چکیده

سیستم‌های پیشرفته کمک‌به‌راننده[۱]، سیستم‌های الکترومکانیکی هستند که به راننده در هنگام رانندگی و یا پارک‌کردن کمک می‌کنند. این سیستم‌ها با جمع‌آوری داده از حسگرهای مختلف و دوربین‌ها، موانع و یا خطای راننده را تشخیص داده و به تناسب پاسخ می‌دهند.

سیستم‌های پیشرفته کمک‌به‌راننده با استفاده از قوانین و مقررات از پیش تعیین‌شده ساخته‌می‌شوند؛ به این صورت که مکانیزم آن‌ها توسط مهندسین مکانیک و الکترونیک طراحی‌شده و در صورت قبولی در تست‌های مربوطه، قابل پیاده‌سازی می‌شوند.

اما این سیستم‌ها اکثراً ایستا هستند، یعنی قابلیت تصمیم‌گیری ندارند و براساس ورودی‌ها و برنامه‌ی داده‌شده به آن‌ها، یک مکانیزم را فعال می‌کنند. برای همین، ممکن است در شرایط خاص، خروجی مطلوب نداشته و یا ساخت سیستم‌های پیچیده‌ای که همه‌ی حالات را در نظر بگیرند غیرممکن شود. در این شرایط، می‌توان از تکنیک های یادگیری ماشین[۲] برای ساخت این سیستم‌ها استفاده‌کرد.

این گزارش در مورد طراحی و پیاده‌سازی یک نمونه ساده سیستم ترمز هوشمند اضطراری[۳] با استفاده از متد یادگیری تقویتی[۴] است.

واژه‌های کلیدی:

سیستم‌های پیشرفته کمک‌به‌راننده، یادگیری ماشین، ترمز هوشمند اضطراری، یادگیری تقویتی

[۱] Advanced driver-assistance systems (ADAS)

[۲] Machine learning

[۳] Automated emergency braking (AEB)

[۴] Reinforcement learning

فهرست مطالب

صفحه

فهرست اشکال

صفحه

فصل اول

مقدمه

مقدمه

توضیح کلی

یادگیری تقویتی

یادگیری تقویتی یکی از نگرش‌ها در یادگیری ما شین ا ست که با تنظیم مقدار پاداش داده شده براساس هر حرکت[۱] در هر حالت[۲]، عامل[۳] را آموزش می‌دهد؛ به این شکل که هرچه‌قدر حرکت انجام‌شده به حالت مطلوب نزدیک‌تر باشد، عامل پاداش بیشتری دریافت خواهدکرد و بالعکس.

برای آموزش عامل، لازم است تا محیط آماده باشد. عامل در ابتدا شروع به کاوش[۴] در محیط کرده و هر چه به انتها نزدیک‌تر می‌شود باید از کاوش‌کردن کاسته و بیشتر از اطلاعاتی که از محیط کسب کرده بهره‌گیری[۵] کند.

---

۱ Action

۲ State

۳ Agent

۴ Explore

۵ Exploit

یادگیری کیو[1]

یادگیری کیو یک الگوریتم برای آموزش عامل است. با این الگوریتم، عامل آموزش می‌بیند تا در هر شرایطی بهترین عمل را اجرا کند. بعد از آموزش، عامل می‌تواند نتایج را در یک جدول ذخیره کرده و هنگام بهره‌گیری، به آن جدول رجوع کند.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right.}^{\text{temporal difference}} \underbrace{\left. - Q(s_t, a_t) \right)}_{\text{old value}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

شکل ۱-۱ الگوریتم یادگیری کیو

که در آن s حالت، a حرکت، α نرخ یادگیری، r پاداش و γ نرخ تاثیر پاداش‌های آینده است که به α و γ ابرپارامتر[2] می‌گویند؛ چرا که تغییر آن‌ها بر همه‌ی مراحل و تصمیمات تأثیرگذار است.

