*فصلنامه علمی - پژوهشی فرماندهی و کنترل، سال ششم، شماره دو، تابستان 1401*

# ناوبری خودمختار ربات چرخ‌دار با رویکرد مبتنی‌بر یادگیری تقویتی عمیق

*کوروش داداش تبار[1]، علی اکبر کیایی[2]، محمد امین عباس زاده[3]*

**تاریخ دریافت:** 1400/11/24 **تاریخ پذیرش:** 1401/05/15

## چکیده

در این پژوهش به بررسی یک رویکرد مبتنی‌بر یادگیری تقویتی عمیق برای ناوبری خودمختار ربات‌ها می‌پردازیم. رویکرد ما در این پژوهش، مبتنی‌بر الگوریتم DDPG و یکی از نسخه‌های بهبود یافته‌ی آن به نام SD3 است. به‌منظور استفاده از این الگوریتم برای مسئله‌ی ناوبری خودمختار، اصلاحاتی بر روی الگوریتم مذکور انجام و برای کاربرد ناوبری بهینه‌سازی شده است. الگوریتم اصلاح شده به علت داشتن لایه‌های کانولوشنی می‌تواند با فضاهای حالت با ابعاد زیاد نیز کار کند. همچنین برای کاهش نوسان ربات در حین حرکت و نیز تشویق برای حرکت سریع‌تر در محیط، استفاده از دو پارامتر پاداش و جریمه براساس سرعت خطی و سرعت زاویه‌ای را پیشنهاد دادیم. و برای بهبود تعمیم پذیری الگوریتم، از الگوریتمی برای تغییر متناوب شکل و چینش موانع در محیط استفاده کردیم. همچنین برای تسریع فرایند یادگیری و بهبود عملکرد ربات، داده های ورودی را نرمال کردیم. سپس الگوریتم پیشنهادی را توسط محیط شبیه‌ساز GAZEBO و سیستم عامل ROS پیاده‌سازی کرده و نتایج بدست آمده را با الگوریتم اولیه‌ی SD3 و الگوریتم DDPG مقایسه نمودیم. الگوریتم پیشنهادی عملکرد بهتری نسبت به این دو روش به نمایش گذاشته است.

**واژگان کلیدی**: ناوبری خودمختار، یادگیری تقویتی عمیق، DDPG، SD3

[1] استادیار، دانشکده مهندسی برق و کامپیوتر دانشگاه صنعتی مالک اشتر (نویسنده‌ی مسئول) dadashtabar@yahoo.com

[2] استادیار، دانشکده مهندسی برق و کامپیوتر دانشگاه صنعتی مالک اشتر ali.a.kiaei@gmail.com

[3] دانشجوی کارشناسی ارشد، دانشکده مهندسی برق و کامپیوتر دانشگاه صنعتی مالک اشتر abbaszadeh.193@gmail.com

[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

## ۱. معرفی

ناوبری خودمختار یکی از مسائلی است که ربات‌ها را قادر می‌سازد تا در محیط‌های مختلف بتوانند وظایف خود را به انجام برسانند. به خاطر کاربرد گسترده‌ی ناوبری و عملکرد آن به‌عنوان بخشی کلیدی در ساختار ربات، روش‌های بسیار زیادی در طول سالیان مختلف ارائه شده است که اکثر آنها در مواجهه با محیط‌های واقعی پیچیده و شلوغ با چالش مواجه می‌شوند. با توسعه‌ی شبکه‌های عصبی عمیق، پژوهشگران آنها را در الگوریتم‌های ناوبری به‌کار گرفتند و این کار منجر به نتایج درخشانی در این زمینه شد. با پیشرفت روش‌های مبتنی‌بر یادگیری تقویتی عمیق در سال‌های اخیر، ناوبری ربات‌ها پیشرفت چشمگیری داشته است و به واسطه‌ی آن ربات‌ها وارد زندگی انسان‌ها شده‌اند. ربات‌هایی که قابلیت ناوبری در محیط‌های پیچیده و ناشناخته را دارند می‌توانند در کاربردهایی مثل جستجوی مصدومین در حوادث طبیعی، خدمات‌رسانی در محیط‌های عمومی‌مثل فرودگاه‌ها و رستوران‌ها، گشت‌زنی و جستجوی دشمن در محیط‌های نظامی و ... استفاده شوند.

در ابتدا از روش‌های کلاسیک برای حل مسائل ناوبری خودمختار ربات‌ها استفاده می‌شد. سپس روش‌های مبتنی‌بر یادگیری تقویتی مورد توجه قرار گرفتند. با توسعه‌ی شبکه‌های عصبی عمیق و عملکرد درخشان آنها، پژوهشگران به فکر استفاده از این شبکه‌ها برای تخمین زدن توابع مورد استفاده در الگوریتم‌های یادگیری تقویتی افتادند و به این ترتیب یادگیری تقویتی عمیق که ترکیبی از یادگیری تقویتی و یادگیری عمیق است پدید آمد. این روش‌ها عملکرد خوبی در بازی‌های ویدئویی به نمایش گذاشتند و پس از آن در حل مسائل ناوبری خودمختار ربات‌ها و نیز مسائل کنترلی به کار گرفته شدند [1].

### 1-1. مقدمه‌ای بر سیستم‌های ناوبری

ناوبری ربات‌ها به معنی قابلیت تعیین موقعیت خود ربات و سپس تولید یک مسیر مناسب و حرکت به سمت هدف است. الگوریتم‌های ناوبری از بخش‌های مختلفی تشکیل شده‌اند.

۱. نقشه‌برداری و محلی‌سازی
۲. تصمیم‌گیری و تعیین هدف
۳. برنامه‌ریزی مسیر
۴. برنامه‌ریزی حرکت
۵. واحد کنترل‌کننده

واحد *نقشه‌برداری و محلی‌سازی*[4] مسئول تهیه‌ی نقشه‌ی سه بعدی از محیط و سپس تعیین موقعیت خود در نقشه‌ی مورد نظر است که به این کار، محلی‌سازی می‌گوییم. همچنین سایز اطلاعات لازم از محیط نیز در این قسمت جمع‌آوری شده و در نهایت این اطلاعات به واحد بعدی ارسال می‌شوند. این بخش برای الگوریتم‌های مربوط به جستجو در محیط حیاتی است.

واحد تصمیم‌گیری و تعیین هدف بر اساس قواعد موجود در محیط و مدل‌های بدست آمده، هدف و یا ناحیه‌ی مورد نظر برای جستجو را تعیین کرده و این اطلاعات را به واحد بعدی ارسال می‌کند [2].

واحد *برنامه‌ریزی مسیر*[5] بر اساس هدف مشخص شده اقدام به طراحی مسیری می‌کند که بتواند در سریع ترین زمان ممکن و بدون برخورد با موانع به نقطه‌ی هدف مشخص شده برسد. مسیر برنامه‌ریزی شده به واحد بعدی داده می‌شود تا ربات مطابق آن به حرکت درآید [2].

واحد *برنامه‌ریزی حرکت*[6] در ربات‌هایی که سیستم دینامیکی پیچیده‌ای دارند، مثل بازوهای رباتیک، ربات‌های انسان نما، ربات‌های چهارپا و... یک واحد حیاتی است. در این سیستم‌ها نیاز است تا بخش‌های مختلف به‌صورتی حرکت داده شوند که کار مورد نظر به خوبی انجام شود. مثلا یک بازوی رباتیک باید طوری حرکت کند که بتواند جسمی را با زاویه‌ی مناسبی بلند کند. یا یک ربات فوتبالیست برای یک شوت محکم باید حالت خاصی به خود بگیرد که توسط واحد برنامه‌ریزی حرکت این کار انجام می‌شود. اما در ربات‌هایی که دینامیک ساده‌ای دارند مثل ربات‌های چرخ‌دار که حرکات ساده ای را انجام می‌دهند، معمولا از این واحد صرف نظر می‌شود [2, 3].

