Formation Control using GQ(λ) Reinforcement Learning

by M. Knopp, C. Aykın, J. Feldmaier and H. Shen

Tahsincan Köse
2188423

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

Introduction

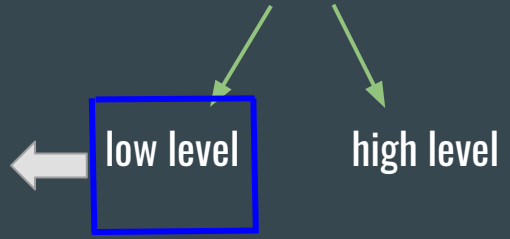
Difficult to implement on low-cost hardware

Problem: Maintaining a specific shape during locomotion.

- Current solutions in the literature are model, potential-field or graph based approaches which yield non-linear control laws.
- Most RL algorithms rely on linear functions which are computationally less complex compared to aforementioned control laws.

Solution: Using RL for Formation Control.

This paper investigates the applicability of the proposed solution on the low level side.



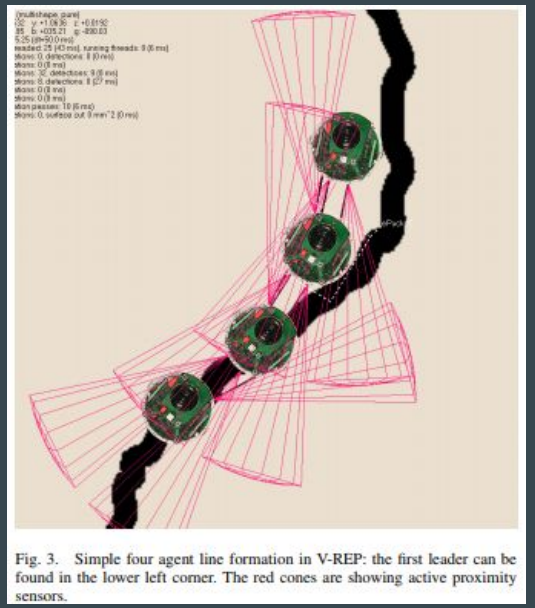


Fig. 3. Simple four agent line formation in V-REP: the first leader can be found in the lower left corner. The red cones are showing active proximity sensors.

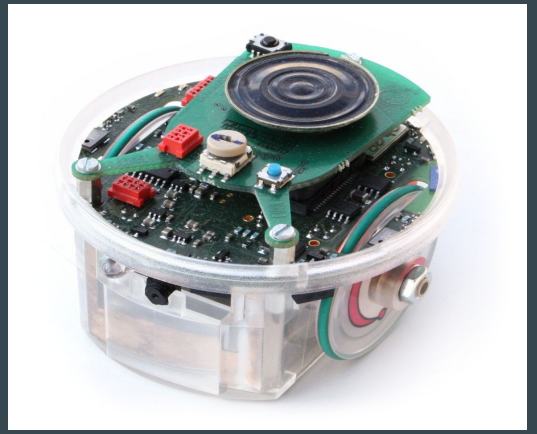
Scenario and Tools

Maximum range approximately 15 cm

Scenario: Simple leader-follower formation which uses proximity sensors as input and motor speeds as actions.

Tools:

- GQ(λ) ➡ state of art reinforcement learning algorithm.
- ε-greedy behavior policy ➡ balances exploration and exploitation.
- E-puck robots ➡ open-source tabletop robots.
- Gumstix Overo COM ➡ Runs embedded Linux and is able to connect to wireless. Used to implement learning algorithms directly on e-puck via Python API.
- V-REP ➡ Open-source robot simulation framework.



ε-greedy GQ(λ) Learning Algorithm

- $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$; target policy to be learned. Incidently, if π is chosen as the greedy policy with respect to the learned value function, then the algorithm will implement a generalization of the Greedy-GQ algorithm (Maei, Szepesvari, Bhatnagar & Sutton 2010).
- $\gamma : \mathcal{S} \to [0,1]$; termination or discounting function ($\gamma(s) = 1 - \beta(s)$ in GQ paper)
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$; reward function

Question Functions

- $b : \mathcal{S} \times \mathcal{A} \to [0,1]$; behavior policy
- $I : \mathcal{S} \times \mathcal{A} \to [0,1]$; interest function (can set to 1 for all state-action pairs or indicate selected state-action pairs to be best approximated)
- $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$; feature-vector function
- $\lambda : \mathcal{S} \to [0,1]$; bootstrapping or eligibility-trace decay-rate function

Answer Functions

Recall:

$$\pi(s) = \begin{cases} \text{random action from } \mathcal{A}(s) & \text{if } \xi < \varepsilon \\ \text{argmax}_{a \in \mathcal{A}(s)} Q(s,a) & \text{otherwise,} \end{cases}$$

ε-greedy exploration/exploitation

ε-greedy GQ(λ) Learning Algorithm - Continued

- $\boldsymbol{\theta} \in \mathbb{R}^n$; the learned weights of the linear approximation: $Q^\pi(s,a) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a) = \sum_{i=1}^n \theta_i \phi_i(s,a)$
- $\boldsymbol{w} \in \mathbb{R}^n$; secondary set of learned weights
- $\boldsymbol{e} \in \mathbb{R}^n$; eligibility trace vector

