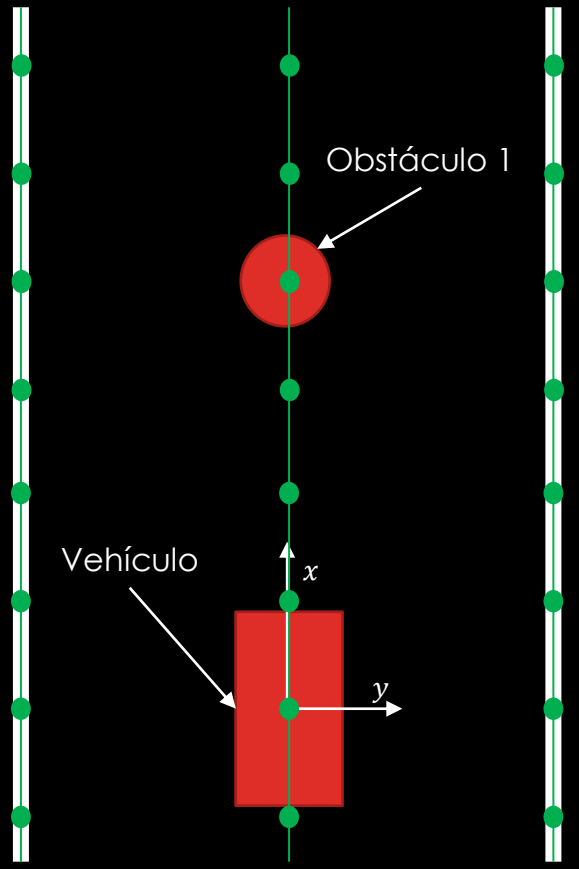
ESTADO DEL TFM 24.01.2021

Borja Pintos

OBJETIVO

- Determinar la trayectoria vehicular óptima en términos de seguridad y confort, conociendo de antemano tanto la posición del vehículo como la posición de los obstáculos de la calzada y los límites de la calzada.
- La posición del vehículo y las coordenadas de obstáculos y líneas de la calzada son determinadas por la percepción del vehículo, que utiliza sensores como cámaras, LIDAR, radar, GPS y IMU.
- La figura muestra un ejemplo de las coordenadas de un obstáculo y de las líneas que delimitan la calzada. La línea central se calcula como la línea equidistante entre las líneas izquierda y derecha, y también es un dato proporcionado por la percepción del vehículo.

