



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)  
دانشکده مهندسی برق

پروژه کارشناسی  
گرایش مخابرات

ماشین های خودران -  
با استفاده از یادگیری تقویتی

نگارش  
محمد رضیئی فیجانی

استاد راهنما  
دکتر وحید پوراحمدی

استاد مشاور  
دکتر حمیدرضا امین داور

شهریور ۱۳۹۸

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

# صفحه فرم ارزیابی و تصویب پایان نامه - فرم تأیید اعضاء کمیته دفاع

در این صفحه فرم دفاع یا تأیید و تصویب پایان نامه موسوم به فرم کمیته دفاع - موجود در پرونده آموزشی - را قرار دهید.

## نکات مهم:

- نگارش پایان نامه/رساله باید به **زبان فارسی** و بر اساس آخرین نسخه دستورالعمل و راهنمای تدوین پایان نامه های دانشگاه صنعتی امیرکبیر باشد.(دستورالعمل و راهنمای حاضر)
- رنگ جلد پایان نامه/رساله چاپی کارشناسی، کارشناسی ارشد و دکترا باید به ترتیب مشکی، طوسی و سفید رنگ باشد.
- چاپ و صحافی پایان نامه/رساله بصورت **پشت و رو(دورو)** بلامانع است و انجام آن توصیه می شود.

به نام خدا

تاریخ: شهریور ۱۳۹۸

## تعهدنامه اصالت اثر



اینجانب **محمد رضیئی فیجانی** متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان‌نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان‌نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان‌نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

محمد رضیئی فیجانی

امضا

این پایان نامه را به پدر، مادر و برادرم که کمک کردند تا این پروژه به اتمام برسد تقدیم  
می‌کنم. همچنین امیدوارم این پروژه گامی نخست برای گام‌های فراتر باشد.

# سپاسگزاری

از دکتر پوراحمدی و دکتر امین داور که کمک های فراوانی جهت پیش برد این هدف بزرگ داشته اند،  
کمال تشکر را دارم. همچنین از سایر دوستانی که من را در این پروژه همراهی و یاری دادند، بسیار  
متشکر هستم.

محمدرضیٰ فیجانی

شهریور ۱۳۹۸

## چکیده

در این پروژه سعی شد تا با استفاده از الگوریتم های یادگیری تقویتی، خودرو خودران تولید شود. در این جا اتومبیل در نقش «عامل» در ادبیات یادگیری تقویتی قرار دارد. در این پروژه با تعریف مناسب «امتیاز»ها، «مشاهده»ها و همچنین پارامترهای الگوریتم های مختلف سعی شد تا هرچه بهتر و بیشتر به این هدف نزدیک شود. گفتنی است که تمامی مراحل کار از قسمت الگوریتم و تعریف پارامتر های ذکر شده تا نوشتن خود نرم افزار محیط شبیه سازی از جمله دستاوردهای این پروژه محسوب می شود. در فصل ۱ توضیح بسیار مختصری در مورد خود مفاهیم یادگیری تقویتی می شود. فصل ۵ راه اندازی کد و نتایج حاصل این پروژه را نشان می دهد. پیش از راه اندازی باید باتوجه به فصل ۲، پیشنیاز های نصب آن تهیه و نصب گردند. همچنین آن فصل توضیح مختصری در مورد چستی هریک از آن پیشنیاز ها ارایه کرده است. در فصل ۳، نحوه تعریف پارامترهای الگوریتم یادگیری تقویتی به صورت کامل بسط داده شده اند. فصل ۴، جزییات پیاده سازی محیط شبیه سازی را نشان می دهد و بر روی جزییات فنی آن تمرکز دارد.

## واژه های کلیدی:

خودرو خودران، یادگیری تقویتی، محیط شبیه سازی، متلب، پایتون، پری اسکن

# فهرست مطالب

عنوان

صفحه

۱	معرفی یادگیری تقویتی	۱
۱-۱	مقدمه	۲
۱-۱-۱	جایگاه یادگیری تقویتی در یادگیری ماشین	۲
۱-۱-۲	وجه تمایز یادگیری تقویتی از دیگر الگوهای یادگیری ماشین	۳
۱-۱-۳	عامل و محیط	۴
۱-۱-۴	حالت	۵
۱-۱-۵	مشاهده پذیری	۶
۱-۱-۶	سیاست	۷
۲-۱	مطالعه بیشتر	۷
۲	پیشنیاز های نصب و معرفی قسمت های مختلف	۸
۲-۱	نرم افزارهای کلی	۹
۲-۲	پیشنیاز های پایتون	۱۰
۲-۳	معرفی دقیق تر اجزای کلی	۱۱
۲-۳-۱	معرفی نرم افزار پری اسکن	۱۱
۲-۳-۲	فرمت های فایل های خروجی	۱۲
۲-۳-۳	نصب موتور متلب	۱۳
۲-۴	معرفی دقیق تر پیشنیاز های پایتون	۱۴
۲-۴-۱	بسته های کمکی	۱۴
۲-۴-۲	بسته gym	۱۴
۲-۴-۳	بسته stable-baseline	۱۶
۳	توضیح مختصری بر الگوریتم	۱۸
۳-۱	معرفی محیط شبیه سازی	۱۹
۳-۲	معرفی رابط برنامه نویسی برنامه و الگوریتم	۲۱
۳-۳	تعریف کردن پارامتر های یادگیری تقویتی	۲۵



۲۶	۱-۳-۳ معرفی برخی توابع رابط برنامه‌نویسی برنامه
۳۲	۲-۳-۳ بررسی تابع <code>_next_observation</code> :
۳۸	۳-۳-۳ بررسی مقدار $\gamma$
۳۹	۴ بررسی جزئیات فنی پروژه
۴۰	۱-۴ مقدمه
۴۰	۲-۴ دورنمای کلی طرح
۴۱	۳-۴ بررسی دقیق‌تر فایل سیمولینک
۴۲	۱-۳-۴ معرفی بلوک Environment و بررسی جزئیات آن
۴۷	۲-۳-۴ بررسی ساختار داده‌های ارسالی و کد آن در سیمولینک
۴۹	۴-۴ بررسی جزئیات بخش پایتون
۴۹	۱-۴-۴ معرفی لایه‌های کد پایتون
۵۱	۵-۴ بررسی دقیق‌تر برخی چالش‌های فنی پروژه
۵۳	۵ شبیه‌سازی و نتایج
۵۴	۱-۵ راه‌اندازی
۵۶	۲-۵ نتایج شبیه‌سازی
۵۸	منابع و مراجع
۵۹	نمایه
۶۰	فهرست اختصارات
۶۱	واژه‌نامه انگلیسی به فارسی
۶۳	واژه‌نامه فارسی به انگلیسی

# فهرست اشکال

صفحه

شکل

۱-۱	جایگاه یادگیری تقویتی	۲
۲-۱		۴
۳-۱	شماتیک تعامل محیط با عامل	۵
۱-۲	تقسیم بندی وظایف اصلی کد پایتون	۱۰
۲-۲	آیکون های اضافه شده بر روی محیط دسکتاپ پس از نصب پری اسکن	۱۱
۳-۲	پنل مدیریت نرم افزار پری اسکن	۱۱
۴-۲	صفحه گرافیکی محیط پری اسکن	۱۲
۱-۳	محیط شبیه سازی	۱۹
۲-۳		۲۰
۳-۳	بررسی دقیق محدوده و مشخصات جاده	۲۰
۴-۳	بررسی تابع action_translate	۲۸
۵-۳	بررسی دقیق تابع افزایش دهنده و کاهنده سرعت ماشین	۳۰
۶-۳	منطق محاسبه done در محیط شبیه سازی	۳۱
۷-۳	نحوه تعریف مشاهده	۳۳
۸-۳	افراز محیط اطراف ماشین برحسب زاویه	۳۳
۹-۳	نمودار تابع محاسبه امتیاز سرعت نرمال شده	۳۵
۱۰-۳		۳۷
۱۱-۳	تابع امتیاز نزدیکی و محل قرار گیری این تابع در فضای مشاهده <sup>۱</sup>	۳۷
۱-۴	بلوک دیالگرام لایه های کلی	۴۰
۲-۴	فایل سیمولینک ایجاد شده توسط نرم افزار پری اسکن همراه با تغییرات	۴۲
۳-۴	روش صحیح اعمال تغییرات روی فایل سیمولینک	۴۳
۴-۴	فایل سیمولینک - شبیه سازی اتومبیل	۴۳
۵-۴	نگاهی از نزدیک به بلوک Environmmnet	۴۵

<sup>1</sup>Observation

۴-۶	فایل سیمولینک - داخل بلوک Environment	۴۵
۴-۷	منطق محاسبه تمام شدن و دستور ریست کردن	۴۷
۴-۸	ساختار جیسون برای بلوک های سیمولینک	۴۸
۴-۹	بلوک دیالگرام لایه های پایتون	۵۰
۵-۱	تصویر شبیه سازی نهایی	۵۷

## فهرست جداول

صفحه

جدول

۱-۲	توضیحات فرمت فایل خروجی	۱۳
۲-۲	معرفی بسته های کمکی پایتون و علت استفاده از آن ها	۱۵
۱-۳	بررسی پارامتر های موجود در <code>env_dict</code>	۲۳
۲-۳	راهنمای توابع اصلی رابط برنامه نویسی برنامه	۲۵
۳-۳	راهنمای توابع کمکی رابط برنامه نویسی برنامه	۲۷
۴-۳	تعریف فضای مشاهده و فضای حرکت در پروژه	۲۷
۱-۴	بررسی ورودی ها و خروجی های مهم در شکل ۴-۴	۴۴
۲-۴	اطلاعات بلوک های فرستندگی گیرندگی در سیمولینک	۴۶
۱-۵	اطلاعات مخزن های پروژه در گیت هاب	۵۴

## فهرست نمادها

نماد	مفهوم
$\mathcal{R}_t$	امتیاز آنی در لحظه $t$
$\mathcal{S}_t$	حالت در لحظه $t$
$\mathcal{S}_t^a$	حالت عامل در لحظه $t$
$\mathcal{S}_t^e$	حالت محیط در لحظه $t$
$\mathcal{O}_t$	مشاهده در لحظه $t$
$\mathcal{A}_t$	حرکت در لحظه $t$
$\ \mathcal{O}\ $	اندازه فضای مشاهده
$\ \mathcal{A}\ $	اندازه فضای حرکت
$\mathcal{H}_t$	تاریخچه تا لحظه $t$
$\mathcal{G}_t$	تابع تجمعی امتیاز، تابع بازگشت
$\gamma$	ضریب کاهنده
$F^{\mathcal{C}} _a$	تابع مرتبط با مفهوم $\mathcal{C}$ با پارامتر $a$
$\pi$	سیاست
$\pi^*$	سیاست بهینه

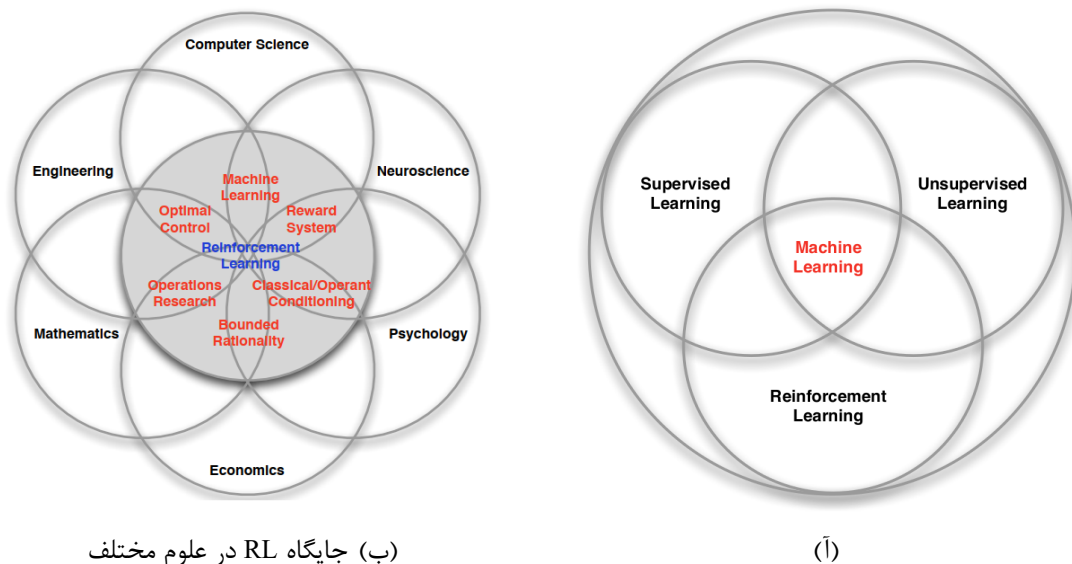
# فصل اول

## معرفی یادگیری تقویتی

## ۱-۱ مقدمه

### ۱-۱-۱ جایگاه یادگیری تقویتی در یادگیری ماشین

بسیاری از صاحب نظران یادگیری ماشین<sup>۱</sup> را به سه دسته تقسیم می‌کنند: (۱) یادگیری با ناظر<sup>۲</sup> (۲) یادگیری بدون ناظر<sup>۳</sup> یا خوشه بندی<sup>۴</sup> (۳) یادگیری شبه ناظر<sup>۵</sup> در این میان، یادگیری تقویتی<sup>۶</sup> را بعضی ها دسته چهارم می‌دانند و بعضی دیگر آن را در دسته سوم قرار می‌دهند. بر اساس دسته‌بندی گروه دوم شکل ۱-۱(آ) رسم شده است. همچنین شکل ۱-۱(ب) کاربرد یادگیری تقویتی را در علوم مختلف نشان می‌دهد.



(ب) جایگاه RL در علوم مختلف

(آ)

شکل ۱-۱: جایگاه یادگیری تقویتی

<sup>1</sup>Machine Learning

<sup>2</sup>Supervised Learning

<sup>3</sup>Unsupervised Learning

<sup>4</sup>Clustering

<sup>5</sup>Semi-Supervised Learning

<sup>6</sup>Reinforcement Learning

## ۱-۱-۲ چه چیزی یادگیری تقویتی را با دیگر الگوهای یادگیری ماشین متمایز می‌کند؟

این سوال از آن جهت حایز اهمیت است که بیان می‌کند چرا ما به سراغ الگوی یادگیری تقویتی رفته‌ایم. پاسخ ملاحظات زیر است.

آ) هیچ ناظر<sup>۷</sup> وجود ندارد و صرفاً امتیاز<sup>۸</sup>ها وجود دارند.

ب) فیدبک<sup>۹</sup> همراه با تاخیر است و به صورت همزمان رخ نمی‌دهد.<sup>۱۰</sup>

ج) مفهوم زمان واقعا مطرح است و یک ترتیب خاص از داده‌ها داریم. شکل ۱-۳ این توالی زمانی را نشان می‌دهد.

یادگیری تقویتی (RL<sup>۱۱</sup>) بر اساس فرضیه امتیازها<sup>۱۲</sup> پایه‌گذاری می‌شود.

**تعریف ۱-۱-۱** (فرضیه امتیازها). همه اهداف می‌توانند براساس بیشینه کردن مقدار میانگین تجمعی امتیازها توصیف کرد.

ممکن است این عبارت کمی عجیب بنظر برسد اما در بسیاری از مسایل که به صورت برد و باخت و به نوعی دو حالت مطلوب و نامطلوب دارند، می‌توان در ساده‌ترین حالت مقدار  $+1$  را برای برد و  $-1$  را برای باخت در نظر گرفت.

**نکته ۱-۱-۲.** در برخی منابع بجای امتیاز از مفهوم هزینه<sup>۱۳</sup> استفاده می‌کنند و هدف الگوریتم آن می‌شود که به سمتی حرکت کند که کمترین هزینه را داشته باشد. برای یک پارچه‌سازی این مفاهیم معمولاً یک علامت منفی برای این دو در نظر می‌گیرند یعنی :

$$r = -c \quad : \quad \text{هزینه} = - \text{امتیاز}$$

<sup>7</sup>Supervisor

<sup>8</sup>Reward

<sup>9</sup>Feedback

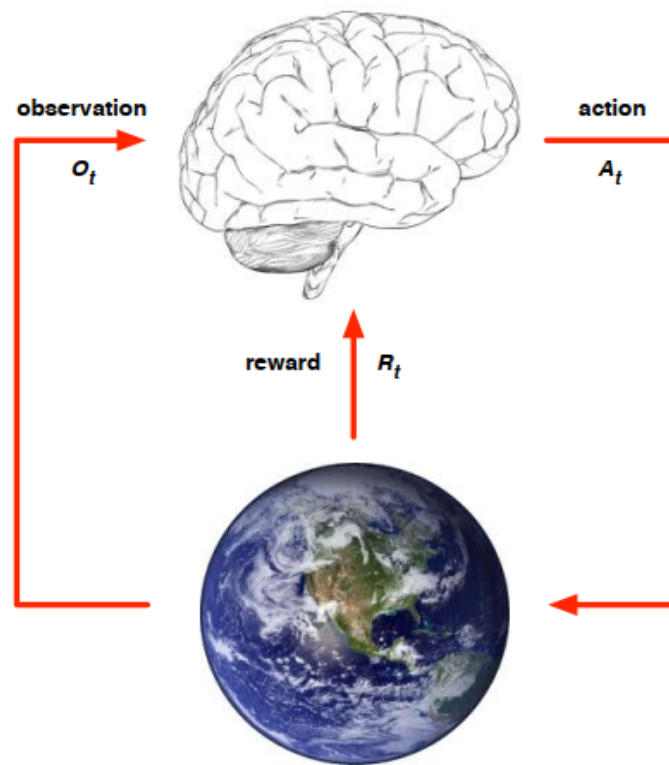
<sup>۱۰</sup>در مورد علت تاخیر در ادامه توضیح داده خواهد شد.