[4] Localization and mapping(SLAM)
[5] Path planning
[6] Motion planning



فصلنامهٔ علمی - پژوهشی فرماندهی و کنترل، سال ششم ، شمارهٔ دو، تابستان 1401

[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

واحد کنترل‌کننده یک کنترل‌کننده‌ی سطح پایین است که براساس مدل دینامیکی ربات، مسیر یا حرکت برنامه‌ریزی شده را دریافت کرده و با اعمال فرمان‌های مناسب برای ربات تلاش می‌کند تا مطابق مسیر یا حرکت طراحی‌شده ربات را به حرکت درآورد [2].

### 2-1. چالش‌های موجود در مسائل ناوبری

برخی از محیط‌های واقعی که دارای پیچیدگی زیادی هستند برای الگوریتم‌های ناوبری چالش‌های زیادی به همراه دارند. الگوریتم ناوبری برای این که در محیط‌های واقعی کارایی مناسبی داشته باشد باید بتواند از منظرهای گوناگون ویژگی‌های مناسبی داشته باشد که به برخی از آنها اشاره می‌کنیم.

**قابلیت تعمیم‌پذیری**

الگوریتم ناوبری باید بتواند در محیط‌های گوناگون عملکرد مناسبی داشته باشد و به عبارتی قابل تعمیم یافتن به محیط‌های مختلفی باشد. الگوریتم‌هایی که صرفا برای یک محیط خاص طراحی می‌شوند به سادگی می‌توانند با قرارگیری ربات در محیطی متفاوت کارایی خود را از دست بدهند. همچنین الگوریتم ناوبری باید طوری طراحی شود که نسبت به تغییرات در پارامترهای هر محیط نیز مقاوم باشد [4].

**قابلیت پردازش بلادرنگ**

الگوریتم طراحی‌شده نباید به‌قدری سنگین باشد که اجرای آن زمان زیادی ببرد. چرا که در محیط‌های واقعی، مخصوصا محیط‌هایی که عامل‌های دینامیکی مختلفی حضور دارند، ربات باید بتواند واکنش سریعی داشته باشد تا منجر به برخورد با عامل‌های دیگر نشود و به موقع تصمیم‌گیری کند [4].

**فضای عمل پیوسته**

از آنجا که سیگنال‌های کنترلی ربات‌ها در محیط‌های واقعی اغلب پیوسته هستند، فضای عمل الگوریتم ناوبری نیز باید پیوسته باشد [2].

**مشاهده‌ی سودمند ربات از محیط**

مشاهده‌ی ربات از محیط باید به اندازه‌ای دارای اطلاعات مفید باشد که ربات بتواند به خوبی به هدف خود دست پیدا کند. بنابراین سنسورها باید اطلاعات مفیدی از محیط جمع‌آوری کرده و تلفیق این اطلاعات به‌عنوان مشاهده‌ی ربات از محیط استفاده شود [4].

## 2. مفاهیم پایه

در این بخش به تشریح الگوریتم عملگر-منتقد[5] به‌عنوان یکی از الگوریتم‌های پرکاربرد در یادگیری تقویتی می‌پردازیم. ابتدا مفاهیم اولیه‌ی یادگیری تقویتی را مرور می‌کنیم. مفاهیم و اصطلاحات پایه ای و پرکاربرد در یادگیری تقویتی به شرح زیر هستند [5].

*اقدام (Action|A)*: شامل تمامی‌واکنش‌های محتملی است که عامل تصمیم‌گیرنده ممکن است در مواجهه با وضعیت ایجاد شده، از خود نشان دهد.

*پاداش (Reward|R)*: بازخورد فوری که پس از ارزیابی هر اقدام عامل تصمیم‌گیرنده، توسط محیط برای آن ارسال می‌شود.

*سیاست ($Policy|\pi$)*: یک استراتژی است که عامل تصمیم‌گیرنده در پاسخ به وضعیت فعلی، برای اقدام بعدی خود در نظر می‌گیرد.

*ارزش (Value|V)*: پاداش بلندمدت مورد انتظار7 تنزیل شده8 که از روی پاداش‌های کوتاه‌مدت (R) به‌دست می‌آید. ارزش به معنی سود مورد انتظار در بلندمدت است که ناشی از وضعیت کنونی s تحت سیاست $\pi$ است.

*Q-value یا اقدام-ارزش (Q)*: مفهوم Q-value بسیار شبیه به مفهوم ارزش (V) است. اما در Q-value یک پارامتر بیشتر وجود دارد. این پارامتر اضافه همان اقدام a می‌باشد. $Q(s,a)$ عبارت است از سود بلندمدت ناشی از اقدام a تحت سیاست $\pi$ در وضعیت کنونی s.

### 2-1. الگوریتم‌های بدون مدل و الگوریتم‌های مبتنی‌بر مدل

مدل یک شبیه‌سازی از پویایی محیط است. یعنی اینکه مدل، تابع احتمال انتقال $T(s'|(s,a))$ را یاد می‌گیرد. اگر یادگیری تابع انتقال احتمال موفقیت‌آمیز باشد، عامل تصمیم‌گیرنده می‌تواند احتمال رخ دادن یک وضعیت مشخص را بر اساس وضعیت و اقدام فعلی محاسبه کند. اما با افزایش تعداد

[7] Expected return

[8] Discounted





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

وضعیت‌ها و اقدام‌ها، الگوریتم‌های مبتنی‌بر مدل کارآمدی خود را از دست می‌دهند [6].

از سوی دیگر، الگوریتم‌های بدون مدل در واقع مبتنی‌بر روش آزمون و خطا هستند و براساس نتیجه آن، دانش خود را به‌روزرسانی می‌کنند. در نتیجه، فضایی برای ذخیره ترکیبات احتمالی وضعیت‌ها و اقدام‌ها نیاز نخواهند داشت [5]. الگوریتم‌هایی که در قسمت بعدی معرفی می‌شوند، همگی در این دسته قرار دارند.

### 2-2. روش On-policy و روش Off-policy

در روش On-policy، عامل تصمیم‌گیرنده ارزش را براساس اقدام a که ناشی از سیاست فعلی است، یاد می‌گیرد. اما در روش دیگر یعنی روش Off-policy فرایند یادگیری به این صورت است که عامل، ارزش را از اقدام $a^*$ (بهترین اقدام موجود) که نتیجه یک سیاست دیگر می‌باشد، می‌آموزد [5].

### 2-3. مروری بر روش عملگر-منتقد

برای درک عملکرد بهتر روش عملگر-منتقد ابتدا روش *گرادیان سیاست*[9][5] را بررسی می‌کنیم. در این روش داشتیم:

$$\nabla_\theta J(\theta) = E_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t] \quad (1)$$

در این معادله، $\theta$ بیانگر پارامترهای تابع تخمین زننده‌ی $\pi$ و $G_t$ بیانگر مجموع تنزیل یافته‌ی پاداش‌ها است. معادله‌ی بالا را به‌صورت زیر نیز می‌توان نوشت:

$$\nabla_\theta J(\theta) = E_{s_0,a_0,\dots,s_t,a_t} [\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)] E_{r_{t+1},s_{t+1},\dots,r_T,s_T}[G_t] \quad (2)$$

در معادله‌ی بالا، امید ریاضی دوم معادل مقدار Q است:

$$E_{r_{t+1},s_{t+1},\dots,r_T,s_T}[G_t] = Q(s_t, a_t) \quad (3)$$

بنابراین می‌توان معادله‌ی 1 را به‌صورت زیر نوشت. در این معادله $\omega$ بیانگر تابع منتقد در الگوریتم عملگر-منتقد است.

$$\begin{aligned} &\nabla_\theta J(\theta) \\ &= E_{s_0,a_0,\dots,s_t,a_t}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] Q_\omega(s_t, a_t) \\ &= E_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] Q_\omega(s_t, a_t) \end{aligned} \quad (4)$$

در معادله‌ی بالا مقدار $Q_w$ می‌تواند توسط یک شبکه‌ی عصبی یاد گرفته شود. این ایده ما را به روش عملگر-منتقد می‌رساند:

منتقد تابع ارزش را تخمین می‌زند. این تابع ارزش می‌تواند ارزش هر عمل (مقدار Q) یا ارزش هر حالت (مقدار V) باشد. عملگر توزیع سیاست را در راستایی که منتقد پیشنهاد می‌کند اصلاح می‌کند.

