平成29度 特別研究報告書

深層強化学習を用いた先進運転支援システム(ADAS)の検討

龍谷大学　理工学部　情報メディア学科

学籍番号　：　T140431
氏名　　：　新井 晶登

指導教員　：　三好力 教授

内容梗概

近年, 自動車市場では, 性能や燃費の良さと共に, 安全性も重要視されている. センシング技術の向上により万が一の事故の時に乗車する人の負傷を防止するだけでなく, 事故そのものを防止する「先進運転支援システム (ADAS)」が急速に成長している. 現在の ADAS では車に取り付けられたセンサや GPS などの位置情報から, 技術者が危険な状態の閾値を決め, それによって制御を考えている. しかしこれでは想定していない状況に対しては危険を回避することが出来ない. また, センシング技術の進歩によりレーダーやカメラからのデータ量が更に多くなると考えられる. 技術者がセンシングからの大量のデータから事故が発生する状況を考えるのは, パターン数が多すぎてすべてを網羅するのは難しいと考える. 本研究では, 技術者が設計していた危険な状況の閾値と制御を深層強化学習を用いた ADAS の検討を行う.

目 次

1 第 1 章 はじめに **3**

2 第 2 章 ADAS(先進運転支援システム) とは **4**

2.1 ADAS の概要 4

2.2 ADAS の例 4

2.3 ADAS の応用 5

3 第 3 章 強化学習 **6**

3.1 強化学習の概要 6

3.2 Q 学習 6

3.3 Q 学習の例 7

3.4 強化学習の限界 8

3.5 ニューラルネットワーク 8

3.6 Deep Q-learning 10

4 第 4 章 提案手法 **11**

4.1 問題定義 11

4.2 提案手法 11

4.3 UI 部及び運転部のプログラム 11

4.4 ADAS 部の Deep Q-network プログラム 18

5 第 5 章 実験 **21**

5.1 シミュレータを使用した学習の実験 21

5.2 ADAS の実証実験 24

5.3 考察 25

6 第 6 章 まとめ **26**

謝辞 **27**

A 付録 **29**

1　第1章 はじめに

近年, 自動車市場では, 性能や燃費の良さと共に, 安全性も重要視されている. 私達が使用する自動車には様々な安全技術が使われている。万一事故が発生した場合に, 乗員の被害を最小限に抑える, シートベルトやエアバックなどの「衝突の対策 (パッシブセーフティ)」や, 事故が起こる前にドライバーの運転を支援し, 事故を未然に防ぐために開発された, アンチロック・ブレーキシステム (ABS) や横滑り防止装置 (ESC) などの「衝突を回避・軽減するための対策 (アクティブセーフティ)」と自動車の安全技術は進化してきた。そして光, カメラからのセンシング技術の向上により万が一の事故の時に乗車する人の負傷を防止するだけでなく, 事故そのものを防止する「先進運転支援システム (ADAS)」が急速に成長している. 最初は高級車向けの高価なオプションとしてだったが, 今ではほとんどの一般車にも搭載されている. 主な機能としては, レーダーやカメラからのセンシングにより障害物を検知して危険を避ける, 白線を検知し車線逸脱を警告するなどがある. このような ADAS を搭載した自動車は,「ぶつからない車」として大きく注目されている.

しかし今の ADAS は技術者が危険な状況の閾値を決め, 制御を考えている. これでは想定していない状況に対しては危険を回避することが出来ない. また, センシング技術の進歩によりレーダーやカメラからのデータ量が更に多くなると考えられる. 技術者がセンシングからの大量のデータから事故が発生する状況を考えるのは, パターン数が多すぎてすべてを網羅するのは難しいと考える.

そこで, 現在のような技術者がすべてを設計しなくても自律的に危険を回避する運転を取得できる様にセンシングのデータから機械学習により ADAS のように危険を回避する運転を学習し, 現在の ADAS が出来ていることが出来るようにすることを最終目的とし研究を行う.

本研究では, 機械学習の中でも強化学習を用いることにより試行錯誤的に様々な状況に対しての行動を学習させる. また、試行錯誤的に学習するので技術者が考えつかなかった状況に対しての運転や, 技術者が設計した運転より優れた運転を発見することが期待される.

2　第2章 ADAS(先進運転支援システム) とは

2.1　ADAS の概要

ADAS(先進運転支援システム) とは, 安全でより良い運転のために事故などの可能性を事前に検知し回避しするために開発されたシステムで,「アドバンスド・ドライバー・アシスタンス・システム (advanced driver assistance system)」の略である.ADAS の機能は基本的に, 車体にレーダー, カメラおよび超音波などのセンサを持ち車体それぞれが独立したシステムとなっている. センシングからのでデータから見通しの悪い状態や, わき見運転などによって事故が起こる危険性を検知し, 自動制御や危険を警告することにより, 事故を未然に防止もしくは軽減するためのものとなる. 解りやすく言うといわゆる「ぶつからない車」である. 車を運転するためには「認知」「判断」「操作」といった動作が必要になるが,ADAS ではセンサが車の目となり, 運転の一連の動作に対して安全を確保するということに重点を置いて, ドライバーの支援を行う.

2.2　ADAS の例

ADAS では以下の様な機能が提供されることになります.

- 車線逸脱警報：車が指示器無しに車線を逸脱しようとしているのを警報する
- 物体の検知/分類：センサから物体を検知し, それが何であるかを分類する
- 前方/後方の駐車支援：駐車するときに死角となる場所の映像を映し, ずれていないか知らせる
- 前方衝突警報：前方に車や障害物が近くにある時にドライバーに警報を鳴らし危険を知らせる
- 車線変更支援：車線を変更するときに変更する車線から車が来ていないか知らせる
- 死角検知：ドライバーから見えない死角にバイクなど居ないか知らせる
- 衝突緩和ブレーキ・システム：前方の車や障害物にぶつかりそうな時に自動で止まる (自動ブレーキとも言う)

- 全車速追従クローズコントロール：前方の車に自動で追従する

最新の技術により, 様々な機能が提供されていますが,ADAS として搭載されている有名なスバルのアイサイトを例としてみてみましょう.

アイサイトには, 人の目と同じように, 左右 2 つのカメラが搭載されており立体的に環境を把握し, 車だけでなく歩行者や自転車なども識別し, 対象の移動速度や距離を正確に認識している. この技術により 5 つの機能を実現している.1 つ目は「プリクラッシュブレーキ」といい, いわゆる自動ブレーキである. 衝突の危険がある場合, ドライバーに注意を喚起し, 回避操作がない場合にはブレーキ制御を行う.2 つ目は「全車速追従機能付クルーズコントロール」といい, 高速道路や自動車専用道路で先行者に追従走行するシステムで, 先行者が停止するとブレーキ制御で減速, 停止を行う.3 つ目は「アクティブレーンキープ」といい, 走行車線両側の区画線を認識してステアリング操作のアシストを行い, 車線内中央付近の維持や車線逸脱抑制により車線からはみ出すのを防ぐ.4 つ目は「AT 誤発進抑制制御/AT 誤後進抑制制御」といい, 壁や生け垣などの障害物が検知され, 誤発進, 誤後進したと判断したら警告表示をし, エンジンの出力を抑え, 緩やかに発進, 後進する.5 つ目は「警報・お知らせ機能」といい, ドライバーが疲れや眠気で注意力が散漫になり, 車がふらついたり, 車線からはみ出しそうになった時に, ドライバーに警告をすることで回避操作を促す.

このように ADAS とは基本的に, センシングによって事故が起きそうならドライバーに注意を喚起し, それでも回避行動がされないなら自動制御するという形になっている. また, 起動条件はセンシングの結果から閾値により決定されている.

2.3 ADAS の応用

この ADAS を応用したものが自動運転になってくる. 自動運転にはレベル 0〜5 が存在しておりそれは, 国土交通省が定義している. レベル 2 まではドライバーの運転を支援する ADAS で, 万が一事故を起こした場合の責任の所在はドライバー側になります. レベル 3 以上は基本的にドライバーが操作を行わなくても運転が行われるものとなり, 事故の責任はシステム側になります. レベル 3 以上を自動運転と定義されている.

3　第3章 強化学習

3.1　強化学習の概要

強化学習には「環境」「エージェント」「状況」「行動」「報酬」という考え方が存在する. エージェントは環境から取得した状況に対し行動を行う. その結果から報酬と新たな状況を受け取る. その報酬もし良い報酬だったら状況に対するその行動が良い行動だと評価される. 報酬を得る前の行動もこの行動あっての報酬ということで, この行動もなかなか良い行動と評価される. ではその前は … と, いったように報酬を得た行動から遡るように逆算して行動を評価していく. このように強化学習では,「連続した行動」に対し評価を行う. この評価を試行錯誤的に様々な状況に対し行動を行うことで, 自らより多くの報酬を得る連続した行動を学習していく. エージェントを学習させるアル

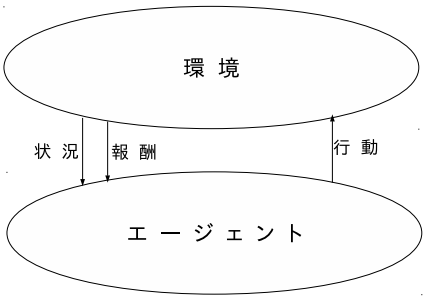

図 1: 強化学習

ゴリズムとして「Q 学習」「Sarsa」「モンテカルロ法」などがあるが, 本研究では「Q 学習」を扱う.

3.2　Q 学習

Q 学習とは, 或る環境状況 s の下で, 行動 a を選択する行動価値観数 Q(s,a) を学習する手法である. もし行動価値観数 Q が完全に分かっているなら,Q(s,a) が最大となる行動 a が最も最良の選択になる. しかし, 最初は正しい Q(s,a) の値は全くわかっていない.Q 関数を学習させるためには, 試行錯誤的に様々な状況を経験させることで, 或る状況の時どの行動がどのくらいの報酬に結びつくかを学習していく. 行動価値関数の一般的な

更新関数は, 以下のようになる.

$$Q(s_{t+1}, a_{t+1}) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{1}$$

ここで,s_t と a_t は, 時刻 t における環境と行動を表す. 行動 a_t により, 状況は s_{t+1} に変化する. r_t は, その状況の変化によって貰える報酬となる.$\gamma \max_a Q(s_{t+1}, a)$ は, 状況 s_{t+1} の時に,Q 値の一番高い行動 a を選んだ場合の Q 値に γ を掛けたものである.γ は割引率と呼ばれ, 未来で得る報酬をどの程度考慮するかを調整し, 定数として 0〜1 に設定する.α は学習率と呼ばれ,Q 値の更新の速さを制御し, これも定数で 0〜1 に設定する. ある程度学習が進むと,Q 関数が最大となる行動が, そこそこ良い行動となっていく. だが Q 関数が最大となる行動だけをしていては, 他に最良の行動があったとしても見つけることが出来ない. そこで, ε -greedy 法という手法を使う. これは, ある割合 ε でランダムな行動を選択,1- ε の割合で Q 関数が最大となる行動を選択する. これにより, いろいろな行動に対する適切な Q 値を学習する. 学習させるためのアルゴリズムは以下のようになる.

すべての Q(s,a) をランダムに初期化する
以下を繰り返す
　開始状態を s にセットする
　以下を繰り返す
　　状態 s_t における行動 a_t を ε -greedy 法で選択
　　行動 a_t を行い, 状態 s_{t+1} に遷移し, 報酬 r_t を得る
　　この行動に該当する $Q(s_t, a_t)$ を更新する
　　$s_t \leftarrow s_{t+1}$
　　終了条件 (目的の達成) を満たせば終了, 満たしていないならループ
　終了条件 (学習回数など) を満たせば終了, 満たしていないならループ

3.3 Q 学習の例

Q 学習の例として, 図 2 を環境とし経路探索を行う.1 をスタートとし,9 をゴールとする.9 に辿り着くと報酬として 100 を得て 1 に戻る. 学習率は 0.3, 割引率は 0.9 に設定し,1000 回学習を行う. 学習の結果, 行動価値関数は図 4 のようになった. 図 3 には状況

ごとに値が多い行動を矢印で表している. 図 3 をみてわかるようにスタートからゴールまでの経路を学習した.

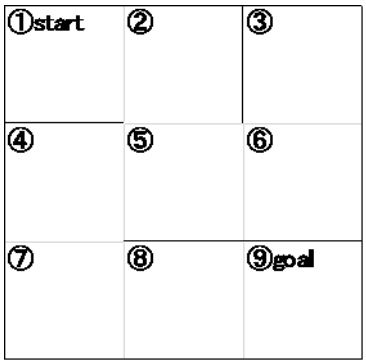

図 2: 環境

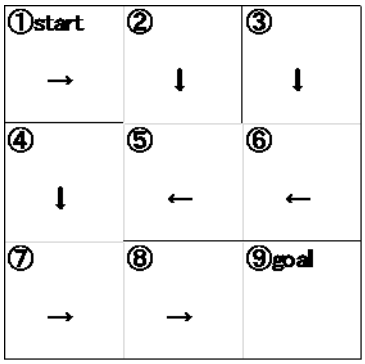

図 3: 学習後の環境

	上	右	下	左
①	31	39	31	31
②	39	39	48	39
③	0	0	39	0
④	58	48	69	58
⑤	39	39	48	58
⑥	25	82	69	69
⑦	58	82	69	69
⑧	82	96	82	82
⑨	0	0	0	0

図 4: 学習後の行動価値関数

3.4 強化学習の限界

強化学習にはある問題点がある. それは小さな環境情報でなければ動かすことが出来ないことにある. 今回行った経路探索は非常に小さな環境だったが, 環境情報が大きくなると行動価値関数の行数が膨大な数になり, 行動価値関数を値を確保するメモリも多く確保しなければいけない. なによりも膨大な学習時間が必要になる. そこで, 行動価値関数をニューラルネットワークで近似する, 深層強化学習へ繋がっていく.

3.5 ニューラルネットワーク

深層強化学習を知る際に,「ニューラルネットワーク」という概念をおさえておく必要がある. ニューラルネットワークとは, 人間の脳内にある神経細胞 (ニューロン) とその繋がり, つまり神経回路網を人工ニューロンという数式的なモデルで表現したものである. ニューラルネットワークには, 入力層, 隠れ層, 出力層から構成され, 層と層のニューロンは重みによって繋がっている. 人工ニューロン単体を図 5 に表す. 図 5 の人工ニューロンを数式で表すと以下の様になる.

$$net = x_1 * w_1 + x_2 * w_2 + \cdots + x_n * w_n \tag{2}$$

$$out = F(net - \theta) \tag{3}$$

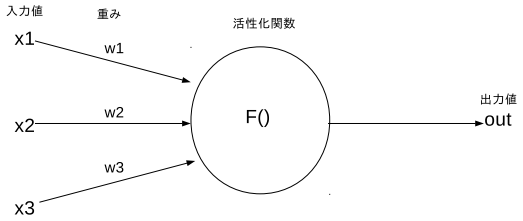

図 5: 人工ニューロン

F() は活性化関数と呼ばれこれにより, 非線形の形も表現することが出来る. このニューラルネットワークを多層構造にすることでより複雑な関数に近似することができるディープラーニングとなる. ディープラーニングの学習の例として「入力層」「隠れ層」「出力層」で構成された 3 層のパーセプトロンについて考えてみる.3 層の多層パーセプトロンを図 6 に表す. データが入力層の x に入ってくると, その値に重みかけ y に, その y の値に重みをかけ出力層に z が出力される. 出力層を教師データに近づけるように, 重みを調整していくことで学習が行われていく.

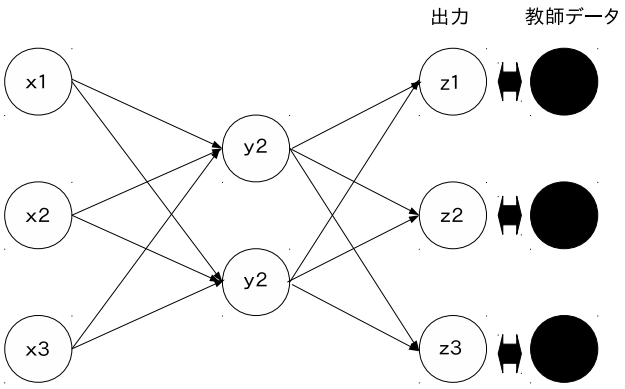

図 6: 多層パーセプトロン

3.6 Deep Q-learning

深層強化学習とは, 入力層を状況 s, 出力層を各行動 a に対する $Q(s,a)$ とし,Q 関数をニューラルネットワークで近似したモノのことを指し, そのネットワークのモデルを DeepQNetwork(DQN) という.DQN の学習の基本は誤差伝搬法となり, 出力結果と正解との誤差を計算して逆方向に伝播させることで,DQN の出力が正解に近くなるように重みを調整する手法である. そのため,DQN を近似する場合「誤差」を求めなければな

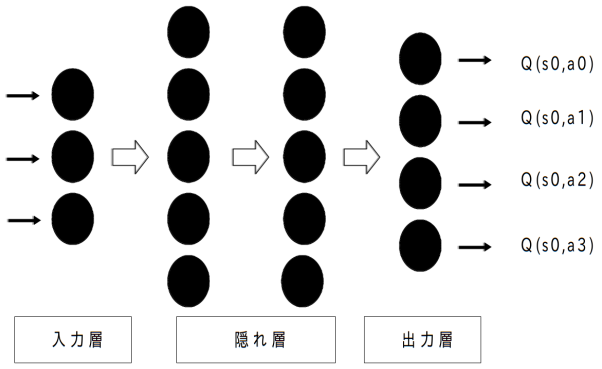

図 7: Deep Q-network(DQN)

らない. 誤差は以下の式によって求める.

$$L_\theta = E[\frac{1}{2}(r_{t+1} + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t))^2] \tag{4}$$

2 乗しているのは誤差のためで,$\frac{1}{2}$ しているのは微分した時に出てくる 2 を消すためにかけてある. そして,(4) の式を微分し得られる誤差伝播させるため勾配法には様々な手法が提唱されており, その中から合うものを選ぶことになる.

