

# A Novel Model-Free Learning Approach for Dynamic Target Tracking Problem and its Validation using Virtual Robot Experimentation Platform

Amr Elhussein and Md Suruz Miah

*Electrical & Computer Eng.*

*Bradley University, Peoria, Illinois, USA*

aelhussein@mail.bradley.edu and smiah@bradley.edu

**Abstract**—This paper advances the previous work done by authors by validating a model-free actor-critic reinforcement learning approach to solve dynamic target tracking problem for a car-like mobile robot. The learning approach generates the linear velocity and steering angle of the robot and it does not require any prior knowledge of the dynamic model of the moving target. Policy iteration approach is employed through Bellman's principle of optimality to assess the cost of the control actions derived by the proposed learning method. The algorithm is tested by conducting a set of computer experiments for complex scenarios using virtual robot experimentation platform widely known as CoppeliaSim.

**Review and expand**

**Index Terms**—Leader-follower formation, mobile robots, reinforcement learning, policy iteration, trajectory tracking

## I. INTRODUCTION

Need to discuss in details the limitations of previous work, we can also add RL applications to robotics navigation

Tracking a random moving target using a mobile robot, for instance, is a challenging task. This is mainly due to their inherent complex nonlinear dynamics. Most of the target tracking algorithms proposed in the literature either rely on A) complex mathematical models, B) simplified mathematical models, and C) nonlinear control techniques or driven by large amount of mostly offline data that leads to an overwhelming degree of computational complexity.

Over the past years mobile robots has been used in several applications in commercial and military sectors such as surveillance, search and rescue missions, coverage optimization, cooperative localization and dynamic target tracking tasks. In all of these mentioned applications a fleet of robot i.e agents interact with each other to achieve a certain goal. Leader-Follower or dynamic target tracking problem has received an extensive amount of study and research in the world of cooperative control theory due to its wide promising applications such as search and rescue missions, wildlife monitoring and surveillance to name a few. In a typical leader follower formation a number of robots referred to as followers apply local control actions to follow leader's robots in a specific predefined path such as **cyclic, circular motion and time varying communication topologies**. **limitations of other methods** many control methods were proposed however they

have had the following limitations:

- (1) a static leader/target is used.
- (2) the leader/target's location or dynamic model is predetermined.
- (3) expensive hardware platforms are used to achieve the mobile target tracking.

The main contribution of this work is the development of a model-free learning approach to control the follower robot by which it overcome many of the limitations mentioned above as it does not rely on any prior information of the mathematical model or the dynamic model. The steering angle and the linear speed of the follower robot are determined by collecting the position and orientation of both the leader and the follower. This set of information is gathered online over a finite period of time. The optimal control actions are then generated by utilizing Bellman's principle of optimality which acts as model free-reinforcement learning approach that allows the follower robot to follow the path of the leader while avoiding collision by maintaining a safe distance. In this paper the proposed algorithm is further validated using a commercially available robot simulator, CoppeliaSim. This paper acts as first milestone in generalizing the algorithm to solve more sophisticated problem such as coverage and mapping. **cite area coverage papers**

The rest of the paper is organized as follows. Section II lays down the problem setting of the leader follower problem and mathematical models of the robots and the state error. The model free actor-critic reinforcement learning and its key steps are described in section III. Section IV illustrates computer simulations for different scenarios that reflects the effectiveness of the proposed method followed by conclusion and future work presented in section V.

## II. DYNAMIC TARGET TRACKING AND PROBLEM SETTING

**Needs to reviewed**

Suppose that a wheeled mobile robot (follower) with coordinate  $(x, y)$  and orientation  $\theta \in [-\pi, \pi)$  rad with respect to the global X-Y coordinate has a linear speed  $v$  and steering angle  $\gamma$  which are considered as the two control actions. Let the position and orientation (pose) of the target i.e leader be  $\mathbf{q}_k \equiv [x_k, y_k, \theta_k]$ . The robot and the target are deployed in a 2D configuration in which the Follower is to follow the

