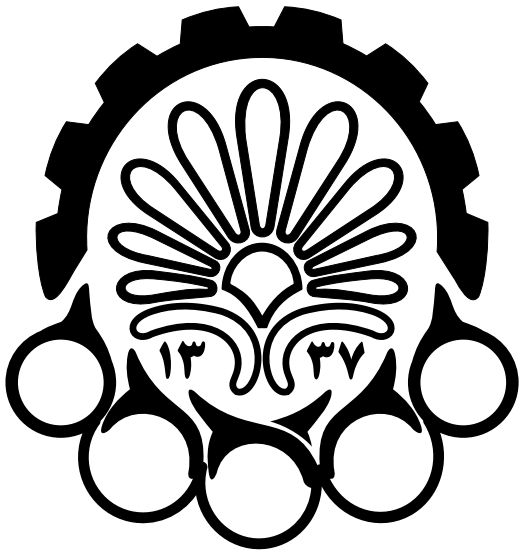


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

تکلیف دوم کارگاه کنترل هوشمند در رباتیک

استاد: دکتر شریفی

اعضای گروه:

تیبا تبشیری نمین ۹۹۲۳۱۰۰

درسا رحمتی ۹۹۲۳۱۱۰

سینا فاضل ۹۹۲۳۰۵۸

امیرحسین دژدار ۹۹۲۳۰۲۶

محدثه رضایی حدادی ۹۹۲۳۰۳۴

فهرست

بخش اول: سوالات تئوری

سوال ۱)

منظور از مفهوم sample efficiency در الگوریتم های یادگیری تقویتی عمیق را توضیح دهید و جستجو کنید چه روشی برای آن ارائه شده است.

Sample Efficiency در الگوریتم‌های یادگیری تقویتی عمیق به میزان کارایی استفاده از داده‌های نمونه برای بهبود سیاست یادگیری اشاره دارد. این مفهوم به‌ویژه در شرایطی که جمع‌آوری داده‌ها هزینه‌بر یا زمان‌بر است، اهمیت پیدا می‌کند. الگوریتم‌های یادگیری تقویتی سنتی معمولاً نیازمند تعداد زیادی نمونه برای یادگیری موثر هستند، که این امر می‌تواند در کاربردهای واقعی چالش‌برانگیز باشد.

یک الگوریتم در صورتی کارآمد است که بتواند از هر نمونه بیشترین بهره را ببرد. به عنوان مثال یک انسان، در عرض چند ثانیه یاد میگیرد که چگونه بازی را بر اساس نمونه های بسیار کمی انجام دهد. این موضوع آن را یک نمونه بسیار کارآمد می کند. الگوریتم های مدرن RL باید هزار بار بیشتر از ما داده ها را ببینند، بنابراین آنها نسبتاً ناکارآمد هستند.

در مورد یادگیری خارج از سیاست (off-policy learning)، همه نمونه‌ها مفید نیستند زیرا بخشی از توزیع مورد علاقه ما نیستند. Importance sampling تکنیکی برای فیلتر کردن این نمونه‌ها است. باید اندازه گیری شود که نمونه های تولید شده چقدر مهم یا مشابه با نمونه هایی هستند که سیاست هدف ممکن است ساخته باشد. با این حال، روش‌های زیادی برای مشخص کردن موارد مهم وجود دارد و اثربخشی آنها ممکن است بسته به کاربرد متفاوت باشد.

۱. Curiosity-Driven Exploration: مقاله (که در بخش منابع ذکر شده است) نشان می دهد که اکتشاف مبتنی بر کنجکاوی می تواند کارآمدتر از اکتشاف تصادفی باشد. این روش شامل ثبت تعداد بازدیدها از هر state و اقدام توسط agent است و کاوش در state های کمتر بازدید شده را ترویج می کند.

۲. Self-Adaptive Double Bootstrapped DDPG (SOUP) : الگوریتم SOUP پیشنهادی، گرادیان خط مشی قطعی عمیق (DDPG) را با استفاده از یک معماری منتقد-بازیگر(actor-critic) بوت استرپ با چندین سر actor و critic گسترش می‌دهد. این معماری کارایی اکتشاف را بهبود می بخشد و تمرین را از طریق مکانیزم self-adaptive confidence که به صورت پویا وزن سرهای بوت استرپ را تنظیم می کند، تثبیت می کند.

۳. Reinforced Representation Learning for Visual Tracking : مقاله یک مدل شبکه عصبی کانولوشنال جدید (CNN) ارائه می‌کند که نمایش‌های ریز دانه و تعبیه‌های معنایی سطح بالا را برای

ردیابی بصری ترکیب می‌کند. این مدل از یک استراتژی یادگیری چند وظیفه‌ای برای انجام تحلیل همبستگی استفاده می‌کند که هم سازگاری و هم قابلیت تعمیم ردیاب را افزایش می‌دهد.