4　第4章 提案手法

4.1　問題定義

現在の ADAS では車に取り付けられたセンサや GPS などの位置情報から, 技術者が危険な状態の閾値を決め, それによって制御を考えている. しかしこれでは想定していない状況に対しては危険を回避することが出来ない. また, センシング技術の進歩によりレーダーやカメラからのデータ量が更に多くなると考えられる. 技術者がセンシングからの大量のデータから事故が発生する状況を考えるのは, パターン数が多すぎてすべてを網羅するのは難しいと考える.

そこで, 今まで技術者が行っていた危険な状態の閾値を決める部分を深層強化学習 (DQN) によって学習させる手法を考えた. 強化学習による試行錯誤的な学習を行うことで, 様々な状態に対しどのように行動すればよいかを学習する. 良い行動, 悪い行動すべてを網羅的に学習するため, 正常に走行している時や, 危険な時を判別できるのではないかと考える. 深層学習を使うことで, センシングからの大量のデータにも対応することができる.

DQN が学習できているかを検証するために, 学習させた DQN を使い安定して道路を走行している時と壁や他車に衝突している時や車線から外れた時の DQN の出力値を比べることで危険な状況にいることを判別するために使えるか検証を行う. また, 危険な状況と判別した時に,DQN に制御を任せたときと, 任せなかった時とで精度がどう変わったかを実験で検証する.

4.2　提案手法

DQN の学習を行うためのシミュレータの作成を行う. シミュレータには UI 部及び運転部と DQN によって学習を行う ADAS 部から構成されている. シミュレータに DQN の「環境」となる部分を作成し, そのシミュレータによって学習を行う.

4.3　UI 部及び運転部のプログラム

図 8 のようなシミュレータを作成した. このシミュレータはｕｂｕｎｔｕ 16.04LTS 上で python2.7 と python のライブラリである pygame1.9.3 と tensorflow1.2.1 と keras2.0.6

を使い作成を行った. シミュレータは現実の道路と同じようにするために, 白線の区画

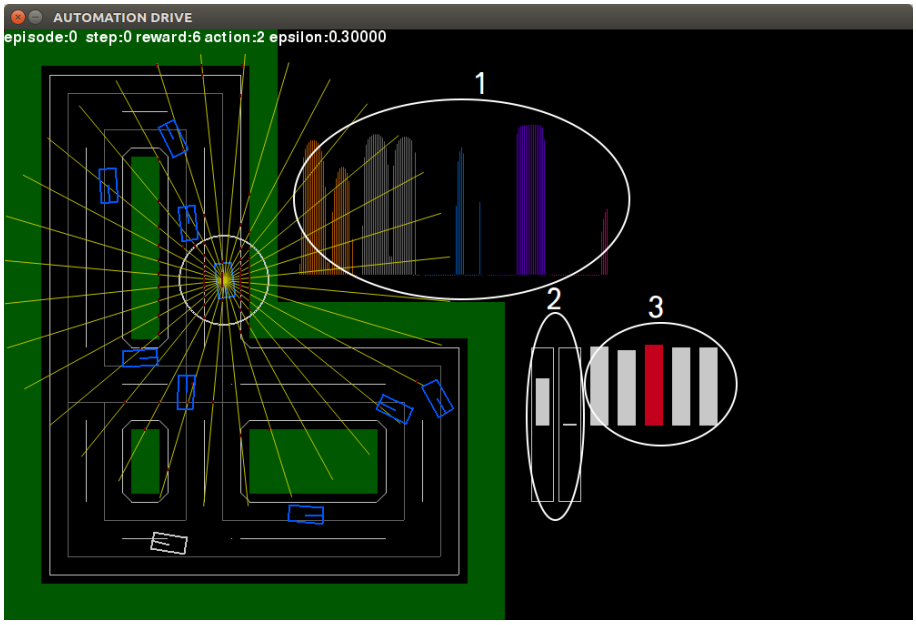

図 8: 作成したシミュレータ

線を使い二車線の道路を再現している. 外側の区画線の先は現実では歩道やガードレールがあるが, このシミュレータでは壁として定義し, その先には行けないようになっている. 道路は十字路,T 字路, カーブを作り, 車はそこらを右折, 左折, 直進, 右カーブ, 左カーブをしなければならない様にしている. このシミュレータが強化学習でいう「環境」に位置する. 車の取れる「行動」はアクセル, ブレーキ, バック, 右にハンドルを切る, 左にハンドルを切るの 5 通りで, スピードとハンドルの加速度を変化させることで車を移動させることができる. スピード, ハンドル共に変化できる値は-5 から 5 の範囲としている. また, 車の中心からは 32 方向に距離センサが出ており, 壁, 車, 区画線, 道路までの距離を計測している. それらの距離と車のアクセルとブレーキの加速度と選択された行動が「状況」となる. 正確には壁, 直近 4 ステップの他車, 区画線, 今進む道路, 次進む道路までの距離を 32 方向なので 256 次元と, 直近 10 ステップのスピードとハンドルの加速度と選択された行動の 30 次元を合わせて 286 次元のデータを「状況」としている. つまり入力が 286 次元, 出力が 5 次元の DQN を作る. 他車だけ直近 4 ステップの情報にしているのは, 他と比べて動くので時系列データが必要だと考えたためである. 一つだけ円で囲まれた車の中心から線が出ているがこれは, 距離センサを視覚化したものとなって

いる. センサが取得したデータをシミュレーターの右の 1 の円の部分に表示している.2 の円の部分は左から今のスピードとハンドルの値を表示している.3 の円の部分は学習させた DQN の出力値を表示していて, 左からアクセル, 右ハンドル, 左ハンドル, ブレーキ, バックの出力値としている. 出力値の 1 番大きな値だけ色を変えており,DQN がどの行動を最適としているのかを分かるようにしている. ソースコード 1 にシステムのメインループを示す.

ソースコード 1: メインループ

```
def main():
  agent = Agent(num_actions=5)
  while(1):
    if mode == TRAIN:
      stage_draw(carlist,carpoint,wallpoint1,wallpoint2,wallpoint3,
        wallpoint4,wallpoint5,routepoint,linepoint,mode)
        for i in range(0,len(carlist)):
          reward = 0
          inside = 0
          agent.get_q_values(state[i])
          state_draw(i,carlist,state,agent.adas_values2[0],
            agent.adas_max_action)
          before_state = np.array(state[i])
          action = agent.get_action(i,state[i])
          inside,reward,terminal =
            move(i,action,reward,car,wallpoint1,wallpoint2,wallpoint3,
            wallpoint6,route,routepoint,carlist,carpoint,linepoint,
            state,agent.episode0,agent.t0,agent.epsilon,
            agent.adas_values2[0],agent.adas_max_action)
          sensor(i,action,reward,car,carlist,wallpoint1,wallpoint2,
            wallpoint3,wallpoint6,carpoint,routepoint,state,inside,TRAIN)
          agent.run(i,before_state,action,reward,np.array(state[i]),terminal)
```

stage_draw() で壁, 道路, 区画線の再描写.state_draw() で今の状況の表示と DQN の出力値を表示している. そして今の状況を before_state に代入しておく.DQN の行動選択として agent.get_action() で行動を ε-greedy 法によって決定する. 行動が決定したら move() によって車が動作し報酬を得て,sensor() で次の状態を取得する.ADAS 部の処理として学習フェーズでは行動前の状況 ($Q(s_t, a)$), 行動 (a), 報酬 (r), 次の状況 ($Q(s_{t+1}, a)$) が揃ったので agent.run() に飛び学習をさせる. これを車の数だけ繰り返す. 動作フェーズでは agent.get_action() で得た行動を move() に渡し動作を行い,sensor() で次の状態を取得するのを車の数だけ繰り返す

シミュレータの動作部分である stage_draw() や state_draw(),move(),sensor() などはすべて自分で制作したオリジナルの部分になる. まず,stage_draw() で壁や道路, 区画線

の座標が定義された定数を引数として関数に渡し, 定義された座標に従い描写を行う. 次に state_draw() は図 8 の 1,2,3 の円の部分の描写になる. 引数で車の情報やセンサ情報,DQN の出力値を関数に渡している. 次に move() で車の移動, 衝突判定を行う. 引数で選択された行動や壁などの座標の定数, 車の情報,DQN の行動数や今の乱数や出力値を関数に渡している. 選択された行動によって車の速度やハンドルの角度を変化させ移動させる. 以下でシミュレーションの動作部分の説明をする.

ソースコード 2: 速度とハンドルの角度の変化と座標移動

```
1  move():
2    (w,h) = (35,15)
3    (dx,dy,deg,flg,cs,speed,handle) = carlist[num]
4
5    if action == 0:
6      if speed < 5:
7        speed += 1
8    if action == 1:
9      if handle < 5:
10       handle += 1
11   if action == 2:
12     if handle > -5:
13       handle -= 1
14   if action == 3:
15     if speed > 1:
16       speed -= 2
17     if speed < -1:
18       speed += 2
19     if -1 <= speed <= 1:
20       speed = 0
21   if action == 4:
22     if speed > -5:
23       speed -= 1
24
25   if speed >= 0:
26     deg += handle
27   elif speed < 0:
28     deg -= handle
29
30   if deg >= 360:
31     deg -= 360
32   elif deg < 0:
33     deg += 360
34
35   rad = math.radians(deg)
36   dx += math.cos(rad)*speed
37   dy += -1*math.sin(rad)*speed
```

ソースコード 2 で行動によって速度とハンドルの変化と座標移動を示す. 変数 w と h は車の大きさを定義している.carlist には車の座標や速度などの情報を保存してあるの

でそこから値を取り出しておく. アクセル, バック, ハンドル操作の行動により速度とハンドルの角度を 1 づつ変化させる. ブレーキが選択された時には速度を 0 に 2 づつ近づける. 速度とハンドルの角度によって車の中心座標を移動させる.

ソースコード 3: 車の描写

```
move():
  carpoint.pop(num)
  carpoint.insert(num,[])
  for i in range(0,4):
    if i == 0:
      w2 = float(w)/2
      h2 = float(h)/2
      hyp = math.sqrt(w2*w2 + h2*h2)
      hyptheta = math.atan2(h,w) * 180 / math.pi
      rad = math.radians(deg+hyptheta)
      dx = math.cos(rad)*hyp + dx
      dy = -1*math.sin(rad)*hyp + dy
      (tx,ty) = (dx,dy)
    if i == 1:
      rad = math.radians(deg+180)
      tx = math.cos(rad)*w + dx
      ty = -1*math.sin(rad)*w + dy
    if i == 2:
      rad = math.radians(deg+270)
      tx = math.cos(rad)*h + tx
      ty = -1*math.sin(rad)*h + ty
    if i == 3:
      rad = math.radians(deg+270)
      tx = math.cos(rad)*h + dx
      ty = -1*math.sin(rad)*h + dy
    pointlist.pop(i)
    pointlist.insert(i,(tx,ty))
    carpoint[num].append(int(tx))
    carpoint[num].append(int(ty))
  if flg == 0:
    color = white
  elif flg == 1:
    color = red
  elif flg == 2:
    color = blue
  car_main = pygame.draw.lines(screen,color,True,pointlist,2)
  (tx,ty) = car_main.center
  (tx2,ty2) = crosspoint(cx1,cy1,cx2,cy2)
  pygame.draw.line(screen,color,(tx,ty),(tx2,ty2),2)
```

ソースコード 3 で車の描写のための処理をします. 車の描写には隅の 4 点の座標が必要なのでそれの計算を車の大きさと角度に合うように行う.4 点の座標の計算が完了したら pygame の関数を使い描写を行う. 車の状態に合わせて描写の色を変化させている. 衝突時は赤色, 正しい道路を走行時は青色, それ以外は白色で描写している.

ソースコード 4: 衝突判定

```
move():
  for i in range(0,len(wallpoint1)-1):
    (x1,y1) = wallpoint1[i]
    (x2,y2) = wallpoint1[i+1]
    k=0
    for j in range(0,4):
      (x3,y3) = (carpoint[num][k],carpoint[num][k+1])
      if j == 3:
        (x4,y4) = (carpoint[num][0],carpoint[num][1])
      else:
        k+=2
        (x4,y4) = (carpoint[num][k],carpoint[num][k+1])
      if mete(x1,y1,x2,y2,x3,y3,x4,y4) != (0,0):
        flg = 1
        coll = 100 + i
        reward -= WALL_PUNISH
        reward -= 3*(abs(speed)+2)
        inside += 0x20
        break
    if flg == 1:
      break
```

ソースコード 4 で衝突判定のための処理をします. 車と壁が衝突しているかを判定している. 壁の座標は wallpoint1,2,3,6 に定義されており, 外側の大きな壁と内側の小さな壁 3 つの合計 4 つ定数が定義してある. まず,wallpoint1 から座標を 2 個取り出し, その 2 点を結んだ線と車を描写した 4 点を結んだ線が交わるかを調べる. それを座標が定義されている数だけ繰り返す. 壁の座標 2 点と車の座標 2 点を mete() 関数に引数として渡す. この関数は引数の前半 2 点を結んだ線と後半 2 点を結んだ線が交わるとき返り値として交点の座標を返し, 交わらない時は (0,0) を返すように作った自作の関数になる. なので, 返り値が (0,0) 以外の時, 衝突が発生しているのでマイナスの報酬を与え終了する. これを壁の座標が定義されている定数の数 4 回分を最大で行う. 他車の衝突と区画線と車体の交わりの判定も同じような流れで行っている. 座標移動によって報酬を与え, 次に sensor() で DQN の入力値となる状況の取得を行う. 引数として車の情報や壁などの座標情報, 今の状況を引数に渡している.

ソースコード 5: センサ情報の取得

```
sensor():
  w = []
  l = []
  c = []
  r1 = []
  r2 = []
  r = deg
```

```
8     for i in range(0,32):
9       (x1,y1) = car[num].center
10      (x2,y2) = (x1+(int)(math.cos(math.radians(-1*r))*250),
11          y1+(int)(math.sin(math.radians(-1*r))*250))
12
13      (walldis,x,y) = wallmete(x1,y1,x2,y2,wallpoint1,wallpoint2,
14        wallpoint3,wallpoint6)
15      if num == 0: pygame.draw.circle(screen,red2,(int(x),int(y)),1)
16      if walldis < 250:
17        w.append(walldis)
18      else:
19        walldis = 250
20        w.append(0)
21
22      (linedis,x,y) = linemete(num,x1,y1,x2,y2,linepoint)
23      if num == 0: pygame.draw.circle(screen,red2,(int(x),int(y)),1)
24      if linedis < walldis:
25        l.append(linedis)
26      else:
27        l.append(0)
28
29      (cardis,x,y) = carmete(num,x1,y1,x2,y2,carpoint)
30      if num == 0: pygame.draw.circle(screen,red2,(int(x),int(y)),1)
31      if cardis < walldis:
32        c.append(cardis)
33      else:
34        c.append(0)
35
36      (x,y) = mete(x1,y1,x2,y2,rx1,ry1,rx2,ry2)
37      if num == 0: pygame.draw.circle(screen,red2,(int(x),int(y)),1)
38      if x != 0 and y != 0:
39        dis = distance(x,y,x1,y1)
40        r1.append(dis)
41      else:
42        r1.append(0)
43
44      (x,y) = mete(x1,y1,x2,y2,rx3,ry3,rx4,ry4)
45      if num == 0: pygame.draw.circle(screen,red2,(int(x),int(y)),1)
46      if x != 0 and y != 0:
47        dis = distance(x,y,x1,y1)
48        r2.append(dis)
49      else:
50        r2.append(0)
```

ソースコード5で車から出ているセンサ情報の取得の処理を示す. センサ情報の取得は基本的に衝突判定と同じで, 車の中心から出ているセンサの線が壁や他車に交わっているかを判定し交点までの距離を計算している. まずセンサで計測するのは壁, 区画線, 他車, 道路, 次の道路の5種類あるのでその値を格納する変数を定義する. センサは32方向に出ているのでfor文にて32回繰り返す. 車の中心の座標の取得とセンサの先の座標

を計算する.wallmete() 関数にセンサの 2 点の座標と壁の座標情報を渡し, この関数の中で一番近い交点の計算を行っている. 交点があった時には返り値として交点までの距離と座標を返してくる. 車の 1 つだけ検知した交点をシミュレータ上に赤点で描写している. センサの計測範囲は 250 までとしているので,250 以下のとき計測値を変数に代入しておく. 区画線, 他車に対しても同様に行うが, 壁を検知している場合はその値以上の値は記録しないようにしている. つまり, センサは壁を超えて検知できない. 道路情報に関しては現実でもセンサ以外の方法で取得しているので 250 以下なら検知できるようにしている.