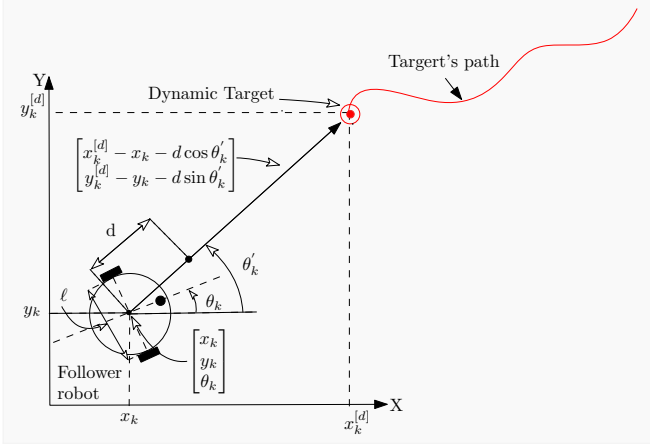


Fig. 1. Mobile robot and its dynamic target tracking problem setup.

target trajectory defined by  $(\mathbf{p}_k^{[d]})^T = [x_k^{[d]}, y_k^{[d]}(t)]$  at time  $t \geq 0$  with  $t = k T_s$ ,  $k \in \mathbb{N}_0$ , and  $T_s > 0$  being the sampling time. Note that this trajectory is completely random. The target motion is modeled by the following discrete-time system:

$$\mathbf{p}_{k+1}^{[d]} = \mathbf{p}_k^{[d]} + T_s \mathbf{u}_k^{[d]}, \quad (1)$$

with  $\mathbf{u}_k^{[d]} \in \mathbb{R}^2$  being the control inputs of the target movement. The follower robot is to maintain a constant safe-distance  $d > 0$  from target. The dynamic car-like model of the follower robot is defined as follows:

$$x_{k+1} = x_k + T_s \nu_k \cos(\theta_k + \gamma_k) + \zeta_1, \quad (2a)$$

$$y_{k+1} = y_k + T_s \nu_k \sin(\theta_k + \gamma_k) + \zeta_2, \quad (2b)$$

$$\theta_{k+1} = \theta_k + T_s \nu_k \frac{\sin(\gamma_k)}{l} + \zeta_3, \quad (2c)$$

Note that the follower dynamic model obeys Ackermann steering principle where  $\gamma_k \in (-\frac{\pi}{2}, \frac{\pi}{2})$  is the front wheel steering angle with respect to the robot's orientation  $\theta_k \in [-\pi, \pi]$ ,  $\nu_k$  is the linear speed,  $l$  is the distance between the drive wheels of the robot, and  $\zeta_1, \zeta_2, \zeta_3 \in \mathbb{R}$  are the model uncertainties. We then define the error vector as:

$$\mathbf{e}_k^T = [\rho_e, \theta_e]^T = [\sqrt{x_e^2 + y_e^2}, \theta_e]^T = [\sqrt{(x_k^{[d]} - x_k - d \cos \theta_k')^2 + (y_k^{[d]} - y_k - d \sin \theta_k')^2}, \theta_k' - \theta_k], \quad (3)$$

where  $\theta_k' = \text{atan2}(y_k^{[d]} - y_k, x_k^{[d]} - x_k)$  and  $\rho_e$  being the euclidean distance.

The control problem can then be formally stated as follows: Find  $\nu_k$  and  $\gamma_k$  such that  $\mathbf{e}_k \rightarrow \mathbf{0}$  as  $k \rightarrow \infty$  subject to (1) and (2).

### III. PROPOSED ACTOR-CRITIC RL APPROACH

Needs to be rewritten

The solution of the leader-follower formation problem is realized using a reinforcement learning approach. It employs

model-free strategies for solving a temporal difference equation developed herein. This solution is equivalent to solving the underlying Bellman optimality equation for the dynamical error model (3). The relative importance of the states in the error vector  $\mathbf{e}_k$  and the control decisions (linear velocity and steering angle) of the follower-robot are evaluated using the performance (cost) index:

$$J = \sum_{k=0}^{\infty} \frac{1}{2} [\mathbf{e}_k^T \mathbf{Q} \mathbf{e}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k], \quad (4)$$