<sup>11</sup>Reinforcement Learning

<sup>12</sup>Reward Hypothesis

<sup>13</sup>Cost





شکل ۱-۲:

### ۳-۱-۱ عامل و محیط

این مفهوم بسیار مفهوم مهمی می‌باشد و بارها از آن در این پروژه یاد شده است. در مسایل یادگیری تقویتی یک **عامل**<sup>۱۴</sup> وجود دارد که در یک **محیط**<sup>۱۵</sup> در حال تعامل است. محیط می‌تواند محیط اطراف عامل باشد و یا هر چیزی که عامل با آن در تعامل است. [۱] این تعامل به این صورت است که عامل که در ابتدا یک حالت<sup>۱۶</sup> اولیه دارد، یک حرکت<sup>۱۷</sup> بر روی محیط در زمان  $t$  انجام می‌دهد. محیط مقدار حرکت در زمان  $t$  را دریافت می‌کند و سپس محیط در زمان  $t + 1$  دو اطلاعات مهم را بر می‌گرداند. (آ) مشاهده (ب) امتیاز شکل ۱-۲ و ۳-۱ این تعامل را نشان می‌دهد. لازم به ذکر است که در پایان هر مرحله<sup>۱۸</sup> مقدار  $t$  یک واحد افزایش می‌یابد.

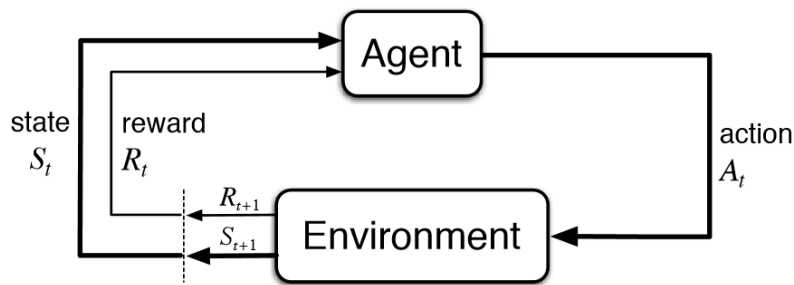
<sup>14</sup> Agent

<sup>15</sup> Environment

<sup>16</sup> State

<sup>17</sup> Action

<sup>18</sup> Step



شکل ۱-۳: شماتیک تعامل محیط با عامل

### ۴-۱-۱ حالت

در بخش قبل تعریف مناسبی از حالت ارائه نشد. برای این تعریف ابتدا مفهوم تاریخچه<sup>۱۹</sup> ارائه می‌شود و از روی آن حالت تعریف خواهد شد.

**تعریف ۱-۱-۳ (تاریخچه).** به سری شامل مشاهده، حرکت و امتیاز می‌باشد:

$$\mathcal{H}_t = \mathcal{O}_1, \mathcal{R}_1, \mathcal{A}_1, \dots, \mathcal{A}_{t-1}, \mathcal{O}_{t-1}, \mathcal{R}_t$$

با این تعریف حالت را می‌توان به شکل زیر تعریف کرد.

**تعریف ۱-۱-۴.** حالت اطلاعاتی است که در محاسبات برای آن که در بعد چه اتفاقی بیافتد، استفاده می‌شود. به عبارت دیگر حالت تابعی از تاریخچه می‌باشد.

$$\mathcal{S}_t = f(\mathcal{H}_t)$$

دو نوع حالت وجود دارد.

(آ) **حالت محیط**<sup>۲۰</sup> که با علامت  $\mathcal{S}_t^e$  نشان داده می‌شود. اطلاعات نهان محیط را نشان می‌دهد و معمولاً برای عامل به‌طور کامل دیده نمی‌شود. حتی اگر برای عامل مشاهده‌پذیر نیز باشد، ممکن است اطلاعات کاملاً بی‌ربطی را همراه داشته باشد.

(ب) **حالت عامل**<sup>۲۱</sup> که با علامت  $\mathcal{S}_t^a$  نشان داده می‌شود. که برابر است با هر اطلاعاتی که عامل برای

<sup>19</sup>History

<sup>20</sup>Environment State

<sup>21</sup>Agent State

رسیدن به حرکت بعدی با استفاده از الگوریتم های RL استفاده می کند.

بنابراین در تعریف ۴-۱-۱ مناسبتر است بجای واژه حالت از حالت عامل استفاده شود. بنابراین:

$$S_t^a = f(\mathcal{H}_t)$$

**یادداشت ۵-۱-۱.** از این پس در سراسر این پایان نامه هر جا صحبت از حالت شد منظور همان حالت عامل است.

**تعریف ۶-۱-۱.** یک حالت  $S_t$  مارکوف<sup>۲۲</sup> است اگر و تنها اگر:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \dots, S_t]$$

در یک حالت مارکوف<sup>۲۳</sup>، آینده از گذشته مستقل است و فقط به زمان حال وابسته است. و این به این معناست که حالت از لحاظ آماری برای توصیف آینده کافی است.

**نکته ۷-۱-۱.** حالت محیط  $S_t^e$  مارکوف است. همچنین تاریخچه نیز مارکوف است.

## ۵-۱-۱ مشاهده پذیری

**مشاهده پذیری کامل**

عامل به طور مستقیم حالت محیط را مشاهده می کند. بنابراین در این حالت داریم:

$$\mathcal{O}_t = S_t^a = S_t^e$$

بنابراین در این حالت عبارت های زیر با یکدیگر برابر هستند.

حالت اطلاعاتی = حالت محیط = حالت عامل

<sup>22</sup>Markov

<sup>23</sup>Markov State

<sup>۲۴</sup> به صورت رسمی، این فرایند یک روند تصمیم‌گیری مارکوف<sup>۲۶</sup> (MDP) می‌باشد. [۲]

## مشاهده پذیری جزئی

عامل به‌طور غیر مستقیم محیط را مشاهده می‌کند.

نمونه ۱-۱-۸. یک ربات با دید دوربین نمی‌تواند موقعیت مطلق را اعلام کند.

نمونه ۱-۱-۹. یک اتومبیل با سنسور تشخیص فاصله نمی‌تواند اطلاعاتی مانند نوع ماشین و قیمت آن را تشخیص دهد.

## ۱-۱-۶ سیاست

سیاست<sup>۲۷</sup> در حقیقت تابعی احتمالی و یا معین است که تصمیم می‌گیرد که در حالت کنونی چه تصمیمی باید گرفت. در واقع رفتار عامل توسط این تابع، بررسی و نشان داده می‌شود.

تعریف ۱-۱-۱۰. اگر تابع معین باشد این تابع به صورت زیر تعریف می‌شود.

$$a = \pi(s)$$

و اگر تابع احتمالاتی باشد به صورت زیر تعریف می‌شود.

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

## ۱-۲ مطالعه بیشتر

جهت مطالعه بیشتر و آشنایی با ادبیات یادگیری تقویتی و همچنین الگوریتم‌های آن می‌توانید به مرجع [۲] و [۱] مراجعه کنید.

<sup>۲۴</sup> حالت اطلاعاتی<sup>۲۵</sup> مفهومی مانند حالت مارکوف دارد.

<sup>۲۶</sup> Markov Decision Process

<sup>۲۷</sup> Policy

## فصل دوم

# پیشنیاز های نصب و معرفی قسمت های مختلف

## ۱-۲ نرم افزارهای کلی

در این پروژه از جهت آنکه نسخه قبلی و پیشینی برای آن نبوده است، به ناچار می‌بایست که کد آن از صفر تا صد آن به صورت دستی نوشته شود. از این‌رو، پیچیدگی‌های بسیار فراوان را به طور خاص در پی داشت. ابزارهای زیادی نیز بنابه شرایط در آن استفاده شد که ارتباط بین آن ابزارها و اجزاء بر این پیچیدگی پیاده سازی طرح افزوده بود.

ابزارهای اصلی و کلی که در این پروژه استفاده شده بود، عبارتند از:

- نرم افزار پری اسکن<sup>۱</sup> ، نسخه 8.5.0

- نرم افزار متلب<sup>۲</sup> ، نسخه R2017b

- زبان برنامه نویسی پایتون ، نسخه 3.6.9

بنابراین برای راه اندازی مجدد کد این پروژه لازم است که موارد بالا روی کامپیوتر شخص به صورت کامل نصب باشد.

همچنین لازم به ذکر است که برخی ابزارات دیگر نیز در این پروژه استفاده شده است که احتمالاً با نصب موارد بالا دیگر نیازی به نصب آن‌ها به صورت جداگانه نیست. هدف این ابزارها ایجاد اتصال بین اجزای اصلی گفته شده است. این گروه شامل موارد زیر هستند:

- سیمولینک<sup>۳</sup> ، جهت اتصال بین متلب و پری اسکن

- شبکه UDP<sup>۴</sup> ، جهت اتصال داده های پویا<sup>۵</sup> بین پایتون و سیمولینک

- موتور متلب<sup>۶</sup> ، جهت اتصال داده های ساکن<sup>۷</sup> بین پایتون و سیمولینک

در این فصل جزئیات بیشتری در مورد لزوم و دلیل استفاده از این ابزارها بررسی می‌شود.

---

<sup>1</sup>PreScan

<sup>2</sup>Matlab

<sup>3</sup>Simulink

<sup>۴</sup>برای این منظور از socket در پایتون استفاده شده است.

<sup>5</sup>Dynamic Data

<sup>6</sup>Matlab Engine

<sup>7</sup>Static Data



شکل ۱-۲: تقسیم بندی وظایف اصلی کد پایتون

## ۲-۲ پیشنهاد های پایتون

یادداشت ۱-۲-۲. کد پایتون در این پروژه شامل دو قسمت کلی زیر می شود. این دو دسته در شکل ۱-۲ مشخص هستند.

۱. دسته اول مربوط به آن بخش از پروژه است که وظیفه اصلی آن ارتباط پیدا کردن با محیط متلب و پری اسکن و ایجاد یک نوع واسط کاربری است. گرفتن و فرستادن اطلاعات مخصوص این قسمت است.

۲. دسته دوم با محیط و نحوه ارتباط آن کاری ندارد و تمرکز خود را بر روی الگوریتم خود که در این جا از الگوریتم های یادگیری تقویتی استفاده شده است، قرار داده است.

دسته اول (سمت چپ تصویر ۱-۲) به پکیج های زیر احتیاج دارد:

- matlab.engine •
- os و time •
- numpy •
- gym •
- pandas •
- socket •

اگر از آناکوندا<sup>۸</sup> برای پایتون استفاده می کنید غیر از دو بسته gym و matlab.engine به صورت پیش فرض نصب شده اند در صورت عدم نصب آن ها را با استفاده از pip<sup>۹</sup> می توان نصب کرد. بسته gym که در این فصل به تفصیل در مورد آن بحث شده است، به راحتی با همان دستور pip نصب می شود. اما نصب matlab.engine یا همان موتور متلب متفاوت است و نمی توان آن را نیز به همان روش نصب کرد.

دسته دوم شامل بسته های زیر است:

<sup>۸</sup>Anaconda

<sup>۹</sup>مثلا بسته numpy را با استفاده از دستور pip install numpy نصب می توان کرد.

• gym[atari] یا gym[all]

• tensorflow

• stable-baseline

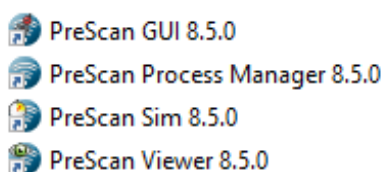
این بسته ها در لایه الگوریتم استفاده شده است. (در مورد این لایه در فصل ۳ بیشتر صحبت خواهد شد). هر سه تایی این بسته ها با همان دستور pip به راحتی نصب می شوند.

## ۳-۲ معرفی دقیق تر اجزای کلی

در این قسمت میخواهیم سه نرم افزار کلی این پروژه را از نگاهی نزدیک تر بشناسیم که عبارتند از :  
(۱) نرم افزار پری اسکن (۲) متلب (۳) پایتون

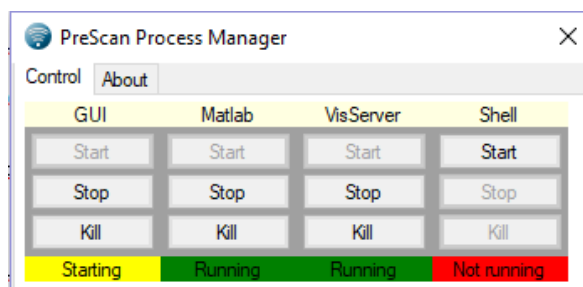
### ۱-۳-۲ معرفی نرم افزار پری اسکن

پس از دانلود و نصب نسخه 8.5.0 این نرم افزار چهار آیکون مانند شکل ۲-۲ به محیط دسکتاپ اضافه می کند. اصلی ترین آن ها PreScan Proccess Manager 8.5.0 نام دارد.



شکل ۲-۲: آیکون های اضافه شده بر روی محیط دسکتاپ پس از نصب پری اسکن

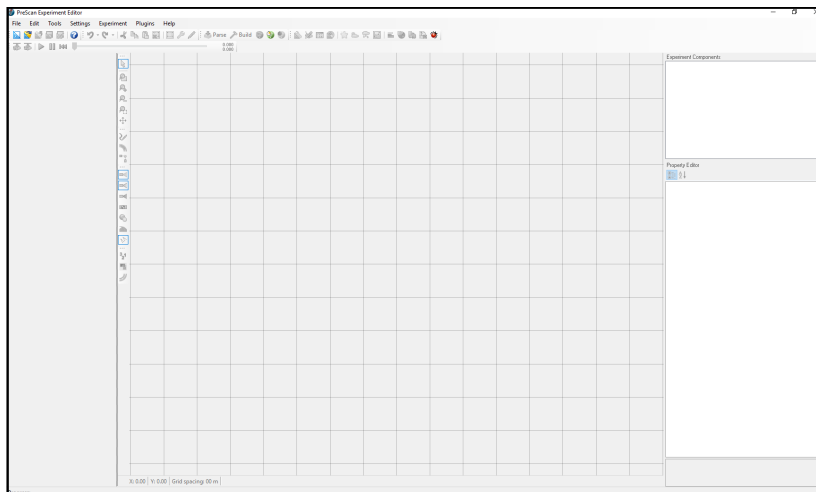
با انتخاب آن صفحه ای مانند زیر باز می شود.



شکل ۳-۲: پنل مدیریت نرم افزار پری اسکن

این پنجره شامل گزینه های زیر است:





شکل ۲-۴: صفحه گرافیکی محیط پری اسکن

• Matlab

• GUI

• Shell

• VisServer

برای ایجاد یک محیط جدید باید GUI را استارت کرد. پس از مدتی صفحه ای مانند شکل ۲-۴ باز می شود.

پس از ایجاد مدل ها و ذخیره آن، فایل های \*\*.pex و \*\*.pb و \*\*\_cs.slx ساخته می شود.<sup>۱۰</sup> جهت استفاده از فایل سیمولینک باید در شکل ۲-۳ متلب را استارت کنید.

**نکته ۲-۳-۱.** برای اجرای فایل های سیمولینک خروجی، لازم است که متلب را فقط و فقط با استفاده از نرم افزار پری اسکن و با استفاده از پنل مدیریت نرم افزار معرفی شده در شکل ۲-۳ باز شود. در صورتی که به صورت مستقیم این کار انجام شود، به مشکل منتهی می شود.

دو قسمت دیگر نیز در شکل ۲-۳ وجود دارد که نیازی به استارت کردن آن ها نیست و خودشان در صورت لزوم به صورت خودکار فراخوانی می شوند.

## ۲-۳-۲ فرمت های فایل های خروجی

نرم افزار پری اسکن پس از ایجاد یک محیط جدید، فایل ها و پوشه های بسیار زیادی را ایجاد می کند. اما در خارج آن پوشه ها ۳ فایل وجود دارد که پسوند آن ها \*\*.pex و \*\*.pb و \*\*\_cs.slx می باشد.

<sup>۱۰</sup>علامت \* به معنای یک اسم مشترک در این سه فایل استفاده شده است.

علامت \*\* همان اسم پروژه ای است که ایجاد کرده ایم. هر یک از این فایل ها به یک بلوک از شکل ۱-۴ مربوط می شود.

فرمت فایل	توضیحات
** .pex	این فایل مربوط به اولین بلوک شکل ۱-۴ است و ارتباط مستقیم با GUI دارد. برای تغییر محیط گرافیکی باید این فایل را باز کرد.
** .pb	این فایل برخی از اطلاعات فایل ** .pex را در اختیار دارد و با تغییر آن فایل این فایل نیز عوض می شود. این فایل حاوی اطلاعات استاتیک محیط ایجاد شده است و مهم ترین کاربرد آن در بلوک موتور متلب که در شکل ۱-۴ نشان داده شده است می باشد. پایتون از طریق این فایل این اطلاعات را دریافت می کند.
**_cs.slx	این فایل سیمولینک است که برای کار کردن با آن باید از پنل مدیریت شکل ۲-۳ استفاده کرد. این فایل پس از ایجاد از فایل ** .pex مستقل می شود. این فایل خود قابلیت تغییر دارد و می توان بلوک های آن را در محیط سیمولینک تغییر داد و بلوک های دیگری به آن افزود. در صورتی که فایل ** .pex تغییر کند، این امکان را نیز دارد که از داخل خود سیمولینک با فشردن دکمه ای این تغییرات جدید اعمال شود بدون آن که به تغییرات خود کاربر لطمه ای وارد شود. در این پروژه این فایل، تغییرات بسیاری را تجربه کرد.

جدول ۱-۲: توضیحات فرمت فایل خروجی

جدول ۱-۲ توضیحات لازم را جهت آشنایی با این خروجی ها آورده است. همچنین در بخش ۳-۴ در مورد فایل \*\_cs.slx توضیحات دقیق تری در مورد جزئیات آن گفته خواهد شد.

### ۳-۳-۲ نصب موتور متلب

برای نصب موتور متلب ابتدا نیاز است که به متلب به طور کامل در سیستم نصب باشد. پس از نصب متلب، محیط Command prompt (admin) را باز کنید و با توجه به نسخه و محل نصب متلب خود به آدرس زیر بروید.

```
<matlabroot>\extern\engines\python
```

مثلا برای Matlab R2017b که در محل پیش فرض خود نصب شده باشد این کار با استفاده از دستور زیر انجام می شود.

```
cd C:\Program Files\MATLAB\R2017a\extern\engines\python
```

در این پوشه یک فایل به نام `setup.py` موجود می باشد. این فایل را با استفاده از دستور

```
python setup.py install
```

در همان محیط `cmd` اجرا کنید.