また, このシミュレータの道路を走行できるが, 衝突は完全に回避出来ないような走行方法を実装した. 道路にいて正しい方向を向いている時にアクセルを選択し, 方向がズレればハンドル操作により正しい方向に戻す. もし道路からはみ出していた場合は, ハンドル操作により道路の方に方向転換をし, アクセルによって直進する. 車の前のセンサが他車を近くで検知しているときは速度を落とすためにブレーキを選択する. この走行方法を用いて実験を行う.

4.4 ADAS 部の Deep Q-network プログラム

Deep Q-network 部分の実装には参考文献の [1] の「DQN を TensorFlow と OpenAI Gym で実装する」のプログラムをもとに作成したシミュレータで動くように改良したもの使用した.DQN の実装には python のライブラリである tensorflow と keras を使用する. 学習させる処理は全て Agent クラスで行っており,Agent クラスの「def __init__(self, num_actions)」では初期化の処理を行っている. 初期化の処理は参考文献 [1] をそのまま使用した. まず, 学習で使うデータを格納するメモリーの初期化を deque() に行うことにより挿入と削除を高速で行える保存領域を作る. このメモリーには ($Q(s_t, a)$,a,s,$Q(s_{t+1}, a)$) のセットを格納して学習の時に取り出す. 次に, ネットワークのモデルを作る.Agent クラスの「def build_network(self)」で Q Network(学習させる network) と Target Network(学習の時に使う教師用の network) の構築を行う. ネットワークの構造を入力 286 次元から出力 5 次元への 6 層 (286-400-200-100-50-5) のネットワークを構築. 活性化関数はすべて ReLU を採用した.Target Network の更新は定期的に Q Network の重みをコピーし更新する. 最適化のための処理の構築を Agent クラスの「def build_training_op(self,

q_network_weights)」で行う. ここの処理は参考文献 [1] をそのまま利用している. ここで誤差の計算と最適化を行う時に, 誤差を-1 から 1 の範囲に絞っといる. それにより学習の安定性を向上させることが出来る. ネットワークを最適化するために勾配降下法を使用するがそのアルゴリズムとして Adam というアルゴリズムを使用する.Adam のパラメータは $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ で最適化を行う. 次に,Agent クラスの「def get_action(self, num, state)」により行動選択を行う. 参考文献 [1] では 4 回に 1 回行動選択を行っていたが, ここでは毎回行う. 行動選択は ε -greedy 法を用いる. なので, 確率 ε でランダムに行動選択, それ以外で Q 値が最大の行動を選択する. 最初の数ステップは完全にランダムに行動させデータを集める. 確率 ε は 0.5 を初期値として, そこから 0.2 まで 100 万回行動する間で線形に減少させる.100 万回以降は 0.2 に固定させる. 行動を行ったあと, 次の状況を獲得したら Agent クラスの「def run(self, num, before_state, action, reward, state, terminal)」に飛び学習が行われる.

ソースコード 6: DQN の学習

```
1  class Agent():
2    def run(self, num, before_state, action, reward, state, terminal):
3
4      self.replay_memory.append((before_state, action, reward, state, terminal))
5
6      if len(self.replay_memory) > NUM_REPLAY_MEMORY:
7        self.replay_memory.popleft()
8
9      if self.t >= INITIAL_REPLAY_SIZE:
10       if self.t % TRAIN_INTERVAL == 0:
11         self.train_network()
12
13       if self.t % TARGET_UPDATE_INTERVAL == 0:
14         self.sess.run(self.update_target_network)
15
16       if self.t % SAVE_INTERVAL == 0:
17         save_path = self.saver.save
18           (self.sess, SAVE_NETWORK_PATH + '/' + ENV_NAME, global_step=(self.t))
19         print('Successfully saved: ' + save_path)
20     self.t += 1
```

ソースコード 6 に DQN の学習部分の処理の流れを示す. 参考文献 [1] では報酬を 1 か-1 に固定していたが, それでは報酬の重み付けができないため無くしている. また, 学習の頻度も変更している. メモリーに $(Q(s_t, a)$,a,s,$Q(s_{t+1}, a))$ のセットを保存し, 一定数を超えたら古い順に削除していく.Q Network の学習は毎回行っていては動作が重くなるだけなので 5 回行動ごとに Agent クラスの「def train_network(self)」で行う.Target

network は 5 千行動ことに更新を行う. また, ネットワークの重みは一定数の行動ごとに保存を行う.

ソースコード 7: ミニバッチ学習

```
class Agent():
  def train_network(self):
    before_state_batch = []
    action_batch = []
    reward_batch = []
    state_batch = []
    terminal_batch = []
    y_batch = []

    minibatch = random.sample(self.replay_memory, BATCH_SIZE)
    for data in minibatch:
      before_state_batch.append(data[0])
      action_batch.append(data[1])
      reward_batch.append(data[2])
      state_batch.append(data[3])
      terminal_batch.append(data[4])

    terminal_batch = np.array(terminal_batch) + 0

    a = np.argmax(self.q_values.eval(
      feed_dict={self.s: np.float32(np.array(state_batch)/250.0)}), axis=1)
    target_q_values_batch = self.target_q_values.eval(
      feed_dict={self.st: np.float32(np.array(state_batch)/250.0)})
    targets = np.zeros((BATCH_SIZE)).astype(np.float32)
    for i in range(0,BATCH_SIZE):
      targets[i] = target_q_values_batch[i][a[i]]
    y_batch = reward_batch + (1 - terminal_batch) * GAMMA * targets

    loss, _ = self.sess.run([self.loss, self.grad_update], feed_dict={
      self.s: np.float32(np.array(before_state_batch)/250.0),
      self.a: action_batch,
      self.y: y_batch
    })

    self.total_loss += loss
```

ソースコード 7 に DQN のミニバッチ学習の処理を示す. ここではメモリーに保存されていた $(Q(s_t, a)$,a,s,$Q(s_{t+1}, a))$ のセットをランダムに 32 個サンプリングし, それを使いミニバッチ学習を行っている. 学習には参考文献 [1] から変更しは DoubleDeepQLearnig を使用した. これにより行動を選択するネットワークモデルと行動を評価するネットワークモデルを分けることで, 行動の過大評価を削減している.

5　第5章 実験

まず, このシミュレータを使用して学習を行うことができるかの実験を行う. 学習時に得た報酬の値の合計や, 事故状態の回数を計測し, 学習経過によって得る報酬が増え, 事故状態の回数が減ることを確認する. この時の事故状態は, 壁や他車に衝突している時と, 車線から外れた時としている. また, 学習させた DQN をある程度ランダム性を持たせ動かす. その時に正しく走行している時の DQN の出力値と, 事故状態の時の DQN の出力値を記録する. その出力値の結果から危険な状態を判別できるかを検証する. そして, 危険な状態と判別する値を決めて自身で設計した走行方法で動作させる. この時 DQN によって危険と判断した時の回数を記録し, 危険と予測され事故状態の回数を記録する. 次に, 危険と判断した時 DQN に任せた時に制御を任せて動作させ事故状態の回数を記録し DQN によって精度がどう変わるかを実験で検証する.

5.1　シミュレータを使用した学習の実験

シミュレータを使用して DQN が ADAS を学習できるかの学習実験を行う. 学習させるときの設定した報酬は以下のようになっている. シミュレータ上で 10 個の車が動作

プラスの報酬	今進む道路に対して並行時の角度を0度とした時, 角度が30度以内で前進している
	道路から外れている時に, 道路に戻る
	次に進む道路に進む
マイナスの報酬	壁に衝突
	他車に衝突
	他車に近すぎる時
	区画線と車体が交わる
	車体が半分以上区画線から出る
	今進む道路を逆走する

図 9: 報酬の設定

し, すべての車は同じ DQN のネットワークモデルを共有して分散学習を行う. 全体で行動数が 100 万回に達して時に学習を終了させる. ランダムで行動させる確率 ε は 0.5 を初期値として, そこから 0.2 まで 100 万回行動する間で線形に減少させる. また全体で 2 万回行動ごとに, ランダム性をなくして全体で 1 万回行動させる. この時に得た報酬の合計と事故状態を計測した結果が図 10 と図 11 になった. 図 10 の縦軸は事故状態の回数の合計となっていて, 図 11 の縦軸は得た報酬の合計となっている. 横軸は図 10, 図 11

共に全体の行動回数になっている. なので, 行動回数が多くなるほど学習が進んでいることになる. 図 10 は学習が進むに連れて事故回数が減少している. つまり, 事故となる状況を学習し事故を回避している. また, 図 11 は学習が進むに連れて得ている報酬が増えている. 道路を走行しないとプラスの報酬は貰えないので, 道路をより走行できる様になっていることが分かる.

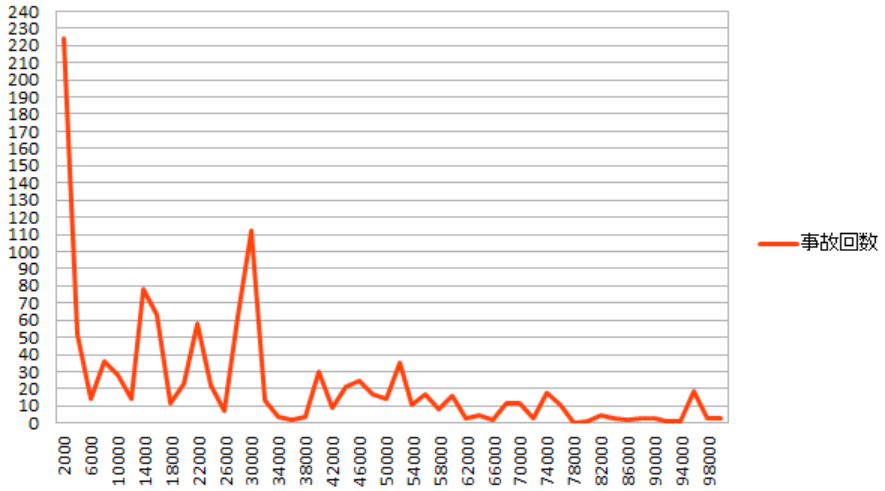

図 10: 学習経過による事故回数の推移

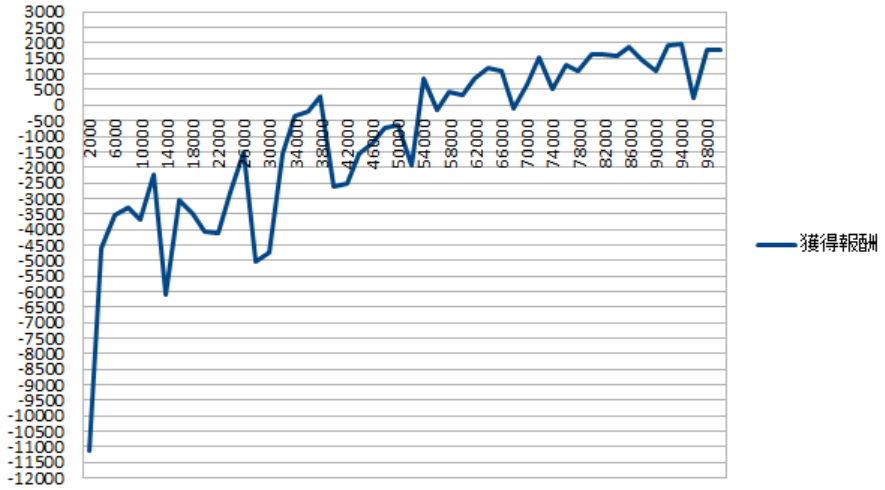

図 11: 学習経過による得る報酬の推移

次に, 危険の判別器として使用できるかの検証を行う. 直近 3 ステップで車が道路を正

しい方向に直進している時を正しく走行しているとし, 正しく走行している時と, 壁や他車に衝突している時と車線から外れている時の事故状態の DQN の出力値を記録した.DQN の出力値は 5 次元あるので,2 パターンの 5 次元の出力値が表 1 と図 12 で表とグラフになっている. 表 1 の表は縦が上からアクセル, 右ハンドル, 左ハンドル, ブレーキ, バックで, 横は正しく走行時と事故状態の出力値の最大と最小と平均となっている. その値のグラフが図 13 になる. 太い縦線が正しく走行時の出力値の範囲で, 縦線に横線が出ている値が平均値になっている.　表 1 を見ると衝突と車線から外れた時の最大値

表 1: DQN の出力値の表

		最大	最小	平均
正しく走行	accelerator	122.058	-72.0315	89.475321876
	right handle	121.843	-39.4181	88.111344946
	left handle	121.326	-22.5893	89.046008865
	brake	119.942	-22.2366	87.13181283
	back	120.179	-22.6086	87.97223787
事故状態	accelerator	23.4599	-257.557	-80.11709339
	right handle	21.7167	-258.762	-80.03703442
	left handle	20.5434	-253.119	-77.04797931
	brake	19.6529	-251.916	-77.40125826
	back	19.2114	-254.518	-78.09806331

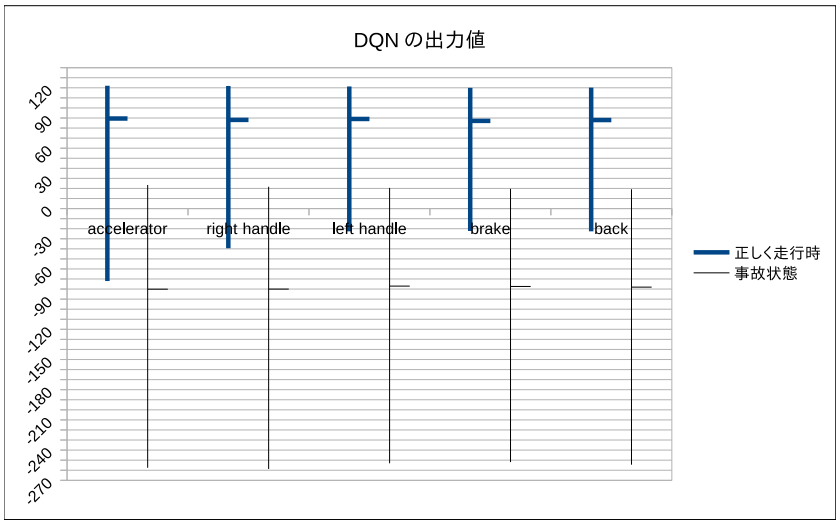

図 12: DQN の出力値のグラフ

は約 20 付近となっている. 正しく走行時の平均は約 90 付近となっている. この結果から

正しく走行時の平均である 90 を下回り,20 に近い時に危険な状態あるといえるだろう. なので, 今回は出力値のどれかが 60 を下回った時に危険な状態とし次の実験を行った.

5.2 ADAS の実証実験

まず，自身で設計した走行方法で DQN に危険か判断させつつ動作させる. この時車 1 台が千回行動ごとに危険と判断した時の回数とその時に事故状態の回数を記録する. 車は全部で 10 台動作するので, 全体で 1 万回行動した時の 1 台あたりの事故回数を記録することになる. 記録が終了したらカウントをリセットし車を初期位置に戻し, 再度動作させる. これを 35 回行う. 次に, 危険と判断した時に DQN に制御を任せて事故回数を千回ごとに 1 台あたりの事故回数を記録した. 同じように 35 回記録を行った. 記録した表が表 2 でグラフが図 13 である. 縦軸が 1 台あたり事故回数で横軸が行動回数の合計になっている.

表 2: 事故率

動作回数	1000	2000	3000	4000	5000	6000	7000	
DQN無し	76.54%	77.57%	80.73%	70.49%	80.20%	65.70%	51.52%	
DQN有り	0.26%	8.50%	2.30%	2.96%	4.48%	6.35%	3.28%	
動作回数	8000	9000	10000	11000	12000	13000	14000	
DQN無し	44.10%	75.68%	54.64%	70.58%	66.95%	79.57%	88.82%	
DQN有り	1.31%	11.60%	2.95%	1.46%	0.84%	1.22%	0.19%	
動作回数	15000	16000	17000	18000	19000	20000	21000	
DQN無し	61.63%	61.18%	70.17%	54.84%	58.52%	75.18%	74.51%	
DQN有り	6.30%	4.73%	7.90%	1.97%	2.32%	2.44%	7.22%	
動作回数	22000	23000	24000	25000	26000	27000	28000	
DQN無し	80.18%	83.17%	79.86%	72.00%	84.86%	80.94%	74.38%	
DQN有り	1.88%	0.78%	1.54%	3.09%	2.77%	9.11%	2.67%	
動作回数	29000	30000	31000	32000	33000	34000	35000	**平均**
DQN無し	68.37%	72.67%	68.01%	64.75%	73.65%	64.21%	67.85%	**70.69%**
DQN有り	1.20%	7.35%	7.09%	1.02%	0.47%	0.68%	6.83%	**3.63%**

図 13 で 1 番大きな値のグラフが DQN が危険と判断した回数,2 番目に大きな値が DQN に制御を任せなかった時の事故回数,1 番値が小さいグラフが危険と判断した時に DQN に制御を任せた時の事故回数になっている. 結果として事故率は 70.69 %から 3.63 %まで大幅に減らすことが出来た.