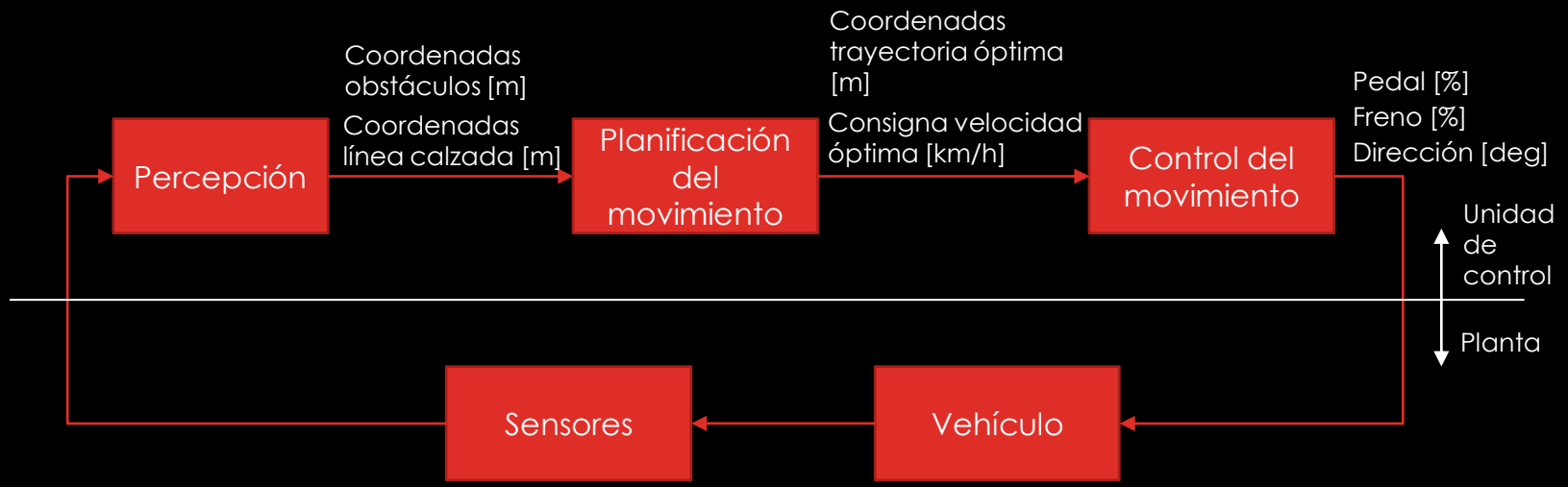


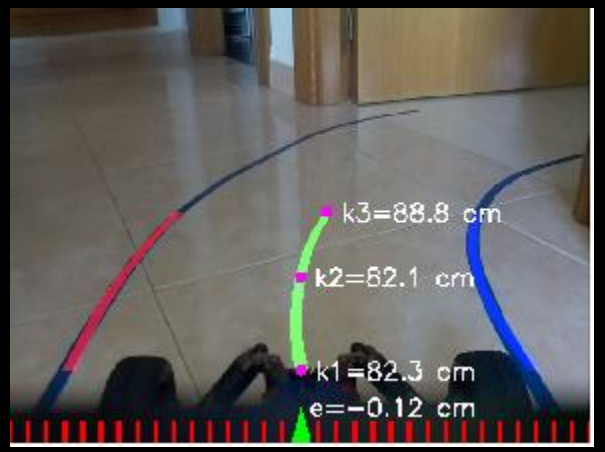
$$obs1 = ([4,0])$$
$$línea_{izq} = ([0,-2],[1,-2],[2,-2],\dots,[6,-2])$$
$$línea_{derecha} = ([0,2],[1,2],[2,2],\dots,[6,2])$$
$$línea_{central} = ([0,0],[1,0],[2,0],\dots,[6,0])$$

Coordenadas x e y en metros

ARQUITECTURA DEL SOFTWARE

- Tanto la posición del vehículo como la posición de los obstáculos de la calzada y los límites de la calzada se conocen de antemano, por lo que no hace falta programar la lógica de percepción.
- Las coordenadas de las líneas de la calzada se pueden detectar a través de una cámara. La siguiente figura es una imagen del proyecto realizado para la asignatura de visión artificial, donde una cámara montada en la Raspberry Pi detecta las líneas de la calzada y calcula la línea central.