**یادداشت ۲-۳-۲.** توجه داشته باشید که باید نسخه متلب و پایتون شما باید با یکدیگر سازگار باشند. برای بررسی این موضوع اگر فایل `setup.py` را با اسفاده از یک ادیتور باز کنید، یک آرایه به نام `_supported_versions` در آن خواهید دید. مقادیر این آرایه، نسخه هایی از پایتون را نشان می دهد که توسط نسخه متل شما پشتیبانی میشود مثلاً در این مورد، با توجه به خط زیر نسخه های ۲/۷، ۳/۴، ۳/۵ و ۳/۶ پایتون پشتیبانی می شود. در غیر این صورت باید نسخه سازگار متلب و یا پایتون را نصب کنید.

```
_supported_versions = ['2.7', '3.4', '3.5', '3.6']
```

## ۴-۲ معرفی دقیق تر پیشنهاد های پایتون

### ۱-۴-۲ بسته های کمکی

این بسته ها نقش حیاتی ندارند و برای برخی از موارد استفاده شده اند. این موارد در جدول ۲-۲ آمده است.

### ۲-۴-۲ بسته gym

#### معرفی

بسته `gym` که توسط `OpenAI` توسعه یافته است. این ابزار فوق العاده این امکان را برای محققین علوم کامپیوتر حرفه ای و یا آماتور فراهم می کند که انواع الگوریتم های یادگیری تقویتی (RL) را بر روی کار خود تست کنند. همچنین پتانسیل این را دارد که محققین محیط خود را بر روی این بسته توسعه دهند. هدف از ایجاد این بسته، استاندارد سازی محیط و نوعی نقطه تراز<sup>۱۶</sup> برای پژوهش های RL محسوب می شود.<sup>[۳]</sup>

در حقیقت می توان این بسته را در وسط شکل ۱-۲ جای داد. جایی که لایه محیط و لایه الگوریتم<sup>۱۷</sup>

<sup>16</sup>Bechmark

<sup>17</sup>Algorithm

نام بسته	دلیل استفاده	روش نصب
numpy	ایجاد ماتریس برای فضای حرکت <sup>۱۱</sup> و فضای مشاهده <sup>۱۲</sup>	pip install numpy
time	جهت ایجاد تاخیر و سقف زمانی <sup>۱۳</sup>	pip install time
os	برای بستن پنجره های باز شده پس از اجرا	pip install os
pandas	برای چاپ اطلاعات آماری امتیاز های بدست آمده در پایان هر اپیزود <sup>۱۴</sup>	pip install pandas
socket	برقراری ارتباط با متلب و فرستان و دریافت کردن داده های پویا	pip install socket
tensorflow	برای لایه الگوریتم و استفاده از الگوریتم های یادگیری تقویتی عمیق <sup>۱۵</sup>	pip install tensorflow

جدول ۲-۲: معرفی بسته های کمکی پایتون و علت استفاده از آنها

به یکدیگر می رسند.

این بسته محیط هایی از پیش ساخته شده دارد. نام این بسته ها در لیست زیر آمده است. <sup>۱۸</sup> اکثر این محیط ها نوعی بازی هستند که عامل سعی در یادگیری آن محیط ها دارد.

- Pong-v0 •
- MsPacman-v0 •
- SpaceInvaders-v0 •
- Seaquest-v0 •
- LunarLanderV2 •
- Reacher-v2 •
- FrozenLake-v0 •
- CartPole-v0 •
- Pendulum-v0 •
- MountainCar-v0 •
- MountainCarContinuous-v0 •
- BipedalWalker-v2 •
- Humanoid-V1 •
- Riverraid-v0 •
- Breakout-v0 •

## نصب

برای نصب نسخه کمینه این نرم افزار با همان روش pip به راحتی می توان نرم افزار مورد نظر را نصب کرد. [۴] این نسخه کمینه برای لایه محیط کافی می باشد. اما اگر بخواهیم بخش الگوریتم را با استفاده از کتابخانه های دیگری مانند stable-baseline نوشت. نیازمند نسخه جامع تری از gym می باشد.

<sup>۱۸</sup> جدول کامل در سایت <https://github.com/openai/gym/wiki/Table-of-environments> قرار دارد.

**یادداشت ۲-۴-۱.** پیشنهاد می شود برای نصب نسخه کامل gym و stable-baseline از لینوکس بجای ویندوز استفاده کنید. زیرا در نصب برخی بسته ها ممکن است با مشکل روبرو شوید.

برای نصب کامل این بسته از دستور `pip install gym[all]` استفاده کنید. ممکن است در نصب mujoco به مشکل برخوردید در این صورت دستور `pip install gym[atari]` استفاده کنید. اگر موفق به نصب این بسته نشدید می توانید مراجل نصب آن را با استفاده از [۳] مراجعه کنید.

## ۲-۴-۳ بسته stable-baseline

این بسته مجموعه ای از الگوریتم های از پیش تعریف شده در یادگیری تقویتی می باشد. در صورتی که محیط در gym رجیستری شده باشد، می توان از آن در پروژه های مختلف استفاده کرد. این الگوریتم ها عبارتند از:

PPO2 •	DQN <sup>20</sup> •	A2C <sup>19</sup> •
SAC •	GAIL •	ACER •
TD3 •	HER •	ACKTR •
TRPO •	PPO1 •	DDPG •

این کتابخانه، یک کتابخانه بسیار پویا می باشد و هر لحظه در حال آپدیت شدن می باشد.

**نکته ۲-۴-۲.** برخی از بسته هایی که در این پروژه کتابخانه استفاده شده است، صرفاً بر روی سیستم عامل لینوکس قابل استفاده هستند. بنابراین برای نصب این بسته، باید از سیستم عامل لینوکس استفاده کرد.

واضح است که این بسته در لایه الگوریتم شکل ۲-۱ استفاده می شود. این بسته برای نصب شدن به gym به صورت کامل نیاز دارد. همچنین این بسته برای کار کردن با شبکه های عصبی مصنوعی از کتابخانه tensorflow استفاده می کند.

پس از نصب `gym[all]` و tensorflow با دستور زیر این بسته نصب خواهد شد. برای ادامه نصب به مرجع [۵] مراجعه شود.

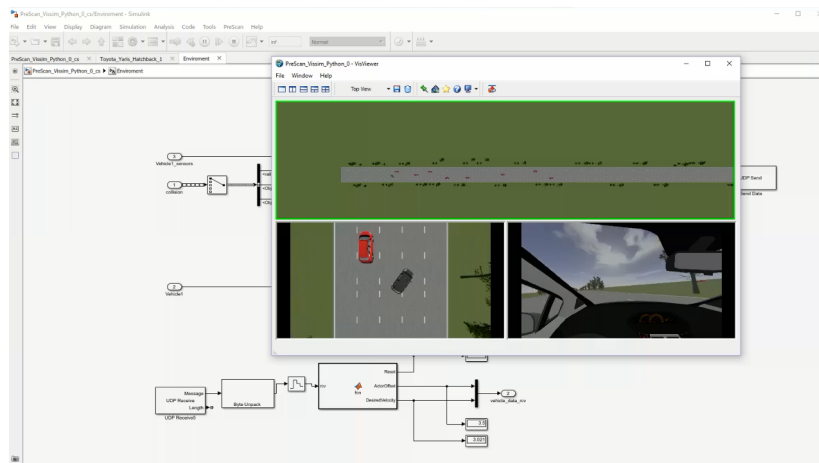
<sup>19</sup>Synchronous Actor Critics

<sup>20</sup>Deep Q-Learning Network

همچنین بسیاری از پروژه های تست شده این کتابخانه در [۶] قابل مشاهده است.

## فصل سوم

### توضیح مختصری بر الگوریتم



شکل ۳-۱: محیط شبیه سازی

در فصل ۱ در مورد مفاهیم یادگیری تقویتی بحث شد. مهمترین مفاهیم عبارتند از:

۱. محیط
۲. عامل
۳. حالت محیط
۴. حالت عامل
۵. امتیاز
۶. مشاهده پذیری<sup>۱</sup>

هدف در این پروژه این بود که یک ماشین خودران<sup>۲</sup> با استفاده از الگوریتم های یادگیری تقویتی ساخته شود. جزییات تئوری الگوریتم و جزییات فنی پروژه به ترتیب در بخش های ۱ و ۴ آورده شده اند. در این بخش به شبیه سازی و جزییات کار و تعریف پارامتر های این پروژه پرداخته می شود.

## ۳-۱ معرفی محیط شبیه سازی

در ابتدا محیط شبیه سازی را معرفی می کنیم. جزییات فنی این محیط در ۴ و همچنین نحوه راه اندازی آن در بخش ۵-۱ به صورت کامل مورد بحث قرار گرفته است. اگر آن محیط را باز کنید محیط مانند شکل ۳-۱ باز خواهد شد. این محیط دو آبجکت مهم دارد؛ (آ) ماشین (اتومبیل) (ب) جاده (شکل ۳-۲) چیزی که اهمیت دارد اندازه ها و نحوه تعریف محدوده هاست. شکل ۳-۳ اندازه ها و محدوده ها را مشخص کرده است. شکل ۳-۳(ب) نشان می دهد که این محدوده ها کاملاً بر روی یک دیگر منطبق نیستند. دلیل اصلی این موضوع عدم اهمیت تطبیق دقیق این دو می باشد. در بخشی که پشت ماشین قرار دارد این محدوده از ۴- (کمی بیشتر از اندازه عرض لاین ها) شروع می شود. زیرا نیازی نیست بیشتر

<sup>1</sup>Observability

<sup>2</sup>Autonomous Vehicle





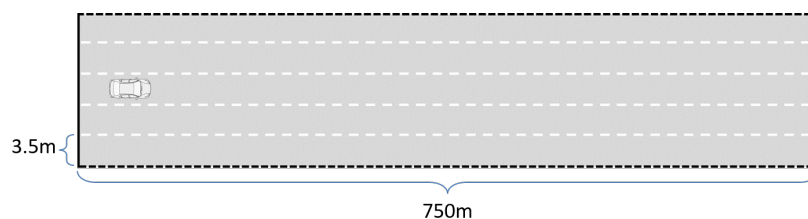
شکل ۳-۲:

از این مقدار ماشین مورد بررسی به عقب برود تا متوجه شویم اشتباه در حال رفتن است. درحقیقت این مورد کمک می‌کند تا تعداد مرحله‌ها را در هر اپیزود اشتباه کاهش یابد. بخش‌های کناری نیز از ۱۱- تا ۱۱+ محدود شده‌اند (بیشتر از عرض خود جاده) تا اگر نوسانی یافت به ماشین این اجازه داده شود تا به مسیر اصلب برگردد.

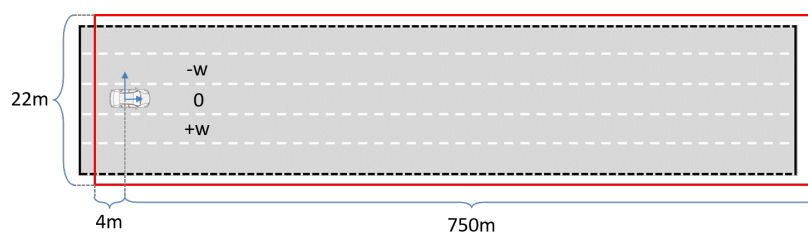
**یادداشت ۳-۱-۱.** ماشین در مبدا صفحه قرار دارد. از این رو اعداد منفی نسبت به همین ماشین نیز سنجیده می‌شوند.

۳ مقدار  $+w$ ،  $-w$  و  $0$  که در شکل ۳-۳(ب) بر روی جاده نوشته شده است در حقیقت مرتبط با بحث فنی ماجرا می‌باشد اما مفهوم آن این است که عامل مورد بررسی می‌تواند این سه لاین را به عنوان حرکت اختیار کند. در حقیقت می‌توان آن‌ها را به عنوان اسم برای هر لاین در نظر گرفت. در مورد حرکت بیشتر صحبت خواهد شد.

**یادداشت ۳-۱-۲.** راه‌اندازی این محیط کمی دردسر خواهد داشت از این‌رو نیاز است پیش از راه‌اندازی



(آ) مشخصات جاده



(ب) محدوده جاده

شکل ۳-۳: بررسی دقیق محدوده و مشخصات جاده

بخش ۱-۵ به طور دقیق مطالعه شود.

## ۲-۳ معرفی رابط برنامه نویسی برنامه و الگوریتم

در این پروژه دو الگوریتم DQN و A2C بهتر از سایر الگوریتم ها عمل کردند اما در نهایت با توجه به آزمایش ها و ملاحظات که انجام شد، الگوریتم DQN از لحاظ سرعت همگرایی بهتر از الگوریتم A2C پاسخ داد. بنابراین صرفاً بر روی این الگوریتم بحث خواهد شد.

```

1 import gym, gym_prescan
2
3 from stable_baselines.common.vec_env import DummyVecEnv
4 from stable_baselines.deepq.policies import MlpPolicy
5 from stable_baselines import DQN
6
7 save_load = "deepq_prescan"
8 env_dict = {
9     'id': 'prescan-without-matlabengine-v0',
10    'verbose': True,
11    'host': '172.21.217.140',
12    # 'delay': 1,
13    'nget': 150
14 }
15
16 env = gym.make(**env_dict)
17 env = DummyVecEnv([lambda: env])
18 model = DQN(MlpPolicy, env, verbose=1, gamma=0.8,
19             prioritized_replay=True)
19 print('Model created!')
20 try:
21     model.learn(total_timesteps=50000)
22 except:
23     print('Error!')
24 model.save(save_load)

```

این کد بخش آموزش<sup>۳</sup> را نشان می دهد. بخش تست<sup>۴</sup> در تمامی الگوریتم ها مشابه یک دیگر است و

<sup>۳</sup>Train

<sup>۴</sup>Test

از جایی که مدل تعریف می‌شود (در اینجا خط ۱۸) شروع خواهد شد. بخش تست در تمامی الگوریتم‌ها کد زیر است.

```

1 model = DQN.load(save_load)
2 obs = env.reset()
3 while True:
4     print('-----TEST-----')
5     action, _states = model.predict(obs)
6     obs, rewards, dones, info = env.step(action)
7     env.render()

```

از روی چند خط آغازین کد DQN می‌توان دریافت که این کد با استفاده از gym [۴] و stable-baseline [۷] نوشته شده است.

بخش مهم بعدی متغیری از جنس دیکشنری به نام env\_dict است. این متغیر برای ساختن متغیر env در دستور env = gym.make(\*\*env\_dict) به کار می‌رود. <sup>۵</sup> توضیح این متغیر و اجزای آن در جدول ۱-۳ آمده است.

همان‌طور که در جدول ۱-۳ توضیح داده شده است؛ دو متغیر nget و delay هر دو از جنس تاخیر می‌باشند. محل تاخیر در تابع step می‌باشد. کد زیر محل تاخیر را نشان می‌دهد.

```

1 def step(self, action):
2     self.send(action)
3     # ----- BEGIN DELAY -----
4     if self.delay > 0 :
5         sleep(self.delay)
6     for _ in range(self.nget):
7         self.render_()
8         if self.done:
9             break
10    # ----- END DELAY -----
11    observation = self._next_observation()
12    reward = self.calc_reward()
13    done = self.done
14    info = {'Collision':self.collision
            , 'Position':self.agent['data']['Position']}

```

<sup>۵</sup> متغیر env در حقیقت نقش محیط را در الگوریتم دارد.

متغیر	توضیحات
id	در این پروژه این متغیر دو حالت بیشتر ندارد که هردو از جنس رشته هستند. اگر این کد با استفاده از موتور متلب استفاده شود، 'prescan-v0' خواهد بود و اگر از موتور متلب استفاده نشده باشد مقدار آن 'prescan-without-matlabengine-v0' خواهد بود. این متغیر مقدار پیش فرض ندارد.
verbose	این متغیر که از جنس بولین می باشد، در صورتی که یک باشد اطلاعات جامعی را در هر مرحله را چاپ می کند. علاوه بر آن اطلاعات آماری امتیازهای بدست آمده در پایان هر اپیزود را نیز چاپ می کند. به طور کلی اجازه گزارش دادن و ندادن اطلاعات درونی الگوریتم توسط این متغیر کنترل می شود.
host	این متغیر برای اتصال شبکه بین دو کامپیوتر به کار می رود و در حقیقت IP کامپیوتری است که ویندوز بر روی آن نصب است. مقدار پیش فرض این متغیر 'localhost' می باشد.
delay	همان طور که مشخص است این متغیر مقدار تاخیر را مشخص می کنیم و مورد کاربرد آن لحظه ای است که action در تابع step فرستاده شده است و پس از گذشت مقداری تاخیر برحسب ثانیه سعی در دریافت اطلاعاتی مانند مشاهده بعدی و محاسبه امتیاز و ... داریم. مقدار پیش فرض این متغیر نیز صفر است.
nget	این متغیر نیز به نوعی متفاوت تاخیر را شکل می دهد. این متغیر از نوع عدد صحیح می باشد و هنگامی که مقدار آن ۱۵۰ است یعنی محل تاخیر، ۱۵۰ بار داده ها را دریافت می کند و مقدار آن ها را می خواند. در حالت عادی تا پایان ۱۵۰ امین دریافت هیچ کاری نمی کند مگر این که مقدار done برابر یک شود؛ در این صورت حلقه را متوقف کرده و باقی عملیات را انجام می دهد. مقدار پیش فرض این متغیر یک می باشد.
experimant_name	این متغیر مربوط به تنظیمات موتور متلب می باشد و به صورت عادی نیازی به تغییر مقدار پیش فرض آن نیست.
close_window	این پارامتر در صورتی قابل اجراست که کد پایتون و نرم افزار پری اسکن هردو بر روی یک کامپیوتر باشند و وظیفه آن این است که محیط گرافیکی را پس از اجرا شدن کد می بندد و مقدار پیش فرض آن صفر می باشد.

