TECHNICAL APPENDIX FOR THE PAPER[**]: TOWARDS SAFE AI: SANDBOXING DNNS-BASED CONTROLLERS IN STOCHASTIC GAMES

BINGZHUO ZHONG[1*], HONGPENG CAO[1], MAJID ZAMANI[2,3], AND MARCO CACCAMO[1]

1. Preliminaries

Maths A topological space S is called a Borel space if it is homeomorphic to a Borel subset of a Polish space (*i.e.,* a separable and completely metrizable space). One of the examples of Borel space is the Euclidean spaces $\mathbb{R}^n$. Here, any Borel space S is assumed to be endowed with a Borel σ-algebra, denoted by $\mathcal{B}(S)$. A map $f : X \rightarrow Y$ is measurable whenever it is Borel measurable.

Definition of Sandbox *Sandbox* [1] is an idea borrowed from the computer security community. In general, the sandbox is a common mechanism that ensures the security of CPSs containing untested and untrusted software components. A verified sandbox is designed to isolate these components from the critical part of a host machine in a sandbox control environment, including its operating system. Consequently, an untrusted controller can only actuate the plant through a tightly-monitored interface so that the plant would not be endangered by accidental or malicious harm caused by the unverified controllers.

Definition of bad prefix [2, Section 2.2] Consider a language $L \subseteq \Sigma^\omega$ of infinite words over the alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for L if and only if $\forall y \in \Sigma^\omega$, we have $x \cdot y \notin L$, where $x \cdot y$ denotes the concatenation between x and y.

2. Proof for Theorem 3.6

Throughout the proof, a probability space is presented by $(\hat{\Omega}, \mathcal{F}_{\hat{\Omega}}, \mathbb{P}_{\hat{\Omega}})$, where $\hat{\Omega}$ is the sample space, $\mathcal{F}_{\hat{\Omega}}$ is a sigma-algebra on $\hat{\Omega}$ which comprises subsets of $\hat{\Omega}$ as events, and $\mathbb{P}_{\hat{\Omega}}$ is a probability measure that assigns probabilities to events. Here, we focus on random variables, denoted by X, that take values from measurable spaces $(S, \mathcal{B}(S))$, *i.e.,* random variables here are measurable functions $X : (\hat{\Omega}, \mathcal{F}_{\hat{\Omega}}) \rightarrow (S, \mathcal{B}(S))$ such that one has $Prob\{\mathcal{Q}\} = \mathbb{P}_{\hat{\Omega}}\{X^{-1}(\mathcal{Q})\}$, $\forall \mathcal{Q} \in \mathcal{B}(S)$. For succinctness, we directly present the probability measure on $(S, \mathcal{B}(S))$ without explicitly mentioning the underlying probability space and the function X itself. Additionally, we denote by $\mathbf{P}(S, \mathcal{B}(S))$ a set of probability measures on the $(S, \mathcal{B}(S))$.

Before showing the proof for Theorem 3.6, some additional definitions and lemmas are required. First, we define the *n-steps reachable state set* of a DFA as follows.

Definition 1. (n-steps Reachable State Set) *Consider a gDTSG* $\mathfrak{D} = (X, U, W, X_0, T, Y, h)$ *and a DFA* $\mathcal{A} = (Q, q_0, \Pi, \tau, F)$ *with a labeling function* $L : Y \rightarrow \Pi$. *An* n-steps reachable state set $\tilde{Q}_n(x_0)$ *of* $\mathcal{A}$ *with respect to an initial state* $x_0 \in X_0$ *is recursively defined as*

$$\tilde{Q}_0(x_0) = \Big\{ q \in Q \,\Big|\, q = \tau(q_0, L \circ h(x_0)) \Big\},$$
$$\tilde{Q}_n(x_0) = \Big\{ q \in Q \,\Big|\, \exists q' \in \tilde{Q}_{n-1}(x_0), \sigma \in \Pi \ s.t.\ q = \tau(q', \sigma) \Big\},$$

with $n \in \mathbb{N}_{>0}$.

*Corresponding Author.

**In the Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence

Additionally, we need the definitions of Markov policy, Product gDTSG, and a cost-to-go function, which are borrowed from [3].