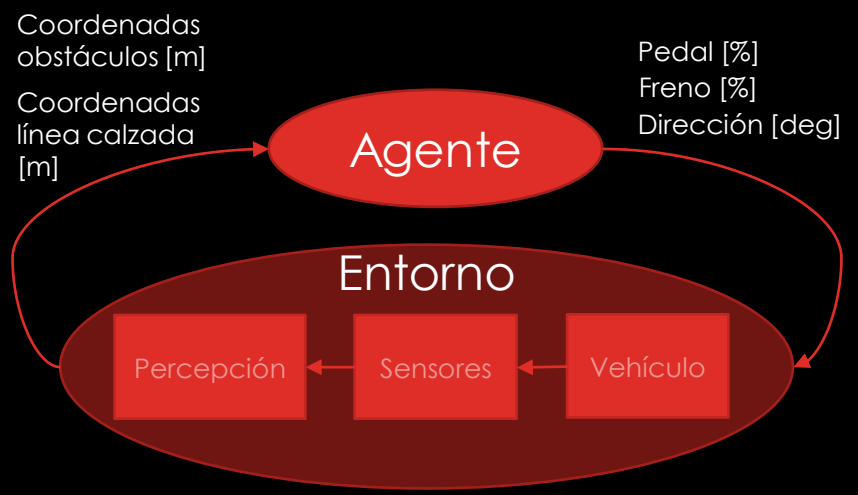


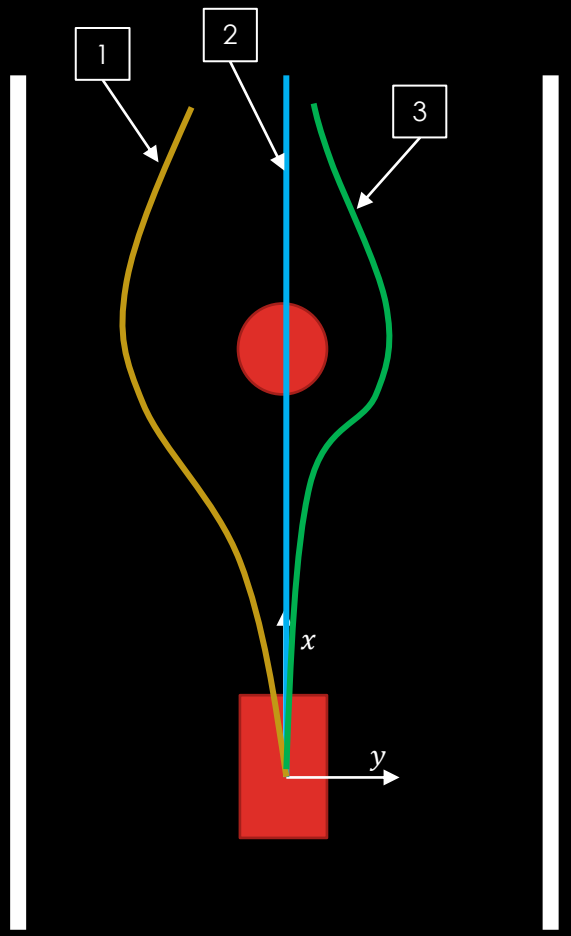


En la imagen, el valor denotado como k hace referencia al radio de curvatura de la línea en el punto seleccionado

ALGORITMO DE APRENDIZAJE POR REFUERZO

- Actualmente, la entrada del algoritmo de aprendizaje son las coordenadas de los obstáculos y las coordenadas de las líneas de la calzada.
- Las salidas son la posición del pedal, el freno y el ángulo de la dirección.
- El algoritmo de RL debe evaluar mediante la función de reward la calidad de la trayectoria generada.
- Por ejemplo, la trayectoria número 2 de la imagen es inaceptable, porque colisiona con un objeto, y la función reward debe penalizar este caso. La trayectoria 3 tiene un alto grado de curvatura, que genera un nivel alto de aceleración lateral. Esto hace que los ocupantes del vehículo perciban dicha trayectoria como incómoda y no segura. La trayectoria 1 sería la óptima, ya que no genera niveles altos de aceleración lateral (buena comodidad) y se aleja lo suficiente del obstáculo (buena seguridad).