نرخ یادگیری(α):

این متغیر تعیین می‌کند که عامل چه‌قدر احتمال دارد تا اطلاعات منتسب به یک حرکت در یک حالت خاص را تغییر دهد. مقدارهای کوچک برای α باعث می‌شود عامل از اطلاعات اولیه‌ای که کسب کرده است استفاده کند ولی مقدارهای بزرگ باعث توجه بیشتر به اطلاعات اخیر می‌شود ($0 < \alpha \leq 1$).

[1] Q-Learning

[2] Hyper-parameter

نرخ پاداش(γ):

این متغیر تعیین می‌کند که عامل چه‌قدر به پاداش‌های آینده اعتنا کند؛ مقدارهای کوچک برای γ باعث می‌شود عامل به پاداش‌های نزدیک توجه داشته‌باشد ولی برای مقدارهای بزرگ، عامل، پاداش‌های درازمدت در آینده را هم در نظر می‌گیرد ($0 \leq \gamma \leq 1$).

سیستم‌های پیشرفته کمک‌به‌راننده

سیستم‌های پیشرفته کمک‌به‌راننده (ADAS) از جمله سیستم‌های ایمنی فعالی هستند که علاوه بر پارامترهای دینامیک خودرو، به کمک انواع حسگرها اطلاعات محیط پیرامون را نیز استخراج می‌نمایند. این سیستم‌ها کمک می‌کنند که امر رانندگی آسان‌تر و ایمن‌تر انجام شده و در مواقع خطر با هشدار به راننده یا اعمال مستقیم دستورات موجب افزایش ایمنی سرنشینان خودرو، عابران پیاده، خودروهای دیگر و حیوانات می‌شوند.

حلقه‌ی اولیه‌ی این سیستم‌ها پایش محیط، دریافت اطلاعات اجسام و موانع در اطراف خودرو است. با اضافه کردن دوربین به پشت آینه، بستر پیاده‌سازی سه سیستم از سیستم‌های ADAS فراهم می‌شود. در این گزارش تمرکز اصلی بر روی توسعه سیستم ترمز هوشمند اضطراری (AEB) خواهد بود و به سبب ایجاد شدن زیر ساخت‌های لازم، امکان توسعه‌ی سیستم هشدار خروج از خط[1] و سیستم تنظیم نور بالا[2] نیز فراهم خواهدشد.

[1] Lane Departure Warning (LDW)

[2] High Beam Assist (HBA)

سیستم ترمز هوشمند اضطراری

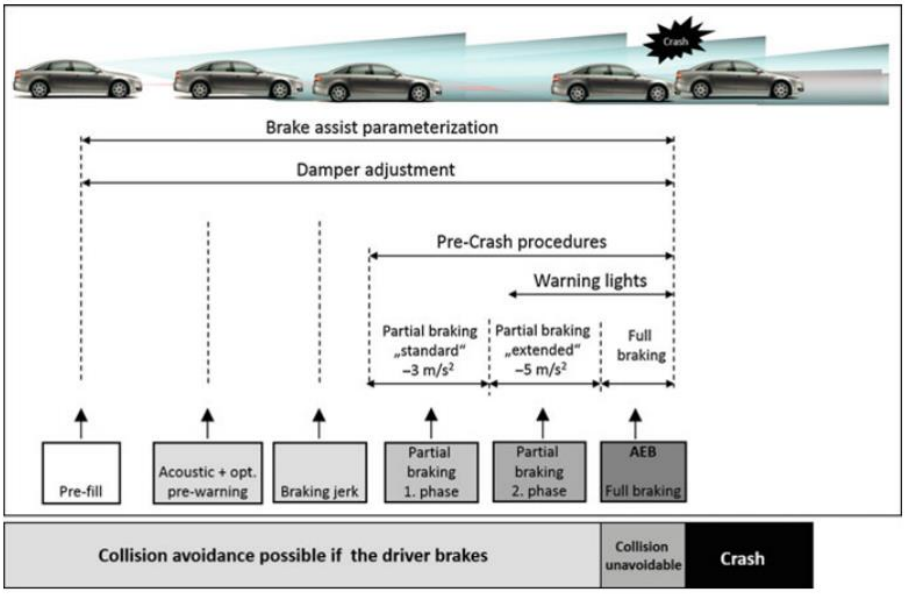
سالانه تعداد بسیار زیادی برخورد از پشت به‌خاطر عدم توجه راننده رخ می‌دهد. حتی یک هشدار کوتاه به راننده نیز می‌تواند به میزان زیادی از تعداد و شدت برخوردها بکاهد. سیستم ترمز هوشمند اضطراری را می‌توان بر اساس نوع حسگر به‌کاررفته در آن‌ها به سه دسته تقسیم کرد: سیستم‌های مبتنی بر دوربین، سیستم‌های مبتنی بر رادار و سیستم‌های مبتنی بر هم‌جوشی داده‌های دوربین و رادار. با نصب سیستم مبتنی بر دوربین بر روی شیشه جلو، وسایل نقلیه‌ای که مستقیماً در جلوی خودرو و در مسیر حرکت آن هستند تا مسافت تقریبی ۶۰ متر قابل‌تشخیص می شوند. سیستم ترمز هوشمند اضطراری این موارد را پایش می‌کند:

- تشخیص حضور خودروهای موجود در جلوی خودرو
- اندازه‌گیری موقعیت و سرعت نسبی خودروها در جلوی خودرو
- تعیین سرعت مطلق خودرو
- تخمین مسیر حرکت خودرو
- ایجاد هشدارهایی برای راننده منطبق با عملکرد و خواسته‌های تعریف شده

این سیستم، به طور پیوسته زمان برخورد[1] و احتمال برخورد را با توجه به سایز و موقعیت خودروهای در تصویر محاسبه می‌کند. اگر به سبب سرعت، خودرو بیش از اندازه به خودروی جلویی نزدیک شود، سیستم، راننده را از احتمال برخورد آگاه می‌کند و در صورت عدم دریافت پاسخ از سمت راننده، ترمز می‌کند. این نکته قابل‌تأمل است که سیستم مذکور، کنترل کامل خودرو را در دست نمی‌گیرد و مانع واکنش راننده نمی‌شود. استفاده از یک دوربین منجر به امکان توسعه یک سیستم ارزان‌قیمت و کارآمد شده که به راحتی قابل نصب است.

[1] Time To Collision (TTC)

اگر مانع طوری ظاهر شود که امکان انجام کاری توسط راننده و سیستم ترمز اضطراری وجود نداشته‌باشد (که به آن حالت برخورد اجتناب‌ناپذیر[۱] می‌گویند)، این سیستم اقدامات کاهشی را انجام می‌دهد؛ مانند فعال کردن تقویت‌کننده‌های ترمز[۲] و آماده کردن کیسه‌های هوا.



شکل ۲-۱ سیستم AEB برای خودروی Audi R8 [2]

اهداف

هدف نهایی شبیه‌سازی سیستم AEB به‌وسیله آموزش یک عامل از طریق یادگیری تقویتی و پیاده‌سازی آن در یک شبیه‌ساز است. جدول حرکات عامل پس از یادگیری استخراج شده و در یک محیط آزموده می‌شود.

[۱] Collision Unavoidable (CU)

[۲] Brake Boosters

فصل دوم

محیط و عامل

فعالیت‌ها و تجربیات کارآموزی

طراحی و پیاده‌سازی عامل

یکی از زیرساخت‌های موجود برای ساخت و توسعه نرم‌افزار برای یادگیری ماشین به شیوه‌ی یادگیری تقویتی، "Gym" ساخته شرکت "OpenAI" است. این کتابخانه، بستر موجود برای ساخت محیط و عامل با در نظر گرفتن جزئیات به همراه خروجی تصویری آن‌ها را فراهم می‌کند. این کتابخانه چند محیط را به‌صورت پیش‌فرض در خود دارد اما هیچ‌کدام از آن‌ها مناسب پیاده‌سازی سیستم AEB نبودند؛ به همین منظور محیط و عامل باید از پایه طراحی شوند.