Definition 2. (Markov Policy) *Consider a gDTSG $\mathfrak{D} = (X, U, W, X_0, T, Y, h)$. A Markov policy ρ defined over the time horizon $[0, H-1] \subset \mathbb{N}$ for Player I is a sequence $\rho = (\rho_0, \rho_1, \ldots, \rho_{H-1})$ of universally measurable maps $\rho_k : X \to \mathbf{P}(U, \mathcal{B}(U))$, with*

$$\rho_k(U \big| x(k)) = 1.$$

Similarly, a Markov policy λ for Player II is a sequence $\lambda = (\lambda_0, \lambda_1, \ldots, \lambda_{H-1})$ of universally measurable maps $\lambda_k : X \times U \to \mathbf{P}(W, \mathcal{B}(W))$, with

$$\lambda_k(W \big| x(k), u(k)) = 1,$$

for all $k \in [0, H-1]$. We use $\mathcal{P}$ and Λ to denote the set of all Markov policies for Players I and II, respectively. Moreover, we denote by $\mathcal{P}^H$ and Λ^H the set of all Markov policies for Players I and II within time horizon $[0, H-1]$, respectively.

Definition 3. (Product gDTSG) *Consider a gDTSG $\mathfrak{D} = (X, U, W, X_0, T, Y, h)$, a DFA $\mathcal{A} = (Q, q_0, \Pi, \tau, F)$, and a labeling function $L : Y \to \Pi$ as in Definition 2.4. The product of $\mathfrak{D}$ and $\mathcal{A}$ is a gDTSG defined as*

$$\mathfrak{D} \otimes \mathcal{A} = \{\bar{X}, \bar{U}, \bar{W}, \bar{X}_0, \bar{T}, \bar{Y}, \bar{h}\},$$

where $\bar{X} := X \times Q$ is the state set; $\bar{U} := U$ is the input set for Player I; $\bar{W} := W$ is the input set for Player II; $\bar{X}_0$ is the initial state set, with $\bar{x}_0 := (x_0, \bar{q}_0) \in \bar{X}_0$, $x_0 \in X_0$ and

$$\bar{q}_0 = \tau(q_0, L \circ h(x_0)); \tag{2.1}$$

$\bar{T}(\mathsf{d}x' \times \{q'\} | x, q, u, w)$ is the stochastic kernel that assigns for any $(x, q) \in \bar{X}$, $u \in \bar{U}$, and $w \in \bar{W}$ the probability $\bar{T}(\mathsf{d}x' \times \{q'\} | x, q, u, w) = T(\mathsf{d}x' | x, u, w)$ when $q' = \tau(q, L \circ h(x'))$, and $\bar{T}(\mathsf{d}x' \times \{q'\} | x, q, w, u) = 0$, otherwise; $\bar{Y} := Y$ is the output set and $\bar{h}(x, q) := h(x)$ is the output map.

Definition 4. *Given a Markov policy $\rho = (\rho_0, \rho_1, \ldots, \rho_{H-1})$ for Player I and $\lambda = (\lambda_0, \lambda_1, \ldots, \lambda_{H-1})$ for Player II of $\widehat{\mathfrak{D}} \otimes \mathcal{A}$, we define a cost-to go function $\underline{V}_n^{\rho,\lambda} : \hat{X} \times Q \to [0, 1]$, which is initialized by $\underline{V}_0^{\rho,\lambda}(\hat{x}, q) = 1$ when $q \in F$, and $\underline{V}_0^{\rho,\lambda}(\hat{x}, q) = 0$, otherwise, and recursively computed as*

$$\underline{V}_{n+1}^{\rho,\lambda}(\hat{x}, q) := \begin{cases} (1-\delta) \displaystyle\sum_{\hat{x}' \in \hat{X}} \underline{V}_n^{\rho,\lambda}(\hat{x}', \bar{q}(\hat{x}', q)) \hat{T}(\hat{x}' | \hat{x}, \hat{u}, \hat{w}) + \delta, & \text{if } q \notin F; \\ 1, & \text{if } q \in F, \end{cases} \tag{2.2}$$

where $\hat{u} = \rho_{H-n-1}(\hat{x}, q)$, $\hat{w} = \lambda_{H-n-1}(\hat{x}, q, \hat{u})$, and

$$\bar{q}(\hat{x}', q) = \underset{q' \in Q'_\epsilon(\hat{x}')}{\arg\max}\, \underline{V}_n^{\rho,\lambda}(\hat{x}', q'), \tag{2.3}$$

with $Q'_\epsilon(\hat{x}')$ as in Definition 3.3.