Parameters internal to GQ:

- α; step-size parameter for learning $\boldsymbol{\theta}$
- $\eta \in [0,1]$; relative step-size parameter for learning $\boldsymbol{w}$ $(\alpha\eta)$

Internal Parameters

$$\rho_t = \frac{\pi(S_t, A_t)}{b(S_t, A_t)}, \tag{1}$$

Importance Sampling Ratio

$$\bar{\boldsymbol{\phi}}_t = \sum_{a \in \mathcal{A}} \pi(S_t, a)\boldsymbol{\phi}(S_t, a) \tag{2}$$

Expected Next Feature Vector

ε-greedy GQ(λ) Learning Algorithm - Continued

Goal is to learn this parameter vector.

Full Specification

$$\delta_t = r(S_t, A_t, S_{t+1}) + \gamma(S_{t+1})\boldsymbol{\theta}_t^\top \bar{\boldsymbol{\phi}}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_t, A_t) \quad (3)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\left[\delta_t \mathbf{e}_t - \gamma(S_{t+1})(1-\lambda(S_{t+1}))(\mathbf{w}_t^\top \mathbf{e}_t)\bar{\boldsymbol{\phi}}_{t+1}\right] \quad (4)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\eta[\delta_t \mathbf{e}_t - (\mathbf{w}_t^\top \boldsymbol{\phi}(S_t, A_t))\boldsymbol{\phi}(S_t, A_t)] \quad (5)$$

$$\mathbf{e}_t = I(S_t)\boldsymbol{\phi}(S_t, A_t) + \gamma(S_t)\lambda(S_t)\rho_t \mathbf{e}_{t-1} \quad (6)$$

weight doubling

The main idea of the algorithm is to minimize the projected Bellman error

$$J(\theta) = \|\Pi T^{\pi_\theta} Q_\theta - Q_\theta\|_\mu^2$$

Pseudocode

Initialize $\boldsymbol{\theta}$ arbitrarily and $\mathbf{w} = 0$
Repeat (for each episode):
  Initialize $\mathbf{e} = 0$
  $S \leftarrow$ initial state of episode
  Repeat (for each step of episode):
    $A \leftarrow$ action selected by policy b in state S
    Take action A, observe next state, S'
    $\bar{\boldsymbol{\phi}} \leftarrow 0$
    For all $a \in \mathcal{A}(s)$:
      $\bar{\boldsymbol{\phi}} \leftarrow \bar{\boldsymbol{\phi}} + \pi(S', a)\boldsymbol{\phi}(S', a)$
    $\rho = \frac{\pi(S,A)}{b(S,A)}$
    GQlearn$(\boldsymbol{\phi}(S, A), \bar{\boldsymbol{\phi}}, \lambda(S'), \gamma(S'), r(S, A, S'), \rho, I(S))$
    $S \leftarrow S'$
  until S' is terminal

GQLearn$(\boldsymbol{\phi}, \bar{\boldsymbol{\phi}}, \lambda, \gamma, R, \rho, I)$
  $\delta \leftarrow R + \gamma\boldsymbol{\theta}^\top \bar{\boldsymbol{\phi}} - \boldsymbol{\theta}^\top \boldsymbol{\phi}$
  $\mathbf{e} \leftarrow \rho\mathbf{e} + I\boldsymbol{\phi}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\delta\mathbf{e} - \gamma(1-\lambda)(\mathbf{w}^\top \mathbf{e})\bar{\boldsymbol{\phi}})$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha\eta(\delta\mathbf{e} - (\mathbf{w}^\top \boldsymbol{\phi})\boldsymbol{\phi})$
  $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e}$

Design of State and Action Space

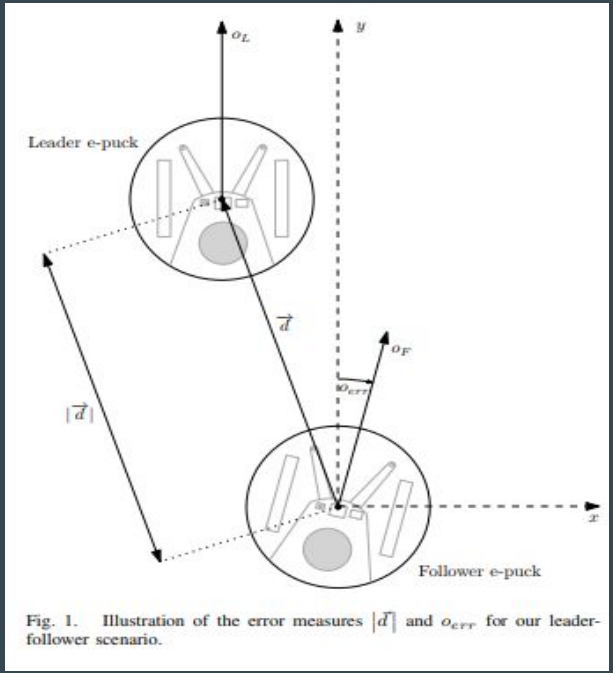


Fig. 1. Illustration of the error measures $|\vec{d}|$ and o_{err} for our leader-follower scenario.