هر دو بخش عملگر و منتقد توسط شبکه‌های عصبی پارامتریزه می‌شوند. اگر منتقد مقدار Q را تخمین بزند، الگوریتم حاصل را Q-Actor Critic[5] می‌نامیم.

**جدول 1 شرح الگوریتم Q Actor Critic**

**Algorithm 1** Q Actor Critic

Initialize parameters $s, \theta, \omega$ and learning rates $\alpha_\theta, \alpha_\omega$; sample $a \sim \pi_\theta(a|s)$

**for** episode = 1, T **do**

Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$

Then sample next state and action $a' \sim \pi_\theta(a'|s')$

Update policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_\omega(s,a) \nabla_\theta \log \pi_\theta(a|s)$

Compute the correction (TD error) for action-value at time t:

$$\delta_t = r_t + \gamma Q_\omega(s', a') - Q_\omega(s,a)$$

And use it to update the parameters of Q function:

$$\omega \leftarrow \omega + \alpha_\omega \delta_t \nabla_\omega Q_\omega(s,a)$$

Move to $a \leftarrow a'$ and $s \leftarrow s'$

**end for**

## 3. مروری بر کارهای پیشین

کارهای انجام شده برای حل مساله‌ی ناوبری خودمختار ربات‌ها را می‌توان به دو دسته‌ی روش‌های کلاسیک و روش‌های مبتنی بر یادگیری تقسیم کرد.

[9] Policy gradient





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

### 3-1. روش‌های کلاسیک

یکی از روش‌های پرکاربرد غیر مبتنی‌بر یادگیری، اجتناب از برخورد با موانع پس از شناسایی آنها و تغییر مسیر دادن و سپس برنامه‌ریزی مسیر است[7]. برای مثال یکی از روش‌ها از تکنیک *شار نوری*[10] برای شناسایی موانع و عدم برخورد با آنها استفاده می‌کند و پس از آن از یک الگوریتم درخت جستجوی تصادفی سریع برای برنامه‌ریزی مسیر استفاده می‌کند[8].

یکی از راه‌های حل مسائل ناوبری خودمختار در محیط‌های ناشناخته، جمع‌آوری قواعد تصمیم‌گیری است. اما علاوه بر دشوار بودن این کار، ربات قابلیت تعمیم‌پذیری ندارد و نمی‌تواند در محیط‌های مختلف فعالیت کند. یکی دیگر از این روش‌ها، روش‌های کنترلی مبتنی‌بر یادگیری هستند. اما این روش‌ها هم راندمان پایینی در یادگیری دارند و در انتقال از محیط شبیه‌سازی به محیط واقعی دچار ضعف هستند.

نقطه ضعف روش‌های کلاسیک ناوبری، عدم انطباق با محیط‌های گوناگون است به این دلیل که به ویژگی‌های محیط و مفروضات مربوط به آن وابستگی زیادی دارند. بنابراین این روش‌ها قابلیت تعمیم‌پذیری اندکی دارند. همچنین طراحی مکانیزمی بر اساس این روش‌ها که هم راندمان محاسباتی بالایی داشته باشد و هم راندمان آن بالا باشد کاری دشوار است[6].

### 3-2. روش‌های مبتنی بر یادگیری

به خاطر مشکلات الگوریتم‌های غیر مبتنی‌بر یادگیری مثل عملکرد نامناسب در محیط‌های پیچیده، روش‌های دیگری که مبتنی‌بر یادگیری هستند ارائه شد که از آنها می‌توان به روش‌های مبتنی‌بر یادگیری تقویتی و *یادگیری تقلیدی*[11] اشاره کرد. یادگیری تقویتی قادر به حل چنین مسائلی با استفاده از حافظه ای شامل اقدام‌ها و مشاهدات گذشته است. در کل روش‌های مبتنی‌بر یادگیری انواع مختلفی دارند و با محیط‌های ناشناخته بهتر می‌توانند تطابق پیدا کنند و قابلیت تعمیم‌پذیری بالایی دارند.

از طرفی با توجه به ویژگی‌هایی که در قسمت‌های قبل برای محیط بیان شد، معمولا ربات قادر به مشاهده‌ی تمام محیط نیست. برای غلبه بر این مشکل مجبور به استفاده از تاریخچه‌ی مشاهدات از محیط هستیم تا در مواجه شدن با هر موقعیتی بتوانیم تصمیمات درستی اتخاذ کنیم. با توجه به این توضیحات، تصمیم‌گیری در روش‌های مبتنی‌بر یادگیری تقویتی باید در قالب فرایندهای تصمیم‌گیری مارکف قابل مشاهده به‌صورت جزئی انجام شود. طبق این فرایند، عامل در حالت جدید، مشاهده‌ی خود از آن حالت را از طریق سنسورها دریافت می‌کند، سپس براساس آن اقدام به تصمیم‌گیری کرده و عملی انجام می‌دهد و به موجب آن عمل، پاداش یا جریمه دریافت می‌کند.

از بین روش‌های مبتنی‌بر یادگیری تقویتی، روش‌های مبتنی‌بر *گرادیان سیاست*[12] مثل روش‌های نسبت درست نمایی[9] و روش‌های عملگر-منتقد برای فرایندهای بدون مدل POMDP با فضای اقدام پیوسته، با انجام عملگر گرادیان نزولی در فضای پارامترها، سیاست‌های تصادفی را استخراج می‌کنند. از طرفی یادگیری تقویتی در حالتی که در آن از تخمین گر تابع غیر خطی برای تخمین تابع ارزش استفاده شود به یک الگوریتم ناپایدار تبدیل می‌شود[10].

از طرفی در روش‌های مبتنی‌بر یادگیری تقویتی نیز باید ویژگی‌های حالت ربات را بشناسیم. همچنین این روش‌ها از مشکل *نفرین ابعاد*[13] رنج می‌برند. به همین دلیل کاربرد روش‌های مبتنی‌بر یادگیری تقویتی به محیط‌های کاملا قابل مشاهده با فضاهای حالتی با ابعاد کم محدود می‌شود.

به همین دلیل برای ارائه‌ی یک روش یکپارچه برای فرایندهای تصمیم‌گیری مارکف و همچنین فرایندهای POMDP، *یادگیری تقویتی عمیق*[14] توسعه داده شده است که ترکیبی از یادگیری تقویتی و یادگیری عمیق است. یادگیری تقویتی عمیق می‌تواند خروجی خام سنسورها را به‌عنوان ورودی دریافت کرده و آنها را به سیگنال‌های کنترلی تبدیل کند.

روش‌های مبتنی‌بر یادگیری تقویتی عمیق قادر هستند تا ویژگی‌های حالت‌ها را به‌صورت اتوماتیک بیاموزند و می‌توانند ابعاد را کاهش دهند[11]. در واقع این روش‌ها، ویژگی‌های حالتی که متعلق به حالت‌هایی با ابعاد زیاد هستند را از روی

[10] Optical flow
[11] Imitation learning
[12] Policy gradient
[13] Curse of dimensionality
[14] Deep Reinforcement Learning(DRL)





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

داده‌های سنسورها یاد می‌گیرند و بدین ترتیب مشکل ابعاد زیاد را حل می‌کنند. بنابراین روش‌های مذکور می‌توانند به خوبی با محیط‌های ناشناخته و شلوغ کنار بیایند.

یادگیری تقویتی عمیق، سیاست‌های کنترلی بهینه را بر اساس سعی و خطا بدست می‌آورد . این موضوع می‌تواند منجر به وارد شدن آسیب‌هایی به ربات شود. برای غلبه بر این مشکل، الگوریتم ما باید قادر به یادگیری با تعداد اندکی نمونه در دنیای واقعی باشد. یکی از الزامات این امر، توانایی به‌روزرسانی پارامترها در هر *گام*[15] است؛ بدون این که نیاز باشد تا انتهای اپیزود صبر کنیم. این موضوع عامل را قادر می‌سازد تا در محیط‌های ناشناخته بهتر عمل کند و سریع تر یاد بگیرد.