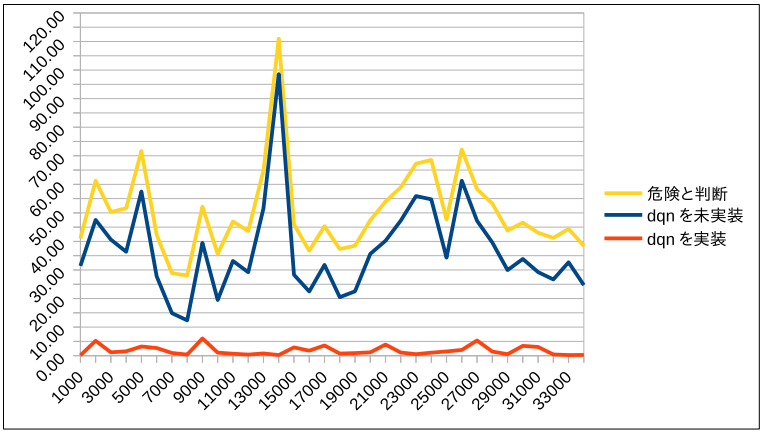

図 13: 衝突と車線外れの回数

5.3 考察

結果から DQN を使うことで大幅に危険を回避することが出来た. 今回深層強化学習を危険判別に使用したが, 出力の結果が正しく走行している時と, 衝突・車線から外れている時とで平均としては分離しているが, 被っている値があり判別器として性能としては他の方法を検討すべきであったと考えます. 制御に関しては完璧に回避することが出来なかったものの, 約 96.37 %も危険を回避することが出来た. 報酬の設定を更に増やしプラスの報酬を得る条件を厳しくすることで, さらに良い結果を出すことができるのではないかと考えます.

6　第6章 まとめ

今回の実験でPC上で動くシミュレータの作成を行い,DQNの学習を行うことが出来た. また,DQNの学習により大幅に事故率を減らすことが出来た. この実験の目的としていた深層強化学習を用いたADAS実現という点では, さらに精度を上げることができればシミュレータ上で完璧に実現できたといえると考える. 単純に精度をあげようと思えば, 報酬の設定をより細かく設定すればよいが, それでは技術者が危険な状況を考え設計しているのと変わらないので, 強化学習を使うメリットがなくなる. 今後の課題としてはより精度を上げれるような深層強化学習のアルゴリズムが必要になる. また, 深層強化学習はランダム性を持って行動するためすべてを網羅的に学習するが, その網羅的に学習したと確証付けれるものがないため, その検証も今後の課題となる.

謝辞

本研究を進めるにあたり, 様々なご指導を頂きました三好力教授に深く感謝いたします. また, 多忙の中研究の過程で様々な助言を頂きました同研究室の皆様に深く感謝いたします.

参考文献

[1] 「DQN を Keras と TensorFlow と OpenAI Gym で実装する」,
http:\\elix-tech.github.io/ja/2016/06/29/dqn-ja.html

[2] 「分散深層強化学習でロボット制御」,
https:\\research.preferred.jp/2015/06/distributed-deep-reinforcement-learning/

[3] Deep Reinforcement Learning with Double Q-learning (Double DQN),
http://wanwannodao.hatenablog.com/entry/2017/03/13/182340

[4] ADAS（先進運転支援システム）
https:\\automotive.ten-navi.com/dictionary/792/

[5] Pygame でゲームプログラミング - ゲーム制作速報
http:\\gamepro.blog.jp/python/pygame/introduction

[6] V. Mnih et al., "Playing atari with deep reinforcement learning"