Additionally, the worst-case adversarial policy $\lambda^*(\rho)$ for Player II with respect to the Markov policy ρ for Player I can be computed as

$$\lambda^*_{H-n-1}(\rho) \in \max_{\lambda_{H-n-1} \in \Lambda} \Big((1-\delta) \sum_{\hat{x}' \in \hat{X}} \underline{V}_n^{\rho,\lambda^*(\rho)}(\hat{x}', \bar{q}(\hat{x}', q)) \hat{T}(\hat{x}' | \hat{x}, \hat{u}, \hat{w}) + \delta\Big), \tag{2.4}$$

for all $n \in [0, H-1]$, with $\hat{u} = \rho_{H-n-1}(\hat{x}, q)$, and $\hat{w} = \lambda_{H-n-1}(\hat{x}, q, \hat{u})$.

With Definition 4, we have the following lemma, which is required for showing the results of Theorem 3.6.

Lemma 5. *Given a gDTSG $\mathfrak{D} = (X, U, W, X_0, T, Y, h)$ and its finite abstraction $\widehat{\mathfrak{D}} = (\hat{X}, \hat{U}, \hat{W}, \hat{X}_0, \hat{T}, Y, \hat{h})$ with $\widehat{\mathfrak{D}} \preceq_\epsilon^\delta \mathfrak{D}$, and a desired safety property (ϕ, H) with its associated DFA $\mathcal{A} = (Q, q_0, \Pi, \tau, F)$. Given a Markov policy $\rho = (\rho_0, \rho_1, \ldots, \rho_{H-1})$ over the product gDTSG $\widehat{\mathfrak{D}} \otimes \mathcal{A}$ within the time horizon $[0, H]$, one has*

$$1 - \underline{V}_{n+1}^{\rho,\lambda^*(\rho)}(\hat{x}, q) = (1-\delta) \min_{\lambda_{H-n-1} \in \Lambda} \sum_{\tilde{x} \in \hat{X}'_{-\epsilon}(q)} \Big(1 - \underline{V}_n^{\rho,\lambda^*(\rho)}(\tilde{x}, \bar{q}(\tilde{x}, q))\Big) \hat{T}(\tilde{x} \,|\, \hat{x}, \hat{u}, \hat{w}), \tag{2.5}$$

with $q \notin F$, $\hat{u} = \rho_{H-n-1}(\hat{x}, q)$, $\hat{w} = \lambda_{H-n-1}(\hat{x}, q, \hat{u})$, $\underline{V}^{\rho,\lambda^*(\rho)}_{n+1}(\hat{x}, q)$ as in (2.2), $\lambda^*(\rho)$ as in (2.4), $\bar{q}$ as in (2.3), and $\hat{X}'_{-\epsilon}(q)$ as in Definition 3.5.

The proof of Lemmas 5 can readily be derived with the help of (2.2) and (2.3). Now, we are ready to show the results for Theorem 3.6.

Proof of Theorem 3.6 For the sake of clarity of the proof, we define

$$r(\hat{x}(k), q(k), \hat{w}(k)) = 1 - \max_{\lambda_k \in \Lambda} \sum_{\hat{x}(k+1) \in \hat{X}} \underline{V}_{*,H-k-1}\Big(\hat{x}(k+1), \bar{q}_*\big(\hat{x}(k+1), q(k)\big)\Big) \hat{T}\big(\hat{x}(k+1) \,\big|\, \hat{x}(k), \hat{u}(k), \hat{w}(k)\big),$$

for $k \in [0, H-1]$ with $\underline{V}_{*,H-k-1}$ and $\bar{q}_*$ as in Definition 3.3, $\hat{u}(k) = \rho'_k(\hat{x}(k), q(k))$, and $\hat{w}(k) = \lambda_k(\hat{x}(k), q(k), \hat{u}(k))$ and