جدول ۳-۱: بررسی پارامتر های موجود در env\_dict

این کد که در حقیقت هسته اصلی<sup>۶</sup> تابع step می‌باشد. در بین محدوده مشخص شده، تاخیر صورت می‌گیرد. همان طور که مشخص است این تاخیر بین فرستادن حرکت و محاسبه پارامترهایی مانند امتیاز و مشاهده (حالت) می‌باشد.

**یادداشت ۱-۲-۳.** علت اصلی وجود تاخیر، مهلت دادن به عامل برای انجام حرکت است.

دو متغیر nget و delay هر دو وظیفه ایجاد تاخیر دارند که نحوه ایجاد این تاخیر با یک‌دیگر کاملاً متفاوت است. همچنین این امکان نیز وجود دارد به صورت ترکیبی نیز این تاخیر را ایجاد کرد. هر کدام از این روش‌ها مزایا و معایب خاص خود را دارند.

مزیت مهم استفاده از nget این است که به صورت مداوم در حال دریافت اطلاعات از محیط شبیه‌سازی است. بنابراین در صورت رخ دادن اتفاق خاصی مانند تصادف کردن و یا از مسیر خارج شدن می‌تواند آن را به موقع تشخیص دهد و تصمیمات لازم را انجام دهد. در صورتی که در زمانی که تاخیر ناشی از delay است، عملاً در آن مدت ارتباط با محیط شبیه سازی قطع شده است و ممکن است رخ دادن موارد گفته شده یا بسیار دیر متوجه شود و یا اصلاً متوجه نشود.

به‌طور مثال در صورتی که دو ماشین با یک‌دیگر تصادف داشته باشند؛ اگر این رخداد سریعاً تشخیص داده نشود، در این صورت احتمال دارد این دو ماشین از روی یک دیگر عبور کنند! و پس از عبور کردن این اطلاعات دریافت شود و برخوردی تشخیص داده نشود. از آن جا که برخورد بیشترین میزان تاثیر در امتیاز دارد؛ بنابراین، این اتفاق تاثیرات خیلی مخربی می‌تواند بر الگوریتم بگذارد.

عیب اصلی روش nget نیز این است که یک مقدار مشخص ندارد و به پارامترهایی از جمله سرعت شبکه نیز وابسته است. بنابراین اگر از یک شبکه به شبکه دیگر منتقل شود می‌تواند مقدار کاملاً متفاوتی به خود گیرد که شاید مطلوب نباشد. اما به راحتی با عوض کردن مقدار این متغیر در لایه الگوریتم می‌توان این مشکل را حل کرد. بنابراین توصیه می‌شود از این متغیر استفاده شود.

به کد **DQN** برگردیم. خط ۱۸ این کد مدل را می‌سازد. چیزی که اهمیت دارد این است که پارامتر  $\gamma$  چه مقداری انتخاب شود. در نسخه نهایی این مقدار روی  $0/8$  تنظیم شده است. در مورد این متغیر در بخش ۱ و در مراجع [۲] و [۱] بحث شده است. در ابتدا این متغیر مقدار پیش فرض  $0/99$  را داشت. پس از بررسی‌های انجام شده این مقدار به  $0/8$  کاهش یافت.

<sup>۶</sup> این کد از آن جهت که کاملاً با تابع اصلی برابر نمی‌باشد، واژه "هسته اصلی" برای آن در نظر گرفته شده است. تفاوت آن با کد اصلی برخی عملیات است که مرتبط با چاپ شدن اطلاعات در حال اجرا می‌باشد که به مقدار verbose مرتبط می‌شود.

نام تابع	توضیحات
<code>--init--</code>	این تابع علاوه بر تنظیم کردن برخی پارامترهای مرتبط به کلاس، اتصال کد پایتون به نرم افزار متلب را نیز برعهده دارد. همچنین تعیین فضای مشاهده و فضای حرکت نیز برعهده این بخش می باشد.
<code>step</code>	این تابع همان مرحله است که در بخش ۱ مطرح شد. محل اصلی اجرای این تابع در داخل یک حلقه متناهی می باشد. این تابع <code>action</code> را به صورت ورودی می گیرد و متغیرهایی مانند <code>observation</code> ، <code>reward</code> ، <code>done</code> و <code>info</code> را خروجی می دهد. در مورد نحوه محاسبه این خروجی ها صحبت خواهد شد.
<code>reset</code>	این تابع <code>env</code> ، را ریست می کند و به عنوان خروجی حالت اولیه را برمی گرداند. موارد استفاده از این تابع معمولاً در اول کد و در آخر هر اپیزود می باشد. آخر هر اپیزود هنگامی فرا می رسد که متغیر <code>done</code> که یکی از خروجی های تابع <code>step</code> است، یک شود.
<code>render</code>	این تابع به صورت معمول کارهای گرافیکی را برعهده دارد. اما از آنجایی که عمل در پس زمینه طرح وجود دارد، پس کار اصلی آن گرفتن داده ها و منظم کردن آن ها می باشد. برای این کار از یک تابع کمکی به نام <code>render_</code> استفاده می کند.

جدول ۳-۲: راهنمای توابع اصلی رابط برنامه نویسی برنامه

### ۳-۳ تعریف کردن پارامترهای یادگیری تقویتی

منظور از پارامترهای یادگیری تقویتی از متغیرهای حرکت و حالت و امتیاز و ... تا تعریف برخی توابع می باشد. ابتدا کلیات توابع را بررسی کنیم و سپس وارد جزئیات آن پارامترها می شویم.

توابع استفاده شده به دو دسته تقسیم می شوند. (آ) توابع اصلی (ب) توابع فرعی یا کمکی. توابع اصلی آن دسته از توابعی هستند که مختص به کتابخانه `gym` هستند و قرار دادن آن ها به شکل صحیح آن، اجباری است. توابع کمکی آن دسته از توابعی هستند که در این توابع نقش های مشخصی را ایفا کردند ولی استفاده کردن از آن ها اجباری نداشته است.

**یادداشت ۳-۳-۱.** در صورت لزوم کاربر می تواند توابع فرعی را تغییر دهد تا خروجی مطلوب خود را حاصل کند اما در لایه الگوریتم صرفاً از توابع اصلی استفاده می شود. زیرا هدف هم که استاندارد سازی کد می باشد با این موضوع سازگار است.

جدول ۳-۲، توابع اصلی را نشان می دهد و جدول ۳-۳ نیز توابع کمکی را نشان می دهد. همچنین لازم به ذکر است که **کد تست استاندارد** این پروژه در به صورت زیر است:

```

1 import gym, gym_prescan
2
3 env_dict = {
4     'id':      'prescan-without-matlabengine-v0',
5     'host':    '172.21.217.140',
6     'verbose': True,
7     'nget':    152
8 }
9 env = gym.make(**env_dict)
10 for i_episode in range(20):
11     observation = env.reset()
12     for t in range(100):
13         env.render()
14         print(observation)
15         action = env.action_space.sample()
16         observation, reward, done, info = env.step(action)
17         if done:
18             print("Episode finished after {} timesteps".format(t+1))
19             break
20 env.close()

```

یادداشت ۳-۳-۲. این کد صرفاً صحت عملکرد و نحوه استفاده از رابط برنامه‌نویسی برنامه<sup>۷</sup> را نشان می‌دهد و شامل هیچ گونه الگوریتمی نمی‌باشد.

### ۳-۳-۱ معرفی برخی توابع رابط برنامه‌نویسی برنامه

بررسی تابع `__init__`:

این تابع سه وظیفه مهم دارد.

(آ) بخشی از وظایف آن، وظایف مشخص آن در کد پایتون است.

(ب) ست کردن برخی پارامترهای مهم که در جدول ۳-۱ نیز آمده‌اند.

(ج) این تابع فضای مشاهده و فضای حرکت را مشخص می‌کند.

اطلاعات فضای مشاهده و فضای حرکت در جدول ۳-۴ آمده‌است.

<sup>۷</sup>API

نام تابع	محل استفاده	توضیحات
make	<code>--init--</code>	این تابع لایه ای را ایجاد می‌کند که هرگونه ارتباط با محیط شبیه‌سازی به آن لایه مرتبط می‌شود.
render_	<code>render</code> و <code>reset</code> و <code>step</code>	این تابع وظیفه دریافت اطلاعات از محیط شبیه‌سازی و استخراج داده‌های مفید از آن است.
send	<code>step</code>	این تابع برای فرستادن حرکت به محیط شبیه‌سازی استفاده می‌شود.
calc_reward	<code>step</code>	این تابع امتیاز را محاسبه می‌کند.
_next_observation	<code>step</code> و <code>reset</code>	این تابع حالت بعدی عامل را محاسبه می‌کند.
action_translate	<code>send</code>	این تابع مانند یک دسته‌بازی در تابع <code>send</code> حرکت را تفسیر می‌کند و دستورات کنترلی قابل اجرا برای محیط شبیه‌سازی ایجاد می‌کند.

جدول ۳-۳: راهنمای توابع کمکی رابط برنامه‌نویسی برنامه

نام متغیر	مفهوم	جنس متغیر	توضیحات
action_space	فضای حرکت	<code>spaces.Discrete(6)</code>	عدد صحیح ۶
observation_space	فضای مشاهده	<code>spaces.Box(shape=(1,38), dtype=np.float16)</code>	ماتریس $1 \times 38$ تایی

جدول ۴-۳: تعریف فضای مشاهده و فضای حرکت در پروژه

### بررسی تابع `action_translation`:

این کد دقیقاً پیاده‌سازی یک دسته‌بازی<sup>۸</sup> می‌باشد. شکل ۴-۳ اطلاعات کامل این موضوع به همراه تفسیر آن‌ها دارد.

از آنجایی که در این پروژه فضای حرکت طبق جدول ۴-۳ مقدار عدد صحیح ۶ را دارد و این به آن معناست که ۶ حالت گسسته بین صفر تا ۵ برای حرکت وجود دارد همچنین نشان می‌دهد که جنس `action` عدد صحیح `int` است. بنابراین، می‌بایست که آن را تفسیر کرد. وظیف اصلی این تابع نیز تفسیر مقدار مختلفی است که `action` می‌تواند بگیرد، می‌باشد.

<sup>۸</sup>Joystick



```

1 def action_translate(self,action):
2     lanewidth =
3         self.enviroment.road.laneWidth
4     self.__action_old__ = self.__action__
5     vel = self.agent['data']['Velocity']
6     offset = self.__action__[0]
7
8     if action == 0 :
9         offset = -lanewidth
10
11    if action == 1 :
12        offset = 0
13
14    if action == 2 :
15        offset = lanewidth
16
17    if action == 3 :
18        vel = action_velocity(vel,True)
19
20    if action == 4 :
21        vel = action_velocity(vel,False)
22
23    self.__action__ = [offset,vel]
24
25    return self.__action__

```



شماره	وظیفه
۰	رفتن به لاین -w
۱	رفتن به لاین 0
۲	رفتن به لاین +w
۳	زیاد کردن سرعت
۴	کم کردن سرعت
۵	بدون تغییر

شکل ۳-۴: بررسی تابع `action_translate`

**یادداشت ۳-۳-۳.** دستور کنترلی اصلی یک بردار دوتایی است (خط ۱۸ کد شکل ۳-۴) که مقدار اولی آن لاین را نشان می‌دهد و مقدار دوم آن سرعتی می‌باشد که انتظار داریم که عامل، سرعت خود را به آن برساند.

**یادداشت ۳-۳-۴.** اگر دقت کنید در کد مذکور دو مقدار کنونی و قدیمی تر action نگهداری شده است.

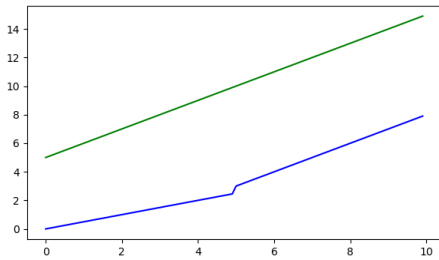
**یادداشت ۳-۳-۵.** همچنین دقت شود که در این کد، مقدار سرعت، همان مقدار حقیقی سرعت است که از محیط شبیه‌سازی می‌آید. و مقداری که در بردار \_\_action\_\_ قرار می‌گیرد مقدار کنترلی سرعت است که با یک‌دیگر تفاوت دارند.

نکته دیگری که حایز اهمیت است این است که هنگامی که گزینه ۳ و یا ۴ انتخاب می‌شوند، سیاستی برای افزایش و کاهش سرعت اتخاذ شده است. این سیاست در قالب یک تابع در تصاویر شکل ۳-۵ ظاهر شده است. همانطور که کد ۳-۵(آ) و نمودار متناظر آن در شکل ۳-۵(ب) نشان می‌دهد، سیاست‌های مختلفی برای زیاد کردن و کم کردن سرعت اتخاذ شده است.

برای زیاد کردن سرعت، سرعت واقعی که از محیط شبیه‌سازی دریافت شده است با ۵ واحد جمع می‌شود. بنابراین انتظار داریم سرعت پس از سه بار افزایش به ۱۵ برسد (شکل ۳-۵(ج)، نمودار قرمز) اما این اتفاق نمی‌افتد. زیرا این افزایش سرعت، کار زمان‌بری است و نیاز به حوصله دارد که اگر حوصله و تحمل و به عبارت دیگر تاخیر را از یه حدی بالاتر ببریم عملاً در کنترل عامل به مشکل خواهیم رسید. همچنین مورد مشابه آن چه که گفته شد، در شکل ۳-۵(د) نیز برقرار است. نقاطی که رنگشان قرمز است نمودار ایدآلی مفروض خواهند بود که معادل تاخیر کم سیستم جهت اعمال سرعت نهایی است. و نمودار دیگر معادل رخدادی است که ۳۰٪ به آن عمل شده است و ۷۰٪ تحت تاثیر مقدار قبلی خواهد بود و به این صورت یک میانگین وزن‌دار گرفته شده است.

مشاهده می‌شود که مسیری که به واقعیت نزدیک‌تر است آرام‌تر از مسیر مدنظر است. همچنین تفاوت تغییر روند کاهشی سرعت‌های کمتر از ۵ واحد، این است که هیچ‌گاه منفی نشود ولی به صورت نمایی کاهش یابد و نزدیک صفر شود.

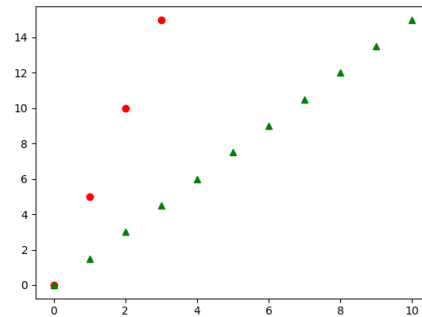
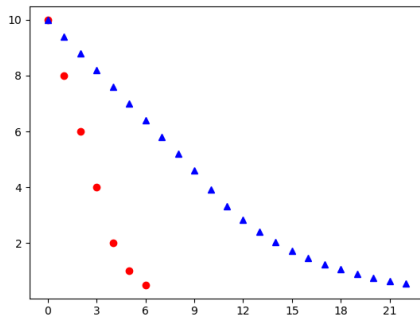
تفاوت دیگر آن است که به هنگام افزایش مقدار ۵ واحد به سرعت افزوده می‌شود و در هنگام کاهش مقدار دو واحد (برای سرعت‌های بالای ۵ واحد) از آن کسر می‌شود. این تفاوت در مقایسه فاصله نقاط بین دو نمودار شکل‌های ۳-۵(ج) و ۳-۵(د) نیز ظاهر شده است. علت اصلی این تفاوت در بررسی



(ب) شکل کلی نمودار افزایش سرعت

```
1 def action_velocity(vel, increase):
2     if increase:
3         return vel+5
4     else:
5         if vel < 5:
6             return vel/2
7         else:
8             return vel - 2
```

(آ) کد اصلی روند افزایشی و یا کاهشی سرعت



(ج) حرکت در جهت افزایش سرعت با شروع از صفر (د) حرکت در جهت کاهش سرعت با شروع از ۱۰

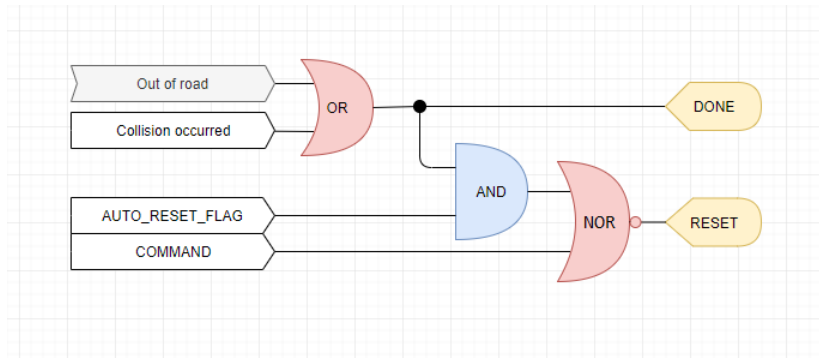
شکل ۳-۵: بررسی دقیق تابع افزایش دهنده و کاهنده سرعت ماشین

calc\_reward خود را نشان می‌دهد. اما پیشتر در نظر داشته باشید که یک سیاست جهت‌دار و تشویقی جهت قرار گرفتن سریع‌تر در مسیر درست می‌باشد.

### بررسی تابع step :

کد این تابع را پیش‌تر بررسی شد. جایگاه این تابع داخل **کد تست استاندارد** در داخل یک حلقه است هنگامی که حلقه تمام شود به اصلاح یک اپیزود تمام شده‌است. دلیل تمام شدن حلقه یا رسیدن به سقف تعداد مرحله در هر اپیزود است و یا یک شدن مقدار done. این مقدار یکی از خروجی‌هایی است که این متغیر می‌تواند اختیار کند.