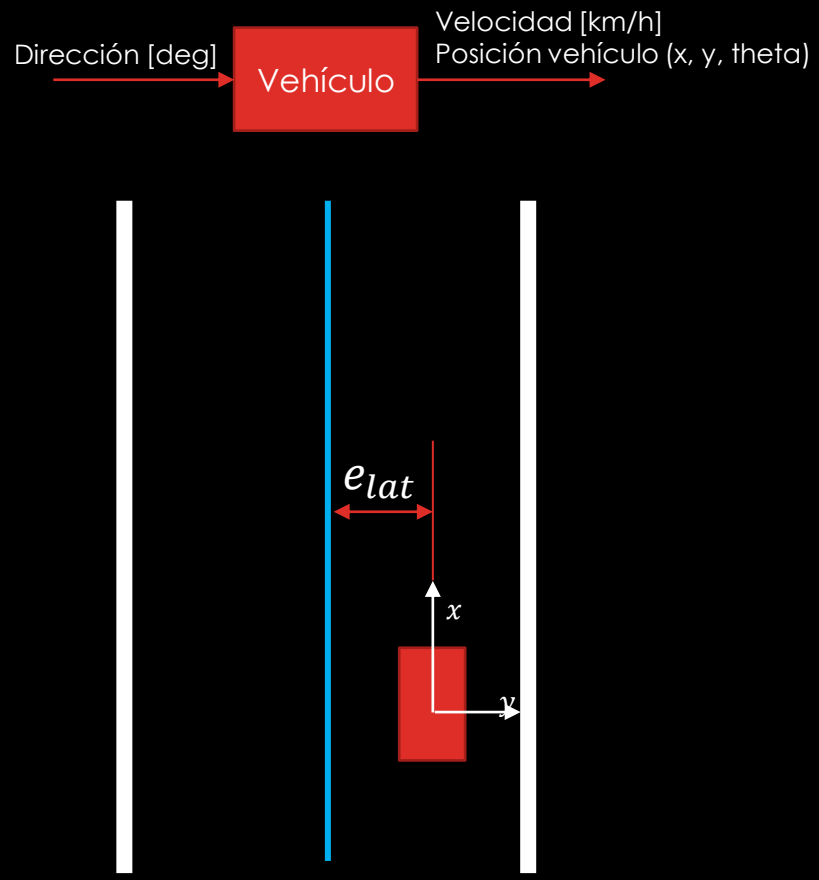


ALGORITMO DE APRENDIZAJE POR REFUERZO. VERSIÓN SIMPLIFICADA

- La primera versión del algoritmo tiene en cuenta dos simplificaciones:
  1. La velocidad del vehículo se mantiene constante a un valor predefinido. Por tanto, las variables de entrada del vehículo 'pedal' y 'freno' se eliminan, reduciendo los grados de libertad.
  2. No se consideran en un principio obstáculos, tan solo las líneas que delimitan la calzada y la línea central.
- En esta primera versión, se ha definido una función de reward para que se siga la línea central proporcionada por la percepción del vehículo:

$$reward = e^{-\frac{e_{lat}^2}{2\sigma^2}} - penalización$$

- Si la distancia lateral del vehículo a la línea central (ver imagen) es siempre cero, se suma a la función de reward el valor 1. Cuanto más nos alejemos de la línea central, más bajo es el valor de reward.
- Existe una penalización que se aplica solamente si el vehículo abandona los límites de la calzada. Si esto ocurriera, se reinicia el entorno y se comienza un nuevo episodio de aprendizaje.
- Teniendo en cuenta la definición anterior de la función de reward, el algoritmo de RL actúa simplemente como un algoritmo de control del movimiento, en vez de ser un algoritmo de planificación del movimiento. Es decir, el algoritmo recibe la trayectoria que se quiere recorrer, y se genera la posición de la dirección para seguir sin error esta trayectoria, tal y como lo haría, por ejemplo, un controlador PI.



ALGORITMO DE APRENDIZAJE POR REFUERZO. VERSIÓN SIMPLIFICADA

- Sin embargo, dada la función de reward de la anterior diapositiva, ¿Qué pasaría si se quiere conducir a lo largo de una curva de 90 grados?
- En este caso, la línea central (línea equidistante entre las líneas derecha e izquierda), no sería una trayectoria óptima, ya que podría generar valores muy altos de aceleración lateral, haciendo que los ocupantes del vehículo perciban esta trayectoria como incómoda.
- De manera simplificada, se puede decir que una trayectoria es cómoda si los valores de aceleración y jerk (derivada de la aceleración) longitudinal y lateral están siempre por debajo de un cierto límite.
- La aceleración lateral se calcula como:

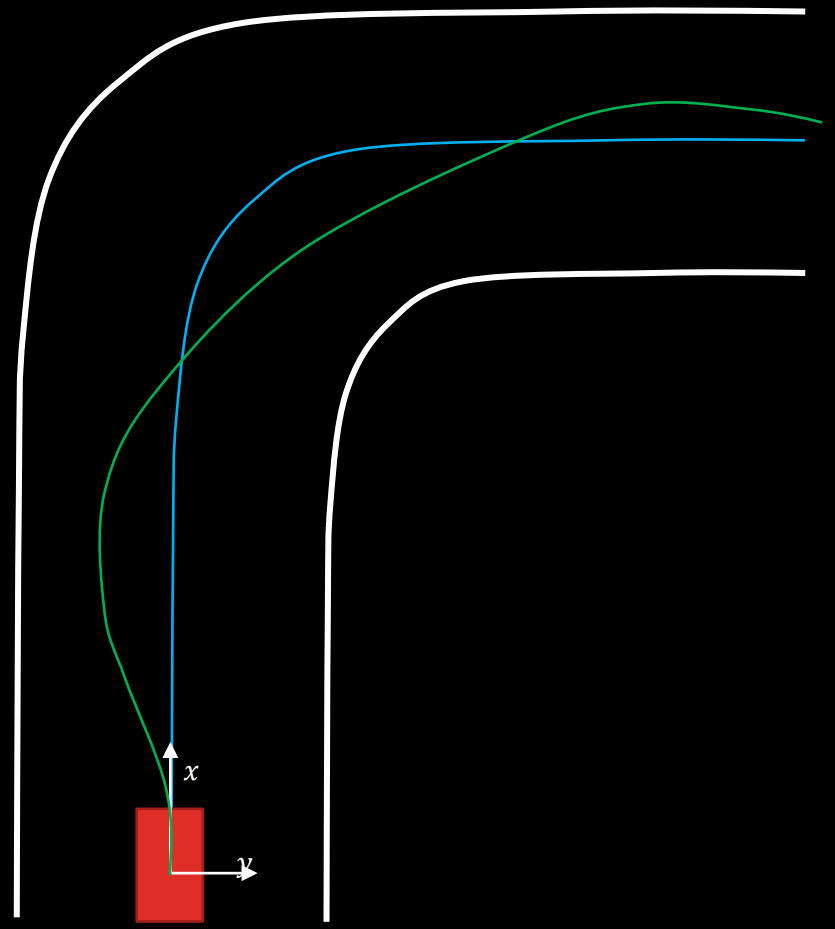
$$a_c = \frac{v^2}{R} \text{ siendo R} = \frac{L}{tan\delta}$$

$v = velocidad$
$R = radio\ de\ curvatura$
$\delta = ángulo\ de\ la\ dirección$
$L = distancia\ entre\ ejes\ delantero\ y\ trasero$

- La función de reward se podría adaptar de la siguiente manera, siempre y cuando la aceleración lateral supere un límite mínimo. De esta forma se esperaría que el algoritmo de RL ya no tenga como objetivo simplemente seguir la línea central (línea azul), sino que genere una trayectoria óptima en cuanto a niveles de aceleración lateral (línea verde en la imagen):

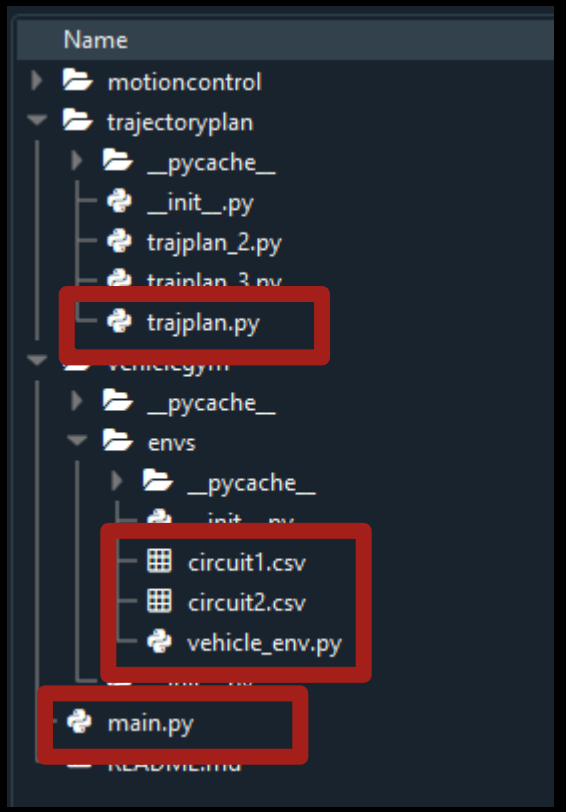
$$reward = e^{-\frac{e_{lat}^2}{2\sigma^2}} - e^{-\frac{\left(\frac{a_c}{a_{c_{max}}}\right)^2}{2\sigma^2}} - penalización$$

- En el caso de incluir el pedal como grado de libertad adicional, aumentaría la complejidad de escribir la función de reward de manera correcta.