۴. Enhanced-Alignment Measure for Binary Foreground Map Evaluation : اندازه گیری جدیدی به نام E-measure برای ارزیابی نقشه های پیش زمینه باینری پیشنهاد شده است. این اندازه‌گیری مقادیر پیکسل محلی را با آمار سطح تصویر ترکیب می‌کند تا هم جزئیات کلی و هم محلی را ثبت کند، که پیشرفت‌های قابل توجهی را نسبت به معیارهای موجود در چندین معیار نشان می‌دهد.

۵. Ensemble Methods: استفاده از مجموعه‌ای از شبکه‌های عصبی که به صورت مستقل آموزش داده می‌شوند. این روش به ارزیابی Epistemic Uncertainty کمک می‌کند و از طریق استفاده از تخمین‌های مختلف، دقت یادگیری را افزایش می‌دهد. نمونه‌ای از این روش‌ها، استفاده از Randomized Prior Functions (RPFs) است که با اضافه کردن پیش‌بینی‌های یک شبکه آموزش‌ندیده به خروجی، میزان عدم قطعیت را بالا نگه می‌دارد تا به کاوش موثرتر در فضای ورودی کمک کند.

۶. Nonparametric Methods: روش‌های ناپارامتری مانند حل معادلات Bellman به صورت غیرپارامتری که نیاز به حل سیستم‌های معادلات خطی دارند و می‌توانند با استفاده از محاسبات ماتریسی و ابزارهای تفاضلی خودکار مانند TensorFlow یا PyTorch سرعت یادگیری را افزایش دهند. این روش‌ها به دلیل اجرای موازی با استفاده از GPUها، کارایی بالاتری نسبت به روش‌های سنتی دارند.

۷. Symmetric Replay Training (SRT): این روش با استفاده از نمونه‌های با پاداش بالا و تشویق به کاوش مناطق تقارن‌دار کمتر کاوش شده، کارایی نمونه را افزایش می‌دهد. SRT می‌تواند بدون نیاز به تعاملات آنلاین اضافی به بهبود سیاست‌های یادگیری کمک کند و در کاربردهای واقعی مانند بهینه‌سازی مولکولی و طراحی سخت‌افزار نتایج مثبتی داشته باشد.

۸. Batch Normalization: استفاده از نرمال‌سازی دسته‌ای در شبکه‌های عصبی منتقد برای تثبیت آموزش و حذف نیاز به شبکه‌های هدف (Target Networks). این روش باعث می‌شود تا فرآیند یادگیری پایدارتر و کارآمدتر شود و به بهبود کارایی نمونه کمک می‌کند.

منبع:

https://ai.stackexchange.com/questions/5246/what-is-sample-efficiency-and-how-can-importance-sampling-be-used-to-achieve-it

https://www.ijcai.org/proceedings/2018/0820.pdf

سوال ۲)

علت استفاده از بافر تجارت در برخی الگوریتم های یادگیری تقویتی را توضیح دهید.

علت استفاده از بافر تجارت (Replay Buffer) در برخی الگوریتم‌های یادگیری تقویتی، بهبود کارایی و پایداری فرآیند یادگیری است. وقتی از بافرهای تجارت در یادگیری تقویتی صحبت می کنیم، به طور کلی منظورمان بافری است که تجربیات جمع آوری شده از تعاملات عامل(های) ما با محیط را ذخیره و بازپخش می کند. در پایتون، یک بافر ساده را می توان با لیستی که عناصر به آن اضافه شده و سپس از آن نمونه برداری می شود، پیاده سازی کرد. چنین بافرهایی بیشتر در الگوریتم های یادگیری خارج از سیاست(off-policy) استفاده می شوند. این به طور شهودی منطقی است زیرا این الگوریتم‌ها می‌توانند از تجربیاتی که در بافر ذخیره می‌شوند، اما در نسخه‌ای از خط‌مشی قبلی (یا حتی یک «سیاست رفتار» کاملاً متفاوت) تولید شده‌اند، یاد بگیرند. هنگام نمونه‌برداری از بافر، ما انتخاب می‌کنیم که با کدام تجربه‌ها agent خود را آموزش دهیم. یک استراتژی ساده که برای بسیاری از الگوریتم‌ها مؤثر بوده است، انتخاب این نمونه‌ها به طور یکنواخت و تصادفی است. یک استراتژی پیشرفته تر (در بسیاری از موارد بهتر است) تکرار تجربیات اولویت دار (PER) است. در PER، به آیتم‌های منفرد در بافر یک مقدار اولویت (اسکالری) اختصاص داده می‌شود که نشان‌دهنده اهمیت آن‌ها است، یا به عبارت ساده‌تر، چقدر انتظار داریم از این آیتم‌ها یاد بگیریم. تجارب با اولویت بالاتر احتمال بیشتری دارد که نمونه گیری شوند.

یک بافر طبیعتاً در ظرفیت خود برای نگهداری تجربیات محدود است. در طول اجرای یک الگوریتم، یک بافر در نهایت به ظرفیت خود می‌رسد و برای اینکه فضایی برای تجربیات جدید ایجاد کنیم، باید تجربیات قدیمی‌تر را حذف کنیم. این کار به طور کلی بر اساس first-in-first-out انجام می شود. برای الگوریتم‌های شما، این بدان معناست که بافرهای با ظرفیت بالا، فرصت یادگیری از نمونه‌های قدیمی‌تر را می‌دهند، در حالی که بافرهای کوچک‌تر، فرآیند یادگیری را روی خط‌مشی‌تر می‌کنند. یک استثنا از این استراتژی در بافرهایی است که نمونه گیری مخزن را اجرا می کنند.