Since there are two motors in each robot, action space is directly 2-dimensional which represents motor speeds. It is tile coded [linear uniform] as 7x7 for discretization.

State space is determined to be 2-dimensional as well:
1) Deviation from optimal distance*, i.e. distance err
2) Angular error.

*optimal distance: middle between the maximum and minimum allowed distance

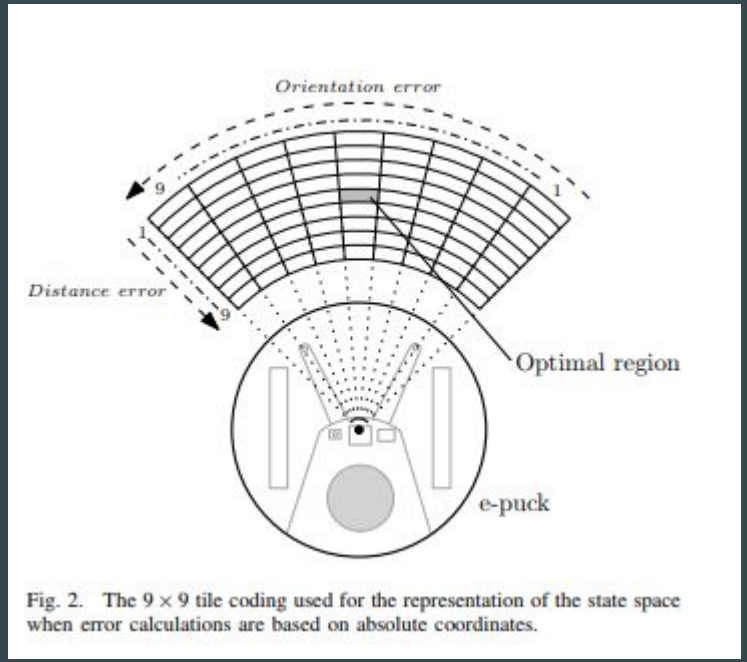


Fig. 2. The 9×9 tile coding used for the representation of the state space when error calculations are based on absolute coordinates.

State space is tile coded [logarithmic] too, as 9x9 for discretization.

Experiments & Results

Simulation

Leader: Simple statistical line follower. Moves arbitrarily.

- Up to 4 followers. Line-like formation. Try to learn the robot directly in front of them.
- Finding: Robots start to stabilize their behavior from first follower towards end of the line formation.

In 74% of all cases, the leader was detected in the dark red area right to optimal region. Additionally, the leader stayed within the orange/red 3x3 area in 98.7% of the time. Logaritmic tiling is used to have rarely visited states distinguishable.

Shift to the right side in the distribution is caused by the overall path which is a closed circular loop.

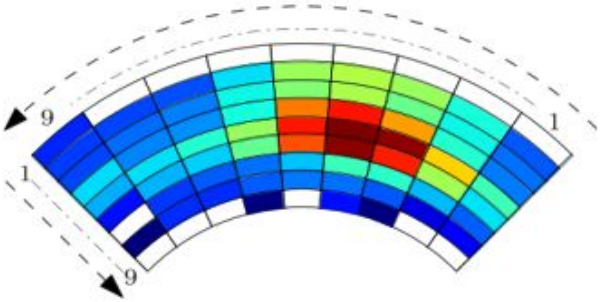


Fig. 4. Distribution of visited states at around 1.3 million steps. In 74% of all cases, the leader was detected in the dark red area right to our optimal region. It stayed within the orange/red 3×3 area on the right side of the optimal region in 98.7% of the time. To make the rarely visited states discernible, a logarithmic scale is used: red and orange values are in the range of 30%–1%, green values are at around 0.1%, and blue values are another magnitude below that. The non-colored states were never visited in this experiment.

Experiments & Results - Continued

Real World

Overero COM instead of a Standart PC is used for whole training.

- Computational bottlenecks are introduced.
- Tiling is reduced from 9x9, 7x7 to 5x5 and 3x3 respectively for state and action values.
- Environmental lighting may heavily distort the sensor readings.
- Tradeoff between smoothness & learning efficiency vs. insurance of followers can keep up with the leader in limited time.

Cure: Transferability of learned parameters from simulation.

✔ Despite all sensor and computational issues, learning algorithm showed that it is able to learn the right parameters for following the leader, i.e. keeping the formation valid.

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

! Nonetheless, stabilization take much more time than the simulation.

Conclusion

- ε-greedy GQ(λ) can be used to have robots learn to keep a line-like formation.
- Required learning parameters can be adapted from a simulation (through transfer of value function estimation).
- “Natural” behavior is obtained through smoothed variant of the leader’s path which would intuitively increase the acceptance of the robot’s behavior by non-experts.