در زمینه‌ی روش‌های مبتنی‌بر یادگیری تقویتی عمیق، در ابتدا برای حل مسائل مربوط به کنترل، الگوریتم DQN ارائه شد[10]. این الگوریتم برای حل مسائلی که دارای فضای عمل با ابعاد محدود و گسسته هستند عملکرد نسبتا خوبی را به نمایش می‌گذارد. الگوریتم DQN برای یادگرفتن بازی‌های آتاری از روی پیکسل‌های تصویر ورودی با موفقیت به کار گرفته شد. اما برای مسائلی که دارای فضای عمل پیوسته و با ابعاد بالا هستند کارایی خود را از دست می‌دهد. همچنین در محیط‌هایی که دارای فضای حالت پیچیده هستند عملکرد خوبی ندارد.

برای حل مسائلی که دارای فضای اقدام پیوسته با ابعاد زیاد هستند، الگوریتم DDPG ارائه شد[12]. این الگوریتم در چنین فضاهایی عملکرد بهتری نسبت به الگوریتم DQN دارد. در الگوریتم DDPG برای رفع مشکل واگرایی در تخمین مقدار Q از روش به‌روزرسانی نرم مقدار هدف استفاده می‌شود. همچنین برای این که داده‌ها از نظر زمانی از یکدیگر مستقل شوند، استفاده از یک حافظه‌ی تکرار مجدد پیشنهاد شده است.

الگوریتم TD3 برای بهبود نواقص الگوریتم DDPG ارائه شد[13]. یکی از مشکلات الگوریتم DDPG مشکل *تخمین بیش از اندازه‌ی*[16] مقدار هدف است[13]. در الگوریتم TD3 برای رفع این مشکل از دو شبکه‌ی مستقل از هم برای تخمین مقدار Q استفاده شده و سپس مقدار هدف توسط معادله‌ی زیر تخمین زده می‌شود.

$$y_i = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \pi_{\phi_1}(s')) \qquad (5)$$

در معادله‌ی بالا، $\phi_1$ بیانگر پارامترهای تابع اول تخمین‌زننده‌ی سیاست است و $\theta_i'$ بیانگر پارامترهای تابع هدف تخمین‌زننده‌ی Q است.

هرچند این روش می‌تواند باعث بروز *بایاس کمتر تخمین زدن*[17] بشود اما چون خطاهای مربوط به کمتر تخمین زدن بر خلاف خطاهای مربوط به تخمین بیش از اندازه، در فرایند پس‌انتشار خطا کمتر در شبکه منتشر می‌شوند، بنابراین این معادله برای ما بهتر است.

در ادامه برخی از روش‌های مبتنی بر یادگیری تقویتی عمیق که برای کاربرد ناوبری خودمختار ربات‌ها بهینه‌سازی شده‌اند را بررسی می‌کنیم.

روش SA-CADRL برای ناوبری در محیط‌های غنی از عابران پیاده طراحی‌شده است[4]. در مقایسه با الگوریتم‌های ناوبری، انسان‌ها به‌طور شهودی می‌توانند متوجه شوند که آیا یک رفتار قابل قبول است یا خیر. ناوبری انسان‌ها از نظر زمانی بهینه است و به‌طور کلی بر اساس مجموعه ای از هنجارهای ساده‌ی اجتماعی انجام می‌شود (مثلا راه رفتن از سمت راست).

در این روش یک شبکه‌ی عصبی عمیق طراحی می‌شود که متقارن بوده و اندازه‌ی آن متناسب با حداکثر تعداد عامل‌های مجاور ربات است که می‌خواهیم رفتار آنها را درنظر بگیریم. ورودی‌های شبکه‌ی عصبی به جای داده‌های خام، شامل دو بردار حالت است. بردار حالت اول متعلق به خود ربات بوده و شامل فاصله‌ی ربات تا هدف، سرعت مطلوب، سرعت ربات در جهت x، سرعت در جهت y و جهت حرکت ربات است. بردار حالت دوم مربوط به سایر عامل‌های موجود در محیط(عابران پیاده) است که شامل فاصله‌ی عامل‌های دیگر تا ربات، موقعیت عامل‌ها، سرعت عامل‌ها و جهت حرکت آنها است.

[15] Step
[16] Over estimation
[17] Under estimation





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

شبکه این دو بردار حالت را به‌عنوان ورودی می‌گیرد و مقدار ارزش هر حالت را به‌عنوان خروجی بر می‌گرداند. برای این که قواعد اجتماعی مثل حفظ حداقل فاصله با عامل‌های دیگر، حرکت از سمت راست و ... توسط ربات یاد گرفته شوند، پارامترهایی بر اساس این قواعد در تابع پاداش قرار داده می‌شود. با ماکزیمم کردن مقدار پاداش، ربات این قواعد را نیز می‌آموزد. در نهایت با استفاده از *سیاست حریصانه*[18] و ارزش‌های تولید شده توسط شبکه، مسیرهای مناسبی تولید می‌شود.

روش GA3C-CADRL ، توسعه‌ی روش SA-CADRL است[14]. در این روش عامل بدون درنظر گرفتن هیچ قاعده‌ی رفتاری خاصی برای عامل‌های موجود در محیط، قادر خواهد بود تا در محیط‌هایی که عامل‌های دینامیکی دیگری نیز حضور دارند، بدون برخورد با آنها به مسیر خود ادامه دهد. همچنین یکی از مشکلات، تعداد عامل‌های موجود در اطراف ربات است. در روش‌های قبلی، تعداد عامل‌های موجود در محیط عدد ثابتی فرض می‌شد و یا از داده‌های خام سنسورها به‌عنوان ورودی شبکه استفاده می‌کردند اما در این روش با به‌کارگیری یک ساختار LSTM ربات قادر به مشاهده‌ی تعداد دلخواهی عامل در اطراف خود خواهد بود و محدودیتی در این زمینه نخواهد داشت.

همچنین در روش‌های پیشین مبتنی‌بر یادگیری، یک سری مفروضات قبلی برای سایر عامل‌ها مثل همگونی[15] و یا مدل رفتاری خاص در بازه‌های زمانی کوتاه[16] در نظر گرفته می‌شد. اما در روش ارائه شده در این مقاله با توسعه‌ی روش‌های قبلی[4, 16] هیچ مفروضاتی برای عامل‌ها در نظر گرفته نمی‌شود.

روش LSTM_LMC نیز برای ناوبری در محیط‌های غنی از عابرین پیاده طراحی‌شده است. اکثر روش‌های مبتنی‌بر یادگیری تقویتی عمیق به محدوده‌ی دید وسیعی نیاز دارند که توسط سنسورهای لیدار تامین می‌شود. در این مقاله روشی برای استفاده از دوربین‌های استریو که محدوده‌ی دید کمتری دارند اما ارزان تر هستند ارائه می‌شود[17]. در این روش برای غلبه بر محدودیت دید دوربین‌های استریو از شبکه‌های عصبی بازگشتی و همچنین از ساختار منتقد نقشه‌ی محلی استفاده می‌شود. همچنین به خاطر محدودیت دید دوربین‌های استریو، محیط به‌صورت POMDP مدل شده است.

همچنین فرزاد نیرویی و دیگران[18] یک روش مبتنی‌بر A3C برای جستجو در محیط‌های شلوغ و ناشناخته ارائه کردند. در مسائل جستجو ربات باید بتواند مناطق مناسب برای جستجو را به خوبی تشخیص داده و سپس با اجتناب از موانع به سمت هدف حرکت کرده و طی مسیر کند.

روش AFCQN نیز برای جستجو در محیط‌های ناشناخته و پیچیده ارائه شده است. ایده‌ی اصلی این روش استفاده از یک ساختار FCQN به همراه یک ساختار کمکی(Auxiliary) برای *قطعه‌بندی*[19] کردن لبه‌ها است. به همین دلیل این روش، AFCQN نام گذاری شده است. این ساختار کمکی باعث می‌شود الگوریتم نسبت به روش‌های ناوبری مبتنی‌بر ساختار FCQN کارایی بهتری داشته باشد[2].

