Universidad EAFIT

Trabajos de semilleros de investigación

Simulación del impacto de motocicletas electricas en Colombia

Semillero de investigación BID

Autores:
Juan Pablo Castaño Morales
Junio 6 de 2025

Introducción

En este informe se presenta el modelamiento matemático y físico de una motocicleta híbrida y eléctrica mediante múltiples capas:

1. Definición de parámetros en los archivos `parameters_electric.py` y `parameters_hybrid.py`.
2. Formulación de las ecuaciones diferenciales en `functions.py` y `model.py`, incluyendo el integrador Runge–Kutta de cuarto orden (RK4) y funciones auxiliares.
3. Implementación del modelo simplificado de partícula en `ModeloMoto1.py`, donde se emplean ecuaciones discretas para fuerzas, potencia y lógica de recarga.
4. Extensión a simulación basada en agentes en `AgentBasedModel.py`, que reutiliza las mismas formulaciones discretas y añade interacción entre múltiples unidades.

Cada sección a continuación detalla el tratamiento matemático asociado, las decisiones binarias y la manera en que se implementó cada bloque en código. Se incluyen tablas que muestran los valores numéricos de todos los parámetros utilizados.

Parámetros del Vehículo y Medio Ambiente

A continuación se presentan las tablas con líneas horizontales y verticales que contienen todos los parámetros necesarios para el modelo eléctrico y el híbrido, agrupados por categorías. Para mayor claridad, se incluyen unidades y, cuando corresponda, fórmulas auxiliares.

Table 1: Parámetros básicos del vehículo y entorno

Parámetro	**Descripción**	**Valor**
m	Masa total (chasis + componentes)	150 kg (eléctrico) 200 kg (híbrido)
A	Área frontal	0.6 m^2
C_d	Coeficiente de arrastre aerodinámico	0.7
C_{rr}	Coeficiente de rodadura	0.01
g	Aceleración de la gravedad	9.8 m/s^2
ρ	Densidad del aire	1.21 kg/m^3
r_w	Radio de rueda	0.2667 m
η_{tren}	Eficiencia del tren motriz eléctrico	0.7

Table 2: Parámetros eléctricos: batería y convertidor Buck

Parámetro	**Descripción**	**Valor**
$E_{\max}$	Capacidad nominal de la batería	700 Wh
E_{umbral}	Umbral mínimo de operación (20% SOC)	$0.2\, E_{\max} = 140$ Wh
L_b	Inductancia del inductor del Buck	0.001 H
R_b	Resistencia interna del inductor del Buck	0.05 Ω
V_{bat}	Voltaje nominal de la batería	48 V
C_{bat}	Capacidad de la batería (Ah)	14.6 Ah
r_1	Resistencia interna de primer polo (Thevenin)	0.165 Ω
c_1	Capacitancia de primer polo (Thevenin)	43.8 F
r_0	Resistencia interna de segundo polo	0.108 Ω
OCV	Voltaje de circuito abierto	74 V
n_{coulomb}	Eficiencia de Coulomb	1.0

Table 3: Parámetros del motor eléctrico y motor de combustión

Parámetro	**Descripción**	**Valor**
R	Resistencia de la armadura del motor eléctrico	$0.1\ \Omega$
L	Inductancia del bobinado del motor eléctrico	0.002 H
k_e	Constante electromotriz (back-emf)	0.1 V s/rad
k_t	Constante de torque del motor eléctrico	0.1 N m/A
J_m	Momento de inercia rotor motor eléctrico	$0.02\ \mathrm{kg\,m^2}$
R_{cil}	Resistencia interna del motor térmico	$0.05\ \mathrm{kg\,m^2}$
J_{ec}	Momento de inercia del motor térmico	$0.05\ \mathrm{kg\,m^2}$
$\eta_{\text{térmica}}$	Eficiencia del motor térmico	0.8
h	Modo de operación (0: eléctrico, 1: térmico)	Variable binaria
$G_i,\ i = 1, \ldots, 5$	Relación de engranajes para marcha i (motor térmico)	$G_1 = 3.0833,\ G_2 = 1.8824,\ G_3 = 1.4,$ $G_4 = 1.1304,\ G_5 = 0.96$

Tratamiento Matemático de los Archivos de Parámetros

`parameters_electric.py` Este módulo define la clase `HEV` para el modelo puramente eléctrico. En el constructor de `HEV` se inicializan los atributos con los valores numéricos de las Tablas 1 y 2. De forma explícita, los parámetros relevantes son:

Parámetro	**Valor**
m	150 kg
A	$0.6\ \mathrm{m^2}$
C_d	0.7
C_{rr}	0.01
g	$9.8\ \mathrm{m/s^2}$
ρ	$1.21\ \mathrm{kg/m^3}$
r_w	0.2667 m
η_{tren}	0.7
$E_{\max}$	700 Wh
E_{umbral}	$0.2 \cdot E_{\max} = 140$ Wh
L_b	0.001 H
R_b	$0.05\ \Omega$
V_{bat}	48 V
C_{bat}	14.6 Ah
r_1	$0.165\ \Omega$
c_1	43.8 F
r_0	$0.108\ \Omega$
OCV	74 V
n_{coulomb}	1.0

Estos parámetros se emplean directamente en las ecuaciones de la batería y del convertidor Buck que se describen en las ecuaciones diferenciales.

`parameters_hybrid.py` Este módulo extiende `parameters_electric.py` añadiendo los parámetros del motor de combustión interna (ICE). La clase `HEV` hereda los atributos del modelo eléctrico y, adicionalmente:

$$\begin{aligned}
&h \in \{0, 1\} \quad \text{(modo eléctrico o térmico)},\\
&\eta_{\text{térmica}} = 0.8,\\
&J_m = 0.02\ \mathrm{kg\,m^2}, \quad J_{\text{ec}} = 0.05\ \mathrm{kg\,m^2},\\
&G_i\ :\ \text{relación de transmisión en cada marcha } i, \quad i = 1, \ldots, 5,
\end{aligned}$$

donde las relaciones de marcha se definen en código como:

$$G_1 = \frac{c_1}{s_1} \cdot \frac{1}{\texttt{Chain.rc}}, \quad G_2 = \frac{c_2}{s_2} \cdot \frac{1}{\texttt{Chain.rc}}, \ \ldots, \ G_5 = \frac{c_5}{s_5} \cdot \frac{1}{\texttt{Chain.rc}}.$$

Los momentos de inercia J_m y J_{ec} aparecen en las ecuaciones rotacionales del motor eléctrico.

Funciones y Ecuaciones Diferenciales

En este bloque se describen las funciones clave de los archivos `functions.py` y `model.py`, que implementan las ecuaciones diferenciales continuas y las decisiones binarias.

Integrador Runge–Kutta 4to (`functions.py`)

La función principal de integración en `functions.py` es:

$$\mathtt{RK}(t_{\mathrm{sim}},\, \Delta t,\, \mathbf{x}_0,\, \delta,\, h_{\mathrm{cont}},\, v_{\text{últ}}),$$

donde:

- t_{sim} es el instante actual de simulación.
- Δt es el paso de integración (e.g., $0.01\,\mathrm{s}$).
- $\mathbf{x}_0$ es el vector de estado en t_{sim}.
- δ es el vector de controles.
- h_{cont} es 0 para modo eléctrico o 1 para modo térmico.
- $v_{\text{últ}}$ es la velocidad del paso anterior (para cálculo de inercia).

Se denota el vector de estado como:

$$\mathbf{X}(t)^\top \;=\; \begin{bmatrix} v(t) & x(t) & i_{\mathrm{ind}}(t) & i_m(t) & u_m(t) & \mathrm{SOC}(t) & \omega_{\mathrm{ice}}(t) & i_{r1}(t) & p_{eb}(t) \end{bmatrix}$$

con $v(t)$ velocidad vehicular [m/s], $x(t)$ posición [m], $i_{\mathrm{ind}}(t)$ corriente de inductor [A], $i_m(t)$ corriente de motor [A], $u_m(t)$ voltaje de motor [V], $\mathrm{SOC}(t)$ estado de carga [%], $\omega_{\mathrm{ice}}(t)$ velocidad angular del cigüeñal [rad/s], $i_{r1}(t)$ corriente en la rama Thevenin [A] y $p_{eb}(t)$ potencia eléctrica instantánea [W]. La función interna `model` calcula:

$$\dot{\mathbf{X}} = \mathbf{F}\big(\mathbf{X}(t),\, \delta(t)\big).$$

El esquema RK4 se expresa como:

$$\begin{aligned}
k_1 &= \mathbf{F}\big(\mathbf{X}_n,\, \delta_n\big),\\
k_2 &= \mathbf{F}\Big(\mathbf{X}_n + \frac{\Delta t}{2}\, k_1,\, \delta_n\Big),\\
k_3 &= \mathbf{F}\Big(\mathbf{X}_n + \frac{\Delta t}{2}\, k_2,\, \delta_n\Big),\\
k_4 &= \mathbf{F}\Big(\mathbf{X}_n + \Delta t\, k_3,\, \delta_n\Big),\\
\mathbf{X}_{n+1} &= \mathbf{X}_n + \frac{\Delta t}{6}\,\big(k_1 + 2\,k_2 + 2\,k_3 + k_4\big).
\end{aligned}$$