where  $\mathbf{Q} \in \mathbb{R}^{2 \times 2}$  and  $\mathbf{R} \in \mathbb{R}^{2 \times 2}$  are symmetric positive definite weighting matrices. The objective of the optimization problem, following [?], is to find an optimal sequence of control policies  $\{\mathbf{u}_k^*\}_{k=0}^{\infty}$  that minimizes the cost index  $J$  along the state-trajectories (1) and (2). Motivated by the structure of the convex quadratic cost functional (4), let the solution of the tracking control problem employ the value function  $V(\mathbf{e}_k, \mathbf{u}_k)$  defined by

$$V(\mathbf{e}_k, \mathbf{u}_k) = \sum_{\kappa=k}^{\infty} \frac{1}{2} (\mathbf{e}_{\kappa}^T \mathbf{Q} \mathbf{e}_{\kappa} + \mathbf{u}_{\kappa}^T \mathbf{R} \mathbf{u}_{\kappa}).$$

This structure yields a temporal difference form (i.e., Bellman equation) as follows

$$V(\mathbf{e}_k, \mathbf{u}_k) = \frac{1}{2} [\mathbf{e}_k^T \mathbf{Q} \mathbf{e}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k] + V(\mathbf{e}_{k+1}, \mathbf{u}_{k+1}).$$

Applying Bellman's optimality principle yields the optimal control policies  $\mathbf{u}_k^*$ ,  $k \geq 0$ , such that [?]

$$\mathbf{u}_k^* = \underset{\mathbf{u}_k}{\text{argmin}} \left[ \frac{1}{2} [\mathbf{e}_k^T \mathbf{Q} \mathbf{e}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k] + V(\mathbf{e}_{k+1}, \mathbf{u}_{k+1}) \right].$$

Alternatively, this optimal policy form is equivalent to  $\mathbf{u}_k^* = \underset{\mathbf{u}_k}{\text{argmin}} [V(\mathbf{e}_k, \mathbf{u}_k)]$ . Therefore, the underlying Bellman optimality equation follows

$$V^*(\mathbf{e}_k, \mathbf{u}_k^*) = \frac{1}{2} [\mathbf{e}_k^T \mathbf{Q} \mathbf{e}_k + \mathbf{u}_k^{*T} \mathbf{R} \mathbf{u}_k^*] + V^*(\mathbf{e}_{k+1}, \mathbf{u}_{k+1}^*),$$

where  $V^*(\cdot, \cdot)$  is the optimal solution for Bellman optimality equation. This temporal difference equation is utilized by reinforcement learning process which solves the following temporal difference approximation form

$$\hat{V}(\mathbf{z}_k) = \frac{1}{2} \mathbf{z}_k^T \bar{\mathbf{P}} \mathbf{z}_k + \hat{V}(\mathbf{z}_{k+1}), \quad (5)$$

where  $\mathbf{z}_k = [\mathbf{e}_k, \mathbf{u}_k]^T \in \mathbb{R}^4$ ,  $V(\mathbf{e}_k, \mathbf{u}_k) \approx \hat{V}(\mathbf{z}_k)$ , and  $\bar{\mathbf{P}}$  is a symmetric block-diagonal matrix formed using  $(\mathbf{Q}, \mathbf{R})$ , i.e.,  $\bar{\mathbf{P}} = \text{blockdiag}(\mathbf{Q}, \mathbf{R})$ . The approximation of the solving value function  $\hat{V}(\mathbf{z}_k)$  employs a quadratic form so that  $\hat{V}(\mathbf{z}_k) = \frac{1}{2} \mathbf{z}_k^T \mathbf{P} \mathbf{z}_k$ , where  $\mathbf{P} \in \mathbb{R}^{4 \times 4}$  is a positive definite matrix. Hence, the optimal control strategy  $\mathbf{u}_k^*$  can be expressed as follows

$$\mathbf{u}_k^* = \underset{\mathbf{u}_k}{\text{argmin}} [\hat{V}(\mathbf{z}_k)] = -\mathbf{P}_{uu}^{-1} \mathbf{P}_{ue} \mathbf{e}_k, \quad (6)$$

where  $\mathbf{P}_{uu}$  and  $\mathbf{P}_{ue}$  are sub-blocks of symmetric matrix  $\mathbf{P}$ .