از مزیت های آن میتوان به موارد زیر اشاره کرد:

۱.پایداری بیشتر: استفاده از بافر تجارت میتواند یادگیری را پایدارتر کند چون از نمونه های متنوع و تصادفی استفاده شده و به روزرسانی پارامتر های شبکه با نوسانات کمتری همراه میشود.

۲.افزایش سرعت یادگیری: بافر تجارت باعث میشود الگوریتم با سرعت به تجربیات جدید واکنش نشان داده و از آنها بیاموزد که باعث افزایش کارایی و سرعت عملکرد میشود.

۳.کاوش موثرتر: با ذخیره سازی و استفاده از تجربیات مختلف گذشته الگوریتم میتواند کاوش موثرتری داشته و راه های بهینه تری کشف کند. این امر به خصوص در محیطهایی که جمع‌آوری داده‌ها هزینه‌بر یا زمان‌بر است، اهمیت زیادی دارد.

۴.کاهش همبستگی نمونه ها: در یادگیری تقویتی، داده‌های جمع‌آوری شده به صورت توالی زمانی (توالی از حالت‌ها، اعمال و پاداش‌ها) هستند که همبستگی بالایی با یکدیگر دارند. این همبستگی می‌تواند باعث

ناپایداری در فرآیند یادگیری شود. بافر تجارت نمونه‌ها را به صورت تصادفی انتخاب می‌کند و این همبستگی را کاهش می‌دهد.

منبع:

https://docs.ray.io/en/latest/rllib/rllib-replay-buffers.html

سوال۳)

یک مقاله در حوزه رباتیک با محوریت بازوی رباتیک بیابید که در کنترل آن از الگوریتم های یادگیری تقویتی استفاده شده باشد. سپس بیان کنید که رویکرد مقاله در استفاده از یادگیری تقویتی چه بوده و از آن در کجا استفاده کرده است. در ادامه بیان کنید که از چه روشی در یادگیری تقویتی استفاده شده (رویکرد های مبتنی بر مدل یا بدون مدل) و همچنین از چه روشی برای یادگیری سیاست استفاده شده است. در نهایت نحوه فرموله کردن مساله را ارائه کنید(معرفی مشاهدات و عمل ها) و نتیجه نهایی مقاله از راه حل ارائه شده را گزارش دهید. کمک گرفتن از ابزار های هوشمند برای جستجو و درک سریعتر مقاله مجاز است.

مقاله۱: https://www.mdpi.com/2076-3417/11/17/7917

مقاله۲:https://www.researchgate.net/publication/341202448_Robotic_Arm_Control_and_Task_Training_through_Deep_Reinforcement_Learning

① مقاله اول:

Vision-Based Robotic Arm Control Algorithm Using Deep Reinforcement Learning for Autonomous Objects Grasping

رویکرد مقاله در استفاده از یادگیری تقویتی:

این مقاله به کنترل بازوی رباتیک با استفاده از الگوریتم‌های یادگیری تقویتی عمیق پرداخته است و هدف این پژوهش بهبود فرآیند گرفتن اشیا به صورت خودکار با استفاده از یادگیری تقویتی عمیق است. در این مقاله از روش سینماتیک معکوس برای دستیابی به موقعیت استفاده شده و ربات آن ۵ درجه آزادی است.

روش یادگیری تقویتی در مقاله:

در این مقاله، الگوریتم Deep Deterministic Policy Gradient (DDPG) برای کنترل بازوی رباتیک به کار گرفته شده است که یک روش بدون مدل (Model-Free) است و برای محیط‌های با عمل‌های پیوسته مناسب است.

رویکرد یادگیری سیاست:

الگوریتم DDPG یک روش مبتنی بر سیاست است که از یک شبکه عصبی برای پیش‌بینی عمل‌ها(actions) و یک شبکه دیگر برای پیش‌بینی مقدار Q استفاده می‌کند. در این مقاله، از یک شبکه CNN برای استخراج ویژگی‌های تصویری و از الگوریتم YOLOv5 برای تشخیص اشیا استفاده شده است.

نحوه فرموله کردن مسئله:

← مشاهدات (Observations): ورودی‌های شبکه شامل تصاویر دریافتی از دوربین، مختصات سه‌بعدی موقعیت هدف، و مقادیر زاویه‌ای اتصالات بازو است.

←عمل‌ها (Actions): این actions شامل زوایه های اتصالات بازوی رباتیک برای حرکت به سمت موقعیت هدف می‌باشند.

نتیجه‌گیری مقاله:

با بررسی شبیه سازی مقاله ، میبینیم روش ارائه شده در مقاله در مقایسه با روش های قدیمی و سنتی دقت بیشتری دارد و بهتر و با خطای کمتری (در زوایای بازو) به موقعیت هدف میرسد. این روش در انجام وظایف مبتنی بر دید نیز عملکرد بهتری ارائه می‌دهد.