$$g(\hat{x}(z-1), q(z-1), \hat{w}(z-1)) = \hat{T}\big(\hat{x}(z) \,\big|\, \hat{x}(z-1), \rho'_{z-1}(\hat{x}(z-1), q(z-1)), \hat{w}(z-1)\big),$$

for $z \in [0, k]$. Consider an initial state $(\hat{x}_0, \bar{q}_0)$, with $\hat{x}_0 \in \hat{X}_0$. By leveraging (2.5), we expand out $\underline{V}^{\rho',\lambda^*(\rho')}_H(\hat{x}_0, \bar{q}_0)$ up to the time instant $k \in [0, H-1]$ as

$$\begin{aligned} &1 - \underline{V}^{\rho',\lambda^*(\rho')}_H(\hat{x}_0, \bar{q}_0) \\ =&(1-\delta)^{k+1} \min_{\lambda_0 \in \Lambda} \sum_{\hat{x}(1) \in \hat{X}'_{-\epsilon}(q(0))} \Big(\min_{\lambda_1 \in \Lambda} \sum_{\hat{x}(2) \in \hat{X}'_{-\epsilon}(q(1))} \Big(\dots \Big(\min_{\lambda_{k-2} \in \Lambda} \sum_{\hat{x}(k-1) \in \hat{X}'_{-\epsilon}(q(k-2))} \Big(\min_{\lambda_{k-1} \in \Lambda} \sum_{\hat{x}(k) \in \hat{X}'_{-\epsilon}(q(k-1))} r\big(\hat{x}(k), q(k), \hat{w}(k)\big) \\ &\times g\big(\hat{x}(k-1), q(k-1), \hat{w}(k-1)\big)\Big) g\big(\hat{x}(k-2), q(k-2), \hat{w}(k-2)\big)\Big) \dots \Big) g\big(\hat{x}(1), q(1), \hat{w}(1)\big)\Big) g\big(\hat{x}(0), q(0), \hat{w}(0)\big), \end{aligned} \tag{2.6}$$

with $\hat{w}(m) := \lambda_m(\hat{x}(m), \rho'_m(\hat{x}(m), q(m)))$, $m \in [0, k-1]$. Firstly, by selecting

$$\hat{x}_*(k) := \underset{\hat{x}(k) \in \hat{X}'_{-\epsilon}(q(k-1))}{\arg\min} r(\hat{x}(k), q(k), \hat{w}(k)),$$

with $q(k-1) \in \tilde{Q}_{k-1}(x_0)$, $\hat{X}'_{-\epsilon}(q(k-1))$ as in Definition 3.5, and $q_*(k) := \bar{q}(q(k-1), \hat{x}_*(k))$ with $\bar{q}$ as in (2.3), one has

$$\begin{aligned} &\min_{\lambda_{k-1} \in \Lambda} \sum_{\hat{x}(k) \in \hat{X}'_{-\epsilon}(q(k-1))} r(\hat{x}(k), q(k), \hat{w}(k)) g(\hat{x}(k-1), q(k-1), \hat{w}(k-1)) \\ &\geq r(\hat{x}_*(k), q_*(k), \hat{w}(k)) \min_{\lambda_{k-1} \in \Lambda} \sum_{\hat{x}(k) \in \hat{X}'_{-\epsilon}(q(k-1))} g(\hat{x}(k-1), q(k-1), \hat{w}(k-1)), \end{aligned}$$

with $\hat{w}(k-1) = \lambda_{k-1}\Big(\hat{x}(k-1), \rho'_{k-1}(\hat{x}(k-1), q(k-1))\Big)$. Hence, proceed with (2.6), one gets