Matemáticamente, este método proporciona error local $\mathcal{O}(\Delta t^5)$ y error global $\mathcal{O}(\Delta t^4)$, lo cual es adecuado para sistemas no lineales acoplados.

Ecuaciones del modelo continuo (`model.py`)

La función interna `model` en `model.py` define cada componente de $\dot{\mathbf{X}}$ de la siguiente manera:

1. Dinámica traslacional

$$m\,\dot{v}(t) = F_{\text{motor}}(t) - \big[F_{\text{aero}}(v) + F_{\text{rodadura}}(\theta) + F_g(\theta)\big].$$

Aquí:

$$F_{\text{aero}}(v) = \frac{1}{2}\,\rho\,A\,C_d\,v^2, \quad F_{\text{rodadura}}(\theta) = m\,g\,C_{rr}\,\cos\big(\theta(t)\big), \quad F_g(\theta) = m\,g\,\sin\big(\theta(t)\big).$$

El término $F_{\text{motor}}(t)$ se obtiene del torque total ajustado por la eficiencia del tren motriz y la relación de transmisión final:

$$F_{\text{motor}}(t) = \frac{T_{\text{motor}}(t)\,\eta_{\text{tren}}}{r_w\,G_{\text{total}}(t)},$$

donde:

$$T_{\text{motor}}(t) = \begin{cases} k_t\,i_m(t), & \text{si } h(t) = 0 \quad \text{(modo eléctrico)}, \\ T_{\text{ec}}\big(\dot{m}_{\text{comb}}(t)\big), & \text{si } h(t) = 1 \quad \text{(modo térmico)}, \end{cases}$$

y $G_{\text{total}}(t) = G_i$ según la marcha seleccionada (ver más abajo).

2. Dinámica rotacional del motor eléctrico

$$\begin{cases} L\,\dfrac{di_{\text{ind}}(t)}{dt} & = V_{\text{bat}}(t) - R\,i_{\text{ind}}(t) - k_e\,\omega_m(t), \\ J_m\,\dfrac{d\omega_m(t)}{dt} & = k_t\,i_m(t) - T_{\text{carga}}(t), \end{cases}$$

donde:

- $i_{\text{ind}}(t)$ es la corriente en el inductor de armadura [A].
- $\omega_m(t)$ es la velocidad angular del rotor [rad/s], con $\omega_m(t) = \omega_w(t)$ al simplificar relaciones de engranaje.
- $T_{\text{carga}}(t)$ agrupa las pérdidas mecánicas (viscosa, fricción) y el torque reflejado de la dinámica traslacional:

$$T_{\text{carga}}(t) = \big(F_{\text{aero}}(v) + F_{\text{rodadura}}(\theta) + F_g(\theta)\big)\,r_w.$$

3. Dinámica rotacional del motor de combustión interna

$$J_{\text{ec}}\,\frac{d\omega_{\text{ice}}(t)}{dt} = T_{\text{ec}}\big(\dot{m}_{\text{comb}}(t)\big) - T_{\text{resistencia}}(t),$$

donde:

- $\omega_{\text{ice}}(t)$ es la velocidad angular del motor [rad/s].
- $T_{\text{ec}}(\dot{m}_{\text{comb}})$ se obtiene del flujo de combustible $\dot{m}_{\text{comb}}(t)$ mediante:

$$T_{\text{ec}}\big(\dot{m}_{\text{comb}}\big) = \frac{h_u\,\eta_{\text{térmica}}}{\omega_{\text{ice}}(t)}\,\dot{m}_{\text{comb}}(t), \quad (h_u \text{ constante de energía del combustible}),$$

  salvo cuando $\omega_{\text{ice}} = 0$, en cuyo caso $T_{\text{ec}} = 0$.

