θ with lr_θ by Adam [25] w.r.t $L^{PPO}(\theta)$
  end for
  // Update value function
  for $k = 1, ..., E_V$ **do**
    $L^V(\phi) = -\sum_{i=1}^{N} \sum_{t=1}^{T_i} (\sum_{t'>t} \gamma^{t'-t} r_i^{t'} - V_\phi(s_i^t))^2$
    Update ϕ with lr_ϕ by Adam w.r.t $L^V(\phi)$
  end for
  // Adapt KL Penalty Coefficient
  if $\mathrm{KL}[\pi_{old} \mid \pi_\theta] > \beta_{high}\mathrm{KL}_{target}$ **then**
    $\beta \leftarrow \alpha\beta$
  else if $\mathrm{KL}[\pi_{old} \mid \pi_\theta] < \beta_{low}\mathrm{KL}_{target}$ **then**
    $\beta \leftarrow \beta/\alpha$
  end if
end for

Figure 17: PPO Algorithm

- During data collection, each robot exploits the same policy to generate trajectories until they collect a batch of data above T_{max}.
- Then the sampled trajectories are used to construct the surrogate loss $L^{PPO}()$, and this loss is optimized with the Adam optimizer for E_π epochs under the Kullback-Leiber (KL) divergence constraint.
- The state-value function $V\phi(s_i^t)$, used as an baseline to estimate the advantage $\hat{A}t_i$, is also approximated with a neural network with parameters ϕ on sampled trajectories.
- The network structure of V_{phi} is the same as that of the policy network π_θ, except that it has only one unit in its last layer with a linear activation.
- We construct the squared-error loss $L^V()$ for V_ϕ, and optimize it also with the Adam optimizer for E_v epochs.
- If the KL divergence seems to go high, the penalty parameter beta is increased, else decreased.

10.2 Different Testing Scenarios

These are also shown in videos in the github repo.

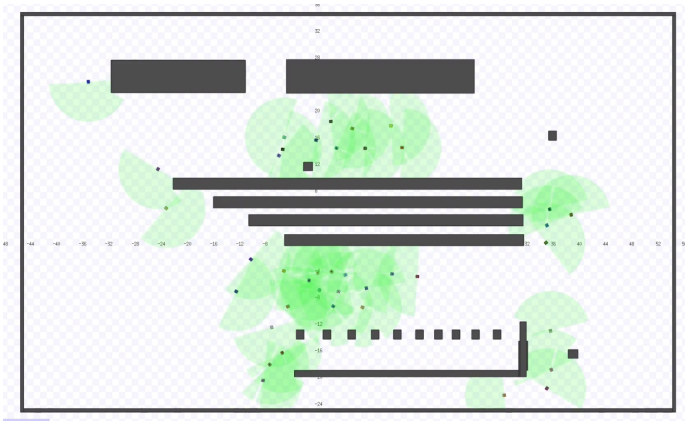


Figure 18: Large System - 44 robots

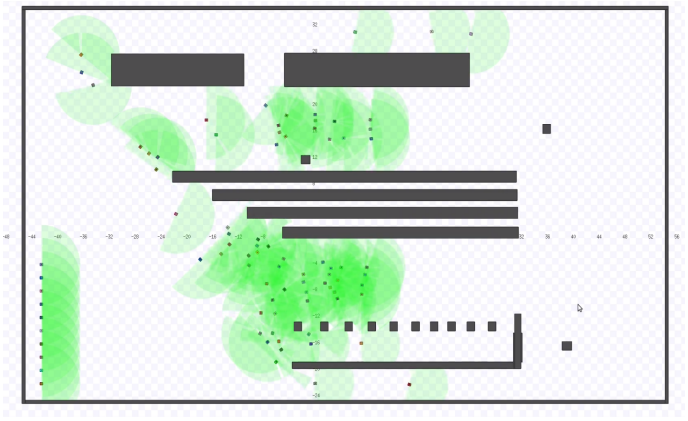


Figure 19: Large System - 70 robots

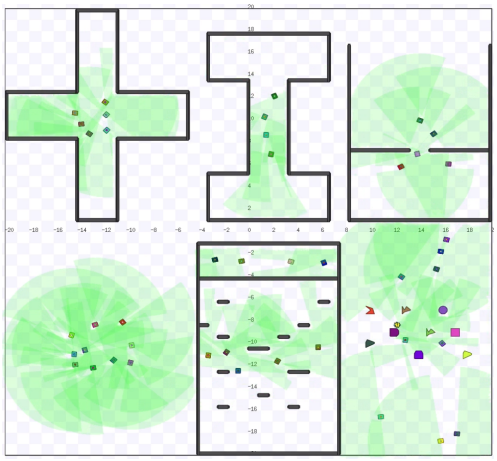


Figure 20: Small Scenario - 44 robots

10.3 Failed Training Loss Graphs

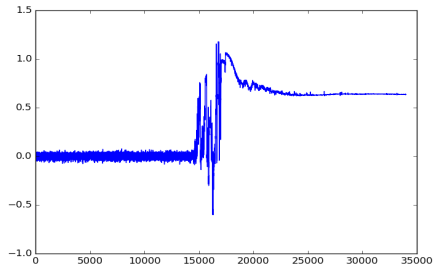


Figure 21: failed policy loss

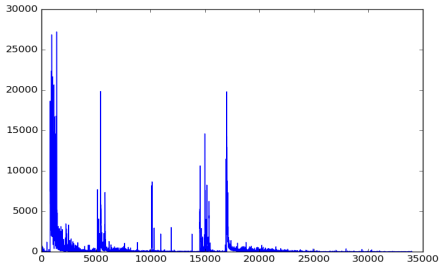


Figure 22: failed value loss

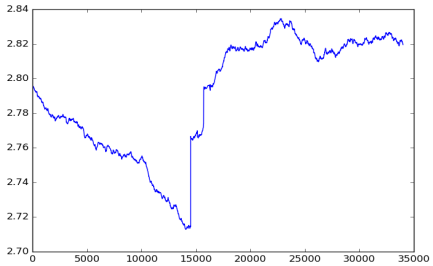


Figure 23: failed entropy

Special Thanks to :

1. **Prof.Donald Sofge ,**

2. **TA - Adheesh Chatterjee.**