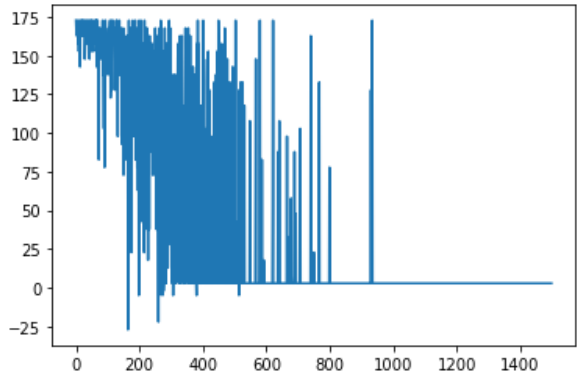
در ابتدا لازم بود تا محیط تا حد امکان ساده‌سازی شود؛ به همین منظور، فرض شده که مانع به صورت ساکن قرار دارد و سرعت ماشین نیز ثابت است؛ بدیهی است که چنین فرضیاتی بخشی از همه‌ی حالات ممکن در دنیای واقعی را پوشش می‌دهد.

سیستم‌های مورد استفاده در حال حاضر، از همان ابتدا از تمام قدرت ترمز ماشین استفاده نمی‌کنند؛ بلکه با فعال کردن در صدی از ترمز سعی می‌کنند تا جلوی برخورد را بگیرند و در صورت وجود احتمال برخورد، ترمز را با قدرت بیشتری فعال می‌کنند؛ اما به دلیل ضیق وقت، تصمیم بر آن شد که عامل از تمام قدرت ترمز استفاده کرده و بر اساس آن، خود را با شرایط وفق دهد.

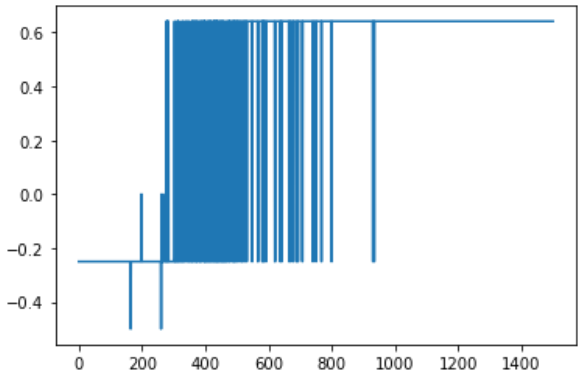
در ابتدا عامل با محیط بیگانه اســـت و نمی‌تواند تصـــمیمی اتخاذ کند؛ برای همین، لازم اســـت تا ترمزگیری را در م سافت‌های مختلف تا مانع امتحان کند تا م سافت ایده‌آل ترمزگیری را بیابد. عامل پس از توقف، پادا شی متنا سب با فا صله از مانع دریافت می‌کند؛ در صورت برخورد با مانع و توقف با فا صله‌ی طولانی از مانع امتیاز منفی و در صـــورت توقف نزدیک مانع، متناســـب با فاصـــله، امتیاز مثبت دریافت می‌کند.

هم‌چنین، در کنار ابرپارامترهای α و γ، نیاز به تعریف یک پارامتر دیگر برای تعیین میزان بهره‌گیری یا کاوش عامل بود و ابرپارامتر ε برای همین منظور درنظر گرفته‌شد که تابعی نمایی است؛ به این صورت که ابتدا عامل تنها به آزمون و خطا پرداخته و با گذر زمان از داده‌های خود برای تصـــمیم‌گیری اســـتفاده می‌کند.

این عامل ۱۵۰۰ بار در شرایط یکسان قرار گرفته‌است. شکل ۳-۱ فاصله تا مانع بعد از توقف و شکل ۳-۲ پاداش دریافتی عامل در هر مرحله را نشان می‌دهد.



شکل ۳-۱ فاصله عامل تا مانع بعد از توقف در هر مرحله



شکل ۳-۲ پاداش دریافتی عامل در هر مرحله

عامل تقریباً پس از ۱۰۰۰ بار آزمون و خطا به حالت بهینه رسیده است.