Second, the policy evaluation step of this process updates the critic weights  $\omega$  in real-time without acquiring any formation about the dynamics of the leader or follower dynamical systems (the calculation mechanism of the critic weights  $\omega$  is explained later on). This is done to search for a strictly better policy.

$$\mathbf{z}_k^T \mathbf{P} \mathbf{z}_k - \mathbf{z}_{k+1}^T \mathbf{P} \mathbf{z}_{k+1} = \mathbf{z}_k^T \bar{\mathbf{P}} \mathbf{z}_k. \quad (7)$$

This equation is utilized repeatedly in order to evaluate a certain policy during at least  $\eta \geq \bar{n}$ ,  $\bar{n} = (2+2)(2+2+1)/2$  evaluation steps (i.e., the lowest evaluation interval spans  $k$  to  $k + \bar{n}$  calculation samples) in order to update the critic weights vector  $\omega = \text{vec}(\mathbf{P})$ , which consists of connection weights between the neurons of the hidden layer and the output layer of the critic neural network shown in Fig. (??). The operator  $\text{vec}(\mathbf{P})$  forms the columns of the  $\mathbf{P}$  matrix into a column vector  $\omega$  of dimension  $\bar{n} = 15$  since the matrix  $\mathbf{P}$  is a symmetric matrix. The left hand side of (7) is expressed using the following critic approximation form

$$\hat{V}(\mathbf{z}_k) - \hat{V}(\mathbf{z}_{k+1}) = \omega^T \tilde{\rho}(\mathbf{z}_{k,k+1}),$$

where  $\tilde{\rho}(\mathbf{z}_{k,k+1}) = \rho(\mathbf{z}_k) - \rho(\mathbf{z}_{k+1}) \in \mathbb{R}^{10 \times 1}$ ,  $\rho(\mathbf{z}_k) = (\mathbf{z}_k^q \otimes \mathbf{z}_k^h)$  ( $q = 1, \dots, 5$ ,  $h = q, \dots, 5$ ), and  $\omega^T = [0.5 P^{11}, P^{12}, P^{13}, P^{14}, 0.5 P^{22}, P^{23}, P^{24}, 0.5 P^{33}, P^{34}, 0.5 P^{44}]^T \in \mathbb{R}^{1 \times 15}$  ( $P^{ij}$  is the  $ij^{th}$  entry of matrix  $\mathbf{P}$ ). The critic weights  $\omega$  are updated using a gradient descent approach, where the tuning error  $\varepsilon_k$  at each computational instance  $k$  follows  $\varepsilon_k = \omega^T \tilde{\rho}(\mathbf{z}_{k,k+1}) - v_k$ , where  $v_k = \frac{1}{2} \mathbf{z}_k^T \bar{\mathbf{P}} \mathbf{z}_k$ . As detailed earlier, it is required to perform at least  $\eta \geq \bar{n}$  evaluation steps before updating the critic weights  $\omega$  (i.e., finding the new improved policy). Hence, it is required to minimize the sum of square errors such that

$$\begin{aligned} \delta_c &= \sum_{\kappa=0}^{\eta-1} \frac{1}{2} (\omega^T \tilde{\rho}(\mathbf{z}_{k+\kappa,k+\kappa+1}) - v_{k+\kappa})^2 = \frac{1}{2} \|\mathbf{v} - \Lambda \omega\|^2 \\ &= \frac{1}{2} (\mathbf{v} - \Lambda \omega)^T (\mathbf{v} - \Lambda \omega), \end{aligned}$$

where  $\Lambda = [\mathbf{o}_0, \mathbf{o}_1, \dots, \mathbf{o}_{\eta-1}]^T \in \mathbb{R}^{\eta \times 15}$  with  $\mathbf{o}_\kappa = \tilde{\rho}^T(\mathbf{z}_{k+\kappa,k+\kappa+1}) \in \mathbb{R}^{1 \times 15}$  and  $\mathbf{v} = [v_0, v_1, \dots, v_{\eta-1}]^T \in \mathbb{R}^\eta$  with  $v_\kappa = \frac{1}{2} \mathbf{z}_{k+\kappa}^T \bar{\mathbf{P}} \mathbf{z}_{k+\kappa}$  for  $\kappa = 0, 1, \dots, \eta - 1$ . Therefore, the update law of the critic weights using the gradient decent approach for at least  $\bar{n}$  samples is given by