$$\begin{aligned} &1 - \underline{V}^{\rho',\lambda^*(\rho')}_H(\hat{x}_0, \bar{q}_0) \\ \geq&(1-\delta)^{k+1} \min_{\lambda_0 \in \Lambda} \sum_{\hat{x}(1) \in \hat{X}'_{-\epsilon}(q(0))} \Big(\min_{\lambda_1 \in \Lambda} \sum_{\hat{x}(2) \in \hat{X}'_{-\epsilon}(q(1))} \Big(\dots \Big(\min_{\lambda_{k-2} \in \Lambda} \sum_{\hat{x}(k-1) \in \hat{X}'_{-\epsilon}(q(k-2))} \Big(\min_{\lambda_{k-1} \in \Lambda} \sum_{\hat{x}(k) \in \hat{X}'_{-\epsilon}(q(k-1))} g\big(\hat{x}(k-1), q(k-1), \\ &\hat{w}(k-1)\big)\Big) g\big(\hat{x}(k-2), q(k-2), \hat{w}(k-2)\big)\Big) \dots \Big) g\big(\hat{x}(1), q(1), \hat{w}(1)\big)\Big) g\big(\hat{x}(0), q(0), \hat{w}(0)\big) r\big(\hat{x}_*(k), q_*(k), \hat{w}(k)\big). \end{aligned} \tag{2.7}$$

with $\hat{w}(m) := \lambda_m(\hat{x}(m), \rho'_m(\hat{x}(m), q(m)))$, $m \in [0, k-1]$. Secondly, we select

$$\hat{x}_*(k-1) = \underset{\hat{x}(k-1) \in \hat{X}'_{-\epsilon}(q(k-2))}{\arg\min} \min_{\lambda_{k-1} \in \Lambda} \sum_{\hat{x}(k) \in \hat{X}'_{-\epsilon}(q(k-1))} g\big(\hat{x}(k-1), q(k-1), \hat{w}(k-1)\big), \tag{2.8}$$

with $\hat{w}(k-1) = \lambda_{k-1}\Big(\hat{x}(k-1), \rho'_{k-1}(\hat{x}(k-1), q(k-1))\Big)$, $q(k-2) \in \tilde{Q}_{k-2}(x_0)$, and

$$q_*(k-1) = \bar{q}(q(k-2), \hat{x}_*(k-1)), \tag{2.9}$$

with $\bar{q}$ as in (2.3). Then, one has

$$\begin{aligned}&\min_{\lambda_{k-2}\in\Lambda}\sum_{\hat{x}(k-1)\in\hat{X}'_{-\epsilon}(q(k-2))}\Big(\min_{\lambda_{k-1}\in\Lambda}\sum_{\hat{x}(k)\in\hat{X}'_{-\epsilon}(q(k-1))} g\big(\hat{x}(k-1),q(k-1),\hat{w}(k-1)\big)\Big)g\big(\hat{x}(k-2),q(k-2),\hat{w}(k-2)\big)\\ \geq\ &\Big(\min_{\lambda_{k-1}\in\Lambda}\sum_{\hat{x}(k)\in\hat{X}'_{-\epsilon}(q_*(k-1))} g\big(\hat{x}_*(k-1),q_*(k-1),\hat{w}(k-1)\big)\Big)\min_{\lambda_{k-2}\in\Lambda}\sum_{\hat{x}(k-1)\in\hat{X}'_{-\epsilon}(q(k-2))} g\big(\hat{x}(k-2),q(k-2),\hat{w}(k-2)\big).\end{aligned}$$

Thus, proceed from (2.7), we have

$$\begin{aligned}&1-\underline{V}_H^{\rho',\lambda^*(\rho')}(\hat{x}_0,\bar{q}_0)\\ \geq&(1-\delta)^{k+1}\min_{\lambda_0\in\Lambda}\sum_{\hat{x}(1)\in\hat{X}'_{-\epsilon}(q(0))}\Big(\min_{\lambda_1\in\Lambda}\sum_{\hat{x}(2)\in\hat{X}'_{-\epsilon}(q(1))}\Big(\dots\Big(\min_{\lambda_{k-2}\in\Lambda}\sum_{\hat{x}(k-1)\in\hat{X}'_{-\epsilon}(q(k-2))} g\big(\hat{x}(k-2),q(k-2),\hat{w}(k-2)\big)\Big)\dots\Big)\\ &\times g\big(\hat{x}(1),q(1),\hat{w}(1)\big)\Big)g\big(\hat{x}(0),q(0),\hat{w}(0)\big)\Big(\min_{\lambda_{k-1}\in\Lambda}\sum_{\hat{x}(k)\in\hat{X}'_{-\epsilon}(q_*(k-1))} g\big(\hat{x}_*(k-1),q_*(k-1),\hat{w}(k-1)\big)\Big)r\big(\hat{x}_*(k),q_*(k),\hat{w}(k)\big).\end{aligned} \tag{2.10}$$