A 付録

chain_keras.py

```
# coding:utf-8
import os
import sys
import time
import math
import random
import pygame
from pygame.locals import *
import numpy as np
import tensorflow as tf
# from skimage.color import rgb2gray
# from skimage.transform import resize
from collections import deque
# from keras.utils import plot_model
from keras.models import Sequential
from keras.layers import InputLayer
from keras.layers.core import Dense, Dropout,
  Flatten
from draw import init, reset, stage_draw, car_set,
  section_act, move, sensor, state_draw, get_move_act
from mete import wallmete, carmete, mete, distance,
  distance_l, angular, crosspoint

KERAS_BACKEND = 'tensorflow'

ENV_NAME = 'AutoDrive'  # Environment name
EXPLORATION_STEPS = 1000000  # Number of steps
  over which the initial value of epsilon is
  linearly annealed to its final value
GAMMA = 0.95  # Discount factor
INITIAL_EPSILON = 0.5  # Initial value of
  epsilon in epsilon-greedy
FINAL_EPSILON = 0.1  # Final value of epsilon in
  epsilon-greedy
INITIAL_REPLAY_SIZE = 1000  # Number of steps to
  populate the replay memory before training
  starts
NUM_REPLAY_MEMORY = 200000  # Number of replay
  memory the agent uses for training
BATCH_SIZE = 32  # Mini batch size
TARGET_UPDATE_INTERVAL = 5000  # The frequency
  with which the target network is updated
ACTION_INTERVAL = 1  # The agent sees only every
  4th input
TRAIN_INTERVAL = 5  # The agent selects 4
  actions between successive updates
SAVE_INTERVAL = 300000  # The frequency with
  which the network is saved
NO_OP_STEPS = 30  # Maximum number of "do
  nothing" actions to be performed by the agent
  at the start of an episode
LOAD_NETWORK = False
TRAIN = True
TEST = False
SAVE_NETWORK_PATH = 'saved_networks/' + ENV_NAME
ADAS_SAVE_NETWORK_PATH = 'saved_networks_adas/'
  + ENV_NAME
SAVE_SUMMARY_PATH = 'summary/' + ENV_NAME
CAR_NUM = 19

class Agent():
 def __init__(self, num_actions):
  self.num_actions = num_actions
  self.epsilon = 0
  self.epsilon_step = (INITIAL_EPSILON -
   FINAL_EPSILON) / EXPLORATION_STEPS
  self.adas_epsilon = 0#INITIAL_EPSILON
  self.t = 0
  self.t0 = 0
  self.t1 = 0
  self.adas_t = 0
  self.adas_t0 = 0
  self.repeated_action = 0
  self.adas_episode = 0

  # Parameters used for summary
  self.total_reward = 0
  self.total_q_max = 0
  self.total_loss = 0
  self.duration = 0
  self.episode = 0
  self.episode0 = 0
  self.success = 0
  self.max_action = 0
  self.adas_max_action = 0
  self.q_values2 = np.zeros((1,5)).astype(np.
   float32)
  self.adas_values2 = np.zeros((1,5)).astype(np.
   float32)

  # Create replay memory
  self.replay_memory = deque()
  self.adas_replay_memory = deque()

  # Create q network
  self.s, self.q_values, self.q_network = self.
   build_network()
  q_network_weights = self.q_network.
   trainable_weights
  self.adas_s, self.adas_values, self.
   adas_network = self.adas_build_network()
  adas_network_weights = self.adas_network.
   trainable_weights

  # Create target network
  self.st, self.target_q_values, self.
   target_network = self.build_network()
  target_network_weights = self.target_network.
   trainable_weights
  self.adas_st, self.target_adas_values, self.
   adas_target_network = self.adas_build_network
   ()
  adas_target_network_weights = self.
   adas_target_network.trainable_weights

  # Define target network update operation
  self.update_target_network = [
   target_network_weights[i].assign(
   q_network_weights[i]) for i in xrange(len(
   target_network_weights))]
  self.adas_update_target_network = [
   adas_target_network_weights[i].assign(
   adas_network_weights[i]) for i in xrange(len(
   adas_target_network_weights))]
  #self.update_adas_network = [
   adas_network_weights[i].assign(
   q_network_weights[i]) for i in xrange(len(
   adas_network_weights))]

  # Define loss and gradient update operation
  self.a, self.y, self.loss, self.grad_update =
   self.build_training_op(q_network_weights)
  self.adas_a, self.adas_y, self.adas_loss, self
   .adas_grad_update = self.
   adas_build_training_op(adas_network_weights)

  # plot_model(self.q_network, to_file='model.
   png')

  self.sess = tf.InteractiveSession()
  self.saver = tf.train.Saver(q_network_weights)
  self.summary_placeholders, self.update_ops,
   self.summary_op = self.setup_summary()
  self.summary_writer = tf.summary.FileWriter(
   SAVE_SUMMARY_PATH, self.sess.graph)
  self.adas_saver = tf.train.Saver(
   adas_network_weights)

  if not os.path.exists(SAVE_NETWORK_PATH):
   os.makedirs(SAVE_NETWORK_PATH)
  if not os.path.exists(ADAS_SAVE_NETWORK_PATH):
   os.makedirs(ADAS_SAVE_NETWORK_PATH)

  self.sess.run(tf.global_variables_initializer
   ())

  # Load network
  if LOAD_NETWORK:
   self.load_network()
  if LOAD_NETWORK:
   self.adas_load_network()

  self.update_adas_network = [
   adas_network_weights[i].assign(
   q_network_weights[i]) for i in xrange(len(
   adas_network_weights))]
  self.sess.run(self.update_adas_network)

  # Initialize target network
  self.sess.run(self.update_target_network)
  self.sess.run(self.adas_update_target_network)

 def adas_build_network(self):
```

```
  model = Sequential()
  model.add(InputLayer(input_shape=(None,286)))
  model.add(Dense(400, activation='relu'))
  #model.add(Dense(300, activation='relu'))
  model.add(Dense(200, activation='relu'))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(50, activation='relu'))
  model.add(Dense(self.num_actions))

  s = tf.placeholder(tf.float32, [None, 286])
  q_values = model(s)

  return s, q_values, model

def adas_build_training_op(self,
 q_network_weights):
 a = tf.placeholder(tf.int64, [None])
 y = tf.placeholder(tf.float32, [None])

 # Convert action to one hot vector
 a_one_hot = tf.one_hot(a, self.num_actions,
  1.0, 0.0)
 q_value = tf.reduce_sum(tf.multiply(self.
  adas_values, a_one_hot), reduction_indices=1)

 # Clip the error, the loss is quadratic when
  the error is in (-1, 1), and linear outside
  of that region
 error = tf.abs(y - q_value)
 quadratic_part = tf.clip_by_value(error, 0.0,
  1.0)
 linear_part = error - quadratic_part
 loss = tf.reduce_mean(0.5 * tf.square(
  quadratic_part) + linear_part)

 #optimizer = tf.train.RMSPropOptimizer(
  LEARNING_RATE, momentum=MOMENTUM, epsilon=
  MIN_GRAD)
 optimizer = tf.train.AdamOptimizer(
  learning_rate=0.001,beta1=0.9,beta2=0.999,
  epsilon=1e-08,use_locking=False,name='Adam')
 grad_update = optimizer.minimize(loss,
  var_list=q_network_weights)
 return a, y, loss, grad_update

def adas_get_action(self, num, state):

 #self.sess.run(self.update_adas_network)

 if self.adas_t % 1 == 0:
  if self.adas_epsilon >= random.random():
   action = random.randrange(0,self.num_actions
    )
  else:
   action = np.argmax(self.adas_values.eval(
    feed_dict={self.adas_s: [np.float32(state
    /250.0)]}))

 if num == 0:
  self.adas_max_action = np.argmax(self.
   adas_values.eval(feed_dict={self.adas_s: [np
   .float32(state/250.0)]}))
  self.adas_values2 = self.adas_values.eval(
   feed_dict={self.adas_s: [np.float32(state
   /250.0)]})

 # # Anneal epsilon linearly over time
 if self.adas_epsilon > FINAL_EPSILON:
  self.adas_epsilon -= self.epsilon_step

 return action

def adas_run(self, num, before_state, action,
 reward, state, terminal):

 self.adas_replay_memory.append((before_state,
  action, reward, state, terminal))

 if len(self.adas_replay_memory) >
  NUM_REPLAY_MEMORY:
  self.adas_replay_memory.popleft()

 if terminal == 1:
  self.adas_episode += 1

 if self.adas_t >= INITIAL_REPLAY_SIZE:
  # Train network
  if self.adas_t % TRAIN_INTERVAL == 0:
   self.adas_train_network()

  # Update target network
  if self.adas_t % TARGET_UPDATE_INTERVAL == 0:
   self.sess.run(self.
    adas_update_target_network)

  # Save network
  if self.adas_t % SAVE_INTERVAL == 0:
   save_path = self.adas_saver.save(self.sess,
    ADAS_SAVE_NETWORK_PATH + '/' + ENV_NAME,
    global_step=(self.adas_t))
   print('Successfully saved: ' + save_path)

 if num == 0:
  self.adas_t0 += 1

 self.adas_t += 1

def adas_train_network(self):
 before_state_batch = []
 action_batch = []
 reward_batch = []
 state_batch = []
 terminal_batch = []
 y_batch = []

 # Sample random minibatch of transition from
  replay memory
 minibatch = random.sample(self.
  adas_replay_memory, BATCH_SIZE)
 for data in minibatch:
  before_state_batch.append(data[0])
  action_batch.append(data[1])
  reward_batch.append(data[2])
  state_batch.append(data[3])
  terminal_batch.append(data[4])

 # Convert True to 1, False to 0
 terminal_batch = np.array(terminal_batch) + 0

 #double deep Q-learning
 a = np.argmax(self.adas_values.eval(feed_dict
  ={self.adas_s: np.float32(np.array(
  state_batch)/250.0)}), axis=1)
 target_adas_values_batch = self.
  target_adas_values.eval(feed_dict={self.
  adas_st: np.float32(np.array(state_batch
  )/250.0)})
 targets = np.zeros((BATCH_SIZE)).astype(np.
  float32)
 for i in range(0,BATCH_SIZE):
  targets[i] = target_adas_values_batch[i][a[i
   ]]
 y_batch = reward_batch + (1 - terminal_batch)
  * GAMMA * targets

 #deep Q-learning
 # target_q_values_batch = self.target_q_values
  .eval(feed_dict={self.st: np.float32(np.array
  (state_batch))})
 # y_batch = reward_batch + (1 - terminal_batch
  ) * GAMMA * np.max(target_q_values_batch,
  axis=1)
 #print np.max(target_q_values_batch, axis=1)

 loss, grad_update = self.sess.run([self.
  adas_loss, self.adas_grad_update], feed_dict
  ={
  self.adas_s: np.float32(np.array(
   before_state_batch)/250.0),
  self.adas_a: action_batch,
  self.adas_y: y_batch
 })

def adas_load_network(self):
 checkpoint = tf.train.get_checkpoint_state(
  ADAS_SAVE_NETWORK_PATH)
 if checkpoint and checkpoint.
  model_checkpoint_path:
  self.adas_saver.restore(self.sess, checkpoint
   .model_checkpoint_path)
  print('Successfully loaded: ' + checkpoint.
   model_checkpoint_path)
 else:
  print('Training new network...')

def build_network(self):
 model = Sequential()
 model.add(InputLayer(input_shape=(None,286)))
 model.add(Dense(400, activation='relu'))
 #model.add(Dense(300, activation='relu'))
 model.add(Dense(200, activation='relu'))
 model.add(Dense(100, activation='relu'))
 model.add(Dense(50, activation='relu'))
 model.add(Dense(self.num_actions))
```

```
  s = tf.placeholder(tf.float32, [None, 286])
  q_values = model(s)

  return s, q_values, model

def build_training_op(self, q_network_weights):
  a = tf.placeholder(tf.int64, [None])
  y = tf.placeholder(tf.float32, [None])

  # Convert action to one hot vector
  a_one_hot = tf.one_hot(a, self.num_actions,
    1.0, 0.0)
  q_value = tf.reduce_sum(tf.multiply(self.
    q_values, a_one_hot), reduction_indices=1)

  # Clip the error, the loss is quadratic when
    the error is in (-1, 1), and linear outside
    of that region
  error = tf.abs(y - q_value)
  quadratic_part = tf.clip_by_value(error, 0.0,
    1.0)
  linear_part = error - quadratic_part
  loss = tf.reduce_mean(0.5 * tf.square(
    quadratic_part) + linear_part)

  #optimizer = tf.train.RMSPropOptimizer(
    LEARNING_RATE, momentum=MOMENTUM, epsilon=
    MIN_GRAD)
  optimizer = tf.train.AdamOptimizer(
    learning_rate=0.001,beta1=0.9,beta2=0.999,
    epsilon=1e-08,use_locking=False,name='Adam')
  grad_update = optimizer.minimize(loss,
    var_list=q_network_weights)
  return a, y, loss, grad_update

def get_q_values(self, state):
  self.max_action = np.argmax(self.q_values.eval
    (feed_dict={self.s: [np.float32(state
    /250.0)]}))
  self.q_values2 =  self.q_values.eval(feed_dict
    ={self.s: [np.float32(state/250.0)]})
  self.adas_values2 =  self.adas_values.eval(
    feed_dict={self.adas_s: [np.float32(state
    /250.0)]})

def get_action(self, num, state):
  action = self.repeated_action

  if self.t % ACTION_INTERVAL == 0:
    if self.epsilon >= random.random() or self.t
      < INITIAL_REPLAY_SIZE:
      action = random.randrange(0,self.num_actions
        )
    else:
      action = np.argmax(self.q_values.eval(
        feed_dict={self.s: [np.float32(state
        /250.0)]}))
    self.repeated_action = action

  if num == 0:
    self.max_action = np.argmax(self.q_values.
      eval(feed_dict={self.s: [np.float32(state
      /250.0)]}))
    self.q_values2 =  self.q_values.eval(
      feed_dict={self.adas_s: [np.float32(state
      /250.0)]})

  # Anneal epsilon linearly over time
  if self.epsilon > FINAL_EPSILON and self.t >=
    INITIAL_REPLAY_SIZE:
    self.epsilon -= self.epsilon_step

  return action

def run(self, num, before_state, action, reward
  , state, terminal):

  if terminal != 2:
    self.replay_memory.append((before_state,
      action, reward, state, terminal))

  if len(self.replay_memory) > NUM_REPLAY_MEMORY
    :
    self.replay_memory.popleft()

  if self.t >= INITIAL_REPLAY_SIZE:
    # Train network
    if self.t % TRAIN_INTERVAL == 0:
      self.train_network()

    # Update target network
    if self.t % TARGET_UPDATE_INTERVAL == 0:
      self.sess.run(self.update_target_network)

    # Save network
    if self.t % SAVE_INTERVAL == 0:
      save_path = self.saver.save(self.sess,
        SAVE_NETWORK_PATH + '/' + ENV_NAME,
        global_step=(self.t))
      print('Successfully saved: ' + save_path)

  self.total_reward += reward
  self.total_q_max += np.max(self.q_values.eval(
    feed_dict={self.s: [np.float32(state
    /250.0)]}))
  self.duration += 1

  if terminal == 1:
    # Write summary
    if self.t >= INITIAL_REPLAY_SIZE:
      stats = [self.total_reward, self.total_q_max
        / float(self.duration),
        self.duration, self.total_loss / (float(
          self.duration) / float(TRAIN_INTERVAL))]
      for i in xrange(len(stats)):
        self.sess.run(self.update_ops[i], feed_dict
          ={
          self.summary_placeholders[i]: float(stats[
            i])
        })
      summary_str = self.sess.run(self.summary_op)
      self.summary_writer.add_summary(summary_str,
        self.episode + 1)

    # Debug
    if self.t < INITIAL_REPLAY_SIZE:
      mode = 'random'
    elif INITIAL_REPLAY_SIZE <= self.t <
      INITIAL_REPLAY_SIZE + EXPLORATION_STEPS:
      mode = 'explore'
    else:
      mode = 'exploit'

    print('EPISODE: {0:6d} / TIMESTEP: {1:8d} /
      DURATION: {2:5d} / EPSILON: {3:.5f} /
      TOTAL_REWARD: {4:3.0f} / AVG_MAX_Q: {5:2.4f}
        / AVG_LOSS: {6:.5f} / MODE: {7}'.format(
      self.episode + 1, self.t, self.duration,
        self.epsilon,
      self.total_reward, self.total_q_max / float(
        self.duration),
      self.total_loss / (float(self.duration) /
        float(TRAIN_INTERVAL)), mode))

    self.total_reward = 0
    self.total_q_max = 0
    self.total_loss = 0
    self.duration = 0
    self.episode += 1
    if num == 0:
      self.episode0 += 1

  if num == 0:
    self.t0 += 1

  self.t += 1

  #return next_state

def train_network(self):
  before_state_batch = []
  action_batch = []
  reward_batch = []
  state_batch = []
  terminal_batch = []
  y_batch = []

  # Sample random minibatch of transition from
    replay memory
  minibatch = random.sample(self.replay_memory,
    BATCH_SIZE)
  for data in minibatch:
    before_state_batch.append(data[0])
    action_batch.append(data[1])
    reward_batch.append(data[2])
    state_batch.append(data[3])
    terminal_batch.append(data[4])

  # Convert True to 1, False to 0
  terminal_batch = np.array(terminal_batch) + 0

  #double deep Q-learning
  a = np.argmax(self.q_values.eval(feed_dict={
```

```
        self.s: np.float32(np.array(state_batch
        )/250.0)}), axis=1)
      target_q_values_batch = self.target_q_values.
        eval(feed_dict={self.st: np.float32(np.array(
        state_batch)/250.0)})
      targets = np.zeros((BATCH_SIZE)).astype(np.
        float32)
      for i in range(0,BATCH_SIZE):
        targets[i] = target_q_values_batch[i][a[i]]
      y_batch = reward_batch + (1 - terminal_batch)
        * GAMMA * targets

      #deep Q-learning
      # target_q_values_batch = self.target_q_values
        .eval(feed_dict={self.st: np.float32(np.array
        (state_batch))})
      # y_batch = reward_batch + (1 - terminal_batch
        ) * GAMMA * np.max(target_q_values_batch,
        axis=1)
      #print np.max(target_q_values_batch, axis=1)

      loss, grad_update = self.sess.run([self.loss,
        self.grad_update], feed_dict={
        self.s: np.float32(np.array(
          before_state_batch)/250.0),
        self.a: action_batch,
        self.y: y_batch
      })

      self.total_loss += loss

    def setup_summary(self):
      episode_total_reward = tf.Variable(0.)
      tf.summary.scalar(ENV_NAME + '/Total Reward/
        Episode', episode_total_reward)
      episode_avg_max_q = tf.Variable(0.)
      tf.summary.scalar(ENV_NAME + '/Average Max Q/
        Episode', episode_avg_max_q)
      episode_duration = tf.Variable(0.)
      tf.summary.scalar(ENV_NAME + '/Duration/
        Episode', episode_duration)
      episode_avg_loss = tf.Variable(0.)
      tf.summary.scalar(ENV_NAME + '/Average Loss/
        Episode', episode_avg_loss)
      summary_vars = [episode_total_reward,
        episode_avg_max_q, episode_duration,
        episode_avg_loss]
      summary_placeholders = [tf.placeholder(tf.
        float32) for _ in xrange(len(summary_vars))]
      update_ops = [summary_vars[i].assign(
        summary_placeholders[i]) for i in xrange(len(
        summary_vars))]
      summary_op = tf.summary.merge_all()
      return summary_placeholders, update_ops,
        summary_op

    def load_network(self):
      checkpoint = tf.train.get_checkpoint_state(
        SAVE_NETWORK_PATH)
      if checkpoint and checkpoint.
        model_checkpoint_path:
        self.saver.restore(self.sess, checkpoint.
          model_checkpoint_path)
        print('Successfully loaded: ' + checkpoint.
          model_checkpoint_path)
      else:
        print('Training new network...')

    def get_action_at_test(self, num, state, act ,
      eps):
      action = self.repeated_action

      action = np.argmax(self.q_values.eval(
        feed_dict={self.s: [np.float32(state /
        250.0)]}))
      self.repeated_action = action

      if num == 0:
        self.t1 += 1

      self.max_action = np.argmax(self.q_values.eval
        (feed_dict={self.s: [np.float32(state
        /250.0)]}))
      self.q_values2 =  self.q_values.eval(feed_dict
        ={self.s: [np.float32(state/250.0)]})

      if eps == 1:
        act = action

      # if self.epsilon >= random.random():
      #   action = random.randrange(0, self.
        num_actions)
      # else:
      #   action = np.argmax(self.q_values.eval(
        feed_dict={self.s: [np.float32(state
        /250.0)]}))
      # self.repeated_action = action
      return action

    def main():
      pointlist = [(0,0),(0,0),(0,0),(0,0)]
      routepoint =
        [(70,70),(240,70),(240,410),(110,410),(110,540),

        (200,540),(200,370),(200,110),(110,110),(110,370),

        (240,370),(480,370),(480,580),(70,580),(70,430),

        (70,70),(240,70),(240,410),(240,540),(440,540),

        (440,410),(200,410),(70,410),(70,70)]
      linepoint  =
        [(50,50,260,50),(260,50,260,340),(260,340,270,350),

        (270,350,500,350),(500,350,500,600),(500,600,50,600),

        (50,600,50,50),(270,430,410,430),(410,430,420,440),

        (420,440,420,510),(420,510,410,520),(410,520,270,520),

        (270,520,260,510),(260,510,260,440),(260,440,270,430),

        (140,430,170,430),(170,430,180,440),(180,440,180,510),

        (180,510,170,520),(170,520,140,520),(140,520,130,510),

        (130,510,130,440),(130,440,140,430),(140,130,170,130),

        (170,130,180,140),(180,140,180,340),(180,340,170,350),

        (170,350,140,350),(140,350,130,340),(130,340,130,140),

        (130,140,140,130),(130,90,180,90),(130,390,180,390),

        (130,560,180,560),(260,390,420,390),(260,560,420,560),

        (250,390,250,390),(250,560,250,560),(90,130,90,350),

        (90,430,90,520),(220,130,220,350),(220,430,220,520),

        (460,430,460,520)]
      wallpoint1 = np.array
        ([(140,140),(170,140),(170,340),(140,340),(140,140)])

      wallpoint2 = np.array
        ([(140,440),(170,440),(170,510),(140,510),(140,440)])

      wallpoint3 = np.array
        ([(270,440),(410,440),(410,510),(270,510),(270,440)])

      wallpoint4 = np.array
        ([(0,0),(40,40),(270,40),(270,340),(270,340),(510,340),

        (510,610),(550,650),(550,300),(300,300),(300,0)])

      wallpoint5 = np.array
        ([(0,0),(40,40),(40,610),(510,610),(550,650),(0,650)])

      wallpoint6 = np.r_[wallpoint4[1:-4], wallpoint5
        [-4:0:-1]]

      car = []
      carpoint = []
      carlist = []
      wall = []
      route = []

      for i in range(0,CAR_NUM): #len(routepoint)-5
        if i != 1 and i != 2 and i != 3 and i != 10
          and i != 11 and i != 13 and i != 14 and i !=
          15 and i != 16 and i != 21 and i != 22 and i
          != 23:
          (x1,y1) = routepoint[i]
          (x2,y2) = routepoint[i+1]
          deg = math.degrees(math.atan2(y2 - y1,x2 - x1
            ))
          deg = -1*deg
          rad = math.radians(deg+45)
          #      x,y,deg,flg,cs,speed,handle
          carlist.append((x1,y1,deg,0,i,0,0))

      state = np.zeros((len(carlist),286)).astype(np.
        float32)
      (w,h) = (35,15)
```

```
speed = 0
deg = 0
action = 0
terminal = 0
cnt = 0


init(car,wall,route,carlist,carpoint,wallpoint1
  ,wallpoint2,wallpoint3,wallpoint6,routepoint)

for i in range(0,len(carlist)):
 deg = carlist[i][2]
 r = sensor(i,0,deg,car,carlist,wallpoint1,
   wallpoint2,wallpoint3,wallpoint6,linepoint,
   carpoint,routepoint,state,inside=0,mode=1)
#reset(car,carlist,wallpoint1,wallpoint4,
  carpoint,routepoint,state)


agent = Agent(num_actions=5)
agent.adas_load_network()
agent.load_network()
coll0_wall=0
coll0_car=0
coll1_wall=0
coll1_car=0
total_reward0=0
total_reward1=0
train_mode = 0
train_cnt = 0
eps = 1
step = 0
theta = 0
adas_flg = [0]*CAR_NUM
adas_mode = 0
flg_mg = [[0]*10]*len(carlist)
coll = 0
total_reward = 0
adas_cnt = 0

mode = 0

while(1):
 #reset(car,carlist,wallpoint1,wallpoint4,
   carpoint,routepoint,state)
 if mode == TRAIN:
  #time.sleep(0.1)
  if terminal == 0 or terminal == 2:
   stage_draw(carlist,carpoint,wallpoint1,
     wallpoint2,wallpoint3,wallpoint4,wallpoint5
     ,routepoint,linepoint)
   for i in range(0,len(carlist)):
    reward = 0
    inside = 0
    (flg,cs,speed,handle) = (carlist[i][3],
      carlist[i][4],carlist[i][5],carlist[i][6])
    before_state = np.array(state[i])
    agent.get_q_values(state[i])
    adas_mode = state_draw(i,carlist,state,
      agent.adas_values2[0],agent.
      adas_max_action) #adas_mode[i]
    action = agent.adas_get_action(i,state[i])
    inside,reward,terminal,theta = move(i,
      action,reward,car,wallpoint1,wallpoint2,
      wallpoint3,wallpoint6,route,routepoint,
      carlist,carpoint,linepoint,state,agent.
      adas_episode,agent.adas_t0,agent.
      adas_epsilon)
    reward = sensor(i,action,reward,car,carlist
      ,wallpoint1,wallpoint2,wallpoint3,
      wallpoint6,linepoint,carpoint,routepoint,
      state,inside,TRAIN)
    agent.adas_run(i,before_state, action,
      reward, np.array(state[i]), terminal)

    total_reward0 += reward
    if 1 <= inside <= 4:
     coll0_wall += 1
    if inside == 5:
     coll0_car += 1
    if agent.adas_t0%(1000*CAR_NUM) == 0:
     total_reward0 = total_reward0/CAR_NUM
     coll0_wall = coll0_wall/CAR_NUM
     coll0_car = coll0_car/CAR_NUM
     with open("coll_cnt.txt","a") as file:
      file.write(str(agent.t0) + " " + str(
        coll0_wall) + " " + str(coll0_car) + " "
         + str(total_reward0) + "\n")
      print "file write"
    if agent.adas_t0%(2000*CAR_NUM) == 0 and i
     == (CAR_NUM-1):
     coll0_wall=0
     coll0_car=0
     total_reward0=0
     car = []
     carpoint = []
     carlist = []
     wall = []
     route = []
     reset(car,wall,route,carlist,wallpoint1,
       wallpoint2,wallpoint3,wallpoint4,
       wallpoint5,wallpoint6,carpoint,routepoint
       ,linepoint,state,CAR_NUM)


  # イベント処理
  for event in pygame.event.get():
   if event.type == KEYDOWN:
    if event.key == K_LCTRL:
     # save_path = agent.saver.save(agent.sess
       , SAVE_NETWORK_PATH + '/' + ENV_NAME,
       global_step=(agent.t))
     # print('Successfully saved: ' +
       save_path)
     save_path = agent.adas_saver.save(agent.
       sess, ADAS_SAVE_NETWORK_PATH + '/' +
       ENV_NAME, global_step=(agent.adas_t))
     print('Successfully saved: ' + save_path)
    if event.key == K_SPACE:
     if eps == 0:
      eps = 1
      coll1_wall=0
      coll1_car=0
      total_reward1=0
      car = []
      carpoint = []
      reset(car,wall,route,carlist,wallpoint1,
        wallpoint2,wallpoint3,wallpoint4,
        wallpoint5,wallpoint6,carpoint,
        routepoint,linepoint,state,CAR_NUM)
     else: eps = 0
    if event.key == K_RSHIFT:
     #mode = 0
     car = []
     carpoint = []
     carlist = []
     wall = []
     route = []
     reset(car,wall,route,carlist,wallpoint1,
       wallpoint2,wallpoint3,wallpoint4,
       wallpoint5,wallpoint6,carpoint,
       routepoint,linepoint,state,CAR_NUM)
   if event.type == QUIT:          # 閉じるボタン
     が押されたら終了
     pygame.quit()               # Pygameの終了(画
       面閉じられる)
     sys.exit()
 elif terminal == 1:
  #reset(car,carlist,wallpoint1,wallpoint4,
    carpoint,routepoint,state)
  terminal = 0
  cnt = 0
 # else:
 #  terminal = 0
 #  cnt = 0
elif mode == TEST:
 if terminal == 0 or terminal == 2:
  stage_draw(carlist,carpoint,wallpoint1,
    wallpoint2,wallpoint3,wallpoint4,wallpoint5
    ,routepoint,linepoint)
  for i in range(0,len(carlist)):
   reward = 0
   inside = 0
   (flg,cs,speed,handle) = (carlist[i][3],
     carlist[i][4],carlist[i][5],carlist[i][6])
   before_state = np.array(state[i])
   agent.get_q_values(state[i])
   adas_mode = state_draw(i,carlist,state,
     agent.adas_values2[0],agent.
     adas_max_action) #adas_mode[i]
   if adas_mode == 1:
    if adas_flg[i] == 0: adas_cnt += 1
    adas_flg[i] = 1
    action = agent.adas_get_action(i,state[i])
   else:
    adas_flg[i] = 0
    action = get_move_act(i,car,carlist,
     carpoint,routepoint,state)
   action = agent.get_action_at_test(i,state[i
     ],0,0)
   #action = get_move_act(i,car,carlist,
     carpoint,routepoint,state)
   #action = agent.adas_get_action(i,state[i])
   inside,reward,terminal,theta = move(i,
```

```
        action,reward,car,wallpoint1,wallpoint2,
        wallpoint3,wallpoint6,route,routepoint,
        carlist,carpoint,linepoint,state,agent.
        adas_episode,agent.adas_t0,agent.
        adas_epsilon)
       reward = sensor(i,action,reward,car,carlist
        ,wallpoint1,wallpoint2,wallpoint3,
        wallpoint6,linepoint,carpoint,routepoint,
        state,inside,TRAIN)

       total_reward0 += reward
       if inside != 0:
        adas_flg[i] = 0
       if 1 <= inside <= 4:
        coll0_wall += 1
       if inside == 5:
        coll0_car += 1
       if i == (len(carlist)-1):
        agent.t += 1
        if agent.t%1000 == 0:
         adas_cnt = float(adas_cnt)/CAR_NUM
         total_reward0 = float(total_reward0)/
          CAR_NUM
         coll0_wall = float(coll0_wall)/CAR_NUM
         coll0_car = float(coll0_car)/CAR_NUM
         with open("coll_cnt.txt", "a") as file:
          file.write(str(agent.t) + " " + str(
           adas_cnt) + " " + str(coll0_wall) + " "
            + str(coll0_car) + " " +  str(
           total_reward0) + "\n")
          print "file write"
         total_reward0 = 0
         coll0_wall = 0
         coll0_car = 0
         car = []
         carpoint = []
         carlist = []
         wall = []
         route = []
         reset(car,wall,route,carlist,wallpoint1,
          wallpoint2,wallpoint3,wallpoint4,
          wallpoint5,wallpoint6,carpoint,
          routepoint,linepoint,state,CAR_NUM)

    for event in pygame.event.get():
     # if event.type == MOUSEBUTTONDOWN and event
       .button == 1:
     #  (x, y) = event.pos
     #  car_set(x,y,carlist,carpoint,car)
     #  stage_draw(carlist,carpoint,wallpoint1,
       wallpoint2,wallpoint3,wallpoint4,wallpoint5
       ,routepoint,linepoint,mode)
     #  (speed,handle) = (0,0)
     #  move(0,action,speed,handle,reward,car,
       wallpoint1,wallpoint2,wallpoint3,wallpoint6
       ,route,routepoint,carlist,carpoint,
       linepoint,state,agent.episode0,agent.t0,
       agent.epsilon,agent.q_values2[0],agent.
       max_action)
     #  sensor(0,action,reward,car,carlist,
       wallpoint1,wallpoint2,wallpoint3,wallpoint6
       ,linepoint,carpoint,routepoint,state,inside
       ,TEST)

      if event.type == KEYDOWN:
       if event.key == K_RSHIFT:
        car = []
        carpoint = []
        carlist = []
        wall = []
        route = []
        reset(car,wall,route,carlist,wallpoint1,
         wallpoint2,wallpoint3,wallpoint4,
         wallpoint5,wallpoint6,carpoint,routepoint
        ,linepoint,state,CAR_NUM)
      if event.type == QUIT:          # 閉じるボタン
       が押されたら終了
       pygame.quit()                  # Pygameの終了(画面
        閉じられる)
       sys.exit()


if __name__ == '__main__':
 main()
```