این تابع مقدار حرکت را می‌گیرد. سپس آن را برای محیط شبیه‌سازی می‌فرستد. در کد زیر این کار توسط تابع send انجام می‌پذیرد. در تابع send ابتدا با استفاده از تابع action\_translate به صورت دستور کنترلی در خواهد آمد و پس از این تبدیل برای محیط شبیه‌سازی ارسال می‌شود. پس از مقداری تأخیر، متغیرهای observation و reward که به ترتیب مشاهده و امتیاز می‌باشند، محاسبه می‌شود. در کنار این دو خروجی، خروجی‌های دیگری وجود دارند. done که از محیط شبیه



شکل ۳-۶: منطق محاسبه done در محیط شبیه‌سازی

سازی می‌آید و نشان می‌دهد که اپیزود تمام شده‌است. متغیر info متغیری است که اطلاعات اضافه‌ای را خروجی می‌دهد که در دیباگ کردن مفید خواهد بود. این اطلاعات اضافی در این پروژه، اطلاعات برخورد و اطلاعات موقعیت عامل انتخاب شده‌اند.

**یادداشت ۳-۳-۶.** نحوه محاسبه done وظیفه محیط شبیه‌سازی می‌باشد. منطق محاسبه آن در شکل ۳-۶ آمده است. بنابراین اطلاعات دو حالت وجود دارد که موجب تمام شدن یک اپیزود و به‌طور معادل یک شدن done می‌شود:

(آ) خارج شدن از محدوده جاده<sup>۹</sup>

(ب) تصادف کردن با ماشین دیگر

**یادداشت ۳-۳-۷.** مدار منطقی شکل ۳-۶ علاوه بر محاسبه متغیر done، در مورد نحوه ریست شدن محیط نیز تصمیم‌گیری می‌کند که مرتبط به بخش فنی است و در این لایه بررسی نمی‌شود.

کد این تابع در زیر آمده است. دو تابع `_next_observation` و `calc_reward` به ترتیب مشاهده و امتیاز را محاسبه می‌کنند که به صورت جداگانه بررسی خواهند شد.

```
1 def step(self, action):
2     self.send(action)
3     # ----- BEGIN DELAY -----
4     if self.delay > 0 :
5         sleep(self.delay)
6     for _ in range(self.nget):
7         self.render_()
8         if self.done:
```

<sup>۹</sup> منظور از محدوده جاده، محدوده‌ای است که در شکل ۳-۳(ب) مشخص شده است.

```

9         break
10    # ----- END DELAY -----
11    observation = self._next_observation()
12    reward = self.calc_reward()
13    done = self.done
14    info = {'Collision':self.collission
           , 'Position':self.agent['data']['Position']}

```

### ۲-۳-۳ بررسی تابع `_next_observation`:

این تابع مشخص می‌کند که چه چیزی به عنوان مشاهده اعلام شود. همان‌طور که در جدول ۴-۳ آمده است، خروجی نهایی این تابع دارای ابعاد  $۱ \times ۳۸$  می‌باشد. شکل ۷-۳ این روند را به‌طور کامل نشان می‌دهد.

بنابراین مشاهده یک بردار ۳۸ تایی است که ۳۶ داده اول آن بر اساس زاویه و اندازه محاسبه شده است و ۲ داده دیگر آن مقدار  $y$  و مقدار سرعت ( $v$ ) می‌باشد.

بر روی ماشین یک سنسور وجود دارد که دو بردار  $۱۰^\circ$  تایی شامل فاصله و زاویه از  $۱۰^\circ$  ماشین کنار خود را می‌دهد. به این ترتیب برای تعریف مشاهده فضای اطراف ماشین به ۳۶ قسمت افراز شد به طوری که هر افراز آن یک محدوده به زاویه  $۱۰^\circ$  درجه را شامل می‌شود. شکل ۸-۳ طرز تعریف و مقدار دهی این بردار ۳۶ تایی را نشان می‌دهد. اگر در آن افراز ماشینی قرار نگرفت مقدار آن صفر است و در غیر این صورت فاصله آن با نزدیک‌ترین ماشین خواهد بود.

**یادداشت ۸-۳-۳.** از آنجایی که مقدار زاویه در بازه  $\theta \in [-۱۸۰^\circ, ۱۸۰^\circ]$  قرار دارد، برای از بین بردن بازه منفی کل زوایا ابتدا با  $۱۸۰^\circ$  جمع شد و سپس افراز صورت گرفت. روش دیگر می‌توانست با استفاده از هم‌نهشتی به پیمانه  $۳۶۰^\circ$  باشد که به هنگام آزمایش روش اول کمک می‌کرد که بهتر یاد بگیرد.

### بررسی تابع `calc_reward`:

این تابع امتیازهای رابط برنامه‌نویسی برنامه را تنظیم می‌کند. بنابراین مهم‌ترین بخش‌های آن محسوب خواهند شد. کد این تابع در زیر آمده است.

```

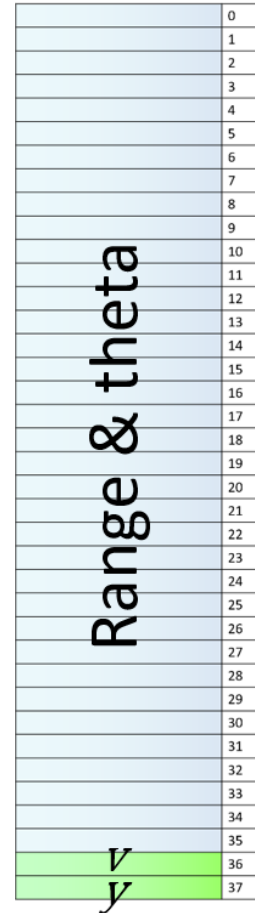
1 def calc_reward(self):
2     lanewidth = self.enviroment.road.laneWidth
3     vel_sim = self.agent['data']['Velocity']

```

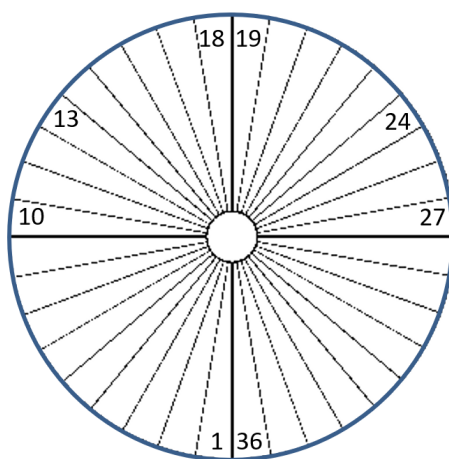
```

1 def _next_observation(self):
2     self.render_()
3     obs = np.zeros((1,36),dtype=np.float)
4     car = self.agent
5     theta = car['Sensors'][0]['data']['theta']
6     Range = car['Sensors'][0]['data']['Range']
7
8     for i in range(len(theta)):
9         t = int((theta[i] + 180 )/10)
10        r = Range[i]
11        if obs[0,t] != 0:
12            obs[0,t] > r
13            continue
14        obs[0,t] = r
15        extra = [car['data']["Velocity"],
16                car['data']["Position"]["y"]]
17        self.__obs__ = np.append(obs, extra)
18        return self.__obs__

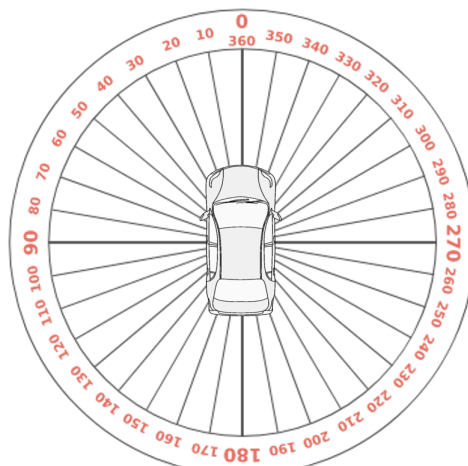
```



شکل ۳-۷: نحوه تعریف مشاهده



(ب)



(ا)

شکل ۳-۸: افراز محیط اطراف ماشین برحسب زاویه و تعریف نحوه قرار دادن آن در ماتریس مشاهده

```

4     vel_cmd = self.__action__[1]
5     Vel = 0.8 * vel_sim + 0.2 * vel_cmd
6
7     Longitudinal_reward = reward_velocity(Vel,28) *1.5
8     Collision_reward = -25 if self.collision['Occurred'] else 0
9     Violation_reward = -0.75 * (np.abs(self.__action__[0] -
        self.__action_old__[0])/lanewidth)
10    Nearby_reward = nearby_reward_linear(self.__obs__[12],-2.5,1.5) +\
11                    nearby_reward_linear(self.__obs__[23],-2.5,1.5)
12    reward_T = Longitudinal_reward + Collision_reward +\
13                Violation_reward + Nearby_reward
14    return reward_T

```

همان طور که کد نیز نشان می‌دهد، امتیاز نهایی به صورت مجموع ۴ امتیاز دیگر می‌باشد. هر یک از این امتیازها مربوط به یک بخش خاص است. بنابراین،

$$\mathcal{R}_T = \mathcal{R}_{\text{Collision}} + \mathcal{R}_{\text{Violation}} + \mathcal{R}_{\text{Longitudinal}} + \mathcal{R}_{\text{Nearby}}$$

$\mathcal{R}_{\text{Collision}}$  در صورتی که تصادف رخ دهد، مقدار آن ۲۵- و در این صورت صفر خواهد بود. بنابراین هنگام تصادف علاوه بر تمام شدن اپیزود یک امتیاز منفی با اندازه زیاد دریافت می‌کند.

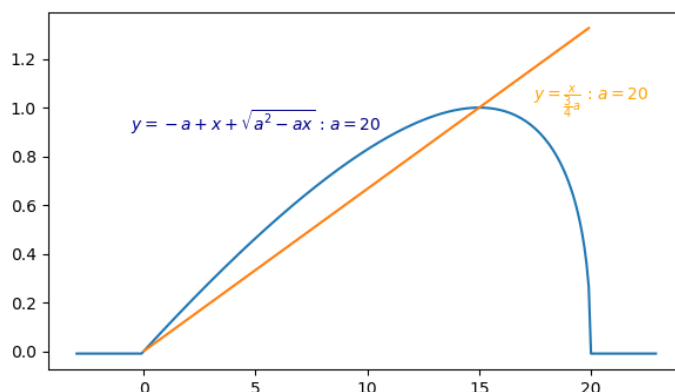
$\mathcal{R}_{\text{Violation}}$  تعریف شد تا عامل متوجه شود که تغییر لاین هزینه خواهد داشت. مقدار این هزینه ۰/۷۵- برابر مقدار تغییر لاین است. یعنی اگر یک واحد لاین خود را تغییر دهد، مقدار آن ۰/۷۵- خواهد شد و اگر دو واحد لاین عوض کند، مقدار آن ۱/۵- خواهد شد.

**یادداشت ۳-۳-۹.** اندازه مقدار  $\mathcal{R}_{\text{Violation}}$  به ازای یک واحد تغییر لاین، نصف مقدار بیشینه  $\mathcal{R}_{\text{Longitudinal}}$  انتخاب شده است.

**یادداشت ۳-۳-۱۰.** تنها امتیاز مثبت فقط  $\mathcal{R}_{\text{Longitudinal}}$  با مقدار بیشینه ۱/۵ می‌باشد.

برای محاسبه  $\mathcal{R}_{\text{Longitudinal}}$  از یک تابع به نام `reward_velocity` با ویژگی های مشخص استفاده شده است. این تابع در ورودی اول خود، ورودی تابع و در ورودی دوم خود پارامتر تابع را دریافت می‌کند که این پارامتر مقدار ثابتی دارد.

تابع مذکور، از لحاظ مقدار بیشینه بسیار خوش تعریف است و دارای ضابطه زیر است.



شکل ۳-۹: نمودار تابع محاسبه امتیاز سرعت نرمال شده

$$F^{\mathcal{R}_v}|_a = \begin{cases} -a + x + \sqrt{a^2 - ax} & : x \in [0, a] \\ -0.01 & : \text{سایر نقاط} \end{cases}$$

این بیشینه مقدار خود را در نقطه  $0.75a$  با مقدار بیشینه  $0.25a$  اختیار می‌کند. اگر این تابع را بر مقدار بیشینه اش تقسیم کنیم، نرمال خواهد شد. بنابراین:

$$F_n^{\mathcal{R}_v}|_a = \frac{F^{\mathcal{R}_v}|_a}{\max\{F^{\mathcal{R}_v}|_a\}} = \begin{cases} \frac{1}{a}(-a + x + \sqrt{a^2 - ax}) & : x \in [0, a] \\ -0.01 & : \text{سایر نقاط} \end{cases}$$

تابع reward\_velocity به عنوان ورودی سوم تصمیم می‌گیرد که خروجی نرمال شده باشد یا نه. اگر normal=True باشد خروجی این تابع برابر خروجی تابع  $F_n^{\mathcal{R}_v}|_a$  همین تابع خواهد بود و اگر این مقدار False باشد، خروجی برابر خروجی تابع  $F^{\mathcal{R}_v}|_a$  خواهد بود. به صورت پیش‌فرض این تابع خروجی نرمال شده خواهد داشت.

نمودار این تابع در ۳-۹ آمده است که با تابع خطی که از مقدار بیشینه آن عبور می‌کند مقایسه شده است.

آ) تا قبل از مقدار بیشینه مقدار این دو تابع نزدیک یک دیگر می‌باشد و شیب آن‌ها نیز به هم نزدیک است.

ب) این تابع پس از مقدار رسیدن به مقدار بیشینه خود، به صورت نزولی افت می‌کند.



ج) اگر از نقطه‌ای که مقدار بیشینه در آن رخ می‌دهد، به سمت مثبت حرکت کنیم سریع تر از هنگامی که به سمت منفی حرکت می‌کند.

د) دیگر ویژگی آن این است که در حدود مقدار بیشینه خود، تقریباً هموار است که این هموار بودن آن موجب آن خواهد شد که عامل بتواند در حوالی سرعت بیشینه خود بدون تفاوت زیادی در مقدار امتیاز تصمیم بگیرد که سرعت را در صورت لزوم کم و یا زیاد کند.

**نکته ۳-۳-۱۱** (خیلی مهم). چیزی که به عنوان متغیر مستقل به تابع ارسال می‌شود سرعت است اما با سرعت واقعی متفاوت است. در حقیقت میانگین وزن‌داری از سرعت واقعی و سرعت دستوری می‌باشد. منظور از سرعت دستوری همان سرعتی است که با استفاده از دستورات کنترلی برای محیط شبیه‌سازی فرستاده می‌شود.

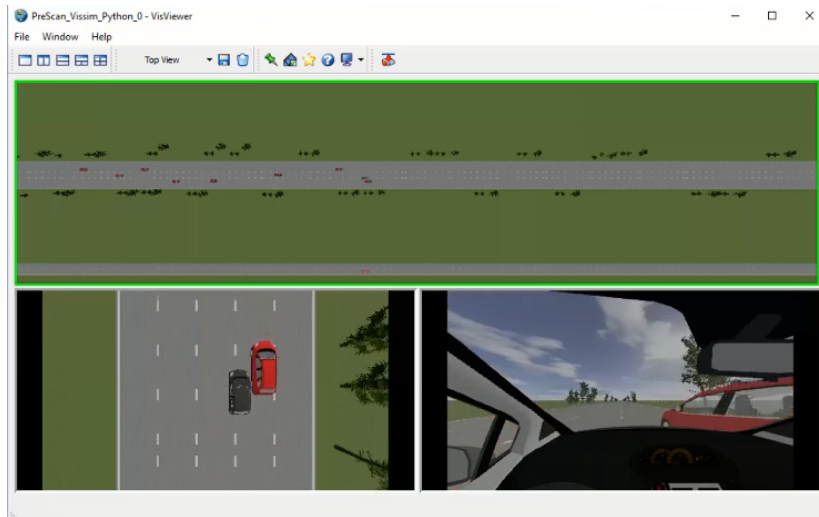
$$V = \alpha V_{sim} + \beta V_{cmd}$$

علت این تفاوت نیز به سیاست‌های تشویقی مربوط می‌شود. فرض کنید عامل تصمیم می‌گیرد سرعت را افزایش دهد. با این تصمیم مقدار سرعت ۵ واحد افزایش می‌یابد. ولی سرعت بعد از گذشت از یک مرحله به مقدار کمی سرعت آن افزایش یافته است از این رو امتیاز کمی می‌گیرد. با این میانگین‌گیری در حقیقت برای تصمیم و نیت خوب عامل مقداری امتیاز در نظر گرفته می‌شود تا مقدار امتیاز بیشتری را بگیرد.

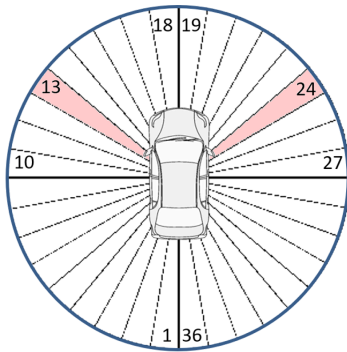
در صورتی که عامل تصمیم بگیرد که سرعت خود را کم کند، بابت این تصمیم که نامطلوب است جریمه می‌شود. مقدار این جریمه کمتر از مقدار تشویق است. این موضوع در نزدیکی‌های سرعت بیشینه کمک‌کننده خواهد بود. زیرا بالا بردن سرعت در آن حوالی مطلوب نیست بنابراین در صورتی که نیت کند که سرعت زیاد شود، این تصمیم عامل همراه با افت امتیاز بیشتری (نسبت به حالتی که نیت در نظر گرفته نشده است) خواهد بود. بنابراین این تفاوت‌ها مطلوب و سازنده خواهد بود.

آخرین امتیاز،  $R_{Nearby}$  می‌باشد. بخاطر مشکلات شبیه‌سازی در اثر تغییر لاین، این تغییر به تمامی اعمال نمی‌شود و ممکن است عامل در فاصله بسیار نزدیک از یک ماشین دیگر رانندگی کند به گونه‌ای که تصادف رخ ندهد. (شکل ۳-۱۰ را مشاهده کنید). هدف از ایجاد این امتیاز نیز ایجاد یک حس خطر است. مسلماً باید علامت آن منفی باشد.