② مقاله دوم:

Robotic Arm Control and Task Training through Deep Reinforcement Learning

رویکرد مقاله در استفاده از یادگیری تقویتی:

هدف از این پژوهش بهبود عملکرد بازوی رباتیک در انجام وظایف مختلف با استفاده از الگوریتم‌های یادگیری تقویتی عمیق است و به بررسی کنترل و آموزش وظایف بازوی رباتیک با استفاده از یادگیری تقویتی عمیق پرداخته است.

روش یادگیری تقویتی در مقاله:

در این مقاله، از الگوریتم Deep Q-Network (DQN) استفاده شده است که یک روش بدون مدل (Model-Free)در یادگیری تقویتی است. این الگوریتم برای کنترل بازوی رباتیک در محیط‌های مختلف به کار گرفته شده و از شبکه‌های عصبی برای تقریب سیاست (پیش‌بینی اعمال) و ارزیابی سیاست استفاده می‌کند.

رویکرد یادگیری سیاست:

الگوریتم DQN از چارچوب Actor-Critic بهره می‌برد که در آن Actor سیاست را با استفاده از شبکه عصبی تخمین می‌زند و Critic ارزش عملکرد آن را ارزیابی می‌کند. این الگوریتم به‌وسیله شبکه‌های هدف برای تثبیت فرآیند یادگیری استفاده می‌شود.

نحوه فرموله کردن مسئله:

←**مشاهدات :** شامل وضعیت مفصل های بازوی رباتیک و ورودی‌های تصویری از سیستم کنترل است.

←**عمل‌ها (actions):** شامل دستورهای کنترلی برای حرکت مفصل های بازوی رباتیک است.

نتیجه‌گیری مقاله:

با توجه به نتایج میتوان فهمید که روش پیشنهاد شده در مقاله وظایف کنترل موقعیت بازوی ربات را به خوبی و با دقت بالا انجام میدهد و در عین حال از روش های قدیمی و سنتی بهتر عمل میکند. این روش همچنین از مصرف منابع و فرسایش فیزیکی ابزارها جلوگیری کند.

بخش دوم: شبیه سازی

کد Reinforcement Learning

در این بخش به توضیح کد RL در پایتون میپردازیم:

```
!pip install shimmy
!pip install stable_baselines3
!pip install gymnasium.wrappers.monitoring
!pip install stable_baselines3 gymnasium
!pip install ppo
```

در این بخش ابتدا کتابخانه های متفاوت را برای کار با محیط RL و مدل های مربوطه نصب میکنیم.

کتابخانه ی shimmy برای تبدیل و هماهنگی نسخه های مختلف API های محیط یادگیری تقویتی مورد استفاده قرار میگیرد.

کتابخانه stable_baselines3 مجموعه ای از الگوریتم های RL است که بر پایه ی pytorch توسعه یافته و میتوان با کمک آن مدل های یادگیری تقویتی مختلف را پیاده سازی کرد.

کتابخانه gymnasium.wrappers.monitoring یک نسخه بهبود یافته از open AI Gym است که برای ایجاد و آزمایش محیط های یادگیری تقویتی استفاده میشود. دراین کتابخانه ابزارهایی برای مانیتور کردن و ضبط اطلاعات اجرای مدل ها وجود دارد.

کتابخانه stable_baselines3 gymnasium دو کتابخانه را بصورت همزمان نصب کرده و ابزار های لازم برای محیط RL را فراهم میکند.

کتابخانه ppo یکی از الگوریتم های معروف یادگیری است که برای بهینه سازی policies استفاده میشود.

```python
import gym
from gym import spaces
import numpy as np
import matplotlib.pyplot as plt
from stable_baselines3 import PPO
from stable_baselines3.common.evaluation import evaluate_policy
```

در ادامه کتابخانه های ضروری را وارد میکنیم.

اول کتابخانه gym را وارد میکنیم که برای ایجاد و مدیریت محیط های یادگیری تقویتی است.

حال از این کتابخانه، ماژول spaces را وارد میکنیم که برای تعریف فضای حالت و action space استفاده میشود.

کتابخانه numpy را با عنوان np وارد میکنیم که برای عملیات عددی و ریاضیاتی در پایتون میباشد.

کتابخانه matplotlib.pyplot با عنوان plt برای ایجاد نمودار ها و تجسم داده ها استفاده میشود.

از کتابخانه stable_baselines3 ، PPO را اضافه میکنیم که یکی از الگوریتم‌های یادگیری تقویتی است و برای بهینه‌سازی policies استفاده می‌شود.