فصل سوم

نتیجه‌گیری

نتیجه‌گیری

خلاصه کار

ابتدا یک عامل از طریق تکنیک‌های یادگیری ماشین برای ترمزگرفتن در زمان درست آموزش داده شد؛ هدف از این کار، طراحی یک سیستم ترمز هوشمند اضطراری ساده شده در محیط نرم‌افزاری بود. سپس، نتایج آموزش در یک محیط شبیه‌ساز صنعتی پیاده‌سازی شد.

کاربرد نتایج

در شرایط دنیای واقعی، جهت کاربردی‌کردن این سیستم، باید مواردی که در بخش تعمیم به آن‌ها اشاره شده‌است، درنظر گرفته شود؛ در هر صورت، می‌توان از این بستر برای توسعه این سیستم و سعی در شبیه‌کردن آن به شرایط دنیای واقعی و یا حتی پیاده‌سازی سیستم‌های دیگر استفاده‌کرد.

پیشنهادات برای توسعه سیستم

در شرایط مفروض، مانع، یک جسم ساکن است. سرعت خودروی مورد بررسی نیز ثابت فرض شده‌است؛ می‌توان برای نزدیک‌تر کردن شرایط به واقعیت، موارد زیر را در نظر گرفت:

- برای جلوگیری از تصادف لزوماً نه حداکثر ترمز بلکه بخشی از آن اعمال شود
- شتاب خودرو ناصفر باشد
- مانع، متحرک با شتاب ثابت باشد
- مانع، متحرک با شتاب متغیر باشد
- شرایط جاده (مانند لغزندگی) درنظر گرفته‌شود
- عامل توانایی تشخیص شرایطی که برخورد حتمی است را داشته‌باشد

منابع و مراجع

[1] H. Winner, S. Hakuli, F. Lotz, C. Singer, Handbook of Driver Assistance Systems, 1st ed, New York, NY: Springer, 2016.

