Robot Flocking Control with Reinforcement Learning and Gibbs Distributions

Myles Foy

Department of Mathematical Sciences, Durham University

Introduction

Multi-robot systems provide numerous advantages over single-robot counterparts in various applications such as environmental surveillance and search and rescue. Flocking behaviour mimics flocking in birds to quickly and safely move a robot swarm from one area to the next. We demonstrate a method for controlling swarm flocking using Reinforcement Learning, where we train each robot to take safe actions given observations of its environment and communication with nearby robots.

Undirected Graphical Models

Each robot is modelled as a node in a graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where edges between nodes indicate that robots are close enough to influence one another. For robot $i \in \mathcal{V}$ we call this set of close robots the **neighbourhood** of i, $\mathcal{N}_i = \{j \in \mathcal{V} : (i,j) \in \mathcal{E}\}$.

Each robot is assigned a random variable called its **state**, $\boldsymbol{X}_i$, containing the robot's position, velocity, relative position to neighbouring robots, distance from obstacles, and the previous action distribution of its neighbours.

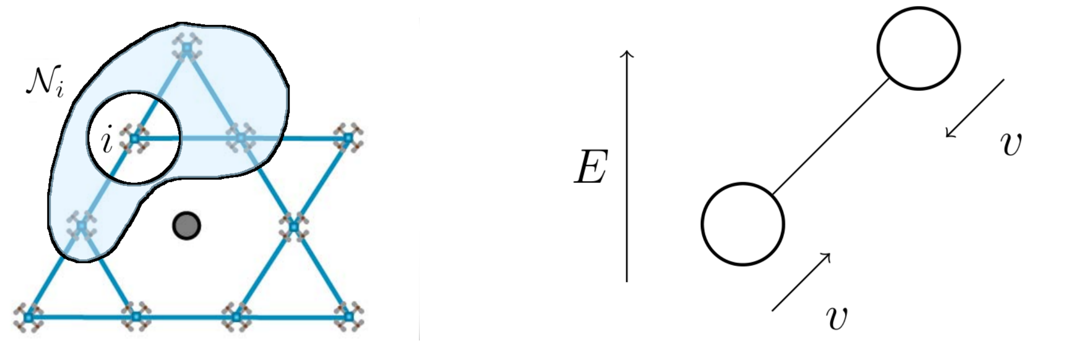

Figure 1:Robots as nodes in a graph, with edges representing influence, and an obstacle in the centre [1]. On the right, two robots are about to collide (undesirable), so an energy function related to their velocities would increase.

We model how robots i and j interact by introducing **energy functions** $\psi(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R}$ that determine how "compatible" their states are. Greater values of ψ indicate incompatibility, while lesser values indicate compatibility. This is a convention from statistical physics.

We can define energy functions over any **clique**, $c \in \mathcal{C}$, of our graph. A clique is a set of nodes that are all connected. In practice, we often use pairwise energy functions, as above, as they are typically more intuitive to specify.

Gibbs Distributions

By combining local energy functions, we can represent the swarm's global behaviour as a **Gibbs Distribution**:

$$P(\boldsymbol{x}) = \frac{1}{Z}\prod_{c\in\mathcal{C}} \exp(-\psi_c(\boldsymbol{x_c})) = \frac{1}{Z}\exp\left(-\sum_{c\in\mathcal{C}}\psi_c(\boldsymbol{x_c})\right)$$

where

$$Z = \int_{\mathcal{X}} \prod_{c\in\mathcal{C}} \exp(-\psi_c(\boldsymbol{x_c}))\mathrm{d}\boldsymbol{x}$$

is a normalisation constant. The exponential transformation of the energy is needed to make sure $P(\boldsymbol{x})$ is always positive; the negation ensures probability increases as energy decreases and vice versa.

Energy Functions

In our model, the total energy, $\sum_{c\in\mathcal{C}}\psi(\boldsymbol{x}_c) = \frac{|\mathcal{V}|}{|\mathcal{E}|}\sum_{(i,j)\in\mathcal{E}}\psi_p(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{i\in\mathcal{V}}\psi_u(\boldsymbol{x}_i)$, is the scaled sum of the pairwise energies of each robot, plus the unary energies of each robot, which each define a specific type of behaviour. We normalise the pairwise energy to account for the dynamic number of edges in the graph.

Example: Distance Energy Function

Robots must remain safely separated to prevent crashes. We define

$$\psi_d(\boldsymbol{x}_i, \boldsymbol{x}_j) = c_{p1}[1 - \exp(-c_{p2}(d_{ij} - d_r))]^2 - c_{p1}$$

where d_r is the desired distance between the two robots, d_{ij} is their actual distance, and c_{p1}, c_{p2} are coefficients to control how much influence robot distance has on the total energy. ψ_d is at a minimum when $d_{ij} = d_r$, and ψ_d takes larger values when the robots are too close rather than too far apart.