- $T_{\text{resistencia}}(t)$ incluye la fricción interna y el torque reflejado del tren motriz según la marcha:

$$T_{\text{resistencia}}(t) = \big(F_{\text{aero}} + F_{\text{rodadura}} + F_g\big)\,\frac{1}{G_{\text{total}}(t)} + f_{\text{viscosa}}\,\omega_{\text{ice}}(t).$$

4. Circuito del convertidor Buck y batería El modelo de batería de primer orden se describe mediante:

$$\begin{cases} L_b \dfrac{di_b(t)}{dt} &= V_{\text{bat}}(t) - V_s(t) - R_b\, i_b(t), \\ \dfrac{d\text{SOC}(t)}{dt} &= -\dfrac{i_b(t)}{C_{\text{bat}}}, \end{cases}$$

donde:

- $i_b(t)$ es la corriente que fluye en el inductor del Buck [A].
- $V_s(t)$ es el voltaje de salida hacia el motor, derivado de la relación $V_s = \delta(t)\, U_b(t)$, con $\delta(t)$ el ciclo de trabajo.
- $V_{\text{bat}}(t)$ es la tensión terminal de la batería, dada por:

  $$V_{\text{bat}}(t) = \text{OCV}(t) - r_1\, i_{r1}(t) - r_0\, i_b(t),$$

  donde $\text{OCV}(t) \approx 74$ V (constante) y $i_{r1}(t)$ satisface:

  $$\frac{di_{r1}(t)}{dt} = -\frac{1}{r_1\, c_1}\, i_{r1}(t) + \frac{1}{r_1\, c_1}\, i_b(t).$$

- C_{bat} es la capacidad en amperios-hora convertida a coulombs: $C_{\text{bat}}(\text{C}) = C_{\text{bat}}(\text{Ah}) \times 3600$.
- El término $-\frac{i_b(t)}{C_{\text{bat}}}$ expresa la disminución del SOC, considerando eficiencia de Coulomb igual a 1.

5. Funciones binarias En `model.py` existen varias condiciones implementadas en notación de función escalón o pieza a pieza, traducidas en código con `if/else`. A modo de ejemplo:

- **Control de frenado:**

  $$\text{brake}(y_c) = \begin{cases} -\, y_c, & -1 \le y_c \le 0, \\ 0, & \text{en otro caso}, \end{cases} \qquad y_c \in [-1, 1].$$

  En código, esto se implementa como:

```
if y_c >= -1 and y_c <= 0:
    brake = -y_c
else:
    brake = 0
```

- **Selección de marcha (γ) según velocidad $v(t)$ y modo híbrido h:**

  $$\gamma = \begin{cases} 5, & \text{si } h = 0 \quad (\text{modo eléctrico}), \\ 1, & \text{si } h = 1 \text{ y } v < \frac{10}{3.6} \text{ m/s}, \\ 2, & \text{si } h = 1 \text{ y } \frac{10}{3.6} \le v < \frac{15}{3.6}, \\ 3, & \text{si } h = 1 \text{ y } \frac{15}{3.6} \le v < \frac{25}{3.6}, \\ 4, & \text{si } h = 1 \text{ y } \frac{25}{3.6} \le v < \frac{41}{3.6}, \\ 5, & \text{si } h = 1 \text{ y } v \ge \frac{41}{3.6}, \end{cases}$$

  implementado en código como:

```
if hybrid_cont == 0:
    gamma = 5
elif v < 10/3.6:
    gamma = 1
elif v < 15/3.6:
```

```
        gamma = 2
    elif v < 25/3.6:
        gamma = 3
    elif v < 41/3.6:
        gamma = 4
    else:
        gamma = 5
```

- **Cambio de modo híbrido:**

$$h(t) = \begin{cases} 0, & \text{si la estrategia de control determina operación eléctrica,} \\ 1, & \text{si determina operación térmica,} \end{cases}$$

usualmente basado en la comparación entre demanda de potencia y estado de carga (SOC).