with $\hat{w}(m):=\lambda_m(\hat{x}(m),\rho'_m(\hat{x}(m),q(m)))$, $m\in[0,k-1]$. For all $z\in[2,k-1]$, by choosing $x_*(z-1)$ similar to (2.8) and $q_*(z-1)$ similar to (2.9), one obtains

$$\begin{aligned}&\min_{\lambda_{z-2}\in\Lambda}\sum_{\hat{x}(z-1)\in\hat{X}'_{-\epsilon}(q(z-2))}\Big(\min_{\lambda_{z-1}\in\Lambda}\sum_{\hat{x}(z)\in\hat{X}'_{-\epsilon}(q(z-1))} g\big(\hat{x}(z-1),q(z-1),\hat{w}(z-1)\big)\Big)g\big(\hat{x}(z-2),q(z-2),\hat{w}(z-2)\big)\\ \geq&\Big(\min_{\lambda_{z-1}\in\Lambda}\sum_{\hat{x}(z)\in\hat{X}'_{-\epsilon}(q_*(z-1))} g\big(\hat{x}_*(z-1),q_*(z-1),\hat{w}(z-1)\big)\Big)\min_{\lambda_{z-2}\in\Lambda}\sum_{\hat{x}(z-1)\in\hat{X}'_{-\epsilon}(q(z-2))} g\big(\hat{x}(z-2),q(z-2),\hat{w}(z-2)\big).\end{aligned} \tag{2.11}$$

with $\hat{w}(m):=\lambda_m\Big(\hat{x}(m),\rho'_m(\hat{x}(m),q(m))\Big)$, $m\in\{z-1,z-2\}$. Then, with (2.11) for all $z\in[2,k-1]$ and (2.10), one gets

$$\begin{aligned}&1-\underline{V}_H^{\rho',\lambda^*(\rho')}(\hat{x}_0,\bar{q}_0)\\ \geq&\ (1-\delta)^{k+1}\Big(\min_{\lambda_0\in\Lambda}\sum_{\hat{x}(1)\in\hat{X}'_{-\epsilon}(q(0))} g(\hat{x}(0),q(0),\hat{w}(0))\Big)\prod_{z=2}^{k}\Big(\min_{\lambda_{z-1}\in\Lambda}\sum_{\hat{x}(z)\in\hat{X}'_{-\epsilon}(q_*(z-1))} g(\hat{x}_*(z-1),q_*(z-1),\hat{w}(z-1))\Big)r(\hat{x}_*(k),q_*(k),\hat{w}(k)).\end{aligned}$$

with $\hat{w}(m):=\lambda_m(\hat{x}(m),\rho'_m(\hat{x}(m),q(m)))$, $m\in[0,k-1]$. Note that $\bar{\omega}'_k:=\big(\hat{x}(0),q(0),\rho_0(\hat{x}(0),q(0)),\hat{x}_*(1),q_*(1),\rho_1(\hat{x}_*(1),q_*(1))\dots\hat{x}_*(k),q_*(k)\big)$ is one of history paths of the memory state of the safe-visor architecture up to the time instant k, and the supervisor as in Definition 3.5 ensures that for all such history paths $\bar{\omega}_k$, we have

$$(1-\delta)^{k+1}\prod_{z=1}^{k}\Big(\min_{\lambda_{z-1}\in\Lambda}\sum_{\bar{\omega}_{\hat{x}k}(z)\in\hat{X}'_{-\epsilon}(\bar{\omega}_{qk}(z-1))} g(\bar{\omega}_{\hat{x}k}(z-1),\bar{\omega}_{qk}(z-1)),\hat{w}(z-1)\Big)r(\bar{\omega}_{\hat{x}k}(k),\bar{\omega}_{qk}(k),\hat{w}(k))\geq\eta.$$

with $\hat{w}(m):=\lambda_m(\hat{x}(m),\rho'_m(\hat{x}(m),q(m)))$, $m\in[0,k-1]$. Therefore, we have $1-\underline{V}_H^{\rho',\lambda^*(\rho')}(\hat{x}_0,\bar{q}_0)\geq\eta$ when applying the supervisor as in Definition 3.5. According to[3, Theorem 5.10], one has $\mathbb{P}\Big\{y_H\models\phi\Big\}\geq\eta$, which completes the proof.