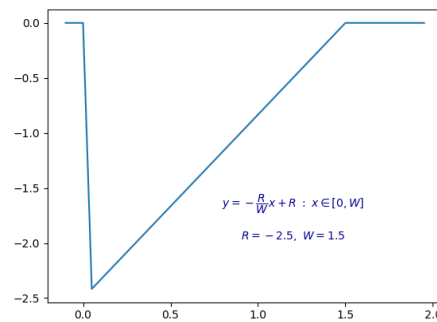
برای این منظور به نوعی دو سنسور مجازی در کناره‌های عامل کار گذاشته شد. در صورتی که ماشین دیگر در فاصله ۱/۵ متری از عامل در جهت‌های داده شده در شکل ۳-۱۱ (ب) قرار داشته‌باشد،



شکل ۳-۱۰:



(ب) محل قرار گیری سنسورهای مجازی بر روی عامل



(آ) تابع امتیاز در زمان نزدیکی

شکل ۳-۱۱: تابع امتیاز نزدیکی و محل قرار گیری این تابع در فضای مشاهده

مقدار آن صفر خواهد بود و در صورتی که فاصله از این مقدار کمتر شود به صورت خطی تا  $2/5$  - امتیاز منفی دریافت می کند.

نمودار تابع در شکل ۳-۱۱ (ب) آمده است و ضابطه آن به صورت زیر می باشد:

$$F^{\mathcal{R}_{\text{Nearby}}}|_{R,W} = \begin{cases} -\frac{R}{W}x + R & : x \in [0, W] \\ 0 & : \text{سایر نقاط} \end{cases}$$

که در رابطه فوق، مقدار  $W$  طول سنسور مجازی و اندازه  $R$  بیشترین امتیاز (اما در جهت منفی) که در این نمودار اتفاق می افتد.

۳-۳-۳ بررسی مقدار  $\gamma$ 

مقدار  $\gamma$  در لایه الگوریتم مطرح می‌شود. در الگوریتم DQN این پارامتر، مقدار دید روبه جلو را نشان می‌دهد. این پارامتر در ابتدا مقدار پیشفرض  $0.99$  داشت و این یعنی تقریباً عامل کل مسیر آینده را نگاه می‌کرد تا ارزش حرکت خود را نشان دهد.

با توجه به تعریف نحوه زیاد و کم شدن سرعت و همچنین تعریف سرعت ورودی تابع محاسبه امتیاز سرعت، در صورتی که عامل، سرعت خود را زیاد کند و بلافاصله سرعت خود را کم کند و دوباره همین سیاست را پیش بگیرد اتفاق بدی می‌افتد.

این اتفاق موجب آن خواهد شد که امتیاز اضافی و غیر واقعی دریافت کند. این موضوع در الگوریتم های یادگیری تقویتی زیاد مطرح نیست چون این الگوریتم ها به سمت بیشینه مقدار امتیاز در حرکت هستند. بنابراین گرچه امتیاز اضافی دریافت می‌کند اما باید یاد بگیرد که سرعت را کم نکند و به صورت مداوم آن را تا حد بیشینه افزایش دهد؛ اما این اتفاق نیفتاد.

اتفاقی که در واقعیت افتاد این بود که ماشین سرعت خود را به صورت نوسانی کم و زیاد می‌کند به طوری که روی سرعت  $2$  واحد تقریباً سرعت خود را نگه می‌دارد.

با کمی تفکر این نتیجه بدیهی می‌شود. چرا که در این صورت بجای حدوداً  $70$  مرحله رفتن در هر اپیزود، آن اپیزود را در حدود  $250$  مرحله می‌گذراند (به خاطر سرعت بسیار پایین آن). و چون هم امتیاز اضافی دریافت می‌کند و هم بخاطر طولانی شدن مسیر، در مجموع امتیاز زیادی دریافت می‌کند. بنابراین سیاست بهینه خود را این گونه پیدا کرده بود.

برای جلوگیری از این پدیده سعی شد تا با کاهش مقدار  $\gamma$  به ازای مقادیر مختلف، دید روبه جلو را کاهش دهد و به دو و یا سه حرکت آینده محدود کند. مقادیر مختلف مورد آزمایش و شبیه سازی قرار گرفتند و در  $0.8$  مقدار آن بهینه شد.

بنابراین این پارامتر اهمیت فراوانی در جلوگیری از قرار گرفتن عامل در مسیر اشتباه دارد و با آن انتخاب از این گونه اشتباهات جلوگیری شده است.

## فصل چهارم

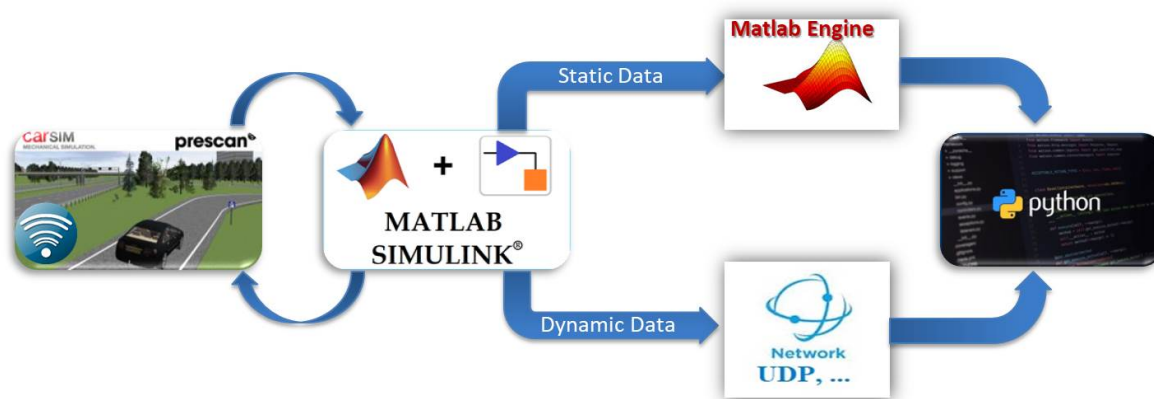
### بررسی جزئیات فنی پروژه

این بخش لایه هایی را مطرح می کند که جهت پیاده سازی پروژه از آن استفاده شده است. لایه ای که به الگوریتم مرتبط می شود در بخش ۳ به صورت کامل بررسی خواهد شد و در این قسمت نحوه کارکردن محیط شبیه سازی مطرح خواهد شد.

## ۱-۴ مقدمه

## ۲-۴ دورنمای کلی طرح

همان طور که گفته شد، در این پروژه از ابزار های مختلفی استفاده شده است. برخی ابزارات دیگر نیز جهت ایجاد اتصال بین آن ابزار ها استفاده شده اند. در این بخش، این اجزا به تفصیل بررسی خواهد شد. هر کدام از این اجزا کار مشخصی را بر عهده دارند. شکل ۱-۴ این ارتباط را نشان می دهد.



شکل ۱-۴: بلوک دیالگرام لایه های کلی

در شکل ۱-۴ از سمت چپ به راست اجزا یاد شده و نحوه ارتباط آن ها بایکدیگر را به خوبی نشان می دهد. این بلاک ها و ارتباط ها عبارتند از:

- اولین بلاک آن، نرم افزار **پری اسکن** می باشد. وظیفه اصلی این نرم افزار، شبیه سازی دینامیک یک اتومبیل و یا موتور و ... می باشد. همچنین ایجاد یک محیط گرافیکی زیبا و یک پنل کاربری گرافیکی برای ساخت ماشین ها از دیگر حسن های این نرم افزار است. فایل های مهم ایجاد شده توسط این بخش، `pex` و `pb` می باشد.
- بلاک بعدی ترکیبی از **متلب و سیمولینک** است. چرا که نرم افزار پری اسکن این امکان را دارد که برای کنترل و دسترسی بیشتر به قسمت های کنترلی مختلف، چیزی به نام API ارائه می دهد.

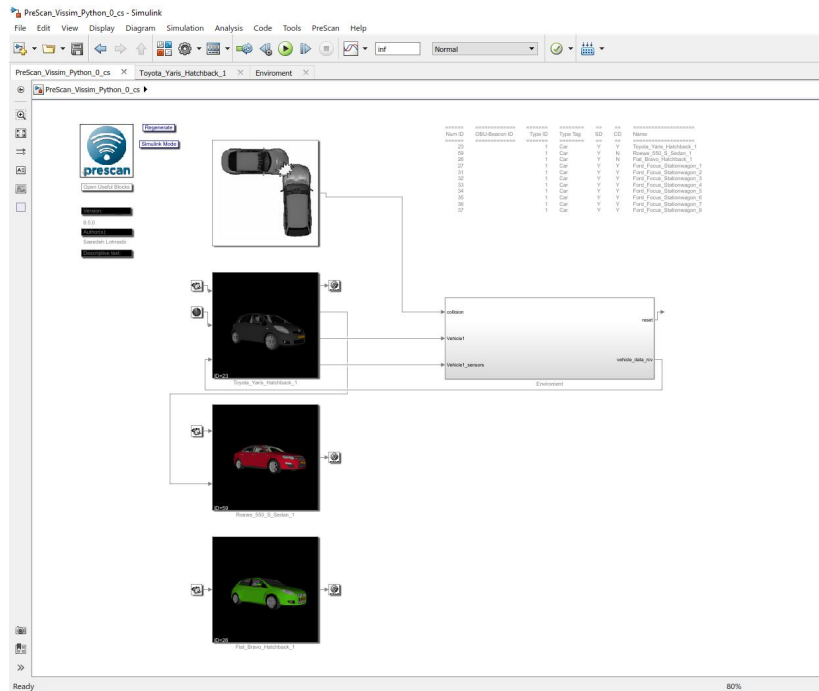
این API یک فایل سیمولینک را در اختیار کاربران قرار میدهد که در آن بلوک های مشخصی به یکدیگر متصل هستند و با مطالعه و تغییر آن بلوک ها می توان کنترل سیستم را به دست گرفت. فایل های مهم این بخش نیز در فرمت .slx و .m در دسترس هستند. همچنین API یاد شده، دستورات دیگری را جهت دریافت داده های استاتیک محیط ساخته شده در این نرم افزار را به کاربران خویش در محیط متلب می دهد.

- دو بلوک بعدی، مربوط به اتصال بین متلب و یا سیمولینک با پایتون هستند. بلوک بالایی این اتصال را بین داده های استاتیک شامل طول جاده و عرض هر لاین، موقعیت اولیه اتومبیل و جاده، و بسیاری اطلاعات دیگر که بسیاری از آن اطلاعات استفاده نشده اند زیرا در این پروژه مفید نبوده اند. این بلوک، فایل سیمولینک را تغییر نمی دهد. بلوک پایینی نیز با استفاده از روش های شبکه کردن، می تواند داده های پویا را از محیط سیمولینک به پایتون منتقل کند. این داده های پویا عبارتند از موقعیت و سرعت و اطلاعات دیگری از اتومبیل در حال حرکت، اطلاعات سنسورها و ... باشد.
- بلوک بعدی پایتون است که خود شامل لایه های دیگری است که در شکل ۴-۹ به تفصیل بیان شده است. نکته جالب در آن این است که در آن لایه ها اثری نیز از دو بلوک پیشین آمده است. همچنین بخش اصلی کار، یا به عبارتی مغز و هوش این کار در این قسمت توسعه یافته است.

## ۴-۳ بررسی دقیق تر فایل سیمولینک

فایل سیمولینک ایجاد شده توسط نرم افزار پری اسکن، قابلیت تغییر به دست کاربر را دارد. شکل ۴-۲ فایل تغییر یافته مربوط به این پروژه را نشان می دهد. بلوک های سمت راست نشان داده شده در سمت راست شکل ۴-۲ توسط نرم افزار پری اسکن ایجاد شده است که البته دست خوش تغییراتی نیز بوده اند. در صورتی که با استفاده از محیط گرافیکی GUI فایل \*.pex تغییر کند، فایل سیمولینک تغییر نمی کند. در برخی موارد این تغییرات ممکن است منجر به پیغام خطا شود.

نکته ۴-۳-۱. در صورتی که فایل \*.pex تغییر کند، برای اعمال این تغییرات، باید روی کلمه Regenerate که در شکل ۴-۳ آمده است، کلیک کرد. با این کار، تغییرات جدید اعمال می شود بی آن که



شکل ۴-۲: فایل سیمولینک ایجاد شده توسط نرم افزار پری اسکن همراه با تغییرات

تغییرات کاربر تحت تاثیر قرار بگیرد.

در شکل ۴-۳ همانطور که در نکته ۴-۳-۱ به آن اشاره شد، دکمه ای تحت عنوان Regenerate وجود دارد که استفاده از آن در همان نکته مشخص شده است. همچنین در این تصویر در زیر لوگوی برنامه پری اسکن، اطلاعاتی مانند شماره نسخه نرم افزار (که در این جا 8.5.0 می باشد)، نام نویسنده مشاهده می شود.

در شکل ۴-۲ اولین بلوک سمت راست همان ماشینی است که ما آنرا تحت کنترل گرفته ایم. اگر به آن وارد شویم، شکل ۴-۴ را مشاهده می کنیم. در این تصویر ورودی و خروجی ها نقش خیلی مهمی دارند. این اطلاعات در جدول ۴-۱ آمده اند.

### ۴-۳-۱ معرفی بلوک Environment و بررسی جزئیات آن

در جدول ۴-۱ صرفا ورودی خروجی های مهم مورد بررسی قرار گرفته است. این ورودی ها و خروجی های مهم به یک بلوک دیگر منتقل می شود. بلوک Environment همان بلوک است که در شکل ۴-۲ نیز مشخص است و در شکل ۴-۵ نیز از زاویه نزدیک تر با جزئیات بیشتر می توان آنرا مشاهده نمود.

<sup>14</sup>json

<sup>۲</sup>بعدا خواهید دید که مشخص بودن طول بسیار بسیار اهمیت دارد!



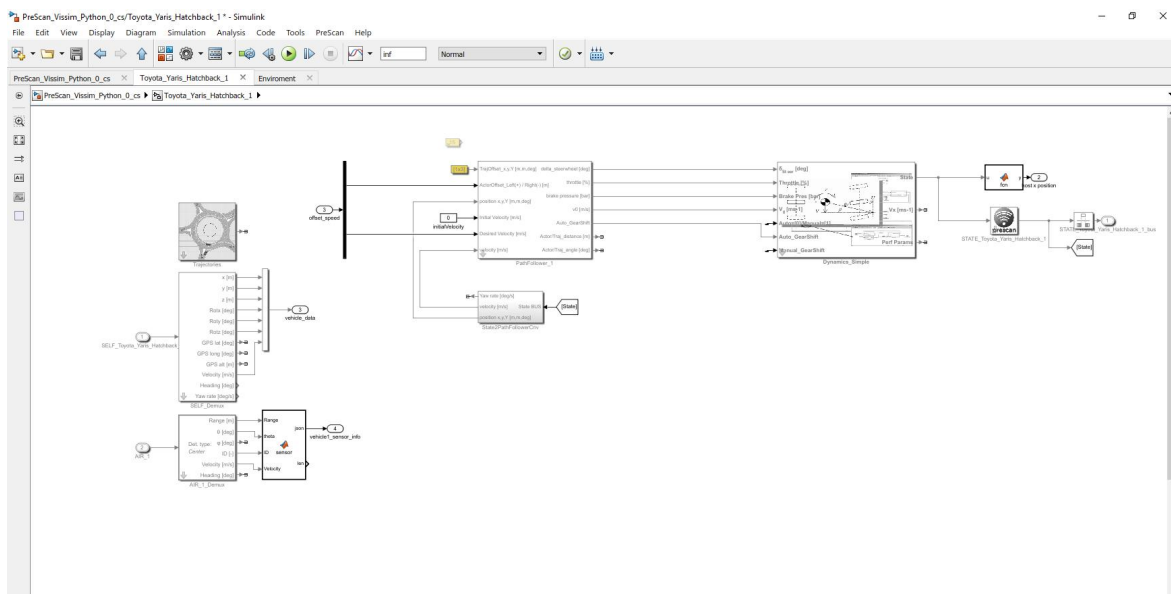
## Simulink Mode

Version:

Author(s):

Descriptive text:

شکل ۴-۳: روش صحیح اعمال تغییرات روی فایل سیمولینک



شکل ۴-۴: فایل سیمولینک - شبیه سازی اتومبیل

وظیفه اصلی بلوک Environment جمع آوری اطلاعات محیط سیمولینک و همچنین ارسال دستورات کنترلی به آن هاست. اطلاعات جمع آوری شده از محیط سیمولینک شامل موارد زیر است.

- اطلاعات ماشینی که نقش «عامل» را در الگوریتم دارد.
- اطلاعات سنسور های ماشین «عامل»
- اطلاعات مربوط به تصادف کردن و کدوم ماشین با کدوم ماشین تصادف کرده است.



نوع	عنوان	بلوک مربوط	توضیحات
خروجی	اطلاعات ماشین	SELF_Demux	اطلاعات ماشین، شامل اطلاعات موقعیت ( $y, x$ ) و $z$ و اطلاعات چرخش (حول $y, x$ و $z$ ) به همراه سرعت ماشین را خروجی می‌دهد. این ۷ داده قبل از خروجی توسط یک mux با یکدیگر ادغام می‌شوند.
خروجی	اطلاعات سنسور ماشین	AIR_1_Demux	اطلاعات سنسور V2C را خروجی می‌دهد. از آنجا که این سنسور فاصله و زاویه و ... تا ده ماشین نزدیک خود را می‌دهد. بنابراین هریک از این اطلاعات یک بردار ده تایی است. برای فرستادن آن اطلاعات به خروجی، ابتدا آن‌ها را به طریقی به فرمت جیسون <sup>۱</sup> تبدیل می‌کند و یک رشته کاراکتر با طول مشخص <sup>۲</sup> را خروجی می‌دهد.
ورودی	کنترل لاین و سرعت ماشین	PathFollower_1	دستورات کنترلی اتومبیل، شامل کدام خط بودن و مقدار سرعت نهایی، به صورت ورودی وارد یک demux می‌شود و پس از جدا سازی، به بلوک مربوط متصل می‌شود.