عملکرد AFCQN به این صورت است که بخش FCQN مسئول تولید ارزش حالت‌های مختلف است و یک شاخه از خروجی لایه‌های کانولوشنی از دو لایه‌ی کانولوشن معکوس عبور کرده و خروجی هم اندازه‌ی نقشه‌ای خواهد بود که به‌عنوان ورودی شبکه‌ی FCQN در نظر گرفته شده است. این نقشه حاوی لبه‌های محیط شامل لبه‌های مربوط به موانع و نواحی مجهول می‌باشد. در این روش ماژول تصمیم از ماژول برنامه ریزی مسیر جدا شده است و این دو ماژول به‌صورت مستقل از هم عمل می‌کنند. ماژول تصمیم‌گیری توسط یک شبکه‌ی FCQN کار می‌کند و ماژول برنامه‌ریزی با الگوریتم کلاسیک A* عمل برنامه‌ریزی را انجام می‌دهد.

برخلاف روش‌های قبلی که تلاش زیادی برای ایجاد یک تابع پاداش متراکم و کامل می‌کردند، تابع پاداش در روش LwH بسیار تنک[20] طراحی‌شده است[1]. عامل تنها در زمانی که به هدف برسد پاداشی دریافت می‌کند.

[18] Greedy policy
[19] Segment
[20] Sparse





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

مزیت تابع پاداش تنک این است که راه حل در یک جهت غیر بهینه بایاس نمی‌شود. اما حذف پاداش‌های میانی باعث می‌شود یادگیری به‌صورت موثر انجام نشود. در این روش برای حل این مشکل از یک سیاست با راندمان پایین در شروع یادگیری استفاده می‌شود تا عامل در ابتدا تشویق به جستجو در محیط شود.

نتایج این الگوریتم از نظر سرعت همگرایی، نرخ موفقیت و بازدهی نهایی بسیار بهتر از سایر الگوریتم‌های با تابع پاداش تنک است.

در ادامه برخی از کارهای انجام شده برای پیاده‌سازی الگوریتم‌های مطرح را به اختصار بررسی می‌کنیم.

بوحامد و دیگران[19, 20] الگوریتم DDPG را برای ناوبری یک کوادروتور به‌کار گرفتند. ریکاردو و دیگران[21] دو الگوریتم SAC[21] و DDPG را به‌طور جداگانه برای ناوبری یک ربات هیبریدی آبی-هوایی استفاده کرده و نتایج را با یکدیگر مقایسه کردند. در این مطالعه الگوریتم SAC عملکرد بهتری به نمایش گذاشته است. همچنین گراندو و دیگران[22] در مطالعه ای دیگر الگوریتم‌های SAC و DDPG را برای ناوبری یک کوادروتور به کار بردند که در این مطالعه نیز الگوریتم SAC عملکرد بهتری را به نمایش گذاشته است.

ژیائوشان و دیگران[23] الگوریتم DDPG را برای ناوبری یک ربات چرخ‌دار بهینه‌سازی کردند. آنها در این مطالعه از دو حافظه برای الگوریتم DDPG استفاده کرده‌اند. حافظه‌ی اول حاوی داده‌های با ارزش (داده‌هایی که در آنها عامل به موانع برخورد نکرده) و حافظه‌ی دوم حاوی داده‌های مربوط به شکست (داده‌هایی که در آنها عامل به موانع برخورد کرده است) است. آموزش به‌صورت ترکیبی از این دو حافظه انجام شده و نتایج بهتری در زمان کمتر حاصل شده است.

جسوس و دیگران[24, 25] برای ناوبری یک ربات چرخ‌دار در محیط شبیه‌ساز Gazebo توسط سیستم عامل ROS از الگوریتم DDPG استفاده کرده‌اند. پس از آموزش در محیط شبیه‌ساز، عامل آموزش داده شده را در محیط واقعی تست کرده‌اند.

چانگ و دیگران[26] الگوریتم‌های RDPG و DDPG را برای ناوبری یک ماشین خودران در محیط شبیه‌ساز AirSim استفاده کرده و نتایج را با یکدیگر مقایسه کرده‌اند. در این مطالعه الگوریتم DDPG به خاطر نداشتن لایه‌های بازگشتی در مسیرهای U شکل عملکرد خوبی نداشته، در حالی که الگوریتم RDPG در این مسیرها موفق عمل کرده است.

کاهن و دیگران[27] یک شبکه‌ی عصبی کانولوشنی را در ترکیب با یک بلوک LSTM برای ناوبری بر اساس تصاویر خام به‌عنوان ورودی طراحی کرده‌اند. این الگوریتم BADGR نام دارد. یکی از ویژگی‌های جالب این الگوریتم این است که آموزش ربات در محیط واقعی انجام می‌شود و در طول آموزش ربات یاد می‌گیرد که چمن‌ها و علف‌های موجود در مسیر را به‌عنوان مانع در نظر نگیرد و از روی آنها رد شود.

سورمان و دیگران[28] الگوریتم GA3C را برای ناوبری یک ربات چرخ‌دار به‌کار گرفتند. در این مطالعه ابتدا عامل در محیط شبیه‌ساز آموزش داده شده و سپس در محیط واقعی نیز تست شده است. در این مطالعه از ترکیب یک دوربین تشخیص عمق و یک سنسور یک بعدی لیدار استفاده شده است. داده‌های این دو سنسور با یکدیگر *ادغام*[22] شده است و یک تصویر واحد به‌دست آمده است که به‌عنوان حالت ورودی الگوریتم استفاده می‌شود. همچنین به علت ابعاد بالای فضای حالت ورودی که یک تصویر عمق از محیط است از لایه‌های کانولوشنی در شبکه استفاده شده است.

## 4. روش پیشنهادی

در کارهای پیشین، الگوریتم DDPG به‌منظور استفاده در کاربرد ناوبری خودمختار ربات‌ها به شیوه‌های گوناگونی بهبود داده شده است.

یکی از نسخه‌های موفق الگوریتم DDPG الگوریتم SD3 است. سرعت همگرا شدن این الگوریتم بیشتر از الگوریتم DDPG است و در محیط‌های مختلف نتایج بهتری تولید می‌کند. اما تا کنون از این الگوریتم برای حل مسائل ناوبری استفاده نشده است.

[21] Soft actor critic

[22] Fusion





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

در این پژوهش الگوریتم SD3 را برای کاربرد ناوبری بهینه‌سازی کرده‌ایم. الگوریتم SD3 تقریبا مشابه الگوریتم TD3 است. در این الگوریتم برای رفع مشکل *تخمین کمتر از حد*[23] که در الگوریتم TD3 وجود دارد، استفاده از تابع *سافتمکس بولتزمن*[24][29] پیشنهاد شده است.

اما اعمال تابع سافتمکس بولتزمن به‌صورت مستقیم به الگوریتم TD3، یعنی اعمال این تابع به خروجی دو شبکه‌ی تولید کننده‌ی Q این مشکل را بدتر می‌کند. داریم:

$$y_i = r + \gamma \min\left(T_{SD2}^{-i}(s'), Q_i\left(s', \pi(s'|\phi^-)|\theta_i^-\right)\right), \quad (6)$$
$$T_{SD2}^{-i}(s') = softmax_\beta(Q_{-i}(s', a', \theta_{-i}^-))$$

در معادله‌ی بالا اثبات می‌شود که $T_{SD2}^{-i}(s') \leq Q_i(s', \pi(s'|\phi^-)|\theta_i^-)$ و به همین دلیل مشکل تخمین کمتر از حد در این حالت بدتر می‌شود[30].

در روش SD3 پیشنهاد می‌شود که مقدار هدف برای منتقد $Q_i$ توسط فرمول $y_i = r + \gamma T_{SD3}(s')$ تخمین زده شود [30]. داریم:

$$T_{SD3}(s') = softmax_\beta\left(\hat{Q}_i(s', a')\right), \quad (7)$$
$$\hat{Q}_i(s', a') = \min(Q_i(s', a', \theta_{-i}^-), Q_{-i}(s', a', \theta_{-i}^-))$$

در این الگوریتم علاوه بر دو شبکه برای منتقد، برای عملگر هم دو شبکه استفاده شده است. این کار بار محاسباتی را اندکی بیشتر می‌کند اما منجر به نتایج بهتری می‌شود. همچنین به‌ازای هر *انتقال*[25] $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'})$ ، K نمونه تولید می‌شود. این K نمونه *حالت‌های*[26] مشابهی دارند اما به اقدام‌های متناظر با هر یک از نمونه‌ها نویز اضافه شده است. در هر مرحله از آموزش تمام این K نمونه استفاده می‌شوند. این کار منجر به تعمیم‌پذیری بهتر شبکه در ازای تحمیل مقدار ناچیزی بار پردازشی بیشتر می‌شود.