3. Training

This section describes the training setup. We leverage the OpenAI gym library [4] to build the environment for simulating quadrotor-vehicle tracking scenario. All the hyper-parameter settings can be found in the configuration file "ddpg_drn.json" provided with the code.

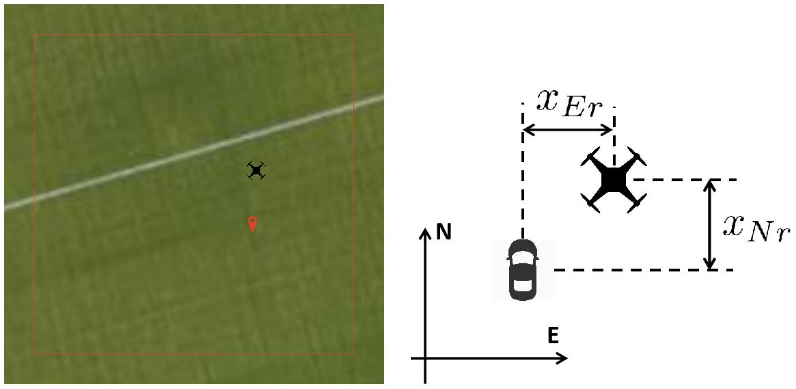

FIGURE 1. Case study for controlling a quadrotor tracking a ground vehicle with an example scenario (**Left**) and the coordinates (**Right**)

3.1. **Quadrotor-vehicle Tracking.** A scenario of a quadrotor tracking a moving vehicle is shown in Figure 1. In this scenario, a quadrotor is flying at a fixed altitude and tracking a vehicle moving on the ground. To control the quadrotor to track the moving vehicle, we need to devise a motion planner to provide the setpoints for the quadrotors. In this work, we formulate this quadrotor-vehicle tracking problem as a *discounted, continuing finite Markov decision process* (MDP)[1] [5] and train a deep reinforcement learning (DRL) agent as a setpoint provider for the quadrotor.

3.2. **State, Action and Observation Spaces.** The state of the quadrotor consists of four elements $s_{d_t} = (x_{E_{d_t}}, \dot{x}_{E_{d_t}}, x_{N_{d_t}}, \dot{x}_{N_{d_t}})$, representing the positions and velocities of the quadrotor at E and N directions, respectively. We omit the z_{d_t} and $\dot{z}_{d_t}$ since we assume the quadrotor is flying at a fixed altitude. Similarly, the state of the vehicle consists of four elements $s_{c_t} = (x_{E_{c_t}}, \dot{x}_{E_{c_t}}, x_{N_{c_t}}, \dot{x}_{N_{c_t}})$, representing the positions and velocities of the vehicle at E and N directions, respectively.

To decrease the learning space and make the learning easier, we take the relative positions and velocities between the quadrotor and the vehicle as: $s_{cd_t} = (x_{E_{dc_t}}, \dot{x}_{E_{cd_t}}, x_{N_{cd_t}}, \dot{x}_{N_{cd_t}})$ as the observations of the vehicle, in which the s_{cd_t} is calculated as $s_{cd_t} = s_{c_t} - s_{d_t}$. With this setting, the observations space can be derived as a tuple of eight elements:

$$o_t = (x_{E_{d_t}}, \dot{x}_{E_{d_t}}, x_{N_{d_t}}, \dot{x}_{N_{d_t}}, x_{E_{cd_t}}, \dot{x}_{E_{cd_t}}, x_{N_{cd_t}}, \dot{x}_{N_{cd_t}}). \tag{3.1}$$