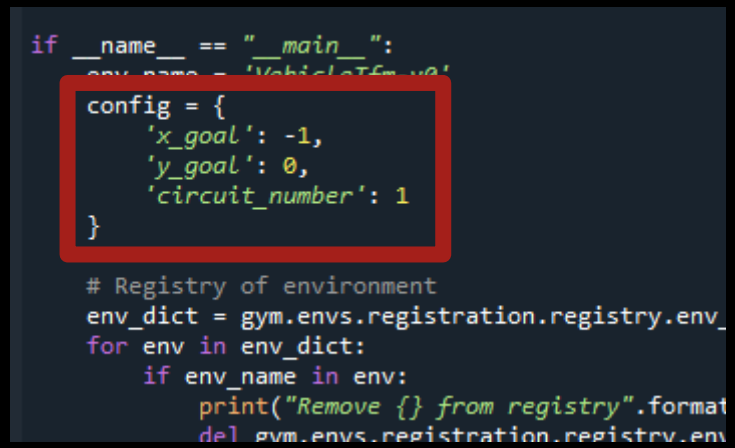
ESTRUCTURA DEL PROYECTO

- El proyecto está actualmente organizado en dos paquetes de software:
  1. La carpeta vehiclegym, donde se encuentra la inicialización del modelo del vehículo, la simulación de su dinámica, el renderizado del vehículo y de su entorno y la definición de la función reward.
  2. La carpeta trajectoryplan, donde se encuentra el algoritmo de aprendizaje por refuerzo
- El script main integra los dos paquetes de software.
- Dentro de la carpeta vehiclegym hay guardadas diferentes configuraciones de circuitos. Estos circuitos están guardados en formato .csv y contienen los puntos de la línea central de la calzada.



SCRIPT MAIN

- EL script main lanza la simulación y el algoritmo de aprendizaje por refuerzo comienza los episodios de aprendizaje.
- La variable config, mostrada en la imagen, permite seleccionar el circuito y las coordenadas de destino del vehículo.
- Si el vehículo llega a las coordenadas de destino, el episodio de aprendizaje se dará por finalizado y se iniciará otro episodio nuevo.
- En el caso de que el vehículo abandone los límites de la calzada, también se finalizará el episodio de aprendizaje y se inicializará otro nuevo. Además, la función de reward será penalizada con un valor negativo por salirse de la calzada. Esto se explica también más adelante dentro de la documentación del paquete vehiclegym.
- El archivo main también contiene la calibración de ciertos parámetros necesarios para el aprendizaje de la red neuronal (por ejemplo, el parámetro tasa de aprendizaje). Estos parámetros se explican dentro del paquete trajectoryplan.



PAQUETE VEHICLEGYM

- El script vehicle_env contiene una clase que define el entorno del algoritmo de aprendizaje con la interfaz necesaria para que funcione con el paquete Gym de Python.
- Este entorno del algoritmo de aprendizaje contiene, entre otras cosas, la simulación de la dinámica del vehículo.
- La clase implementa 4 métodos esenciales para que un agente de aprendizaje por refuerzo sea capaz de correr sin problemas:
    1. Método __init__: inicializa las variables
    2. Método step: ejecuta un paso de la simulación de la dinámica del vehículo. La dinámica del vehículo se ha modelado con OpenModelica y se importa a Python mediante una función FMU.
    3. Método reset: se resetea el estado del vehículo a un estado inicial. Cada vez que se finaliza un episodio de aprendizaje, se resetea el entorno a este estado inicial. En este caso, se resetea el estado del vehículo a una posición inicial y a un ángulo inicial del vehículo (x, y, theta).
    4. Método render: se renderiza la posición del vehículo en cada instante de tiempo. Además se grafica el circuito seleccionado y los obstáculos.

```
74
75      def step(self, action):
76          # Execute one time step within the environment
77
78          # Convert steering from normalized value to real value
79          action = action[0]*self.max_steering
80          action = [np.array(action)]
81          self.model.set(list(['delta']), list(action))
82
83          # Simulation options
84          opts = self.model.simulate_options()
85          opts['ncp'] = 10
86          opts['initialize'] = False
87
88          # Run the simulation one sample time
89          res = self.model.simulate(start_time=self.start_time, final_time=self.stop_time, options=opts)
90
91          # Get sensor measurements from the vehicle model
92          self.sensors = tuple([res.final(k) for k in ['x','y','theta_out']])
93
94          # Get the state forwarded to the agent
95          self.state = self._lateralcalc()
96
97          # Check if environment is terminated
98          self.done = self._is_done()
99
100         # If environment is not terminated, increase simulation start and stop times one sample time
101         if not self.done:
102             self.start_time = self.stop_time
103             self.stop_time += self.sample_time
104
105         return self.state, self._reward(action), self.done, {}
```