draw.pyx

```
# -*- coding:utf-8 -*-
import numpy as np
cimport numpy as np
cimport cython
import pygame
from pygame.locals import *
import sys
import math
import time
import random
from fractions import Fraction

from mete import wallmete,linemete,carmete,mete,
   distance,distance_l,angul,angular,crosspoint

black  = [  0,  0,  0]
white  = [200,200,200]
white2 = [100,100,100]
blue   = [  0, 88,255]
blue2  = [  0, 44,122]
green  = [  0, 88,  0]
green2 = [  0, 44,  0]
red = [194,  0, 28]
red2   = [ 97,  0, 14]
yellow = [200,200,  0]
yellow2 = [100,100,  0]

WALL_PUNISH = 5
CAR_PUNISH = 5
LINE_PUNISH = 3

(x,y) = (1000,650)
pygame.init()
screen = pygame.display.set_mode((x,y),0,32)
clock = pygame.time.Clock()
font = pygame.font.Font(None, 25)

cpdef init(car,wall,route,carlist,carpoint,
   wallpoint1,wallpoint2,wallpoint3,wallpoint6,
   routepoint):

 pointlist = [(0,0),(0,0),(0,0),(0,0)]
 (w,h) = (35,15)
 speed = 0
 deg = 0
 pygame.init()

 pygame.display.set_caption("AUTOMATION DRIVE")

 for i in range(0,len(wallpoint1)-1):
  (x1,y1) = wallpoint1[i]
  (x2,y2) = wallpoint1[i+1]
  if x1 == x2 or y1 == y2:
   wall.append(pygame.draw.line(screen,green,(x1
      ,y1),(x2,y2),1))
  else:
   if x1 > x2:
    (tmp_x,tmp_y) = (x2,y2)
    (x2,y2) = (x1,y1)
    (x1,y1) = (tmp_x,tmp_y)
   slope = Fraction(y2-y1,x2-x1)
   for j in range(x1,x2,slope.denominator):
    y1 += slope.numerator
    wall.append(pygame.draw.rect(screen,green,
       Rect(j,y1,slope.denominator,slope.
       numerator)))

 for i in range(0,len(wallpoint2)-1):
  (x1,y1) = wallpoint1[i]
  (x2,y2) = wallpoint1[i+1]
  if x1 == x2 or y1 == y2:
   wall.append(pygame.draw.line(screen,green,(x1
      ,y1),(x2,y2),1))
  else:
   if x1 > x2:
    (tmp_x,tmp_y) = (x2,y2)
    (x2,y2) = (x1,y1)
    (x1,y1) = (tmp_x,tmp_y)
   slope = Fraction(y2-y1,x2-x1)
   for j in range(x1,x2,slope.denominator):
    y1 += slope.numerator
    wall.append(pygame.draw.rect(screen,green,
       Rect(j,y1,slope.denominator,slope.
       numerator)))

 for i in range(0,len(wallpoint3)-1):
  (x1,y1) = wallpoint1[i]
  (x2,y2) = wallpoint1[i+1]
  if x1 == x2 or y1 == y2:
   wall.append(pygame.draw.line(screen,green,(x1
      ,y1),(x2,y2),1))
  else:
   if x1 > x2:
    (tmp_x,tmp_y) = (x2,y2)
    (x2,y2) = (x1,y1)
    (x1,y1) = (tmp_x,tmp_y)
```

```
    slope = Fraction(y2-y1,x2-x1)
    for j in range(x1,x2,slope.denominator):
     y1 += slope.numerator
     wall.append(pygame.draw.rect(screen,green,
        Rect(j,y1,slope.denominator,slope.
        numerator)))

  for i in range(0,len(wallpoint6)-1):
   (x1,y1) = wallpoint6[i]
   (x2,y2) = wallpoint6[i+1]
   if x1 == x2 or y1 == y2:
    wall.append(pygame.draw.line(screen,green,(x1
       ,y1),(x2,y2),1))
   else:
    if x1 > x2:
     (tmp_x,tmp_y) = (x2,y2)
     (x2,y2) = (x1,y1)
     (x1,y1) = (tmp_x,tmp_y)
    slope = Fraction(y2-y1,x2-x1)
    for j in range(x1,x2,slope.denominator):
     y1 += slope.numerator
     wall.append(pygame.draw.rect(screen,green,
        Rect(j,y1,slope.denominator,slope.
        numerator)))

  for i in range(0,len(routepoint)-1):
   (x1,y1) = routepoint[i]
   (x2,y2) = routepoint[i+1]
   route.append([])
   if x1 == x2 or y1 == y2:
    route[i].append(pygame.draw.line(screen,white
       ,(x1,y1),(x2,y2),1))
   else:
    if x1 > x2:
     (tmp_x,tmp_y) = (x2,y2)
     (x2,y2) = (x1,y1)
     (x1,y1) = (tmp_x,tmp_y)
    slope = Fraction(y2-y1,x2-x1)
    for j in range(x1,x2,slope.denominator):
     y1 += slope.numerator
     route[i].append(pygame.draw.rect(screen,
        white,Rect(j,y1,slope.denominator,slope.
        numerator)))

  for i in range(0,len(carlist)): #len(carlist)
   carpoint.append([])
   (dx,dy,tmp_deg,flg,cs,speed,handle) = carlist[
      i]
   (tx,ty) = (dx,dy)
   rad = math.radians(deg)
   dx += math.cos(rad)*speed + math.cos(rad)*20
   dy += -1*math.sin(rad)*speed - math.sin(rad
      )*10
   k=0
   for j in range(0,4):
    if j == 1:
     rad = math.radians(tmp_deg+180)
     tx = math.cos(rad)*w + dx
     ty = -1*math.sin(rad)*w + dy
    if j == 2:
     rad = math.radians(tmp_deg+270)
     tx = math.cos(rad)*h + tx
     ty = -1*math.sin(rad)*h + ty
    if j == 3:
     rad = math.radians(tmp_deg+270)
     tx = math.cos(rad)*h + dx
     ty = -1*math.sin(rad)*h + dy
    pointlist.pop(j)
    pointlist.insert(j,(tx,ty))
    carpoint[i].append(int(tx))
    carpoint[i].append(int(ty))
   if flg == 0:
    car.append(pygame.draw.lines(screen,white2,
       True,pointlist,1))
   elif flg == 1:
    car.append(pygame.draw.lines(screen,red2,True
       ,pointlist,1))
   else:
    car.append(pygame.draw.lines(screen,blue2,
       True,pointlist,1))
   rad = math.radians(deg)
   dx -= math.cos(rad)*20
   dy += math.sin(rad)*10

  screen.fill((0,0,0))

  pygame.display.update()


cpdef reset(car,wall,route,carlist,wallpoint1,
    wallpoint2,wallpoint3,wallpoint4,wallpoint5,
    wallpoint6,carpoint,routepoint,linepoint,state
    ,num):
 pointlist = [(0,0),(0,0),(0,0),(0,0)]
 (w,h) = (35,15)

 screen.fill((0,0,0))

 for i in range(0,1):
  carlist.pop(0)

 for i in range(0,num): #len(routepoint)-5
  if i != 1 and i != 2 and i != 3 and i != 10
     and i != 11 and i != 13 and i != 14 and i !=
      15 and i != 16 and i != 21 and i != 22 and
     i != 23:
   (x,y) = routepoint[i]
   (x1,y1) = routepoint[i]
   (x2,y2) = routepoint[i+1]
   deg = math.degrees(math.atan2(y2 - y1,x2 - x1
      ))
   deg = -1*deg
   rad = math.radians(deg+45)
   #      x,y,deg,flg,cs,speed,handle
   carlist.append((math.cos(rad)*30+x,-1*math.
      sin(rad)*10+y,deg+3.5,0,i,0,0))

 for i in range(0,len(wallpoint1)-1):

  (x1,y1) = wallpoint1[i]
  (x2,y2) = wallpoint1[i+1]
  if x1 == x2 or y1 == y2:
   wall.append(pygame.draw.line(screen,green,(x1
      ,y1),(x2,y2),1))
  else:
   if x1 > x2:
    (tmp_x,tmp_y) = (x2,y2)
    (x2,y2) = (x1,y1)
    (x1,y1) = (tmp_x,tmp_y)
   slope = Fraction(y2-y1,x2-x1)
   for j in range(x1,x2,slope.denominator):
    y1 += slope.numerator
    wall.append(pygame.draw.rect(screen,green,
       Rect(j,y1,slope.denominator,slope.
       numerator)))

 for i in range(0,len(wallpoint2)-1):
  (x1,y1) = wallpoint1[i]
  (x2,y2) = wallpoint1[i+1]
  if x1 == x2 or y1 == y2:
   wall.append(pygame.draw.line(screen,green,(x1
      ,y1),(x2,y2),1))
  else:
   if x1 > x2:
    (tmp_x,tmp_y) = (x2,y2)
    (x2,y2) = (x1,y1)
    (x1,y1) = (tmp_x,tmp_y)
   slope = Fraction(y2-y1,x2-x1)
   for j in range(x1,x2,slope.denominator):
    y1 += slope.numerator
    wall.append(pygame.draw.rect(screen,green,
       Rect(j,y1,slope.denominator,slope.
       numerator)))

 for i in range(0,len(wallpoint3)-1):
  (x1,y1) = wallpoint1[i]
  (x2,y2) = wallpoint1[i+1]
  if x1 == x2 or y1 == y2:
   wall.append(pygame.draw.line(screen,green,(x1
      ,y1),(x2,y2),1))
  else:
   if x1 > x2:
    (tmp_x,tmp_y) = (x2,y2)
    (x2,y2) = (x1,y1)
    (x1,y1) = (tmp_x,tmp_y)
   slope = Fraction(y2-y1,x2-x1)
   for j in range(x1,x2,slope.denominator):
    y1 += slope.numerator
    wall.append(pygame.draw.rect(screen,green,
       Rect(j,y1,slope.denominator,slope.
       numerator)))

 for i in range(0,len(wallpoint6)-1):
  (x1,y1) = wallpoint6[i]
  (x2,y2) = wallpoint6[i+1]
  if x1 == x2 or y1 == y2:
   wall.append(pygame.draw.line(screen,green,(x1
      ,y1),(x2,y2),1))
  else:
   if x1 > x2:
    (tmp_x,tmp_y) = (x2,y2)
    (x2,y2) = (x1,y1)
    (x1,y1) = (tmp_x,tmp_y)
   slope = Fraction(y2-y1,x2-x1)
```

```
    for j in range(x1,x2,slope.denominator):
     y1 += slope.numerator
     wall.append(pygame.draw.rect(screen,green,
        Rect(j,y1,slope.denominator,slope.
        numerator)))

  for i in range(0,len(routepoint)-1):
   (x1,y1) = routepoint[i]
   (x2,y2) = routepoint[i+1]
   route.append([])
   if x1 == x2 or y1 == y2:
    route[i].append(pygame.draw.line(screen,white
       ,(x1,y1),(x2,y2),1))
   else:
    if x1 > x2:
     (tmp_x,tmp_y) = (x2,y2)
     (x2,y2) = (x1,y1)
     (x1,y1) = (tmp_x,tmp_y)
    slope = Fraction(y2-y1,x2-x1)
    for j in range(x1,x2,slope.denominator):
     y1 += slope.numerator
     route[i].append(pygame.draw.rect(screen,
        white,Rect(j,y1,slope.denominator,slope.
        numerator)))

  stage_draw(carlist,carpoint,wallpoint1,
     wallpoint2,wallpoint3,wallpoint4,wallpoint5,
     routepoint,linepoint)

  for i in range(0,len(carlist)): #len(carlist)
   carpoint.append([])
   (dx,dy,tmp_deg,flg,cs,speed,handle) = carlist[
      i]
   rad = math.radians(deg)
   dx += math.cos(rad)*speed + math.cos(rad)*w
   dy += -1*math.sin(rad)*speed - math.sin(rad)*h
   (tx,ty) = (dx,dy)
   k=0
   for j in range(0,4):
    if j == 1:
     rad = math.radians(tmp_deg+180)
     tx = math.cos(rad)*w + dx
     ty = -1*math.sin(rad)*w + dy
    if j == 2:
     rad = math.radians(tmp_deg+270)
     tx = math.cos(rad)*h + tx
     ty = -1*math.sin(rad)*h + ty
    if j == 3:
     rad = math.radians(tmp_deg+270)
     tx = math.cos(rad)*h + dx
     ty = -1*math.sin(rad)*h + dy
    pointlist.pop(j)
    pointlist.insert(j,(tx,ty))
    carpoint[i].append(int(tx))
    carpoint[i].append(int(ty))
   if flg == 0:
    car.append(pygame.draw.lines(screen,white2,
       True,pointlist,1))
   elif flg == 1:
    car.append(pygame.draw.lines(screen,red2,True
       ,pointlist,1))
   else:
    car.append(pygame.draw.lines(screen,blue2,
       True,pointlist,1))
   rad = math.radians(deg)
   dx -= math.cos(rad)*w
   dy += math.sin(rad)*h
   pygame.draw.lines(screen,blue,True,pointlist
      ,2)

  state = np.zeros((len(carlist),286)).astype(np.
     float32)

  for i in range(0,len(carlist)):
   deg = carlist[i][2]
   r = sensor(i,0,deg,car,carlist,wallpoint1,
      wallpoint2,wallpoint3,wallpoint6,linepoint,
      carpoint,routepoint,state,inside=0,mode=1)

  pygame.display.update()


cpdef stage_draw(carlist,carpoint,wallpoint1,
   wallpoint2,wallpoint3,wallpoint4,wallpoint5,
   routepoint,linepoint):
  cdef int i
  pointlist = [(0,0),(0,0),(0,0),(0,0)]
  (w,h) = (35,15)
  screen.fill((0,0,0))

  #wall depiction
  pygame.draw.polygon(screen,green,wallpoint1)
  pygame.draw.polygon(screen,green,wallpoint2)
  pygame.draw.polygon(screen,green,wallpoint3)
  pygame.draw.polygon(screen,green,wallpoint4)
  pygame.draw.polygon(screen,green,wallpoint5)

  #route depiction
  for i in range(0,len(routepoint)-1):
   pygame.draw.aaline(screen, white2, routepoint[
      i], routepoint[i+1])

  for i in range(0,len(linepoint)):
   (x1,y1,x2,y2) = linepoint[i]
   pygame.draw.aaline(screen, white, (x1,y1), (x2
      ,y2))


cpdef section_act(int num,int action,car,carlist
   ,carpoint,routepoint,q_values):

  if min(q_values) < 0:
   return action

  return action

cpdef car_set(x,y,carlist,carpoint,car):
  pointlist = [(0,0),(0,0),(0,0),(0,0)]
  (w,h) = (35,15)
  rect = pygame.draw.circle(screen,red,(x,y),1)
  ind = rect.collidelist(car)
  if ind == -1:
   n = len(carlist)
   carlist.append((x,y-h/2,0,0,0,0,0))
   carpoint.append([])
   (dx,dy,deg,flg,cs,speed,handle) = carlist[n]
   rad = math.radians(deg)
   dx += math.cos(rad)*speed + math.cos(rad)*w
   dy += -1*math.sin(rad)*speed - math.sin(rad)*h
   (tx,ty) = (dx,dy)
   k=0
   for j in range(0,4):
    if j == 1:
     rad = math.radians(deg+180)
     tx = math.cos(rad)*w + dx
     ty = -1*math.sin(rad)*w + dy
    if j == 2:
     rad = math.radians(deg+270)
     tx = math.cos(rad)*h + tx
     ty = -1*math.sin(rad)*h + ty
    if j == 3:
     rad = math.radians(deg+270)
     tx = math.cos(rad)*h + dx
     ty = -1*math.sin(rad)*h + dy
    pointlist.pop(j)
    pointlist.insert(j,(tx,ty))
    carpoint[n].append(int(tx))
    carpoint[n].append(int(ty))
   if flg == 0:
    car.append(pygame.draw.lines(screen,white2,
       True,pointlist,1))
   elif flg == 1:
    car.append(pygame.draw.lines(screen,red2,True
       ,pointlist,1))
   else:
    car.append(pygame.draw.lines(screen,blue2,
       True,pointlist,1))
   rad = math.radians(deg)
   pygame.draw.lines(screen,blue,True,pointlist
      ,2)
  elif ind != 0:
   carpoint.pop(ind)
   carlist.pop(ind)
   car.pop(ind)

  pygame.display.update()



cpdef state_draw(int num,carlist,state,
   adas_values,adas_max_action):
  if num == 0:
   q = 0
   r = 0
   (flg,cs,speed,handle) = (carlist[num][3],
      carlist[num][4],carlist[num][5],carlist[num
      ][6])
   (x,y) = (320,270)
   for i in range(0,160):
    if i % 32 == 0:
     x+=5
    if int(i/32) == 0: pygame.draw.aaline(screen,
       (255*2/3,127*2/3,0), (x,y), (x, y+state[
       num][i]*-2/3), 2)
```

```
    if int(i/32) == 1: pygame.draw.aaline(screen,
       (100,100,100), (x,y), (x, y+state[num][i
      ]*-2/3), 2)
    if int(i/32) == 2: pygame.draw.aaline(screen,
       (0,127*2/3,255*2/3), (x,y), (x, y+state[
      num][i]*-2/3), 2)
    if int(i/32) == 3: pygame.draw.aaline(screen,
       (127*2/3,0,255*2/3), (x,y), (x, y+state[
      num][i]*-2/3), 2)
    if int(i/32) == 4: pygame.draw.aaline(screen,
       (255*2/3,0,127*2/3), (x,y), (x, y+state[
      num][i]*-2/3), 2)
    x+=2

  (x,y) = (570,520)
  x+=10
  y-=170
  pygame.draw.rect(screen,white, Rect(x,y
     ,25,170),1)
  x+=5
  y+=85
  pygame.draw.rect(screen,white, Rect(x,y,15,
     speed*-10))
  x+=25
  y-=85
  pygame.draw.rect(screen,white, Rect(x,y
     ,25,170),1)
  x+=5
  y+=85
  pygame.draw.rect(screen,white, Rect(x,y,15,
     handle*-10))
  x+=30

  if adas_values[0] <= 45 or adas_values[1] <=
     45 or adas_values[2] <= 45 or adas_values[3]
      <= 45 or adas_values[4] <= 45:

    for i in range(0,5):
     if adas_max_action == i:
      pygame.draw.rect(screen,red, Rect(x+i*30,y
         ,20,adas_values[i]*-1))
     else:
      pygame.draw.rect(screen,white, Rect(x+i*30,
         y,20,adas_values[i]*-1))

 if adas_values[0] <= 60 or adas_values[1] <= 60
    or adas_values[2] <= 60 or adas_values[3] <=
    60 or adas_values[4] <= 60:
  return 1
 else:
  return 0

cpdef get_move_act(int num,car,carlist,carpoint,
   routepoint,state):

 (dx,dy,deg,flg,cs,speed,handle) = carlist[num]
 rdeg = 0

 cs2 = cs + 1
 if cs2 >= len(routepoint)-1:
  cs2 = 0

 (cx1,cy1,cx2,cy2) = (carpoint[num][0],carpoint[
    num][1],carpoint[num][6],carpoint[num][7])

 (tx,ty) = (dx,dy)
 (tx2,ty2) = crosspoint(cx1,cy1,cx2,cy2)

 (rx1,ry1) = routepoint[cs]
 (rx2,ry2) = routepoint[cs+1]
 (rx3,ry3) = routepoint[cs2]
 (rx4,ry4) = routepoint[cs2+1]

 if rx1 == rx2:
  if ry1 > ry2:
   rdeg = 90
  if ry1 > ry2:
   rdeg = 270
 if ry1 == ry2:
  if rx1 > rx2:
   rdeg = 180
  if rx1 > rx2:
   rdeg = 0
 sdeg = deg - rdeg