[2] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed, Canada, CA: O'Reilly, 2019.

```
# Core Library
import math
import random
from typing import Any, Dict, List, Tuple

# Third party
import gym
import numpy as np
import matplotlib.pyplot as plt
from gym import spaces


# Hyperparameters
alpha = 0.1
gamma = 0
epsilon = 1

# For plotting metrics
all_epochs = []
all_penalties = []


q_table = np.zeros([210, 2])


class AEBEnv(gym.Env):
    """
    Define a simple Track environment.

    The environment defines which actions can be taken at which point and
    when the agent receives which reward.
    """

    def __init__(self, start, object_position, maximum_brake_acc, initial_speed):
        self.starting_distance = start
        self.object_position = object_position
        self.is_car_stopped = False

        # self.car_weight = weight  # measured in kg
        self.maximum_brake_acc = maximum_brake_acc  # measured in m/s2
        self.initial_speed = initial_speed  # measured in m/s
```

```
    self.current_speed = initial_speed
    self.current_position = 0

    # Define what the agent can do
    ### Brake at 0.0 or 1.0 intensity
    self.action_space = spaces.Discrete(2)

    # Observation is the remaining distance
    low = np.array([0.0])
    high = np.array([self.current_position - object_position])
    self.observation_space = spaces.Box(low, high, dtype=float)

    # Store what the agent tried
    self.current_episode = -1
    self.action_episode_memory: List[any] = []

def step(self, action: int) -> Tuple[int, float, bool, int]:
    """
    The agent takes a step in the environment.

    Parameters
    ----------
    action : float

    Returns
    -------
    ob, reward, episode_over, info : tuple
        ob : List[int]
            an environment-specific object representing your observation of
            the environment.
        reward : float
            amount of reward achieved by the previous action. The scale
            varies between environments, but the goal is always to increase
            your total reward.
        episode_over : bool
            whether it's time to reset the environment again. Most (but not
            all) tasks are divided up into well-defined episodes, and done
            being True indicates the episode has terminated. (For example,
            perhaps the pole tipped too far, or you lost your last life.)
        info : Dict
```

```
            diagnostic information useful for debugging. It can sometimes
            be useful for learning (for example, it might contain the raw
            probabilities behind the environment's last state change).
            However, official evaluations of your agent are not allowed to
            use this for learning.
        """
        self.curr_step += 1

        #brake_intensity = float(action / 10)

        self._take_action(action)
        reward = self._get_reward()
        ob = self._get_state()
        info = self.object_position - self.current_position
        return ob, reward, self.is_car_stopped, info

    def _take_action(self, action: int) -> None:
        self.action_episode_memory[self.current_episode].append(action)

        #self.decrease_speed(action)
        if action == 0:
            return
        else:
            self.calculate_stopping_point(action)

            ##is_car_stopped = self.current_speed == 0
            self.is_car_stopped = True

    def _get_reward(self) -> float:
        """Reward is given according to the distance to the object."""
        distance = self.object_position - self.current_position
        desired_distance = 10
        if self.is_car_stopped == True:
            if distance > desired_distance:
                return -.5
            elif distance < 0:
                #print('-10')
                return -1
            else:
                #reward_tmp = math.exp(-(desired_distance - distance) / desired_distance)
                reward_tmp = math.exp(-(desired_distance - distance)/5)
```

```
            return reward_tmp
        else:
            return 0

    def reset(self, initial_distance) -> List[int]:
        """
        Reset the state of the environment and returns an initial observation.

        Returns
        -------
        observation: List[int]
            The initial observation of the space.
        """
        self.curr_step = -1
        self.current_episode += 1
        self.action_episode_memory.append([])
        self.is_car_stopped = False
        self.current_speed = self.initial_speed
        self.current_position = 0
        self.object_position = initial_distance
        return 0

    def _render(self, mode: str = 'human', close: bool = False) -> None:
        return None

    def _get_state(self) -> List[int]:
        """Get the observation."""
        self.current_position = self.current_position + 5
        ob = self.object_position - self.current_position
        ob = int(ob)
        return ob



    def decrease_speed(self, multiplier):
        self.current_speed = self.current_speed - (multiplier * self.maximum_brake_acc)
        if self.current_speed <= 0:
            self.is_car_stopped = True

    # calculates the distance the car travels before stopping
    def calculate_stopping_point(self, action):
        # time_travelled = speed / brake
```

```
        #brake_acc_applied = float(self.maximum_brake_acc * action / 10)
        distance_travelled = (self.current_speed ** 2) / (2 * self.maximum_brake_acc)
        self.current_position = self.current_position + distance_travelled


starting_distance = random.randint(150, 200)
env = AEBEnv(0, starting_distance, 9, 20)

total_epochs = []
total_penalties = []
total_rewards = []
total_distances = []

episodes_tr = 5001
for i in range(1, episodes_tr):
    epsilon = math.exp(-i/512)

    starting_distance = random.randint(150, 200)
    state = env.reset(starting_distance)

    epochs, penalties, reward, = 0, 0, 0
    done = False
    info = starting_distance
    while not done:
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample() # Explore action space which is (only braking) braking intensity
        else:
            action = np.argmax(q_table[state]) # Exploit learned values

        next_state, reward, done, info = env.step(action)

        if next_state < 0 or next_state > starting_distance:
            done = True
            break

        old_value = q_table[state, action]
        next_max = np.max(q_table[next_state])


        new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
        q_table[state, action] = new_value
```

```
        if reward < 0:
            penalties += 1

        state = next_state
        epochs += 1

    total_distances.append(info)
    total_epochs.append(epochs)
    total_penalties.append(penalties)
    total_rewards.append(reward)

    if i % 500 == 0:
        #print(f"Episode: {i}")
        print(f"Done: %{i/50}")


print("Training finished.\n")

plt.plot(total_rewards, linestyle = 'None', marker='.', alpha = .3, label = 'Reward')
plt.legend(loc = 'lower right')
plt.show()

plt.plot(total_distances, marker='.', linestyle='None', alpha = .1, label = 'Distance to object')
plt.legend()
plt.show()
```