$$\begin{aligned} \omega^{[r+1]} &= \omega^{[r]} - \ell_c \frac{\partial \delta_c}{\partial \omega} = \omega^{[r]} - \ell_c (-\Lambda^T \mathbf{v} + \Lambda^T \Lambda \omega^{[r]}) \\ &= \omega^{[r]} - \ell_c \Lambda^T (\Lambda \omega^{[r]} - \mathbf{v}), \quad (8) \end{aligned}$$

where  $0 < \ell_c < 1$  is a critic learning rate and  $r$  is the update index of the critic weights.

$$\mathbf{P} = \begin{bmatrix} 2\omega^1 & \omega^2 & \omega^3 & \omega^4 \\ \omega^2 & 2\omega^5 & \omega^6 & \omega^7 \\ \omega^3 & \omega^6 & 2\omega^8 & \omega^9 \\ \omega^4 & \omega^7 & \omega^9 & 2\omega^{10} \end{bmatrix} \in \mathbb{R}^{4 \times 4},$$

where  $\omega^i$  is the  $i^{th}$  entry of the weight vector  $\omega$ .

merge the actor weights discussion with the rest

The critic weights are then used to update the actor weights which maps the state error to the desired policy (control actions) at every time step  $t$  with the following equation:

$$\mathbf{u}_k = \omega_a^T \mathbf{e}_k \quad (9)$$

Applying a similar gradient descent approach as with the critic weights the actor weights are updated as follows:

$$\omega_a^{[r+1]} = \omega_a^{[r]} - \ell_a (\mathbf{u}_k - \mathbf{u}_k^*) \mathbf{e}_k, \quad (10)$$

where  $\ell$  is the actor learning rate.

The complete policy iteration solution process for the leader-follower problem is detailed out in Algorithm 1.

---

**Algorithm 1:** Model-free actor-critic reinforcement learning using the policy iteration solution.

---

**Input:** Sampling-time  $T_s$ ,  $\mathbf{Q}$ , and  $\mathbf{R}$

**Output:** Error trajectory  $\mathbf{e}_k$ , for  $k = 0, 1, \dots$

---

```

1 begin
2    $k = 0, r = 0$  /* Discrete time and
   policy indices */
3    $\eta = (n + m)(n + m + 1)/2$ 
4   Initialize  $\mathbf{P}^{[0]}$  /* Positive definite */
5   Set offset distance  $d$ 
6   Given approximate initial poses of leader and
   follower, compute  $\mathbf{e}_0$  using error model (3)
7   Compute follower's input  $\mathbf{u}_0^{[0]}$  using policy (9)
8   repeat /* Main timing loop */
9     Find  $\mathbf{e}_{k+1}$  using (3)
10    Compute policy  $\mathbf{u}_{k+1}^{[r]}$  using (9)
11    if  $[(k + 1) \text{ modulo } \eta] == 0$  then
12       $r \leftarrow r + 1$  /* Evaluate policy
      */ /* using (6)
13      Solve for the critic-weights  $\omega_c$ 
      using (8)
14
15      Solve for the actor-weights  $\omega_a$  using (10)
16
17      Construct matrix  $\mathbf{P}^{[r]}$  using vector  $\omega_c$ 
18    until Tracking errors are zero
19     $k \leftarrow k + 1$ 

```

---

#### IV. COMPUTER EXPERIMENTS AND RESULTS

This section adopts the theoretical results discussed in the previous section by simulating the Algorithm using the Pioneer 3-DX in CoppeliaSim. The results obtained by the authors in [cite flairs paper here](#) are validated using the commercial robot simulator, CoppeliaSim as a preliminary step to implement the Algorithm experimentally in real world. we present the dynamics of the tracking error and the convergence characteristics of