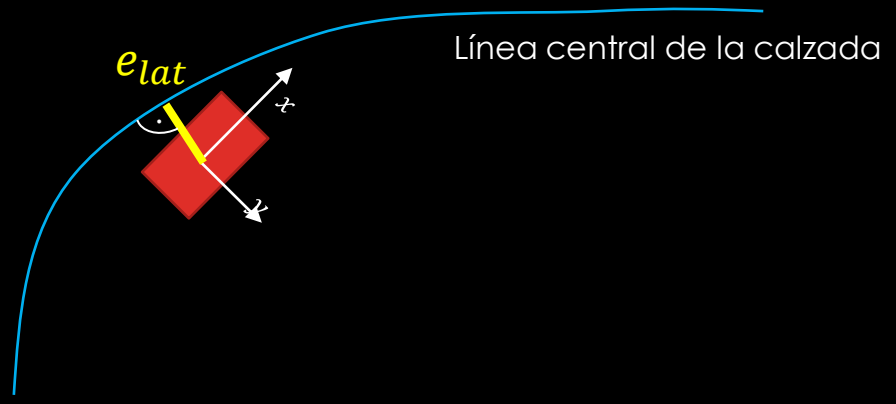
PAQUETE VEHICLEGYM

- Otros métodos implementados en esta clase son los siguientes:
  1. Método _reward: método donde se implementa la función de reward. La función de reward se define de la siguiente manera:

     $$reward = e^{-\frac{e_{lat}^2}{2\sigma^2}} - penalización$$

     Donde e_{lat} es la distancia lateral entre el vehículo y la línea central de la calzada. La penalización se aplica solamente cuando el vehículo abandona los límites de la calzada.
  2. Método _lateralcalc: esta función calcula la distancia lateral entre la posición actual del vehículo y la línea central. La distancia lateral se define como la distancia menor entre la posición del vehículo y la línea central (ver imagen).
  3. Método _isdone: método que evalúa si se finaliza el episodio de aprendizaje o no. El episodio de aprendizaje se finaliza si el vehículo abandona la calzada, si se llega al destino final o si se sobrepasa un tiempo máximo de simulación.
  4. Método close: simplemente cierra la ventana de renderizado.

```python
    def _reward(self, action):
        # Calculate reward value

        # action = action[0]

        # Get state
        e_lat = self.state

        # Reward due to vehicle exceeding road boundaries
        reward_out_boundaries = -100

        # Reward if vehicle follows the center lane of the road
        lane_std = 0.05
        reward = np.exp(-e_lat**2/(2*lane_std**2))

        # Reward out of boundaries only applied if vehicle exceedes road boundaries
        if abs(e_lat) > 1:
            reward = reward + reward_out_boundaries

        return reward
```