از کتابخانه stable_baselines3.common.evaluation ، evaluate_policy را اضافه میکنیم که برای ارزیابی عملکرد مدل یادگیری تقویتی استفاده می‌شود.

```python
class ThreeDOFRobotEnv(gym.Env):
    def __init__(self):
        super(ThreeDOFRobotEnv, self).__init__()
        self.max_torque = 1.0
        self.max_speed = np.pi
        self.dt = 0.05
        self.goal = np.array([1.5, 1.5])
        self.max_distance = 3.0  # Maximum allowable distance from the goal
        self.barrier = np.array([0.75, 0.75])
        self.barrier_radius = 0.1  # Radius of the barrier

        self.action_space = spaces.Box(low=-self.max_torque, high=self.max_torque, shape=(3,), dtype=np.float32)
        self.observation_space = spaces.Box(low=-self.max_speed, high=self.max_speed, shape=(6,), dtype=np.float32)

        self.reset()
```

در این بخش یک کلاس به اسم ThreeDOFRobotEnv ایجاد میکنیم. این کلاس یک محیط برای یادگیری تقویتی با رباتی با سه درجه آزادی فراهم می‌کند.

بصورت کلی، در این بخش میخواهیم یک محیط RL را با تنظیمات و پارامترهای اولیه ایجاد کنیم. حال تنظیمات اولیه کلاس والد را انجام میدهیم:

ابتدا پارامترهای محیط را تنظیم میکنیم. ابتدا self.max_torque = 1.0 را داریم که حداکثر گشتاور مجاز برای هر مفصل ربات است. سپس self.max_speed=np.pi که حداکثر سرعت زاویه ای مجاز برای هر مفصل ربات است. Self.dt=0.05 که گام زمانی در هر مرحله برای به روزرسانی حالت ها است. Self.goal=np.array([1.5,1.5]) نیز نشان دهنده موقعیت هدف(موقعیت نهایی) ربات میباشد. Self.max_distance=3.0 حداکثر فاصله مجاز از هدف میباشد و زمانیکه ربات از این فاصله تعیین شده دورتر برود آموزش تمام میشود. Self.barrier=np.array([0.75,0.75]) موقعیت یک مانع در محیط را نشان میدهد و شعاع مانع (که محدوده برخورد را تعیین میکند) با self.barrier_radius=0.1 مشخص میشود.

در ادامه برای فضای مشاهده، self.action_space را برای مشخص کردن عمل های مجاز ربات (این عمل ها گشتاور های اعمال شده به هریک از سه مفصل هستند) و self.observation_space را برای محدوده حالت های مجاز ربات (حالت ها شامل زوایا و سرعت های زاویه ای سه مفصل ربات هستند) تعریف میکنیم.

در آخر برای برگرداندن حالت محیط به حالت اولیه از self.reset استفاده میکنیم. توجه شود که در حالت اولیه زاویه ها و سرعت های زاویه ای برای هر مفصل صفر است.

```
def reset(self):
    self.state = np.zeros(6)
    self.prev_distance = self._get_distance_to_goal()
    self.elapsed_time = 0  # Reset the elapsed time
    return self.state
```

در این بخش، ابتدا حالت اولیه ربات را به یک آرایه ۶ عنصری از درایه های صفر تبدیل میکنیم. این آرایه شامل سرعت های زاویه ای و زاویه های سه مفصل میباشد.

در ادامه در خط self.prev_distance ، فاصله ربات تا مقصد(هدف) را در حالت اولیه ذخیره میکنیم که برای محاسبه پاداش (reward) و تغییر فاصله مورد استفاده قرار میگیرد.

در self.elapsed_time زمان گذشته از زمان شروع را به صفر برمیگرداند و برای تعیین مدت زمان مورد استفاده قرار میگیرد.

در خط آخر نیز حالت (اولیه) ربات را دریافت میکنیم که شامل زوایا و سرعت های زاویه ای میباشد.

```python
def step(self, action):
    th1, th2, th3, dth1, dth2, dth3 = self.state
    u1, u2, u3 = action

    dth1 = np.clip(dth1 + self.dt * u1, -self.max_speed, self.max_speed)
    dth2 = np.clip(dth2 + self.dt * u2, -self.max_speed, self.max_speed)
    dth3 = np.clip(dth3 + self.dt * u3, -self.max_speed, self.max_speed)

    th1 += self.dt * dth1
    th2 += self.dt * dth2
    th3 += self.dt * dth3

    self.state = np.array([th1, th2, th3, dth1, dth2, dth3])
    self.elapsed_time += self.dt  # Increment the elapsed time

    reward, done = self._compute_reward()
    return self.state, reward, done, {}
```

در این بخش ابتدا توابع را تعریف میکنیم. درواقع ربات یک قدم برمیدارد و وضعیت آن بر اساس اقدامات اعمال شده به روزرسانی میشود. این وضعیت شامل زاویه ها و سرعت های زاویه ای میشود و اقدامات اعمال شده به ربات (یعنی گشتاورها) بدست می آیند.

سپس سرعت های زاویه ای جدید بر اساس گشتاور اعمال شده و step زمانی محاسبه میشوند. ب استفاده از np.clip، مقادیر سرعت زاویه ای را در محدوده [−π,π] نگه میداریم.

در ادامه، زوایای جدید مفاصل با استفاده از سرعت های زاویه ای جدید و step زمانی محاسبه میشوند.

سپس وضعیت جدید ربات (که به روز شده است) ذخیره میشود.