برای این که عامل توانایی کار با فضاهای حالت با ابعاد بالا را داشته باشد از لایه‌های کانولوشنی استفاده کرده‌ایم. الگوریتم حاصل را به اختصار SD3_Nav می نامیم. در ادامه موارد مهم استفاده شده در روش پیشنهادی را به اختصار بررسی می‌کنیم.

**جدول 2 شرح الگوریتم SD3**

**Algorithm 2** SD3 algorithm

Initialize critic network $Q_1, Q_1$, and actor networks $\pi_1, \pi_2$ with random parameters $\theta_1, \theta_2, \varphi_1, \varphi_2$

Initialize target networks $\theta_1^- \leftarrow \theta_1$, $\theta_2^- \leftarrow \theta_2$, $\varphi_1^- \leftarrow \varphi_1$, $\varphi_2^- \leftarrow \varphi_2$

Initialize replay buffer $\beta$

**for** t = 1 to T **do**

  Select action $a$ with exploration noise $\epsilon \sim \mathcal{N}(0, \sigma)$ based on $\pi_1$ and $\pi_2$ .

  Execute action $a$, observe reward $r$, new state $s'$ and done $d$.

  Store transition tuple $(s, a, r, s', d)$ in $\beta$ // $d$ is done flag.

  **for** i=1, 2 **do**

    Select a minibatch of N transitions $\{(s, a, r, s', d)\}$ from $\beta$

    Sample K noises $\epsilon \sim \mathcal{N}(0, \bar{\sigma})$

    $\hat{a}' \leftarrow \pi_i(s'|\emptyset_i^-) + clip(\epsilon, -c, c)$

    $\hat{Q}(s', \hat{a}') \leftarrow min_{j=1,2}(Q_j(s', \hat{a}', \theta_j^-))$

    $$\leftarrow E_{\hat{a}' \sim p}\left[\frac{\exp\left(\beta \hat{Q}(s', \hat{a}')\right) \hat{Q}(s', \hat{a}')}{p(\hat{a}')}\right] \Big/ E_{\hat{a}' \sim p}\left[\frac{\exp\left(\beta \hat{Q}(s', \hat{a}')\right)}{p(\hat{a}')}\right]$$

    $y_i \leftarrow r + \gamma(1-d) softmax_\beta\left(\hat{Q}(s', .)\right)$

    Update the critic $\theta_i$ according to Bellman loss:

    $$\frac{1}{N}\sum_s (Q_i(s, a|\theta_i) - y_i)^2$$

    Update actor $\varphi_i$ by policy gradient:

    $$\frac{1}{N}\sum_s \left[\nabla_{\varphi_i}(\pi(s|\varphi_i)) \nabla_a Q_i(s, a|\theta_i)|_{a=\pi(s|\varphi_i)}\right]$$

    Update target networks: $\theta_i^- \leftarrow \tau\theta_i(1-\tau)\theta_i^-$, $\varphi_i^- \leftarrow \tau\varphi_i + (1-\tau)\varphi_i^-$

  end for

end for

## 4-1. تعمیم‌پذیری

یکی از چالش‌های موجود در آموزش عامل‌ها برای مساله‌ی ناوبری، مشکل تعمیم‌پذیر بودن است. باید عامل طوری آموزش

[23] Under estimation
[24] Boltzmann Softmax
[25] Transition
[26] State





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

داده شود که در محیط‌های جدید که چینش و شکل موانع تغییر می‌کند، عامل به مشکل برنخورد. برای جلوگیری از بایاس شدن عامل به موانع محیط و به‌منظور تعمیم‌پذیری بهتر، از یک الگوریتم استفاده کرده‌ایم که در هر K اپیزود به‌صورت تصادفی چینش و شکل فیزیکی موانع موجود در محیط را تغییر می‌دهد. به این ترتیب عامل در محیط‌های جدید که چینش موانع متفاوت خواهد بود عملکرد بهتری دارد.

### 4-2. تابع پاداش

برای این که عامل یاد بگیرد به‌صورت مناسب در محیط حرکت کند باید تابع پاداش را طوری طراحی کنیم که ملاک‌های اصلی مد نظر برای ناوبری را در بر بگیرد. برای مثال حرکت روان با نوسانات کم، عدم برخورد با موانع، انتخاب کوتاه‌ترین مسیر و حفظ فاصله‌ی ایمن نسبت به موانع از ملاک‌های مهم در ناوبری هستند. بنابراین باید تابع پاداش را به‌صورت مناسبی مدل کنیم که این ملاک‌ها را در بر بگیرد. در اینجا برخی از پارامترهای تابع پاداش را نرمال کرده‌ایم تا بتوانیم پارامترها را به‌صورت مناسبی نسبت به یکدیگر وزن‌دهی کنیم. پارامترهای انتخاب شده برای تابع پاداش را در ادامه بررسی می‌کنیم.

**پاداش زاویه نسبت به هدف**

برای این که عامل هرچه بیشتر تشویق شود تا زاویه‌ی خود را نسبت به هدف نزدیک به صفر نگه دارد، از مجذور زاویه طبق معادله‌ی زیر استفاده کرده‌ایم تا میزان پاداش به صورت نمایی در حوالی زاویه‌ی صفر افزایش یابد. با این کار عامل به صفر نگهداشتن این زاویه بیشتر تشویق می‌شود.

$$R_\phi = 1 - 2\sqrt{\left|\frac{\phi}{\pi}\right|}\ , -\pi \le \phi \le \pi \qquad (8)$$

**پاداش فاصله نسبت به هدف**

برای تشویق عامل برای نزدیک‌تر شدن به نقطه‌ی هدف، پاداشی مطابق معادله‌ی زیر برای این منظور اختصاص می‌دهیم.

$$R_d = 1 - \sqrt{d}\ , \qquad 0 < d < 1 \qquad (9)$$

در معادله‌ی بالا نیز برای تشویق عامل به حرکت به سمت هدف، از تابعی غیر خطی استفاده کرده‌ایم به طوری که در حوالی صفر، پاداش به صورت نمایی افزایش یابد.

**پاداش سرعت خطی، سرعت زاویه‌ای و پاداش‌های ثابت**

در تابع پاداش به‌منظور تشویق عامل به حرکت سریع‌تر و همچنین به‌منظور جلوگیری از چرخش‌های سریع و نیز هموارتر شدن مسیرهای تولیدی، به ترتیب ضریبی از سرعت خطی و سرعت زاویه‌ای را به‌عنوان پاداش و جریمه در تابع پاداش لحاظ کرده‌ایم ($C_0$ و $C_1$ ضرایب ثابتی هستند).

$$R_v = C_0\frac{v}{v_{max}}\ , \quad R_\omega = -C_1\frac{\omega}{\omega_{max}}$$

$$R_c = \begin{cases} -K_0 & if\ collision \\ 0 & otherwise \end{cases}$$

$$R_{cd} = \begin{cases} -K_1 & if\ near\ obstacle \\ 0 & otherwise \end{cases} \qquad (10)$$

$$R_{goal} = \begin{cases} K_2 & if\ get\ goal \\ 0 & otherwise \end{cases}$$

در معادله‌ی بالا $R_c$ بیانگر جریمه‌ی برخورد به موانع، $R_{cd}$ بیانگر جریمه‌ی نزدیک شدن بیش از حد به موانع و $R_{goal}$ بیانگر پاداش رسیدن به هدف است. پاداش نهایی نیز از جمع پاداش‌های قبلی به‌دست می‌آید.

$$R_T = R_\phi + R_d + R_v + R_\omega + R_c + R_{cd} + R_{goal} \qquad (11)$$

### 4-4. نرمال کردن داده‌های ورودی

نرمال کردن داده‌های ورودی منجر به عملکرد بسیار بهتر الگوریتم می‌شود. استفاده از داده‌های نرمال نشده به‌عنوان ورودی الگوریتم باعث می‌شود قابلیت تعمیم‌پذیری در حالت‌های مختلف از بین برود. برای مثال اگر ورودی فاصله نسبت به هدف را به‌صورت نرمال نشده به شبکه بدهیم، شبکه نسبت به فاصله‌هایی که در طی فرایند آموزش دیده است بایاس می‌شود و در مرحله‌ی تست در صورت دیدن فاصله‌ای که تاکنون با آن برخورد نداشته، عملکرد آن دچار اختلال می‌شود.