 sdis2 = distance_l(tx,ty,rx1,ry1,rx2,ry2)
 thetag2 = angular(tx,ty,tx2,ty2,rx3,ry3,rx4,ry4
    )
 thetas2 = angular(tx,ty,tx2,ty2,rx1,ry1,rx2,ry2
    )

 if (state[num][64] > 220 or state[num][65] >
    220 or state[num][66] > 220 or state[num][94]
     > 220 or state[num][95] > 220) and sdis2 <
    20 and thetag2 < 40:
  return 3

 if sdis2 < 20:
  if thetag2 > 30:
   if speed < 1:
    return 3
   if (0 <= sdeg <= 180) or (-360 <= sdeg <=
      -180):
    return 2
   elif (-180 < sdeg < 0) or (180 < sdeg <
      360):
    return 1
  else:
   if handle > 2:
    return 2
   elif handle < -2:
    return 1
   if speed < 5:
    return 0
 else:
  if thetas2 > 50:
   if speed != 0:
    return 3
   return 1
  else:
   if handle > 2:
    return 2
   elif handle < -2:
    return 1
   if speed < 5:
    return 0

 return random.randrange(0,5)



cpdef move(int num,int action,int reward,car,
   wallpoint1,wallpoint2,wallpoint3,wallpoint6,
   route,routepoint,carlist,carpoint,linepoint,
   state,episode,step,epsilon):
 cdef int i,tx,ty,cs,inside=0,cnt=0,terminal=0

 pointlist = [(0,0),(0,0),(0,0),(0,0)]
 (w,h) = (35,15)
 (dx,dy,deg,flg,cs,speed,handle) = carlist[num]
 (tmp_x,tmp_y) = (dx,dy)
 rect = car[num]
 tmp_deg = deg
 tmp_speed = speed
 tmp_handle = handle

 if action == 0:
  if speed < 5:
   speed += 1
 if action == 1:
  if handle < 5:
   handle += 1
 if action == 2:
  if handle > -5:
   handle -= 1
 if action == 3:
  if speed > 1:
   speed -= 2
  if speed < -1:
   speed += 2
  if -1 <= speed <= 1:
   speed = 0
 if action == 4:
  if speed > -5:
   speed -= 1

 if speed >= 0:
  deg += handle
 elif speed < 0:
  deg -= handle

 if deg >= 360:
  deg -= 360
 elif deg < 0:
  deg += 360



 rad = math.radians(deg)
 dx += math.cos(rad)*speed*1.5
 dy += -1*math.sin(rad)*speed*1.5

 (mx,my) = (dx,dy)
```

```
 cs2 = cs + 1
 if cs2 >= len(routepoint)-1:
  cs2 = 0

 # gdis1 = distance_l(tx,ty,rx3,ry3,rx4,ry4)
 # sdis1 = distance_l(tx,ty,rx1,ry1,rx2,ry2)
 # thetag1 = angular(tx,ty,tx2,ty2,rx2,ry2)
 # thetas1 = angular(tx,ty,tx2,ty2,rx1,ry1)
 # b_sdis = distance(tx,ty,rx1,ry1)


 #Depiction of the car
 (tx,ty) = (dx,dy)
 carpoint.pop(num)
 carpoint.insert(num,[])
 for i in range(0,4):
  if i == 0:
   w2 = float(w)/2
   h2 = float(h)/2
   hyp = math.sqrt(w2*w2 + h2*h2)
   hyptheta = math.atan2(h,w) * 180 / math.pi
   rad = math.radians(deg+hyptheta)
   dx = math.cos(rad)*hyp + dx
   dy = -1*math.sin(rad)*hyp + dy
   (tx,ty) = (dx,dy)
  if i == 1:
   rad = math.radians(deg+180)
   tx = math.cos(rad)*w + dx
   ty = -1*math.sin(rad)*w + dy
  if i == 2:
   rad = math.radians(deg+270)
   tx = math.cos(rad)*h + tx
   ty = -1*math.sin(rad)*h + ty
  if i == 3:
   rad = math.radians(deg+270)
   tx = math.cos(rad)*h + dx
   ty = -1*math.sin(rad)*h + dy
  pointlist.pop(i)
  pointlist.insert(i,(tx,ty))
  carpoint[num].append(int(tx))
  carpoint[num].append(int(ty))

 rad = math.radians(deg)
 (cx1,cy1) = pointlist[0]
 (cx2,cy2) = pointlist[3]

 if flg == 0:
  color = white
 elif flg == 1:
  color = red
 elif flg == 2:
  color = blue
 car_main = pygame.draw.lines(screen,color,True,
    pointlist,2)
 (tx,ty) = car_main.center
 (tx2,ty2) = crosspoint(cx1,cy1,cx2,cy2)
 pygame.draw.line(screen,color,(tx,ty),(tx2,ty2
    ),2)


 for i in range(0,len(route[cs2])):
  if car_main.colliderect(route[cs2][i]) == 1:
   if speed>=0 and speed < 4:
    reward += 5
   (rx4,ry4) = routepoint[cs]
   cs += 1
   if cs >= len(routepoint)-1:
    cs = 0
   cs2 = cs + 1
   if cs2 >= len(routepoint)-1:
    cs2 = 0
   #terminal = 1
   (rx5,ry5) = routepoint[cs2]
   (rx6,ry6) = routepoint[cs2+1]
   if rx4 != rx6 or ry4 != ry6:
    thetag3 = angular(tx,ty,tx2,ty2,rx5,ry5,rx6,
       ry6)
    if speed>=0 and speed < 4:
     if thetag3 <= 70:
      reward += 10
      if thetag3 <= 70:
       reward += 10
       if thetag3 <= 50:
        reward += 10
   break

 car.pop(num)
 flg = 0
 coll = 0
 for i in range(0,len(wallpoint1)-1):
  (x1,y1) = wallpoint1[i]
  (x2,y2) = wallpoint1[i+1]
  k=0
  for j in range(0,4):
   (x3,y3) = (carpoint[num][k],carpoint[num][k
      +1])
   if j == 3:
    (x4,y4) = (carpoint[num][0],carpoint[num
       ][1])
   else:
    k+=2
    (x4,y4) = (carpoint[num][k],carpoint[num][k
       +1])
   if mete(x1,y1,x2,y2,x3,y3,x4,y4) != (0,0):
    flg = 1
    coll = 100 + i
    reward -= WALL_PUNISH
    reward -= 3*(abs(speed)+2)
    inside += 0x20
    break
  if flg == 1:
   break

 if flg != 1:
  for i in range(0,len(wallpoint2)-1):
   (x1,y1) = wallpoint2[i]
   (x2,y2) = wallpoint2[i+1]
   k=0
   for j in range(0,4):
    (x3,y3) = (carpoint[num][k],carpoint[num][k
       +1])
    if j == 3:
     (x4,y4) = (carpoint[num][0],carpoint[num
        ][1])
    else:
     k+=2
     (x4,y4) = (carpoint[num][k],carpoint[num][k
        +1])
    if mete(x1,y1,x2,y2,x3,y3,x4,y4) != (0,0):
     if reward >= 100: reward = 0
     flg = 1
     coll = 200 + i
     reward -= WALL_PUNISH
     reward -= 3*(abs(speed)+2)
     inside += 0x20
     break
   if flg == 1:
    break

 if flg != 1:
  for i in range(0,len(wallpoint3)-1):
   (x1,y1) = wallpoint3[i]
   (x2,y2) = wallpoint3[i+1]
   k=0
   for j in range(0,4):
    (x3,y3) = (carpoint[num][k],carpoint[num][k
       +1])
    if j == 3:
     (x4,y4) = (carpoint[num][0],carpoint[num
        ][1])
    else:
     k+=2
     (x4,y4) = (carpoint[num][k],carpoint[num][k
        +1])
    if mete(x1,y1,x2,y2,x3,y3,x4,y4) != (0,0):
     if reward >= 100: reward = 0
     flg = 1
     coll = 300 + i
     reward -= WALL_PUNISH
     reward -= 3*(abs(speed)+2)
     inside += 0x20
     break
   if flg == 1:
    break

 if flg != 1:
  for i in range(0,len(wallpoint6)-1):
   (x1,y1) = wallpoint6[i]
   (x2,y2) = wallpoint6[i+1]
   k=0
   for j in range(0,4):
    (x3,y3) = (carpoint[num][k],carpoint[num][k
       +1])
    if j == 3:
     (x4,y4) = (carpoint[num][0],carpoint[num
        ][1])
    else:
     k+=2
     (x4,y4) = (carpoint[num][k],carpoint[num][k
        +1])
    if mete(x1,y1,x2,y2,x3,y3,x4,y4) != (0,0):
     if reward >= 100: reward = 0
     flg = 1
     coll = 400 + i
```

```
      reward -= WALL_PUNISH
      reward -= 3*(abs(speed)+2)
      inside += 0x20
      break
    if flg == 1:
     break

if flg != 1:
 for i in range(0,len(car)):
  if car_main.colliderect(car[i]) == 1:
   if i >= num:
    i+=1
   k=0
   for j in range(0,4):
    (x1,y1) = (carpoint[num][k],carpoint[num][k
      +1])
    if j == 3:
     (x2,y2) = (carpoint[num][0],carpoint[num
       ][1])
    else:
     k+=2
     (x2,y2) = (carpoint[num][k],carpoint[num][
       k+1])
    m=0
    for l in range(0,4):
     (x3,y3) = (carpoint[i][m],carpoint[i][m
       +1])
     if l == 3:
      (x4,y4) = (carpoint[i][0],carpoint[i][1])
     else:
      m+=2
      (x4,y4) = (carpoint[i][m],carpoint[i][m
        +1])
     if mete(x1,y1,x2,y2,x3,y3,x4,y4) != (0,0):
      if reward >= 100: reward = 0
      flg = 1
      coll = 500 + i
      reward -= CAR_PUNISH
      reward -= 3*(abs(speed)+2)
      inside += 0x40
      break
    if flg == 1:
     break
  if flg == 1:
   break

for i in range(0,len(route[cs])):
 if car_main.colliderect(route[cs][i]) == 1:
  if flg == 0:
   flg = 2
  break

flg2 = 0
for i in range(0,len(linepoint)):
 (x1,y1,x2,y2) = linepoint[i]
 k=0
 for j in range(0,4):
  (x3,y3) = (carpoint[num][k],carpoint[num][k
    +1])
  if j == 3:
   (x4,y4) = (carpoint[num][0],carpoint[num
     ][1])
  else:
   k+=2
   (x4,y4) = (carpoint[num][k],carpoint[num][k
     +1])
  if mete(x1,y1,x2,y2,x3,y3,x4,y4) != (0,0):
   flg2 = 1
   reward -= LINE_PUNISH
   break
 if flg2 == 1:
  break

car.insert(num,car_main)

(rx1,ry1) = routepoint[cs]
(rx2,ry2) = routepoint[cs+1]
(rx3,ry3) = routepoint[cs2]
(rx4,ry4) = routepoint[cs2+1]

# gdis2 = distance_l(tx,ty,rx3,ry3,rx4,ry4)
sdis2 = distance_l(tx,ty,rx1,ry1,rx2,ry2)
thetag2 = angular(tx,ty,tx2,ty2,rx3,ry3,rx4,ry4
  )
thetas2 = angular(tx,ty,tx2,ty2,rx1,ry1,rx2,ry2
  )
routedis = distance(rx1,ry1,rx2,ry2)
sdis = distance(tx,ty,rx1,ry1)
gdis = distance(tx,ty,rx2,ry2)

# cnt_speed = 0
# cnt_speed += speed
# for i in range(256,277,3):
#  cnt_speed += (state[num][i]/25)-5
# if cnt_speed == 0:
#  reward -= 20

if state[num][64] > 220 or state[num][65] > 220
   or state[num][66] > 220 or state[num][94] >
  220 or state[num][95] > 220:
 reward -= 5

if flg == 2:
 if thetag2 <= 80:
  #reward += 1
  if thetag2 <= 60:
   #reward += 1
   if thetag2 <= 30:
    reward += 1
    if speed >= 0:
     reward += 1*(speed)
   if speed < -1:
    reward -= abs(speed)
 elif thetag2 > 90:
  reward -= 1
  if thetag2 > 110:
   reward -= 1
   if thetag2 > 130:
    reward -= 1
  reward -= abs(speed)
elif flg == 0 and (rx1<tx<rx2 or rx2<tx<rx1 or
  ry1<ty<ry2 or ry2<ty<ry1) and sdis2 < 20:
 if thetag2 <= 60:
  if speed > 0:
   reward += speed
  elif speed < -1:
   reward -= abs(speed)
 elif thetag2 > 90:
  reward -= abs(speed)
else:
 if thetas2 <= 80:
  reward += 1
  if thetas2 <= 50:
   reward += 1
   if thetas2 <= 40:
    reward += 2
   if speed > 0:
    reward += speed
   else:
    reward -= abs(speed)
 reward -= 10

type=0
(dx,dy)  =(mx,my)
if flg == 0:
 if dx < 40 or dx > 510 or dy < 40 or dy > 610
   or (dx > 270 and dy < 340) or (140 < dx <
   170 and 140 < dy < 340) or (140 < dx < 170
   and 440 < dy < 510) or (270 < dx < 410 and
   440 < dy < 510):
  (dx,dy) = routepoint[0]
  cs = 0
 carlist.pop(num)
 carlist.insert(num,(dx,dy,deg,0,cs,speed,
   handle))
elif flg == 1:
 type = int(coll/100)
 sub = coll-type*100
 if type == 1:
  (x1,y1) = wallpoint1[sub]
  (x2,y2) = wallpoint1[sub+1]
  thetacoll = angular(tx,ty,tx2,ty2,x1,y1,x2,y2
    )
 elif type == 2:
  (x1,y1) = wallpoint2[sub]
  (x2,y2) = wallpoint2[sub+1]
  thetacoll = angular(tx,ty,tx2,ty2,x1,y1,x2,y2
    )
 elif type == 3:
  (x1,y1) = wallpoint3[sub]
  (x2,y2) = wallpoint3[sub+1]
  thetacoll = angular(tx,ty,tx2,ty2,x1,y1,x2,y2
    )
 elif type == 4:
  (x1,y1) = wallpoint6[sub]
  (x2,y2) = wallpoint6[sub+1]
  thetacoll = angular(tx,ty,tx2,ty2,x1,y1,x2,y2
    )
 elif type == 5:
  (x,y) = car[sub].center
  thetacoll = angular(tx,ty,tx2,ty2,x,y,x,y)

 if 0 <= thetacoll < 70:
  tmp_x += math.cos(rad)*-2
```

```
    tmp_y += -1*math.sin(rad)*-2
   elif 180 >= thetacoll > 100:
    tmp_x += math.cos(rad)*2
    tmp_y += -1*math.sin(rad)*2
   else:
    tmp_deg += 10

   carlist.pop(num)
   carlist.insert(num,(tmp_x,tmp_y,tmp_deg,1,cs,
     speed,handle))
  elif flg == 2:
   carlist.pop(num)
   carlist.insert(num,(dx,dy,deg,2,cs,speed,
     handle))
  # else:
  #   carlist.pop(num)
  #   carlist.insert(num,(tmp_x,tmp_y,tmp_deg,3,cs
     ,speed,handle))

  if num == 0:
   r = deg
   (pos_x,pos_y) = car_main.center
   pygame.draw.circle(screen,white,(pos_x,pos_y
     ),50,2)
   for i in range(0,32):
    line = pygame.draw.aaline(screen, yellow, (
      pos_x,pos_y), (pos_x+math.cos(math.radians
      (-1*r))*250, pos_y+math.sin(math.radians
      (-1*r))*250), 1)
    r += 11.25
   text = font.render(("episode:%d  step:%d
     reward:%d action:%d epsilon:%0.5f" % (
     episode,step,reward,action,epsilon)), True,
     (255,255,255))
   screen.blit(text, [0,0])
  #reward = reward/3
  return type,reward,terminal,thetag2

cpdef sensor(int num,int action,int reward,car,
    carlist,wallpoint1,wallpoint2,wallpoint3,
    wallpoint6,linepoint,carpoint,routepoint,state
    ,inside,mode):
  #cdef int i,x,y,x1,y1,x2,y2,dx,dy,tmp_deg,flg,
     cs,cs2,r
  cdef int i=0,dis1=250,dis2=250,tmp_dis1=250,
     tmp_dis2=250,rcnt=0
  font = pygame.font.Font(None, 25)
  (dx,dy,deg,flg,cs,speed,handle) = carlist[num]
  w = []
  l = []
  c = []
  r1 = []
  r2 = []

  # for i in range(64,96):
  #   dis = state[num][i]
  #   if dis < tmp_dis1:
  #     tmp_dis1 = dis

  # for i in range(96,128):
  #   dis = state[num][i]
  #   if dis < tmp_dis2:
  #     tmp_dis2 = dis

  cs2 = cs + 1
  if cs2 >= len(routepoint)-1:
   cs2 = 0

  (rx1,ry1) = routepoint[cs]
  (rx2,ry2) = routepoint[cs+1]

  (rx3,ry3) = routepoint[cs2]
  (rx4,ry4) = routepoint[cs2+1]

  r = deg

  #s = time.time()
  for i in range(0,32):

   (x1,y1) = car[num].center
   (x2,y2) = (x1+(int)(math.cos(math.radians(-1*r
     ))*250), y1+(int)(math.sin(math.radians(-1*r
     ))*250))

   (walldis,x,y) = wallmete(x1,y1,x2,y2,
     wallpoint1,wallpoint2,wallpoint3,wallpoint6)
   if num == 0: pygame.draw.circle(screen,red2,(
     int(x),int(y)),1)
   if walldis <= 250:
    w.append(walldis)
   else:
    walldis = 250
   w.append(0)

   (linedis,x,y) = linemete(num,x1,y1,x2,y2,
     linepoint)
   if num == 0: pygame.draw.circle(screen,red2,(
     int(x),int(y)),1)
   if linedis <= walldis:
    l.append(linedis)
   else:
    l.append(0)

   (cardis,x,y) = carmete(num,x1,y1,x2,y2,
     carpoint)
   if num == 0: pygame.draw.circle(screen,red2,(
     int(x),int(y)),1)
   if cardis <= walldis:
    c.append(cardis)
   else:
    c.append(0)

   (x,y) = mete(x1,y1,x2,y2,rx1,ry1,rx2,ry2)
   if num == 0: pygame.draw.circle(screen,red2,(
     int(x),int(y)),1)
   if x != 0 and y != 0:
    dis = distance(x,y,x1,y1)
    r1.append(dis)
   else:
    r1.append(0)

   (x,y) = mete(x1,y1,x2,y2,rx3,ry3,rx4,ry4)
   if num == 0: pygame.draw.circle(screen,red2,(
     int(x),int(y)),1)
   if x != 0 and y != 0:
    dis = distance(x,y,x1,y1)
    r2.append(dis)
   else:
    r2.append(0)

   j=i+64
   t1=i+160
   t2=i+192
   t3=i+224
   state[num][t3] = state[num][t2]
   state[num][t2] = state[num][t1]
   state[num][t1] = state[num][j]

   j=i
   if w[i] == 0:
    state[num][j] = 0
   else:
    state[num][j] = 250-w[i]
   j += 32
   if w[i] == 0:
    state[num][j] = 0
   else:
    state[num][j] = 250-l[i]
   j += 32
   if c[i] == 0:
    state[num][j] = 0
   else:
    state[num][j] = 250-c[i]
   j += 32
   if r1[i] == 0:
    state[num][j] = 0
   else:
    state[num][j] = 250-r1[i]
   j += 32
   if r2[i] == 0:
    state[num][j] = 0
   else:
    state[num][j] = 250-r2[i]

   rcnt += r1[i]
   r += 11.25

  if flg == 2 and rcnt == 0:
   j=96
   for i in range(0,32):
    r1[i] = 0
    state[num][j] = 250
    j+=1

  j = 282
  for i in range(0,9):
   tmp = state[num][j]
   j+=3
   state[num][j] = tmp
   j-=4
   tmp = state[num][j]
   j+=3
   state[num][j] = tmp
   j-=4
```

```
    tmp = state[num][j]
    j+=3
    state[num][j] = tmp
    j-=4

  j = 256
  state[num][j] = (speed+5)*25
  j += 1
  state[num][j] = (handle+5)*25
  j += 1
  state[num][j] = (action+1)*50


  if num == len(carlist)-1 or mode == False:
   pygame.display.update()

  return reward
```