در آخر، پاداش و وضعیت پایان و یک dictionary برای اطلاعات اضافی بازگردانده میشود.

```
def _get_distance_to_goal(self):
    x = np.cos(self.state[0]) + np.cos(self.state[0] + self.state[1]) + np.cos(self.state[0] + self.state[1] + self.state[2])
    y = np.sin(self.state[0]) + np.sin(self.state[0] + self.state[1]) + np.sin(self.state[0] + self.state[1] + self.state[2])
    distance = np.linalg.norm(self.goal - np.array([x, y]))
    return distance
```

این تابع فاصله نوک ربات تا مقصد(هدف) را بدست می آورد و در بدست آوردن فاصله فعلی تا هدف به ما کمک میکند.

در بخش x ، موقعیت نوک ربات را با استفاده از زوایای سه مفصل محاسبه میکنیم. با جمع کردن زوایای مفصل اول تا سوم و با استفاده از تابع np.cos، موقعیت نهایی در جهت x بدست می آید.

در خط بعد در مولفه ی y مشابه x (اما اینبار با کمک np.sin) موقعیت نهایی در جهت y بدست می آید.

سپس فاصله اقلیدسی بین موقعیت نوک ربات ([x,y]) و موقعیت هدف (self.goal) محاسبه میشود و این فاصله بازگردانده میشود.

```
def _get_end_effector_position(self):
    th1, th2, th3, _, _, _ = self.state
    x = np.cos(th1) + np.cos(th1 + th2) + np.cos(th1 + th2 + th3)
    y = np.sin(th1) + np.sin(th1 + th2) + np.sin(th1 + th2 + th3)
    return np.array([x, y])
```

در این بخش، موقعیت EE را بدست می آوریم. درواقع با استفاده از زاویه های مفصل ها، موقعیت نهایی EE در فضای دو بعدی تعیین میشود.

ابتدا وضعیت فعلی زوایای سه مفصل بدست می آید. (مقادیر سرعت زاویه ای در این تابع استفاده نشده و بدست نمی آیند.)

در خط بعد، مولفه ی x موقعیت نوک ربات را با استفاده از موقعیت افقی لینک ها به ما میدهد. در خط بعد موقعیت y نیز به همین ترتیب (با کمک موقعیت عمودی).

در آخر موقعیت EE به صورت یک آرایه بدست می آید.

```python
def _check_collision_with_barrier(self):
    end_effector_pos = self._get_end_effector_position()
    return np.linalg.norm(end_effector_pos - self.barrier) <= self.barrier_radius
```

در این تابع برخورد EE با یک مانع را بررسی میکنیم. این کار را با استفاده از موقعیت نوک ربات و موقعیت مانع و فاصله بین آنها و همچنین شعاع مانع انجام میدهیم.

```python
def _compute_reward(self):
    current_distance = self._get_distance_to_goal()
    reward = 0
    done = False

    if self._check_collision_with_barrier():
        reward -= 5.0  # Heavy penalty for collision with barrier
        done = True
    elif current_distance > self.max_distance:
        reward -= 1.0  # Penalty for being too far from the goal
        done = True
    elif current_distance < 0.2:
        reward += 1.0  # Reward for reaching the goal
        done = True
    else:
        reward += (self.prev_distance - current_distance) * 15  # Reward for getting closer to the goal
        reward -= 0.01 * self.elapsed_time  # Small penalty for time taken

    self.prev_distance = current_distance

    reward -= 0.06 * (abs(action[0]) + abs(action[1]) + abs(action[2]))
    # Encourage smoother movements by penalizing large velocities
    reward -= 0.06 * (abs(self.state[3]) + abs(self.state[4]) + abs(self.state[5]))

    return reward, done
```

حال در این قسمت، تابع reward را با توجه به وضعیت ربات ایجاد میکنیم. با بررسی برخورد به مانع و فاصله، پاداش محاسبه میشود. در ابتدا، پاداش برابر صفر قرار داده میشود و مقدار اولیه وضعیت اتمام بخش را false قرار میدهد. سپس در بخش های بعدی، پاداش و جریمه های مربوطه داده میشود و در هرکدام با توجه به شرط وضعیت بخش مشخص میشود(با done=true بخش به پایان میرسد).

در صورتی که هیچکدام از شروط برقرار نباشند، پاداش بر اساس کاهش فاصله تا هدف بدست می آید. همچنین جریمه ای برای زمان سپری شده در نظر گرفته میشود.