جدول ۴-۱: بررسی ورودی ها و خروجی های مهم در شکل ۴-۴

این اطلاعات به صورت ورودی به این بلوک وارد می‌شود و دستورات کنترلی شامل کنترل لاین ماشین به همراه مقدار سرعت نهایی ماشینی که نقش «عامل» در الگوریتم دارد، از این بلوک خارج می‌شود.

پیشتر صحبت شد که وظیفه تصمیم گیری بر عهده کد پایتون است. با این حساب سیمولینک قدرت تشخیص و تصمیم گیری ندارد. از این رو وظیفه بلوک Environment نیز تصمیم گیری نمی‌باشد بلکه با تکنیک هایی سعی بر برقراری ارتباط با کد پایتون دارد. در حقیقت این بلوک نقش واسط بین سیمولینک و پایتون را دارد. این بلوک علاوه بر دستورات کنترلی لاین و سرعت، دستور «شروع کردن دوباره» و یا ریست را نیز دریافت می‌کند. با این دستور تمامی اطلاعات به حالت اول بر خواهد گشت و ماشین «عامل» به جای اول بر خواهد گشت و منتظر دستورات جدید می‌ماند.

شکل ۴-۱ توضیح بسیار کلی در مورد این نحوه ارتباط نشان داده است و در شکل ۴-۶ جزئیات بیشتری را در مورد داخل این بلوک را نشان می‌دهد.

اگر به شکل ۴-۶ دقت شود دو بلوک UDP دیده می‌شود. (بلوک UDP Send در بالا سمت راست و بلوک UDP Receive در پایین سمت چپ شکل ۴-۶) این دو بلوک برای فرستادن داده های مورد نیاز



نام بلوک	نوع بلوک	IP	Port
UDP Receiver	گیرنده	localhost	۸۰۷۰
Send Data	فرستنده	localhost	۸۰۳۱

جدول ۲-۴: اطلاعات بلوک های فرستندگی گیرندگی در سیمولینک

به پایتون و گرفتن دستور های کنترلی از پایتون در سیمولینک تعبیه شده اند.

**یادداشت ۲-۳-۴.** اگر نمودار شکل ۱-۴ در نظر داشته باشیم، خواهیم یافت که این بلوک در شاخه پایینی ارتباط بین سیمولینک و پایتون قرار دارد که از روش های شبکه کردن بین این دو انجام می شود. این بلوک صرفا داده های پویا را از متلب به سیمولینک انتقال می دهد و با داده های استاتیک کاری ندارد.

در جدول ۲-۴ اطلاعات دقیق شبکه برای این دو بلوک دیده می شود. گفتنی است که این داده ها به نحوه مناسب کد شده اند تا تنها با استفاده از این دو بلوک بتوان داده ها را منتقل کرد.

روش کد شدن<sup>۳</sup> قبل از ارسال و یا دریافت در این دو بلوک کاملا با یکدیگر متفاوت هستند. این کار توسط بلوک های قبل و بعد دو بلوک مذکور انجام می شود.

در شکل ۲-۴ سه بلوک تابع متلب<sup>۴</sup> وجود دارد که از مهم ترین نقش را دارند. این نقش ها در ادامه توضیح توضیح مفصل داده شده اند.

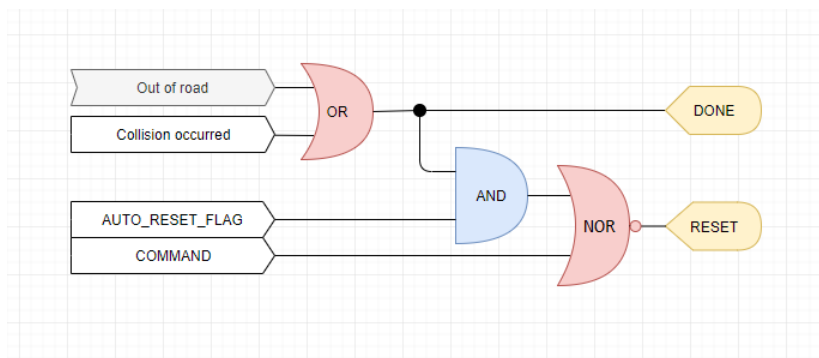
خلاصه این وظایف در زیر آمده است:

- **بلوک فرستنده:** این بلوک در سمت بالا سمت راست تمام اطلاعاتی که به بلوک کلی Environment وارد می شود را به نحوی مناسب به همراه متغیر done که از یک تابع متلب دیگر خارج می شود را برای پایتون طی ساختار مشخصی (جیسون) ارسال می کند. موضوع به صورت بسیط در بخش ۲-۳-۴ مورد بررسی قرار گرفته است.

- **بلوک گیرنده:** این بلوک تنها وظیفه آن این است که داده هایی که پس از بلوک Byte Unpacking آمده است، را انتخاب می کند که هر یک از این دیتا ها چه مفهومی هستند. از آن جا که ۳ داده کنترلی (لاین و سرعت و ریست) از طرف پایتون ارسال می شود، بلوک Byte Unpacking وظیفه این ۳ داده را به فرمت double در می آورد (این فرمت در سمت پایتون نیز به

<sup>3</sup>Encoding

<sup>4</sup>Matlab-function block



شکل ۴-۷: منطق محاسبه تمام شدن و دستور ریست کردن

همین شکل قرار داد شده است). بنابراین این بلوک Byte Unpacking است که کار اصلی را انجام می‌دهد و تنها وظیفه این بلوک تفکیک و اسم گذاری بر روی خروجی بلوک Byte Unpacking است.

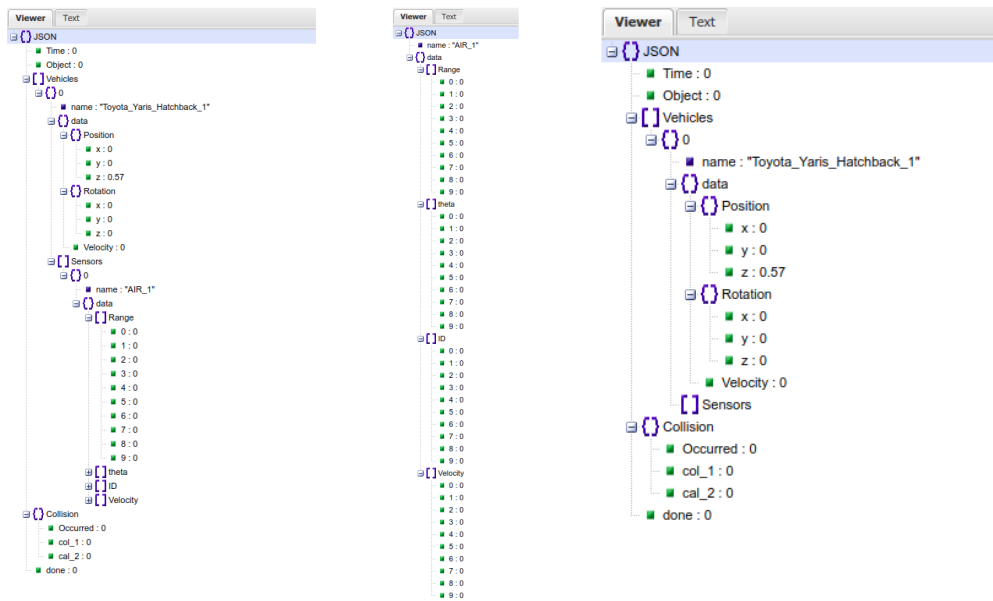
- **بلوک محاسبه تمام شدن و ریست کردن** : این بلوک با دو خروجی مهم می‌دهد. یکی از آن خروجی‌ها done است که توسط بلوک فرستنده نیز برای پایتون ارسال می‌شود. و مقدار دیگری را که خروجی می‌دهد دستور ریست کردن است. این دستور در لبه مثبت (از صفر به یک برسد) ریست می‌کند. منطق این بلوک یک منطق باینری است که در شکل ۴-۷ روش محاسبه آن نیز آمده است. همچنین این بلوک یک مقدار به اسم Auto\_reset\_flag نیز به عنوان ورودی دریافت می‌کند. این مقدار که می‌تواند صفر و یا یک باشد، تصمیم می‌گیرد که در صورتی که  $done = 1$  شد آیا ریست کند و یا ریست نکند. اگر یک باشد ریست می‌کند و گرنه منتظر رسیدن دستور از پایتون می‌ماند.

## ۴-۳-۲ بررسی ساختار داده‌های ارسالی و کد آن در سیمولینک

بلوک فرستنده در شکل ۴-۶ اطلاعات دریافت شده را طی ساختار مشخصی به ساختار جیسون در می‌آورد. نمونه‌ای از این ساختار در شکل ۴-۱ آمده است.

شکل ۴-۸ (ب) همان ساختاری است که پیشتر در مورد آن صحبت شد. با توجه به این که این ساختار (در ادامه خواهید دید که) در اطلاعات سنسور ماشین به کار گرفته می‌شود و هر سنسور ممکن است اطلاعات خاص خود را داشته باشد. برای یکسان‌سازی این اطلاعات، در اولین لایه دو گزینه data

<sup>۶</sup> این عکس با استفاده از سایت <http://jsonviewer.stack.hu/> تهیه شده است.



(آ) اطلاعات خام جیسون کلی (ب) جیسون سنسور (ج) شکل ادغام شده

شکل ۴-۸: تصویر (آ) نحوه ساختاردهی کلی در بلوک تابع متلب واقع در زیرسیستم Environment را نشان می‌دهد. در این قسمت اطلاعات ماشین شامل دو قسمت data و Sensors است. اطلاعات سنسور های این ماشین از تصویر (ب) تامین می‌شود. این ساختار در همان زیرسیستم ماشین تبدیل به جیسون شده و برای زیرسیستم Environmnet ارسال می‌شود. قسمت سنسور یک آرایه است تا بتوان چندین سنسور مختلف را برای آن ارسال کرد. بنابراین ساختاری مانند تصویر (ج) باید را در نهایت برای پایتون ارسال می‌کند.<sup>۶</sup>

و name را به طور قرار دادی بین همه سنسور ها یکسان است تا در کد پایتون بتوان با توجه به اسم آن ها ادامه ساختار که در data می‌باشد، قابل تشخیص باشد. اما از آن جا که صرفا از اطلاعات یک سنسور استفاده شده است این شکل ساختاردهی تنها یک گزینه برای توسعه های آتی آن است.

در این ساختار ۴ اطلاعات نشان داده شده که هر یک از آن اطلاعات خود یک بردار ۱۰ تایی است، برای پایتون ارسال می‌شود.

شکل ۴-۸(آ) ساختار کلی است که محیط Environmnet از آن استفاده می‌کند. در این بلوک علاوه بر داده هایی مانند اطلاعات ماشین و سنسورش، اطلاعات تصادف و اطلاعات تمام شدن و یا نشدن شبیه سازی که پیش تر در مورد آن اطلاعات صحبت شد، دو اطلاعات اضافه دیگر نیز می‌فرستد.

- Time: زمان شبیه سازی را همراه با دیتا دیگر ارسال می‌کند و به اصلاح یک ساختار زمانی ایجاد می‌شود.<sup>۷</sup>

- Object: این عبارت کمک به کد پایتون می‌کند که ماشین هدف و یا عامل را از بین ماشین

<sup>7</sup>Time-struct

های موجود در لیست Vehicles بیاید.<sup>۸</sup>

در نهایت با ادغام شدن اطلاعات سنسور نیز، ساختار کلی به شکل ۴-۸(ج) در خواهد آمد. در بخش ۴-۵ از برخی از چالش های ساختاردهی کردن به این شکل، صحبت خواهد شد. همچنین خروجی نهایی آن نیز را می توانید در همان بخش بیابید.

## ۴-۴ بررسی جزئیات بخش پایتون

بارها اشاره شد که پایتون بخش اصل تصمیم گیری را برعهده دارد. در بخش ۴-۳ سعی شد تا به نحو مناسبی دیتای محیط شبیه سازی را به پایتون منتقل کند و دستورات کنترلی را نیز از پایتون به محیط شبیه سازی ارسال کند. در این بخش بر روی مفاهیم پایتونی آن مانور می دهیم. از آن جا که کد پایتون باید به سیمولینک متصل شود و دیتا را به الگوریتم کنترلی خود (که از الگوریتم های یادگیری تقویتی استفاده شده است)، برساند و از آن جا دستورات را دریافت کند و به سیمولینک برساند، نیازمند لایه هایی است که هر لایه بخشی از این کار ها انجام دهد.

### ۴-۴-۱ معرفی لایه های کد پایتون

کد پایتون از لایه های مختلف تشکیل شده است. از هر لایه به لایه بعدی سطح زبان بالاتر می رود. این لایه ها در شکل ۴-۴-۱ مشخص شده اند. در لایه های ابتدایی، سطح استفاده از دستورات بسیار ابتدایی است و در هر لایه با تعریف توابع و کلاس هایی این امکان را ایجاد کرده اند که بدون در نظر گرفتن این که در سطوح پایین تر چه اتفاقاتی می افتد، در سطوح بالاتر از آن امکانات استفاده کرد. در ادامه این بحث مفصل توضیح داده خواهد شد.

در شکل ۴-۹ لایه برنامه نویسی شده با ۵ عدد نشان داده شده است. توضیحات زیر متناسب با هریک از این شماره ها در نظر گرفته شده اند:

① **Model**: در این لایه، با استفاده از موتور متلب با متلب و سیمولینک ارتباط برقرار می کند و با این ارتباط دو وظیفه بسیار مهم را انجام می دهد.

(آ) ساخت مدل هایی مانند اتومبیل و جاده و دریافت اطلاعات ضروری آن ها در متلب

<sup>۸</sup> از آنجایی که در این پروژه لیست Vehicles یک آبجکت بیشتر ندارد، پس این بخش نیز صرفاً ظرفیت توسعه پذیری این کد را بالا می برد.



شکل ۴-۹: بلوک دیالگرام لایه های پایتون

ب) تعریف کلاس sim و ارسال دستوراتی مانند شروع کردن، توقف کردن، مکث کردن، ادامه دادن و ... به سیمولینک.

این لایه صرفاً از اطلاعات استاتیک سیمولینک استفاده می‌کند.

② **Environment**: این لایه برای ارتباط با زیرسیستم Environment که پیشتر آن‌را در شکل ۴-۵ مشاهده کردید، ساخته شده است. در این لایه با استفاده از ابزار هایی که لایه Model در اختیار آن قرار می‌دهد، به سادگی آجکت هایی مانند ماشین و جاده را می‌سازد و با استفاده از اطلاعات شبکه که در جدول ۴-۲ آمده است، با سیمولینک ارتباط برقرار می‌کند و آن اطلاعات پویا را دریافت می‌کند و دستورات کنترلی را برای آن ارسال می‌کند.

این لایه از موتور متلب استفاده نمی‌کند بلکه از روش های شبکه ای استفاده می‌کند.

③ **Env**: این لایه یک آجکت از کلاس Environment را دریافت می‌کند و تمامی نیاز های خود را در مورد هر مسئله‌ای که به ارتباط با محیط متلب و یا سیمولینک به آن واگذار می‌کند و تمرکز آن بر روی مفاهیم یادگیری تقویتی مانند (۱) فضای مشاهده (۲) فضای حرکت (۳) تعریف حرکت (۴) تعریف حالت (۵) تعریف امتیاز و ... می‌باشد.

④ **gym**: در نهایت یک مجموعه داریم که هر یک وظایف مربوط به خود را دارند. در این قسمت کد ما که در لایه Env به gym نزدیک شده است، با فراخوانی لایه های پیشین به ترتیب از بالا به پایین (از نظر سطح) فراخوانی می‌شود و در هر لایه آجکت‌هایی از کلاس‌های موجود در لایه

پایین تر فراخوانی می‌شود. از این رو جهت پیکان بر عکس است.

در مورد gym در فصل ۲ به صورت مفصل بحث شده‌است.

**Algorithm ۵ :** در لایه های پیشین سینتکس کد gym ایجاد شده است و این لایه بر توسعه الگوریتم های یادگیری تقویتی که در فصل ۱ مورد بررسی مفصل قرار گرفته است، تمرکز دارد. می‌توان گفت این لایه نقش رییس و یا مغز کار را دارد و باقی لایه ها کارگرانی هستند که وظیفه دارند که اطلاعات را به شکلی مناسب این لایه، برای آن منتقل کنند.

## ۴-۵ بررسی دقیق تر برخی چالش های فنی پروژه

یکی از مهمترین چالش های این پروژه برقرار ارتباط محیط شبیه سازی سیمولینک با پایتون بود. با استفاده از بلوک UDP Send می‌توان یک یا چند داده هم نوع را منتقل کرد. این کار از لحاظ فنی، موضوع انتقال را دشوار کرده بود. زیرا یا باید برای هر دیتا، یک بلوک فرستنده قرار داد و یا باید همه دیتا ها را به یک نوع تبدیل کرد و سپس اطلاعات به ترتیب خاصی برای فرستنده فرستاد.

روش دوم دو تا مشکل دارد یکی آن‌که به شدت به ترتیب قرار گیری دیتا وابسته خواهد شد و دیباگ آن بسیار سخت می‌شود. دوم آنکه از دیتای ارسالی هیچ مفهومی را نمی‌توان بدون داشتن آن ترتیب استخراج نمود و به عبارت دیگر اصلا ماژولار نخواهد بود و کد بسیار نامفهوم و سخت خواهد شد و در صورت توسعه بخش فنی عملا کار از اول شروع می‌شود.

برای حل این مشکلات بلوکی به نام Environment در نظر گرفته شد که هرگونه ارتباط با پایتون توسط این بلوک رخ می‌دهد. سپس در قسمت فرستنده یک تابع پیش‌پردازنده تعبیه شد تا داده ها را جمع آوری و ساختار دهی کند. ساختار استفاده شده json است که در شکل ۴-۸ معرفی شد. این ساختار، کد را به شدت ماژولار می‌کند.