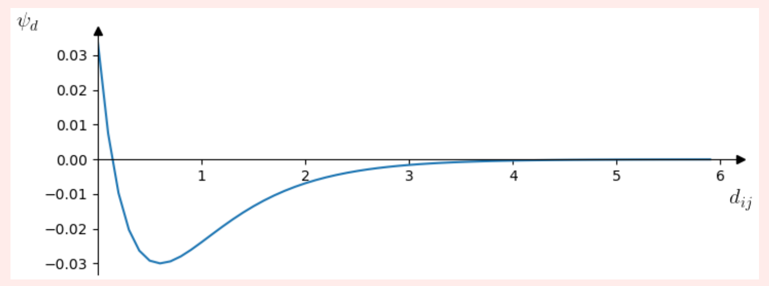

Figure 2:Plot of the distance energy with $d_r = 0.6, c_{p1} = 0.03, c_{p2} = 1.5$.

Remember that lower energy means higher probability, so we are constructing our Gibbs distribution such that the robots have a high probability of maintaining their desired separation.

Reinforcement Learning

Reinforcement Learning is a type of machine learning where an agent interacts with its environment. The agent begins in some state, s, then takes an action, a, which leads to the agent being in a new state, s', and receiving a reward, $r \in \mathbb{R}$.

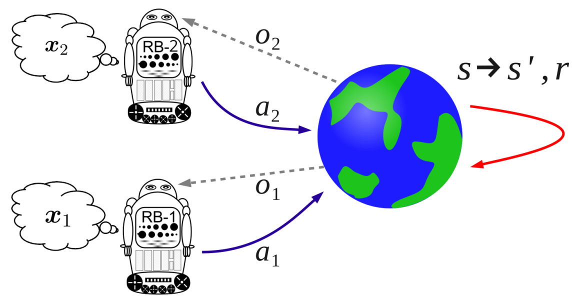

Figure 3:Robots take individual actions and make individual noisy observations of the state, which they use to form a belief state [2].

We have multiple robot agents, which receive noisy **observations**, o_i, of the state, due to sensor limitations. They maintain a **belief state** which approximates the true state. This is $\boldsymbol{x}_i$, the random variable in our graphical model.

Our goal is to specify a **policy**, $\pi_i(a|\boldsymbol{x}_i)$, for each robot, which determines the action to take in a given belief state, in order to maximise the expected **total reward**, the expectation of the sum of all rewards from this time step into the future. We then train this policy using simulations of the decision process.

Constructing Reward

We want our reward to be high when robots are in a desirable configuration, and low when in an undesirable one. Recall that we constructed our Gibbs distribution to have exactly these properties. We use our unnormalised probability distribution as the reward function, since Z is intractable, and rewards do not need to be normalised.

$$r = \exp\left(-\frac{|\mathcal{V}|}{|\mathcal{E}|}\sum_{(i,j)\in\mathcal{E}}\psi_p(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i\in\mathcal{V}}\psi_u(\boldsymbol{x}_i)\right) \propto P(\boldsymbol{x}).$$

Computing this reward requires calculating energies for all robots, not just neighbours. We introduce the decentralised pairwise energy

$$\hat{H}_p = \sum_{i\in\mathcal{V}}\frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}\psi_p(\boldsymbol{x}_i, \boldsymbol{x}_j).$$

It can be shown $\hat{H}_p$ has the same minimum value and condition as the total pairwise energy. Intuitively, minimising the global energy requires all neighbourhood energies to be minimised, hence we define decentralised rewards

$$r_i = \exp\left(-\frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}\psi_p(\boldsymbol{x}_i, \boldsymbol{x}_j) - \psi_u(\boldsymbol{x}_i)\right).$$

Policy

Our policy function is a neural network that uses attention layers trained to pay attention to dangerous possible neighbour actions. The policy anticipates such actions and takes action accordingly, allowing for collision avoidance.

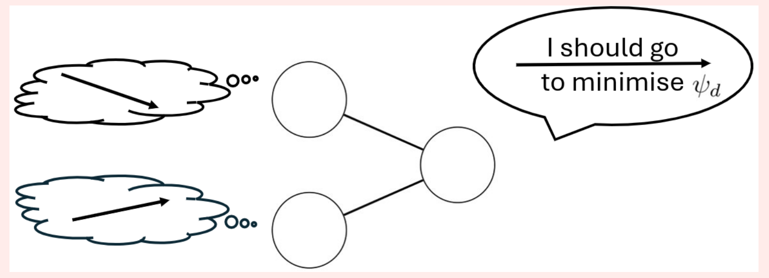

Figure 4:Danger is avoided to maximise reward (minimise energy)

Summary and Future Work

By embedding our goals into a graphical model and constructing a policy that selects actions to meet these goals, our robots can flock autonomously and effectively.
The policy will be trained using proximal policy optimisation, a gradient-based method developed by OpenAI. I will be studying this method, then writing code to simulate drone flocking in a congested environment.

References

[1] Feng Xue Qingrui Zhang Dengyu Zhang, Chenghao Yu. Learning efficient flocking control based on gibbs random fields. *IEEE*, 2025.

[2] Christopher Amato Frans A. Oliehoek. *A Concise Introduction to Decentralised POMDPs*. Springer, 1 edition, 2016.