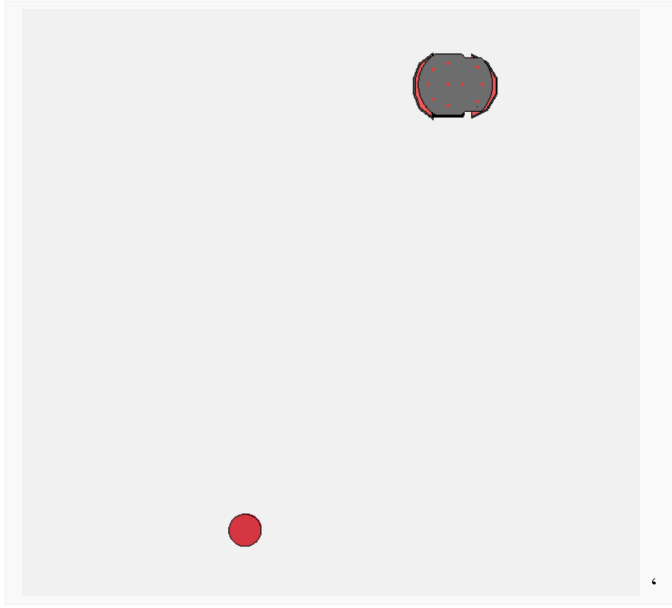


Fig. 2. Coppeliasim Scene.

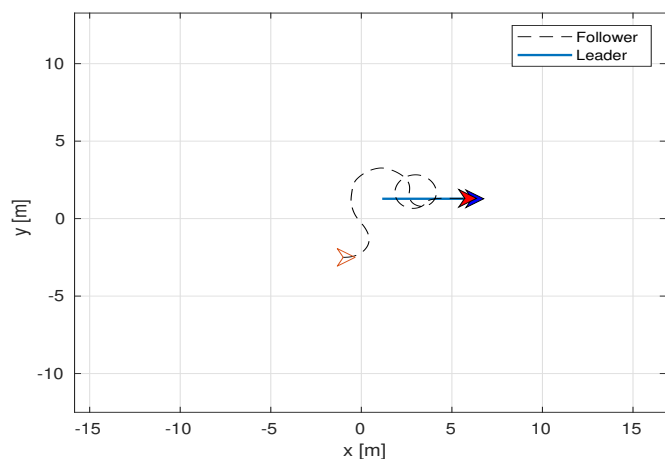
the actor and critic weights. The weighting matrices are set to  $\mathbf{Q} = \text{diag}[0.001, 0.001]$  and  $\mathbf{R} = \text{diag}[10^{-5}, 10^{-5}]$ .

$$\mathbf{Q} = 0.001 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{R} = 0.00001 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

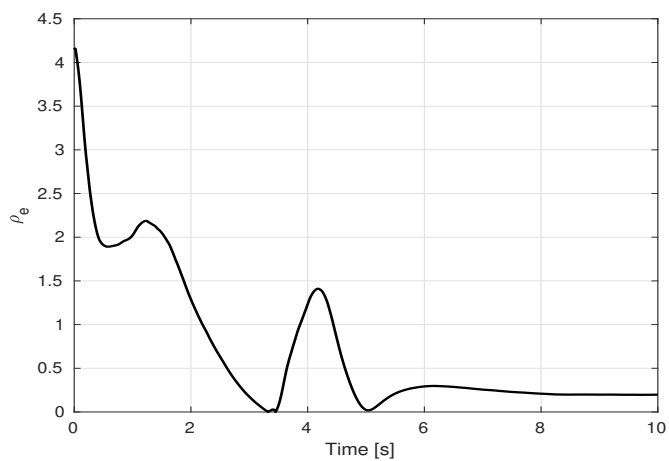
The actor and critic learning rates  $\ell_c$  are set to 0.01 and 0.0001. The sampling time  $T_s$  is set to 0.001 sec. The desired distance offset between the leader and the follower is set to  $d = 0.5$  [m] for all scenarios. The Coppeliasim simulator accurately mimic what would happen in a realworld scenario following idustry standatrds and guidelines. The Pioneer robot model and the was initally placed at position (-1,-2.5) m. The Target modeled as a cylinder was initially positioned on a random position around the origin (0,0). The simulation was run for

## V. CONCLUSION

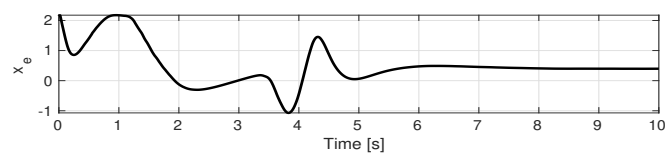
an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks . . .”. Instead, try “R. B. G. thanks . . .”. Put sponsor acknowledgments in the unnumbered footnote on the first page.