جریمه ای برای گشتاورهای بزرگ و سرعت های زاویه ای بزرگ (برای داشتن حرکت نرم تر و کنترل حرکات سریع) در نظر گرفته شده است.

```python
def render(self, mode='human'):
    th1, th2, th3, _, _, _ = self.state
    x1 = np.cos(th1)
    y1 = np.sin(th1)
    x2 = x1 + np.cos(th1 + th2)
    y2 = y1 + np.sin(th1 + th2)
    x3 = x2 + np.cos(th1 + th2 + th3)
    y3 = y2 + np.sin(th1 + th2 + th3)

    plt.figure()
    plt.plot([0, x1], [0, y1], 'ro-')
    plt.plot([x1, x2], [y1, y2], 'ro-')
    plt.plot([x2, x3], [y2, y3], 'ro-')
    plt.plot(self.goal[0], self.goal[1], 'go')
    plt.plot(self.barrier[0], self.barrier[1], 'ko')
    circle = plt.Circle(self.barrier, self.barrier_radius, color='k', fill=True)
    plt.gca().add_patch(circle)
    plt.xlim(-3, 3)
    plt.ylim(-3, 3)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.title("3-DOF Planar Robot")
    plt.legend(["Link 1", "Link 2", "Link 3", "Goal", "Barrier"])
    plt.grid()
    plt.show()
```

این تابع محیط ربات ۳ درجه آزادی را نشان میدهد. به عبارت دیگر بخش محیط ربات را بصورت گرافیکی رسم میکند. برای اینکار وضعیت فعلی زوایای سه مفصل ربات، موقعیت انتهای اولین و دومین لینک ربات و موقعیت EE بدست می آیند.

یک شکل جدید برای رسم نمودار ایجاد میشود. لینک های ربات با رنگ قرمز رسم میشوند. نقطه هدف با رنگ سبز و مانع با سیاه رسم میشود. در ادامه کد محدوده محور x و y، برچسب محور ها، عنوان نمودار و... تعیین میشود.

در انتها نمودار نمایش داده میشود.

```python
# Callback to store rewards during training
class RewardCallback(BaseCallback):
    def __init__(self, verbose=0):
        super(RewardCallback, self).__init__(verbose)
        self.rewards = []

    def _on_step(self) -> bool:
        self.rewards.append(self.locals['rewards'])
        return True

    def _on_training_end(self) -> None:
        plt.plot(self.rewards)
        plt.xlabel('Steps')
        plt.ylabel('Rewards')
        plt.title('Training Rewards')
        plt.show()
```

این کلاس برای ذخیره و رسم پاداش ها در طول آموزش مدل یادگیری تقویتی است.

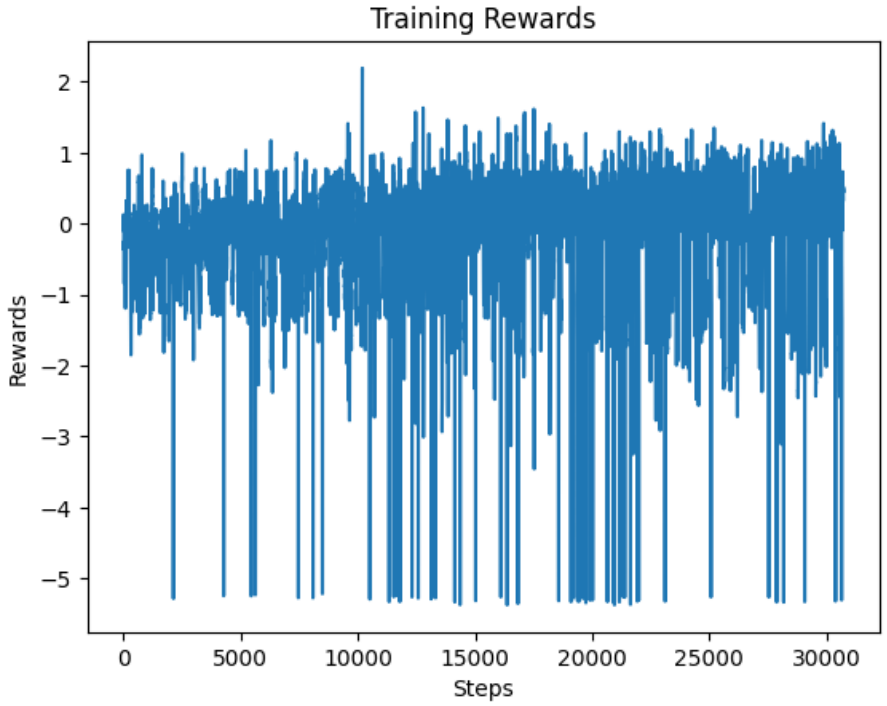
```python
# Create the environment
env = ThreeDOFRobotEnv()

# # Create and train the model
# model = PPO('MlpPolicy', env, verbose=1)
# model.learn(total_timesteps=50000)
# Create and train the model with callback
callback = RewardCallback()
model = PPO('MlpPolicy', env, verbose=1)
model.learn(total_timesteps=30000, callback=callback,progress_bar = True)

# Evaluate the model
mean_reward, std_reward = evaluate_policy(model, env, n_eval_episodes=10)
print(f"Mean reward: {mean_reward} +/- {std_reward}")
```

این بخش از کد یک محیط یادگیری تقویتی برای ربات سه درجه آزادی ایجاد میکند و یک مدل PPO را برای ذخیره و نمایش پاداش ها در طول آموزش ایجاد کرده و آموزش میدهد و مدل را ارزیابی میکند. مدل PPO با استفاده از سیاست MlpPolicy و محیط env ایجاد میشود. این مدل برای ۳۰۰۰۰ step آموزش داده میشود. همچنین برای نمایش پیشرفت آموزش از progress_bar=True استفاده میکنیم.