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

همین موضوع برای داده‌های لیدار و سرعت و ... نیز برقرار است. همچنین چون داده‌های ورودی با نوع‌های مختلف دارای بازه‌های متفاوتی هستند، استفاده از داده‌های نرمال نشده باعث بایاس شدن شبکه‌ی عصبی به فاصله‌ی بین این داده‌ها می‌شود.

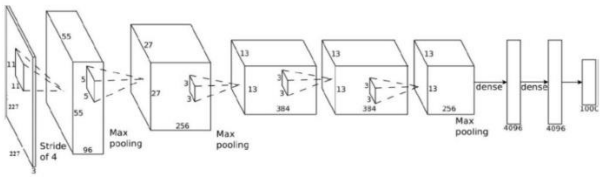
## 5. نتایج تجربی

در ابتدا الگوریتم اولیه‌ی SD3 را مورد ارزیابی قرار دادیم. برای این منظور الگوریتم ارائه شده توسط نویسندگان را برای کاربرد ناوبری استفاده کردیم. از آنجایی که الگوریتم اولیه قابلیت کار با فضاهای حالت با ابعاد زیاد را ندارد، حالت ورودی برای این الگوریتم را یک بردار شامل موارد زیر در نظر گرفتیم.

- زاویه نسبت به هدف
- فاصله نسبت به هدف
- زاویه نسبت به نزدیک‌ترین مانع
- فاصله نسبت به نزدیک‌ترین مانع

این الگوریتم در برخورد با موانع انبوه کارایی مناسبی نداشت. همچنین در هنگام عبور از موانع، مسیرهای شکسته توسط الگوریتم تولید می‌شد و از طرفی مسیرهای مستقیم نیز با نوسان زیاد توسط الگوریتم طی می‌شد.

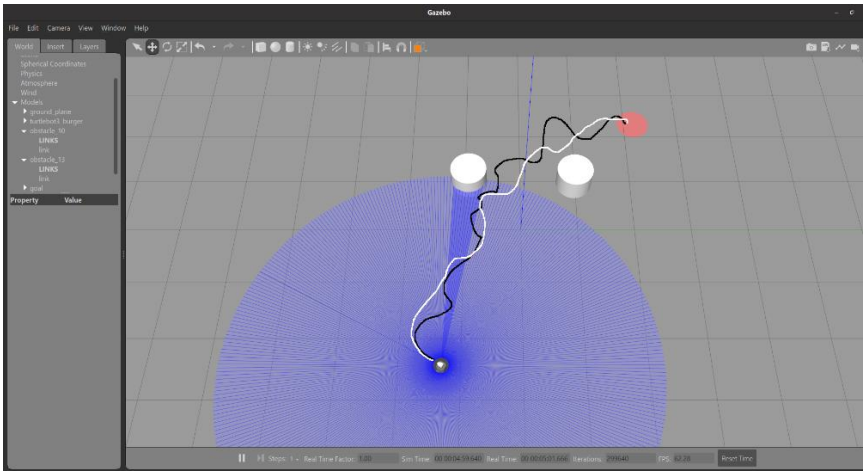
پس از انجام اصلاحات گفته شده در قسمت قبل و همچنین استفاده از لایه‌های کانوالوشنی در ساختار شبکه‌های عملگر و منتقد، الگوریتم SD3 را برای کاربرد ناوبری بهینه‌سازی کردیم. لایه‌های کانوالوشنی به این صورت عمل می‌کنند که ابتدا یک ماتریس از پیش تعریف شده را با ماتریس تصویر ورودی کانوالو کرده و سپس آن را از یک تابع تحریک عبور می‌دهند. این لایه‌ها قابلیت استخراج ویژگی‌های مورد نظر از داخل یک تصویر و یافتن ارتباط بین یک پیکسل با پیکسل‌های مجاور را به ما می‌دهند. در شکل زیر عملکرد یک لایه‌ی کانوالوشنی را ملاحظه می‌کنید.




**شکل 1. عملکرد یک لایه ی کانوالوشنی [31].**

بردار حالت ورودی برای این الگوریتم شامل داده‌های سنسور لیدار، فاصله و زاویه نسبت به هدف و اقدام انجام شده در گام قبلی است.

مسیرهای تولید شده توسط الگوریتم‌های مذکور پس از اجرا کردن در شبیه‌ساز و مشاهده‌ی مسیرهای تولید شده توسط آنها، به‌صورت تقریبی در شکل‌های زیر ترسیم شده‌اند. مسیر با رنگ سفید مربوط به الگوریتم پیشنهادی و مسیر با رنگ سیاه مربوط به الگوریتم اولیه‌ی SD3 است. همانطور که در شکل‌های 1 و 2 مشاهده می کنیم، الگوریتم پیشنهادی هم در عبور از موانع و هم در طی مسیرهای مستقیم عملکرد بهتری دارد.

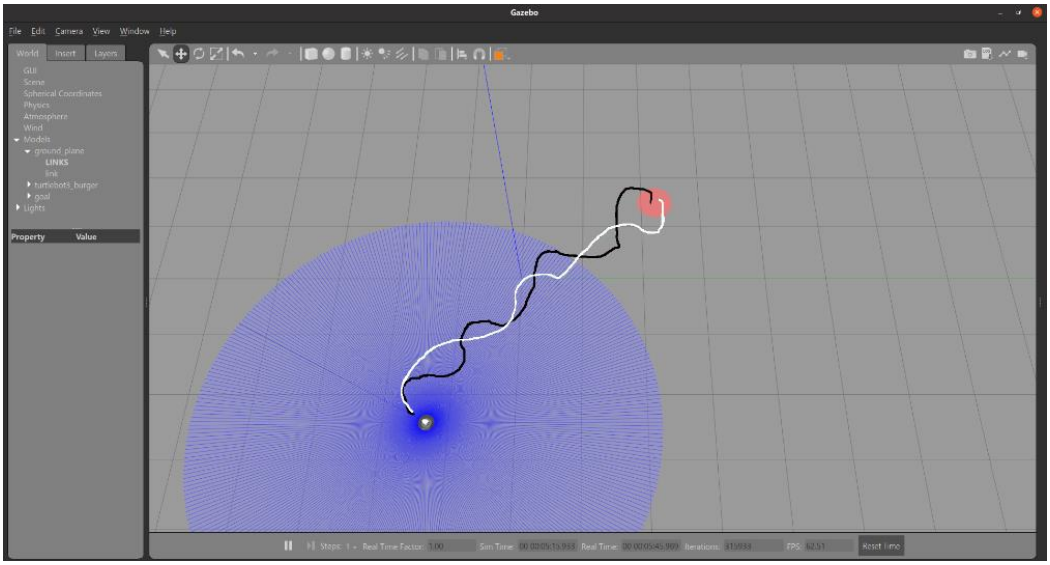



**شکل 2. مقایسه‌ی مسیرهای تولید شده برای عبور از موانع توسط دو الگوریتم مورد اشاره (مسیر سیاه‌رنگ متعلق به الگوریتم پایه و مسیر سفیدرنگ متعلق به الگوریتم پیشنهادی است).**



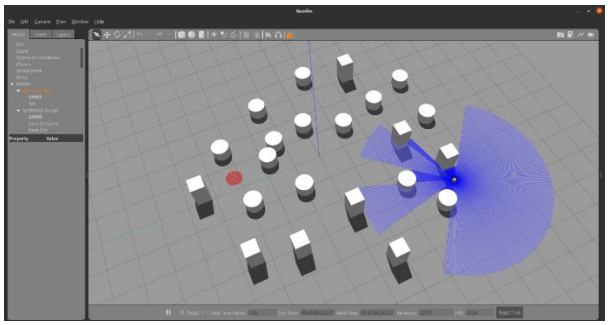


[ Downloaded from ic4i-journal.ir on 2026-11-23 ]



**شکل 3. مقایسه‌ی مسیرهای تولید شده برای طی مسیرهای مستقیم توسط دو الگوریتم مورد اشاره (مسیر سیاه‌رنگ متعلق به الگوریتم پایه و مسیر سفیدرنگ متعلق به الگوریتم پیشنهادی است).**

یکی از ملاک‌های مهم برای ارزیابی عملکرد الگوریتم، نرخ موفقیت در رسیدن به اهداف ازپیش‌تعیین‌شده است. برای این منظور، الگوریتم DDPG را نیز پیاده‌سازی کرده و پس از تست بر روی محیط یکسان، نتایج را با نتایج حاصل از تست الگوریتم پیشنهادی مقایسه می‌کنیم.