نکته خیلی مهم در استفاده از این ساختار این است که این ساختار به شکل رشته می‌باشد. از آنجایی که در سیمولینک باید ابعاد ماتریس ورودی و خروجی مشخص باشد و از آنجایی که متلب رشته را مانند یک بردار با طول متغیر می‌بیند بنابراین محیط سیمولینک به محض آن که خروجی و یا ورودی داده ای از جنس کاراکتر باشد، پیغام خطا می‌دهد.

روشی که برای دور زدن این مورد به کار رفت؛ به شرح زیر است.

۱. طول ساختار جیسون را بر حسب کاراکتر ثابت کشت. برای این کار لازم شد که طول اعداد نیز



ثابت شود.

۲. خروجی به uint8 تبدیل شد. زیرا دیگر جنس آن کاراکتر نیست ولی از لحاظ حافظه دقیقا برابر char می باشد.

خروجی تابع پیش پردازنده به صورت زیر خواهد بود.

```
'{"Time":0,"Object":0,"Vehicles":[{"name":"Toyota_Yaris_Hatchback_1","data":{"Position":{"x": 0.00000e+00,"y": 0.00000e+00,"z": 5.70000e-01},"Rotation":{"x": 0.00000e+00,"y": 0.00000e+00,"z": 0.00000e+00},"Velocity": 0.00000e+00},"Sensors":[{"name":"AIR_1","data":{"Range": [ 0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00],"theta": [ 0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00],"ID": [ 0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00],"Velocity": [ 0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00,0.00000e+00]}]}],"Collision":{"Occurred":0,"col_1": 0,"cal_2": 0},"done":0}'
```

در این ساختار اعداد هر محدوده ای که داشته باشد با طول و دقت مشخص در کنار یک دیگر قرار می گیرند. بنابراین این مشکل نیز حل می شود.

## فصل پنجم

### شبیه سازی و نتایج

نام مخزن	توضیحات	نیاز به موتور متلب
gym-Prescan	پروژه کامل در این مخزن قرار دارد. در آن بر بخش الگوریتم از کتابخانه هایی استفاده شده است که بر روی سیستم عامل لینوکس قابل اجرا هستند بنابراین در صورتی که از لایه الگوریتمی که این پروژه از آن استفاده می کند استفاده نشود، در هر سیستم عاملی می توان از آن استفاده کرد.	×✓
gym-Prescan-minimal	در این جا بخش فایل های پری اسکن به همراه فایل هایی که به موتور متلب نیاز دارد حذف شده است.	×

جدول ۵-۱: اطلاعات مخزن های پروژه در گیت هاب

در فصل های گذشته در خصوص ابزار هایی که در این پروژه استفاده شده اند، صحبت شد و توضیح مفصلی بر چپستی آن ابزار ها و ضرورت استفاده از آن ها داده شد. اما سوالات بی جوابی نیز ماند که در این فصل به آن ها خواهیم پرداخت.

یکی از آن سوالات نحوه راه اندازی کد پروژه می باشد و سوال دیگر نتیجه حاصل از شبیه سازی نهایی چگونه است می باشد.

## ۵-۱ راه اندازی

این کد در گیت هاب در دو مخزن<sup>۱</sup> قرار دارد. جدول ۵-۱ اطلاعات این مخزن ها<sup>۲</sup> را نشان می دهد.<sup>۳</sup> ابتدا پیش نیاز های لازم را که در بخش ۲ توضیح داده شدند، را نصب کنید. پیشنهاد می شود که تمامی نسخه های لازم توضیح داده شده در بخش ۲ را در سیستم عامل لینوکس استفاده کنید. اگر به توصیه استفاده از لینوکس عمل نکرده باشید، می توانید با استفاده شبکه که کد در اختیار تان قرار می دهد کد را روی دو کامپیوتر ران کنید به طوری که در یک کامپیوتر ویندوز و نرم افزار های گفته شده نصب باشد و روی دیگری لینوکس و پایتون و پیش نیاز های پایتون که در بخش ۲-۴ بررسی شدند

<sup>۱</sup>Repository

<sup>۲</sup>غیر از این دو مخزن، مخزن دیگری به آدرس زیر وجود دارد که تاریخچه و روند رسیدن به این نتیجه نهایی را نشان می دهد.

<sup>۳</sup>آدرس این مخزن به این شکل بدست می آید: <https://github.com/mohammadraziei/<REPOSITORY-NAME>>

نصب باشد.

**یادداشت ۵-۱-۱.** بهتر است این دو کامپیوتر در یک شبکه داخلی به یک دیگر متصل شده باشند.

حال وارد کامپیوتری شوید که ویندوز بر روی آن نصب است. این کامپیوتر قرار است نقش محیط را برای ما ایجاد کند.

**یادداشت ۵-۱-۲.** کد های پایتون صرفاً بر روی کامپیوتری که لینوکس دارد اجرا کنید.

که دستور زیر را در ترمینال<sup>۴</sup> خود وارد کنید.


```
1 git clone https://github.com/MohammadRaziei/gym-Prescan.git
2 pip install -e gym-Prescan
```

خط دوم این کد، اختیاری می باشد و صرفاً کار را ساده می کند. همچنین می توان آن را به صورت زیر نیز نوشت:

```
1 pip install git+https://github.com/MohammadRaziei/gym-Prescan
```

سپس وارد مسیر زیر شوید.

`gym-Prescan/gym_prescan/envs/PreScan`

سپس با استفاده از آیکون <sup>۵</sup> کلیک کنید. در این صورت بر روی نوار Toolbar این آیکون نیز ظاهر می شود. با فشردن آن، پنجره شکل ۲-۳ باز می شود. بر روی Matlab کلیک کنید تا محیط متلب باز شود. اجرای فایل `startup.m` برای هنگامی که از کد پایتون بر روی همان سیستم استفاده نمی کنید، اختیاری است.

فایل سیمولینک را باز کنید و به صورت دستی IP بلوک فرستنده را به IP مورد نظر تغییر دهید و یا با استفاده از کد زیر در کامندلاین متلب تغییرات لازم را انجام دهید.

```
1 ExperimentName = 'PreScan_Vissim_Python_0';
2 send_ip = 'localhost';
3 sys = load_system([ExperimentName '_cs']);
4 set_param([ExperimentName '_cs/Environment/Send Data'],
    'remoteURL', ['' send_ip '']);
5 save_system(sys);
```

<sup>۴</sup>Terminal

<sup>۵</sup> این آیکون پس از نصب نرم افزار پری اسکن بر روی دستکتاپ تشکیل می شود.

در این کد کافیت مقدار `send_ip` را مطابق با IP کامپیوتر دیگر تنظیم کنید. پس از تنظیمات فایل سیمولینک را اجرا کنید برای این کار می‌توانید آن را باز کرده و اجرا کنید و یا با استفاده از دستور زیر در کامندلاین متلب آن را انجام دهید.

```
1 ExperimentName = 'PreScan_Vissim_Python_0';
2 sys = load_system([ExperimentName '_cs']);
3 set_param(bdroot, 'SimulationCommand', 'start');
```

حال در سیستم لینکوس خود می‌توانید بسته زیر را دانلود کنید.

```
1 git clone https://github.com/MohammadRaziei/gym-PreScan-minimal.git
2 cd gym-PreScan-minimal
```

در این پوشه تعدادی از الگوریتم‌های معروفی که در حوزه یادگیری تقویتی عمیق (DRL<sup>6</sup>) نوشته شده است، قرار دارد. در بین این الگوریتم‌ها دو الگوریتم DQN و A2C نسبت به بقیه بهتر جواب داده‌اند. این الگوریتم‌ها در بخش ۱ و در [۷] توضیح کامل داده شده‌اند.

برای اجرای الگوریتم DQN باید ابتدا IP کامپیوتر ویندوزی را در قسمت مشخص در متغیر `env_dict` که در جدول ۱-۳ به‌طور کامل بررسی شده است، بنویسید و سپس دستور `python dqn.py` را در ترمینال لینوکس وارد کنید. بدین صورت محیط شبیه‌سازی روی هر دو کامپیوتر شروع به کار می‌کند.

## ۲-۵ نتایج شبیه‌سازی

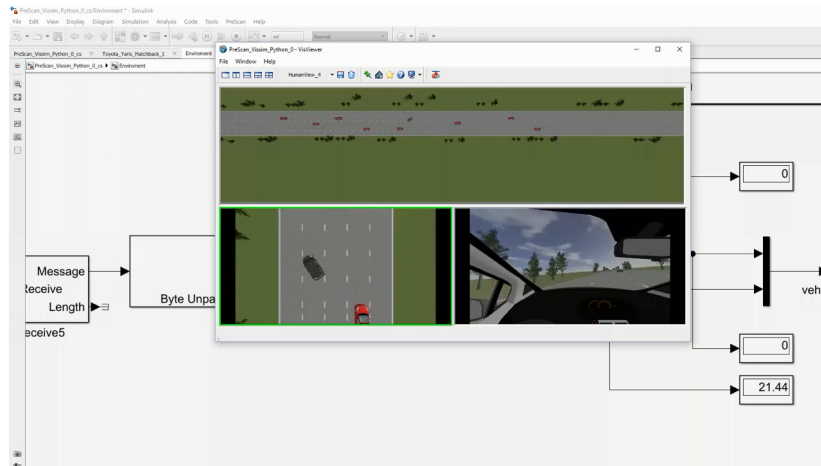
الگوریتم استفاده شده در این پروژه DQN می‌باشد. مقدار  $\gamma$  در این الگوریتم  $\frac{7}{8}$  انتخاب شده است. یا توجه به حالت و امتیازهای تعریف شده در فصل ۳ پس از ۲۰۰ بار تلاش به سیاست بهینه<sup>۷</sup> می‌رسد. اما برای بالا رفتن دقت، ۲۵۰۰ بار عامل مسیر را طی کرده است. دلیل آن بالا رفتن تعداد اپیزودها می‌باشد. با این کار مقدار  $\epsilon$  کاهش یافته و احتمال جست و جوی سیاست جدید آن کاهش می‌یابد.

در کل پس یافتن سیاست بهینه، تعداد مرحله‌ها به ۷۳ رسید. سرعت عامل حول ۲۱ در حال نوسان است و میانگین بدون وزن امتیازها برابر ۱/۳۳۸۲۱ می‌باشد.

از آنجایی که نتایج این پروژه به صورت تصویر قابل بیان نیستند و نمی‌توان حتی چندین تصویر مختلف از وضعیت‌های مختلف آن گرفت و به عنوان نتیجه منتشر کرد، از نتایج این پروژه، فیلمی تهیه

<sup>6</sup>Deep Reinforcement Learning

<sup>7</sup>Optimal Policy



شکل ۵-۱: شبیه سازی نهایی، شکلی مانند این دارد. از این رو فیلمی از این محیط در حال اجرا تهیه شده است که در آدرس <https://youtu.be/chgpBWU9XCA> بارگذاری شده است.

شده است که در آدرس زیر بارگذاری شده است.

<https://youtu.be/chgpBWU9XCA>

پایان

## منابع و مراجع

- [1] D. Silver, “UCL course on RL.” <http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>, 2015.
- [2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.
- [3] G. Hayes, “How to install openai gym in a windows environment.” <https://medium.com/p/338969e24d30>, 2018.
- [4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” 2016.
- [5] A. Hill, “Stable baselines: a fork of openai baselines — reinforcement learning made easy.” <https://stable-baselines.readthedocs.io/en/master/>.
- [6] A. Hill, “Stable baselines: a fork of openai baselines — reinforcement learning made easy.” <https://medium.com/p/df87c4b2fc82/>.
- [7] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, “Stable baselines.” <https://github.com/hill-a/stable-baselines>, 2018.

## نمایه

حرکت، ۸۶، ۲۱	مخزن، ۲۷
عامل، ۹۶، ۱۷، ۲۰، ۲۱	امتیاز، ۷۵، ۲۰، ۲۲
حالت عامل، ۸، ۲۰	فرضیه امتیازها، ۵
الگوریتم، ۱۶، ۱۷	یادگیری تقویتی، ۶۴، ۱۶، ۲۰، ۲۲
ماشین خودران، ۲۰	یادگیری شبه ناظر، ۴
نقطه تراز، ۱۶	حالت، ۸۶، ۲۲
خوشه بندی، ۴	مرحله، ۶، ۲۱
هزینه، ۵	ناظر، ۵
یادگیری تقویتی عمیق، ۱۷، ۲۸	یادگیری با ناظر، ۴
محیط، ۶، ۷، ۹، ۱۶، ۱۷، ۲۰، ۲۷	ترمینال، ۲۷
حالت محیط، ۷، ۸، ۲۰	تست، ۲۲، ۲۳
اپیزود، ۲۱	آموزش، ۲۲
فیدبک، ۵	یادگیری بدون ناظر، ۴
تاریخچه، ۷، ۸	
حالت اطلاعاتی، ۹	
مارکوف، ۸	
حالت مارکوف، ۸، ۹	
متلب، ۱۷	
یادگیری ماشین، ۴	
مشاهده، ۶، ۷	
مشاهده پذیری، ۸، ۲۰	
پری اسکن، ۲۸	



# فهرست اختصارات

## A

A2C ..... Synchronous Actor Critics

## D

DQN ..... Deep Q-Learning Network

DRL ..... Deep Reinforcement Learning

## R

RL ..... Reinforcement Learning

# واژه‌نامه انگلیسی به فارسی

## E

Environment ..... محیط  
Environmnet State ..... حالت محیط  
Episode ..... اپیزود

## F

Feedback ..... بازخورد  
Full observability ... مشاهده‌پذیری کامل

## H

History ..... تاریخچه

## I

Information State ..... حالت اطلاعاتی

## J

Joystick ..... دسته‌بازی

## M

Machine Learning ..... یادگیری ماشین

## A

Action ..... حرکت  
Action Space ..... فضای حرکت  
Agent ..... عامل  
Agent State ..... حالت عامل  
Algorithm ..... الگوریتم  
Anaconda ..... نرم‌افزار آناکوندا  
API ..... رابط برنامه‌نویسی برنامه  
Autonomous Vehicle .... ماشین خودران

## B

Bechmark ..... نقطه تراز

## C

Clustering ..... خوشه بندی  
Cost ..... هزینه

## D

Deep ..... یادگیری تقویتی عمیق  
Reinforcement Learning  
Dynamic Data ..... داده‌های پویا

Simulink ..... سیمولینک  
 State ..... حالت  
 Step ..... مرحله  
 Supervised Learning ..... یادگیری با ناظر  
 Supervisor ..... ناظر

## T

Terminal ..... ترمینال  
 Test ..... تست  
 Timeout ..... سقف زمانی  
 Train ..... آموزش

## U

Unsupervised Learning ..... یادگیری بدون ناظر

Markov ..... مارکوف  
 Markov Decision Process ..... روند تصمیم‌گیری مارکوف  
 Markov State ..... حالت مارکوف  
 Matlab ..... نرم‌افزار متلب  
 Matlab Engine ..... موتور متلب

## O

Observability ..... مشاهده‌پذیری  
 Observation ..... مشاهده  
 Observation Space ..... فضای مشاهده  
 Optimal Policy ..... سیاست بهینه

## P

Partial observability ..... مشاهده‌پذیری جزئی  
 Policy ..... سیاست  
 PreScan ..... نرم‌افزار پری‌اسکن

## R

Reinforcement Learning ..... یادگیری تقویتی  
 Repository ..... مخزن (گیت‌هاب)  
 Reward ..... امتیاز  
 Reward Hypothesis ..... فرضیه امتیازها

## S

Semi-Supervised Learning ..... یادگیری شبه ناظر

# واژه‌نامه فارسی به انگلیسی

خ	ا
Clustering ..... خوشه بندی	Train ..... آموزش
	Episode ..... اپیزود
	Algorithm ..... الگوریتم
د	Reward ..... امتیاز
Dynamic Data ..... داده‌های پویا	
Joystick ..... دسته‌بازی	ب
	Feedback ..... بازخورد
ر	
API ..... رابط برنامه‌نویسی برنامه	ت
	History ..... تاریخچه
س	Terminal ..... ترمینال
Timeout ..... سقف زمانی	Test ..... تست
Policy ..... سیاست	
Optimal Policy ..... سیاست بهینه	ح
Simulink ..... سیمولینک	State ..... حالت
	Information State ..... حالت اطلاعاتی
ع	Agent State ..... حالت عامل
Agent ..... عامل	Markov State ..... حالت مارکوف
	Environmnet State ..... حالت محیط
	Action ..... حرکت

## ف

Reward Hypothesis ..... فرضیه امتیازها  
 Action Space ..... فضای حرکت  
 Observation Space ..... فضای مشاهده

## م

Markov ..... مارکوف  
 Autonomous Vehicle ..... ماشین خودران  
 Environment ..... محیط  
 Repository ..... مخزن (گیت‌هاب)  
 Step ..... مرحله  
 Observation ..... مشاهده  
 Observability ..... مشاهده‌پذیری  
 Partial observability ..... مشاهده‌پذیری جزئی  
 Full observability ..... مشاهده‌پذیری کامل  
 Matlab Engine ..... موتور متلب

## ن

Supervisor ..... ناظر  
 Anaconda ..... نرم‌افزار آناکوندا  
 PreScan ..... نرم‌افزار پری‌اسکن  
 Matlab ..... نرم‌افزار متلب  
 Bechmark ..... نقطه تراز

## ه

Cost ..... هزینه

## ی

Supervised Learning ..... یادگیری با ناظر  
 Unsupervised Learning ..... یادگیری بدون ناظر  
 Reinforcement Learning ..... یادگیری تقویتی  
 Deep ..... یادگیری تقویتی عمیق  
 Reinforcement Learning ..... یادگیری تقویتی عمیق  
 Semi-Supervised ..... یادگیری شبه ناظر  
 Learning ..... یادگیری  
 Machine Learning ..... یادگیری ماشین