mete.pyx

```
# -*- coding: utf-8 -*-
import numpy as np
from numpy import*
cimport numpy as np
cimport cython
import sys
import math
import time
import random
from fractions import Fraction

DTYPE = np.int
ctypedef np.int_t DTYPE_t

cpdef angul(int x1,int y1,int x2,int y2,int x3,
    int y3):
  x = np.array([x2-x1,y2-y1])
  y = np.array([x3-x1,y3-y1])
  dot_xy = np.dot(x, y)
  norm_x = np.linalg.norm(x)
  norm_y = np.linalg.norm(y)
  cos = dot_xy / (norm_x*norm_y)
  rad = np.arccos(cos)
  theta = rad * 180 / np.pi
  return theta

cpdef angular(int x1,int y1,int x2,int y2,int x3
    ,int y3,int x4,int y4):
  if x3 == x4 and (y3 < y1 < y4 or y4 < y1 < y3):
   y3 = y1
  if y3 == y4 and (x3 < x1 < x4 or x4 < x1 < x3):
   x3 = x1
  x = np.array([x2-x1,y2-y1])
  y = np.array([x3-x1,y3-y1])
  dot_xy = np.dot(x, y)
  norm_x = np.linalg.norm(x)
  norm_y = np.linalg.norm(y)
  cos = dot_xy / (norm_x*norm_y)
  rad = np.arccos(cos)
  theta = rad * 180 / np.pi
  return theta

cpdef crosspoint(int x1,int y1,int x2,int y2):

  if x1 > x2:
   tmp = x1
   x1 = x2
   x2 = tmp
  if y1 > y2:
   tmp = y1
   y1 = y2
   y2 = tmp

  dfx = x1 + (x2 - x1)/2
  dfy = y1 + (y2 - y1)/2

  return dfx,dfy


cpdef distance_l(int x,int y,int rx1,int ry1,int
     rx2,int ry2):

  if rx1 > rx2:
   tmp = rx1
   rx1 = rx2
   rx2 = tmp
  if ry1 > ry2:
   tmp = ry1
   ry1 = ry2
   ry2 = tmp
  if (rx1<x<rx2) or (ry1<y<ry2):
   u = np.array([rx2-rx1,ry2-ry1])
   v = np.array([x-rx1,y-ry1])
   L = abs(cross(u,v)/linalg.norm(u))
   return L
  else:
   dis1 = distance(x,y,rx1,ry1)
   dis2 = distance(x,y,rx2,ry2)
   if dis1 < dis2:
    return dis1
   else:
    return dis2

cpdef distance(int x1,int y1,int x2,int y2):
  #s = time.time()
  a = np.array([x1,y1])
  b = np.array([x2,y2])
  u = b - a
  #print ("distance    :%f [sec]" % (time.time() -
      s))
  return np.linalg.norm(u)

cpdef carmete(int num,int x1,int y1,int x2,int
    y2,point):
  #s = time.time()
  cdef int i,d,ps=250,xx=0,yy=0
  (cx,cy) = (x1,y1)
  # if x2 < x1:
  #  tmp = x1
  #  x1 = x2
  #  x2 = tmp
  # if y2 < y1:
  #  tmp = y1
  #  y1 = y2
  #  y2 = tmp
  for i in range(0,len(point)):
   if i != num:
    (cx1,cy1,cx2,cy2,cx3,cy3,cx4,cy4) = point[i]
    #if (x1<=cx1<=x2 and y1<=cy1<=y2) or (x1<=cx2
       <=x2 and y1<=cy2<=y2) or (x1<=cx3<=x2 and
       y1<=cy3<=y2) or (x1<=cx4<=x2 and y1<=cy4<=
       y2):
    (x,y) = mete(x1,y1,x2,y2,cx1,cy1,cx2,cy2)
    if x != 0 and y != 0:
     d = distance(cx,cy,x,y)
     if ps > d or ps == 0:
      (xx,yy) = (x,y)
      ps = d
    (x,y) = mete(x1,y1,x2,y2,cx2,cy2,cx3,cy3)
    if x != 0 and y != 0:
     d = distance(cx,cy,x,y)
     if ps > d or ps == 0:
      (xx,yy) = (x,y)
      ps = d
    (x,y) = mete(x1,y1,x2,y2,cx3,cy3,cx4,cy4)
    if x != 0 and y != 0:
     d = distance(cx,cy,x,y)
     if ps > d or ps == 0:
      (xx,yy) = (x,y)
      ps = d
    (x,y) = mete(x1,y1,x2,y2,cx4,cy4,cx1,cy1)
    if x != 0 and y != 0:
     d = distance(cx,cy,x,y)
     if ps > d or ps == 0:
      (xx,yy) = (x,y)
      ps = d
  #print ("car_mete    :%f [sec]" % (time.time() -
      s))
  return ps,xx,yy

cpdef linemete(int num,int x1,int y1,int x2,int
    y2,point):
  cdef int i,d,ps=250,xx=0,yy=0
  (cx,cy) = (x1,y1)
  for i in range(0,len(point)):
   (lx1,ly1,lx2,ly2) = point[i]
   (x,y) = mete(x1,y1,x2,y2,lx1,ly1,lx2,ly2)
   if x != 0 and y != 0:
    d = distance(cx,cy,x,y)
    if ps > d or ps == 0:
     (xx,yy) = (x,y)
     ps = d

  return ps,xx,yy

cpdef wallmete(int x1,int y1,int x2,int y2,np.
    ndarray point1,np.ndarray point2,np.ndarray
    point3,np.ndarray point4):
  cdef int i,x=0,y=0,wx1,wy1,wx2,wy2,dis=250,xx
     =0,yy=0
  for i in range(0,len(point1)-1):
   (wx1,wy1) = point1[i]
```

```
    (wx2,wy2) = point1[i+1]
    (x,y) = mete(x1,y1,x2,y2,wx1,wy1,wx2,wy2)
    if x != 0 and y != 0:
     dis1 = distance(x1,y1,x,y)
     if dis1 <= dis:
      (xx,yy) = (x,y)
      dis = dis1

  for i in range(0,len(point2)-1):
    (wx1,wy1) = point2[i]
    (wx2,wy2) = point2[i+1]
    (x,y) = mete(x1,y1,x2,y2,wx1,wy1,wx2,wy2)
    if x != 0 and y != 0:
     dis1 = distance(x1,y1,x,y)
     if dis1 <= dis:
      (xx,yy) = (x,y)
      dis = dis1

  for i in range(0,len(point3)-1):
    (wx1,wy1) = point3[i]
    (wx2,wy2) = point3[i+1]
    (x,y) = mete(x1,y1,x2,y2,wx1,wy1,wx2,wy2)
    if x != 0 and y != 0:
     dis1 = distance(x1,y1,x,y)
     if dis1 <= dis:
      (xx,yy) = (x,y)
      dis = dis1

  for i in range(0,len(point4)-1):
    (wx1,wy1) = point4[i]
    (wx2,wy2) = point4[i+1]
    (x,y) = mete(x1,y1,x2,y2,wx1,wy1,wx2,wy2)
    if x != 0 and y != 0:
     dis1 = distance(x1,y1,x,y)
     if dis1 <= dis:
      (xx,yy) = (x,y)
      dis = dis1

  return dis,xx,yy

cpdef mete(int x1,int y1,int x2,int y2,int x3,
   int y3,int x4,int y4):
  cdef int dev,d1,d2

  dev = (y2-y1)*(x4-x3)-(x2-x1)*(y4-y3)

  if dev == 0:
   return (0,0)

  d1 = y3*x4-x3*y4
  d2 = y1*x2-x1*y2
  x = d1*(x2-x1)-d2*(x4-x3)
  x /= dev
  y = d1*(y2-y1)-d2*(y4-y3)
  y /= dev

  if x1 > x2:
   tmp = x1
   x1 = x2
   x2 = tmp

  if y1 > y2:
   tmp = y1
   y1 = y2
   y2 = tmp

  if x3 > x4:
   tmp = x3
   x3 = x4
   x4 = tmp

  if y3 > y4:
   tmp = y3
   y3 = y4
   y4 = tmp

  #print("x:%d y:%d (%d,%d):(%d,%d),(%d,%d):(%d,%
     d)" % (x,y,x1,y1,x2,y2,x3,y3,x4,y4))

  if (x1 <= x < x2 and x3 <= x < x4) or (y1 <= y
    < y2 and y3 <= y < y4):
   return (x,y)
  elif x1 == x2 and y3 == y4 and y1 < y < y2 and
    x3 < x < x4:
   return (x,y)
  elif y1 == y2 and x3 == x4 and x1 < x < x2 and
    y3 < y < y4:
   return (x,y)
  elif x3 == x4 and y1 == y2 and y3 < y < y4 and
    x1 < x < x2:
   return (x,y)
  elif y3 == y4 and x1 == x2 and x3 < x < x4 and
    y1 < y < y2:
   return (x,y)

  return (0,0)

# cpdef dot(a,b):
#   return a[0]*b[0]-a[1]*b[1]

# cpdef dot(a,b):
#   return a[0]*b[0]+a[1]*b[1]

# cpdef cross_l(a,b):
#   return a[0]*b[1]-a[1]*b[0]

# cpdef ccw(p1,p2,p3):
#   sw = 0
#   a[2],b[2],EPS=1.0e-8

#   a[0] = p2[0] - p1[0]
#   a[1] = p2[1] - p1[1]
#   b[0] = p3[0] - p1[0]
#   b[1] = p3[1] - p1[1]
#   if (cross_l(a, b) > EPS):
#    sw = 1
#   elif (cross_l(a, b) < -EPS):
#    sw = -1
#   elif (dot(a, b) < -EPS):
#    sw = 2
#   elif (norm(a) < norm(b)):
#    sw = -2

#   return sw


# cpdef islntersect(c1,c2,c3,c4,l1,l2):
#   int sw = 0

#   if (ccw(p1, p2, p5) * ccw(p1, p2, p6) <= 0
    and ccw(p5, p6, p1) * ccw(p5, p6, p2) <= 0 or
#    ccw(p2, p3, p5) * ccw(p2, p3, p6) <= 0 and
    ccw(p5, p6, p2) * ccw(p5, p6, p3) <= 0 or
#    ccw(p3, p4, p5) * ccw(p3, p4, p6) <= 0 and
    ccw(p5, p6, p3) * ccw(p5, p6, p4) <= 0 or
#    ccw(p4, p1, p5) * ccw(p4, p1, p6) <= 0 and
    ccw(p5, p6, p4) * ccw(p5, p6, p1) <= 0):
#    sw = 1

#   return sw
```