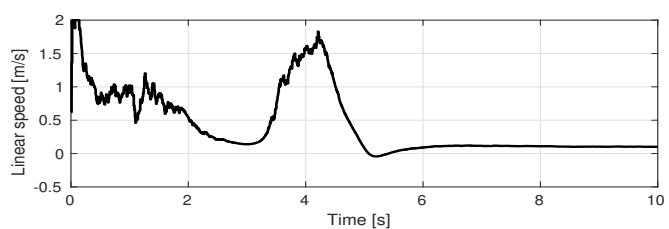
(a)



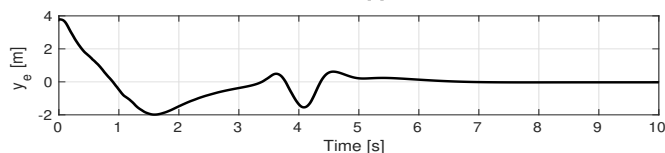
(b)



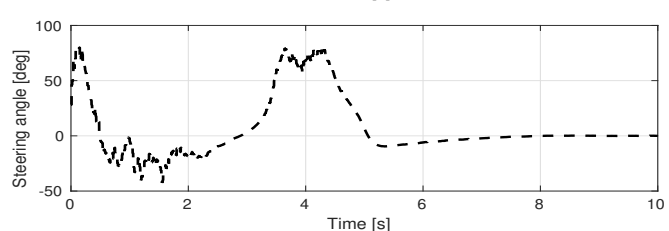
(c)



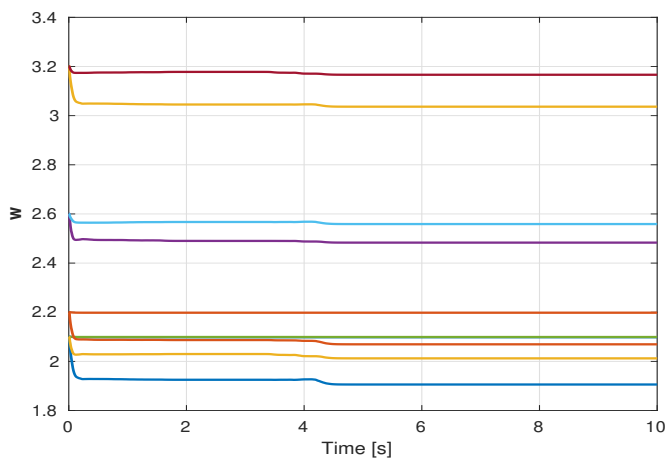
(d)



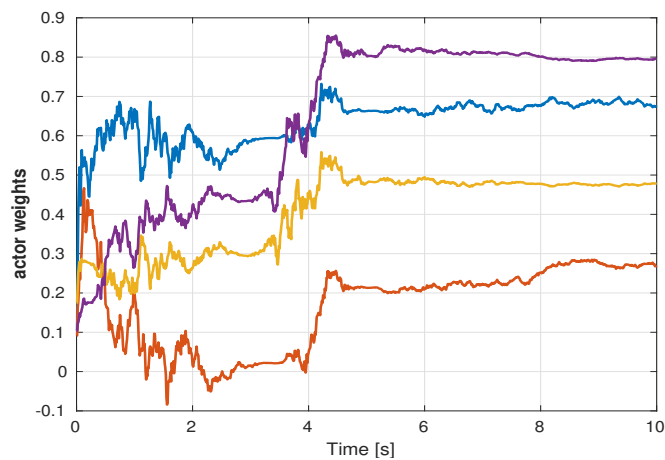
(e)



(f)



(g)



(h)

Fig. 3. Performance in tracking random trajectory: (a) leader-follower trajectories, (c) state error, (d) linear speed and steering angle of the follower; and (e) learning weights.