این مدل برای ۱۰ بخش(episode) ارزیابی و میانگین و انحراف استاندارد پاداش ها محاسبه و نمایش داده میشود.

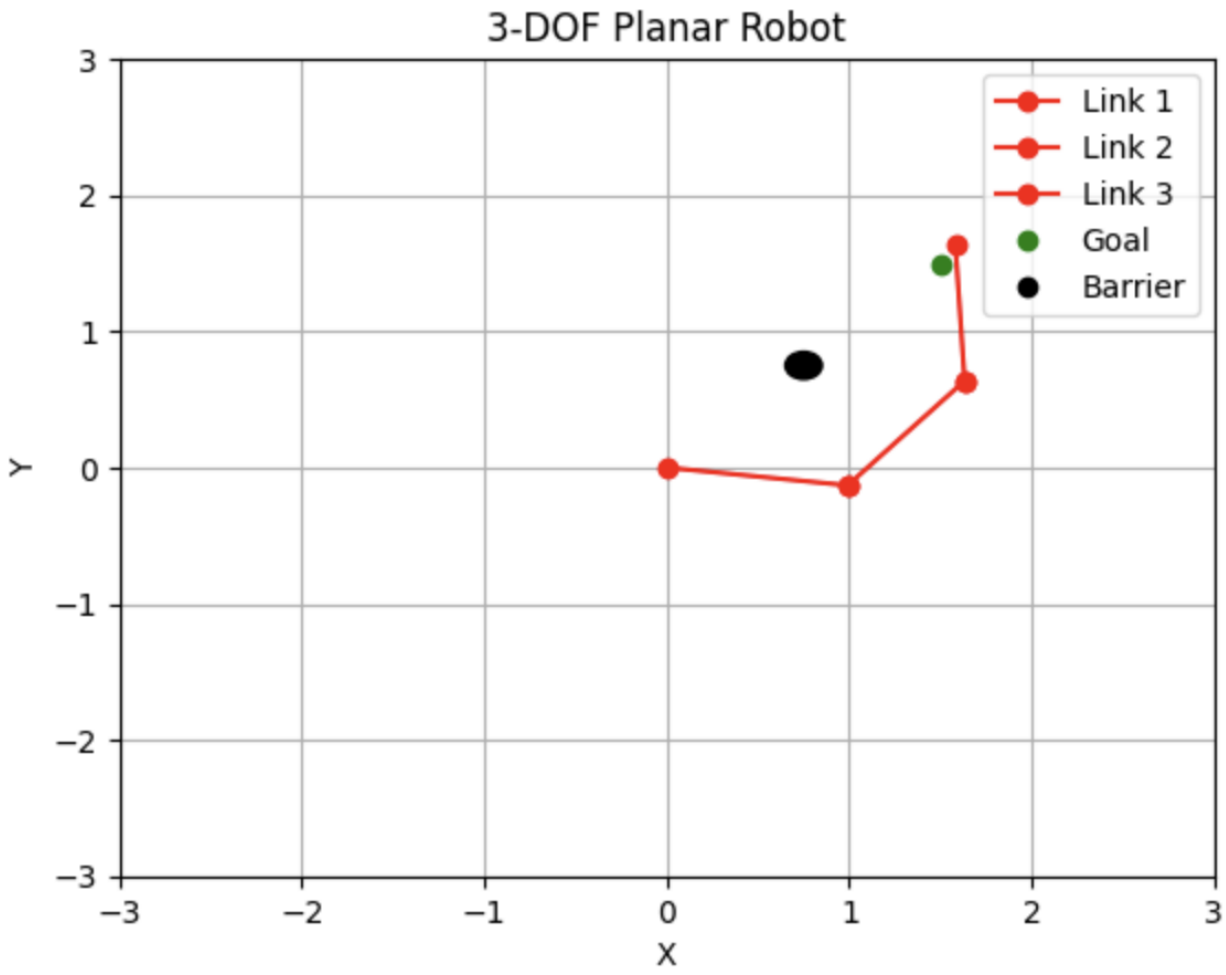


```
# Visualize the robot in the environment
obs = env.reset()
for _ in range(100):
    action, _states = model.predict(obs)
    obs, rewards, dones, info = env.step(action)
    env.render()
    if dones:
        obs = env.reset()
```

این کد وضعیت ربات را در محیط نمایش میدهد. ربات ۱۰۰ قدم میرود و وضعیت آن در هر قدم به روز رسانی میشود.

وضعیت اولیه محیط دریافت شده، مدل با استفاده از وضعیت فعلی اقدام بعدی را پیش بینی میکند. این اقدام پیش بینی شده به محیط اعمال شده و وضعیت جدید و پاداش و وضعیت اتمام episode و... دریافت میشود.

در صورت تمام شدن episode، محیط reset شده و حرکت ادامه میابد.


3-DOF Planar Robot
Link 1
Link 2
Link 3
Goal
Barrier
X
Y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3