Since vehicle's state s_{c_t} is observable, we can use it as the "guidance setpoints" and only train the agent to output the setpoints residuals $a_t = (x_{E_{dr_t}}, \dot{x}_{E_{dr_t}}, x_{N_{dr_t}}, \dot{x}_{N_{dr_t}})$ as the action to reach the optimal. Therefore, the final setpoints of the quadrotor are calculated as:

$$s_{T_t} = (x_{E_{d_t}} + x_{E_{dr_t}}, \dot{x}_{E_{d_t}} + \dot{x}_{E_{dr_t}}, x_{N_{d_t}} + x_{N_{dr_t}}, \dot{x}_{N_{d_t}} + \dot{x}_{N_{dr_t}}). \tag{3.2}$$

The idea of the residual training in this work is inspired by [6], where the analytical controller is used as baseline and the DRL only learns the residuals part to compensate the weakness of the analytical controller. In our training, we use the observed vehicle's states as baseline setpoint provider and use the DRL to learn the residuals to correct the baseline setpoints. The benefits of such hybrid settings is that the baseline setpoint provider can significantly improve the exploration efficiency, and the DRL agent can learn to deal with the

[1]The continuing task means the agent is interacting with its environment with a long life span with the final time step $T = \infty$. In this settings, a discount factor is introduced to discount the future expect returns to avoid the expect return at time step t being infinite.

uncertainties to further improve the tracking performance. Moreover, we add terminal states to the discounted, continuing finite MDP that stops the process when safety bounds are violated. A state is terminal if

$$\beta(s_t) = \begin{cases} 1, & \text{if } |x_{E_{d_t}}| \geq x_{max} \text{ or } |x_{N_{d_t}}| \geq y_{max} \\ 0, & \text{otherwise,} \end{cases} \tag{3.3}$$

i.e., if quadrotor exceeds the bounds of the flying zone.

3.3. **Reward Function.** In the MDP, the agent aims to minimize the Euclidean distance d between the position of the quadrotor and the position of the vehicle, where d is defined as:

$$d := \sqrt{(X_{E_r})^2 + (X_{N_r})^2}. \tag{3.4}$$

Then,we derive the reward function R based on distance d at the time step t as:

$$R(s_t, a_t, s_{t+1}) = e^{-\delta d_t} - v\beta(s_{t+1}). \tag{3.5}$$

The first term is rewarding smaller distances between the quadrotor and the vehicle. We chose the exponential function because it limits this term to $(0, 1]$ and has strong gradients when the distance is close to zero. This helps the agent to control the quadrotor to follow the vehicle closely. The second term penalizes safety violations. The parameter δ stretches the exponential, and the parameter v balances the weight of the penalty term.

3.4. **DNNs implementation and simulation training.** The actor and critic networks in the DDPG [7] algorithm are both implemented as a Multi-Layer Perceptron (MLP) with three fully connected hidden layers of 256, 128, and 64 neurons activated with ReLU. The observations o_t form the input of the actor, and the observations o_t and the action a_t form the input of the critic. The output layer of the actor contains four neurons activated with $Tanh$, and the output layer of the critic is a single neuron without activations. We split the training process into episodes with a maximum length of $T_e = 1000$ steps to evaluate the training performance. After every twenty training episodes, we run an evaluation episode, in which no exploration noise is applied. We train the agent in the simulation for one millions steps to converge. And the model evaluated with lowest moving average distance is saved for the following experiements.

References

[1] C. Reis, A. Barth, C. Pizano, Browser security: lessons from google chrome, Communications of the ACM 52 (8) (2009) 45–49.

[2] O. Kupferman, M. Y. Vardi, Model checking of safety properties, Formal Methods in System Design 19 (3) (2001) 291–314.

[3] B. Zhong, A. Lavaei, M. Zamani, M. Caccamo, Automata-based controller synthesis for stochastic systems: A game framework via approximate probabilistic relations, arXiv:2104.11803(2021) .

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540(2016) .

[5] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[6] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, S. Levine, Residual reinforcement learning for robot control, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 6023–6029.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: Proceedings of International Conference on Learning Representations(Poster), 2016.

[1]TUM School of Engineering and Design, Technical University of Munich, Germany.

Email address: bingzhuo.zhong@tum.de

Email address: cao.hongpeng@tum.de

[2]Department of Computer Science, University of Colorado Boulder, USA.

[3]Department of Computer Science, LMU Munich, Germany.