**شکل 4. یکی از محیط‌هایی که الگوریتم پیشنهادی را در آن تست کرده‌ایم.**

برای ارزیابی، الگوریتم‌ها را 10 بار و هربار به میزان 100 اپیزود اجرا کرده و تعداد اپیزودهای موفق (بدون برخورد به موانع) را ثبت کرده‌ایم. نتایج حاصل را در جدول 3 مشاهده می‌کنیم. الگوریتم پیشنهادی با صرف زمان آموزش 14000 اپیزود عملکرد بهتری نسبت به الگوریتم DDPG با زمان آموزش 170000 اپیزود به نمایش می‌گذارد.

**جدول 3 ارزیابی الگوریتم پیشنهادی برحسب نرخ موفقیت**

| نام الگوریتم | میانگین نرخ موفقیت(درصد) |
|---|---|
| SD3_Nav | 54 |
| DDPG[3] | 45.5 |

## 6. نتیجه‌گیری

تا کنون از الگوریتم SD3 در کاربردهای ناوبری استفاده نشده است. در این پژوهش الگوریتم SD3 که یک الگوریتم پایه‌ای یادگیری تقویتی عمیق است را برای حل مسائل ناوبری مناسب‌سازی کردیم. الگوریتم SD3 مشکل خطای بیشتر از حد و کمتر از حد در الگوریتم DDPG را تا حدودی برطرف نموده است و به همین دلیل عملکرد بسیار بهتری نسبت به الگوریتم DDPG در سرعت همگرا شدن و نیز پاداش دریافتی از محیط دارد [30]. همچنین با استفاده از لایه‌های کانولوشنی قابلیت کار با فضاهای حالت با ابعاد بالا نیز فراهم شد .

الگوریتم حاصل نسبت به الگوریتم اولیه‌ی SD3 عملکرد بسیار بهتری در طی مسیرهای مستقیم و نیز عبور از موانع به





[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

نمایش گذاشت و همچنین نسبت به الگوریتم DDPG نیز عملکرد بهتری دارد.

در ادامه برای بهبود عملکرد الگوریتم پیشنهادهایی برای کارهای آینده ارائه می‌دهیم. با توجه به هزینه‌ی بالای سنسورهای لیدار، استفاده از تصاویر عمق به علت هزینه‌ی اندک دوربین‌های استریو مناسب‌تر است. بنابراین می‌توان الگوریتم را به نحوی ارتقا داد که بتواند با تصاویر عمق که ماتریس‌هایی دوبعدی هستند (برخلاف داده‌های لیدار استفاده شده در این مطالعه که یک بعد دارند) کار کند. همچنین می‌توان از لایه‌های بازگشتی برای کار در محیط‌های پیچیده استفاده کرد و الگوریتم را برای پیاده‌سازی در محیط واقعی بهبود داد.



[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

## 7. مراجع


[1] C. Wang, J. Wang, J. Wang, and X. Zhang, "Deep Reinforcement Learning-based Autonomous UAV Navigation with Sparse Rewards," *IEEE Internet of Things Journal,* 2020.

[2] H. Li, Q. Zhang, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE transactions on neural networks and learning systems,* 2019.

[3] C. Wang, J. Wang, Y. Shen, and X. Zhang, "Autonomous navigation of UAVs in large-scale complex environments: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology,* vol. 68, no. 3, pp. 2124-2136, 2019.

[4] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017: IEEE, pp. 1343-1350.

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

[6] H. Li, Q. Zhang, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE transactions on neural networks and learning systems,* vol. 31, no. 6, pp. 2064-2076, 2019.

[7] J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg, "Automatic collision avoidance for manually tele-operated unmanned aerial vehicles," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014: IEEE, pp. 6638-6643.

[8] X.-Z. Peng, H.-Y. Lin, and J.-M. Dai, "Path planning and obstacle avoidance for vision guided quadrotor UAV navigation," in *2016 12th IEEE International Conference on Control and Automation (ICCA)*, 2016: IEEE, pp. 984-989.

[9] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning,* vol. 8, no. 3, pp. 229-256, 1992.

[10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *nature,* vol. 518, no. 7540, pp. 529-533, 2015.

[11] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine,* vol. 34, no. 6, pp. 26-38, 2017.

[12] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971,* 2015.

[13] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, 2018: PMLR, pp. 1587-1596.

[14] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018: IEEE, pp. 3052-3059.

[15] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018: IEEE, pp. 6252-6259.

[16] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, 2017: IEEE, pp. 285-292.

[17] J. Choi, K. Park, M. Kim, and S. Seok, "Deep reinforcement learning of navigation in a complex and crowded environment with a limited field of view,"






[ Downloaded from ic4i-journal.ir on 2026-11-23 ]

[24] J. C. Jesus, J. A. Bottega, M. A. Cuadros, and D. F. Gamarra, "Deep deterministic policy gradient for navigation of mobile robots in simulated environments," in *2019 19th International Conference on Advanced Robotics (ICAR)*, 2019: IEEE, pp. 362-367.

[25] J. C. de Jesus, J. A. Bottega, M. A. de Souza Leite, and D. F. T. Gamarra, "Deep deterministic policy gradient for navigation of mobile robots," *Journal of Intelligent & Fuzzy Systems,* no. Preprint, pp. 1-13, 2021.

[26] C.-C. Chang, J. Tsai, J.-H. Lin, and Y.-M. Ooi, "Autonomous Driving Control Using the DDPG and RDPG Algorithms," *Applied Sciences,* vol. 11, no. 22, p. 10659, 2021.

[27] G. Kahn, P. Abbeel, and S. Levine, "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters,* vol. 6, no. 2, pp. 1312-1319, 2021.

[28] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, "Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments," *arXiv preprint arXiv:2005.13857,* 2020.

[29] L. Pan, Q. Cai, Q. Meng, W. Chen, L. Huang, and T.-Y. Liu, "Reinforcement learning with dynamic boltzmann softmax updates," *arXiv preprint arXiv:1903.05926,* 2019.

[30] L. Pan, Q. Cai, and L. Huang, "Softmax deep double deterministic policy gradients," presented at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), 2020.

[31] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, 2017: Ieee, pp. 1-6.

in *2019 International Conference on Robotics and Automation (ICRA)*, 2019: IEEE, pp. 5993-6000.

[18] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters,* vol. 4, no. 2, pp. 610-617, 2019.

[19] O. Bouhamed, H. Ghazzai, H. Besbes, and Y. Massoud, "Autonomous UAV navigation: A DDPG-based deep reinforcement learning approach," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020: IEEE, pp. 1-5.

[20] O. Bouhamed, X. Wan, H. Ghazzai, and Y. Massoud, "A DDPG-based Approach for Energy-aware UAV Navigation in Obstacle-constrained Environment," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, 2020: IEEE, pp. 1-6.

[21] R. B. Grando *et al.*, "Deep Reinforcement Learning for Mapless Navigation of a Hybrid Aerial Underwater Vehicle with Medium Transition," *arXiv preprint arXiv:2103.12883,* 2021.

[22] R. B. Grando, J. C. de Jesus, and P. L. Drews-Jr, "Deep Reinforcement Learning for Mapless Navigation of Unmanned Aerial Vehicles," in *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, 2020: IEEE, pp. 1-6.

[23] X. Gao, L. Yan, G. Wang, T. Wang, N. Du, and C. Gerada, "Toward Obstacle Avoidance for Mobile Robots Using Deep Reinforcement Learning Algorithm," in *2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA)*, 2021: IEEE, pp. 2136-2139.



[ Downloaded from ic4i-journal.ir on 2026-11-23 ]