PAQUETE TRAJECTORYPLAN

- El algoritmo de aprendizaje se llama deep deterministic policy gradient (DDPG) y ha sido obtenido a través de un ejemplo de los tutoriales de Keras (https://keras.io/examples/rl/ddpg_pendulum/).
- Este algoritmo es capaz de aprender acciones continuas a partir de observaciones también continuas.
- Utiliza dos redes neuronales: una para la política de acciones, llamada actor model, y otra que evalúa si la acción seleccionada es buena o no, llamada critic model.
- Al valor de la acción generado por la red neuronal actor model se le suma un valor aleatorio que sirve para explorar nuevos escenarios. Este valor aleatorio es generado de acuerdo con el proceso Ornstein-Uhlenbeck. La generación de la acción se ejecuta en la línea 94 del script main.
- Una vez generada la acción se envía al modelo del vehículo y se ejecuta un paso de simulación. Dicha acción tiene como consecuencia un nuevo estado que genera una nueva observación de dicho estado y un valor de la función de reward (línea 96).

```
82      # Takes about 4 min
83      for ep in range(total_episodes):
84          prev_state = env.reset()
85          episodic_reward = 0
86
87          while True:
88              # Uncomment this to see the Actor in action
89              # But not in a python notebook.
90              env.render()
91
92              tf_prev_state = tf.expand_dims(tf.convert_to_tensor(prev_state), 0)
93
94              action = policy(tf_prev_state, ou_noise, actor_model, lower_bound, upper_bound)
95              # Recieve state and reward from environment.
96              state, reward, done, info = env.step(action)
97              print("Reward ->  {}".format(reward))
98              print("Action ->  {}".format(action))
99
100             buffer.record((prev_state, action, reward, state))
101             episodic_reward += reward
102
103             buffer.learn(target_actor, target_critic, actor_model, critic_model,\
104               actor_optimizer, critic_optimizer, gamma)
105             update_target(target_actor.variables, actor_model.variables, tau)
106             update_target(target_critic.variables, critic_model.variables, tau)
107
108             # End this episode when `done` is True
109             if done:
110                 break
111
112             prev_state = state
113
114         ep_reward_list.append(episodic_reward)
115
116         # Mean of last 40 episodes
117         avg_reward = np.mean(ep_reward_list[-40:])
118         print("Episode * {} * Avg Reward is ==> {}".format(ep, avg_reward))
119         avg_reward_list.append(avg_reward)
```

PAQUETE TRAJECTORYPLAN

- Tras obtener la nueva observación y el nuevo valor de reward, se guardan la observación previa, la nueva observación, la acción y el valor de reward en un buffer.
- El algoritmo no utiliza un único valor de la acción, observación y reward para aprender, sino que escoge de manera aleatoria una cantidad n de valores de dicho buffer para realizar el episodio de aprendizaje.
- Para añadir estabilidad al algoritmo de aprendizaje, se utilizan dos redes neuronales para el actor model y otras dos para el critic model. Esta segunda red neuronal se denomina target network. La diferencia es que los pesos de las target networks se van actualizando de manera mucho más lenta que los pesos de sus redes neuronales homólogas. Para fijar la velocidad de aprendizaje de las target networks existe un parámetro llamado tau.
- Para actualizar los pesos de las redes neuronales se utiliza el algoritmo DDPG, que se muestra en la siguiente imagen.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

PAQUETE TRAJECTORYPLAN

- Los parámetros que se utilizan para el entrenamiento de las redes neuronales son los siguientes:
  1. Desviación estándar del ruido generado para explorar nuevas acciones
  2. Tasa de aprendizaje de las redes neuronales critic model y actor model
  3. Número total de episodios de aprendizaje
  4. Valor de descuento para valores de reward futuros
  5. Velocidad de actualización de los pesos de las target networks

```
49      std_dev = 0.2
50      ou_noise = OUActionNoise(mean=np.zeros(1), std_deviation=float(std_dev) * np.ones(1))
51  1
52      actor_model = get_actor(num_states, upper_bound)
53      critic_model = get_critic(num_states, num_actions)
54
55      target_actor = get_actor(num_states, upper_bound)
56      target_critic = get_critic(num_states, num_actions)
57
58      # Making the weights equal initially
59      target_actor.set_weights(actor_model.get_weights())
60      target_critic.set_weights(critic_model.get_weights())
61
62      # Learning rate for actor-critic models
63      critic_lr = 0.002
64      actor_lr = 0.001                                    2
65
66      critic_optimizer = tf.keras.optimizers.Adam(critic_lr)
67      actor_optimizer = tf.keras.optimizers.Adam(actor_lr)
68
69      total_episodes = 100                                3
70      # Discount factor for future rewards
71      gamma = 0.99                                        4
72      # Used to update target networks
73      tau = 0.005                                         5
74
75      buffer = Buffer(num_states, num_actions, 50000, 64)
```