Perfil de Velocidades (`wmtc_profile` en `functions.py`)

La función `wmtc_profile` interpola un vector discreto de velocidades $\{v_k\}$ para generar un perfil de velocidad continuo en el tiempo:

$$\texttt{wmtc\_profile}(t_s,\, t_{\text{ini}},\, t_{\text{fin}},\, \{v_k\}_{k=0}^{N}),$$

donde t_s es el paso de muestreo t_{ini} y t_{fin} definen el intervalo de simulación, y $\{v_k\}$ corresponde a valores de velocidad tomados a intervalos unitarios (1 s). Matemáticamente:

$$\begin{aligned} &\text{Definir } t_q = \big\{\, t_{\text{ini}},\, t_{\text{ini}} + t_s,\, \ldots,\, t_{\text{fin}} \,\big\}, \\ &\text{y un vector índice } k = 0, 1, \ldots, N-1. \\ &\text{Entonces } v_{\text{profile}}(t_q) = \text{interp}\big(\{v_k\},\, k,\, (t_q - t_{\text{ini}})\big), \end{aligned}$$

donde se realiza interpolación lineal entre los puntos discretos. El resultado es un vector de velocidades con resolución t_s apropiada para el integrador RK4.

Modelado (`ModeloMoto1.py`)

En `ModeloMoto1.py` se implementa un modelo discreto de "partícula" que recorre perfiles de velocidad y pendiente obtenidos de un archivo `.pkl`. A continuación se describen los bloques matemáticos y las funciones binarias incluidas, con notación consistente en todas las derivadas.

Conversión de Unidades y Definición de Variables

Para cada paso i (índice entero que recorre los vectores originales):

$$v_i^* = \frac{v_i}{3.6} \ \ [\text{m/s}], \quad \theta_i^* = \theta_i \, \frac{\pi}{180} \ \ [\text{rad}],$$

donde v_i está en km/h y θ_i en grados. Además, se definen los siguientes estados discretos:

$$\begin{aligned} \text{SOC}_i &: \text{Estado de carga en Wh en el paso } i, \\ \text{en\_recarga}_i &: \begin{cases} 1, & \text{si está recargando en } i, \\ 0, & \text{si no,} \end{cases} \\ \text{tiempo\_en\_recarga}_i &: \text{Número de pasos transcurridos en recarga,} \\ \text{posiciones}_i &: \text{Par de coordenadas geográficas en el paso } i. \end{aligned}$$

Cálculo de Fuerzas (Paso i)

En cada iteración discreta (asumiendo $\Delta t = 1\,\mathrm{s}$), se calculan:

$$\begin{aligned}
F_{\text{aero},\,i} &= \frac{1}{2}\,\rho\,A\,C_d\,\big(v_i^*\big)^2,\\
F_{\text{rodadura},\,i} &= m\,g\,C_{rr}\,\cos\!\big(\theta_i^*\big),\\
F_{g,\,i} &= m\,g\,\sin\!\big(\theta_i^*\big),\\
\Delta v &= v_i^* - v_{i-1}^*, \quad (\text{con } v_{-1}^* = 0),\\
F_{\text{inercia},\,i} &= m\,\frac{\Delta v}{\Delta t},\\
F_{\text{total},\,i} &= F_{\text{aero},\,i} + F_{\text{rodadura},\,i} + F_{g,\,i} + F_{\text{inercia},\,i}.
\end{aligned}$$

Todas las derivadas en este bloque están expresadas con notación de diferencia finita. Obsérvese que $F_{\text{inercia},\,i}$ es una aproximación discreta a $m\,\dot{v}$.

Cálculo de Potencia y Energía

De la fuerza total se obtiene la potencia en la rueda:

$$P_{\text{rueda},\,i} = F_{\text{total},\,i}\,\cdot\,v_i^*.$$

Luego, en modo eléctrico ($h = 0$), la potencia eléctrica demandada de la batería es:

$$P_{\text{eléctrica},\,i} = \frac{P_{\text{rueda},\,i}}{\eta_{\text{tren}}}, \quad (h = 0),$$

y en modo combustión ($h = 1$) se desprecia consumo eléctrico, pudiéndose modelar el consumo de combustible aparte. En el código se define:

$$E_{\text{eléctrica},\,i} = \frac{P_{\text{eléctrica},\,i}}{3600} \quad [\text{Wh}],$$

pues $\Delta t = 1\,\mathrm{s} = 1/3600\,\mathrm{h}$. De este modo:

$$\text{SOC}_i = \text{SOC}_{i-1} - E_{\text{eléctrica},\,i}.$$

Actualización del Estado de Carga y Lógica Binaria de Recarga

La recarga se activa cuando SOC_i cae por debajo del umbral E_{umbral} y no se estaba recargando en el paso previo:

$$\begin{cases}
\text{en_recarga}_i = 1, & \text{si } \text{SOC}_{i-1} < E_{\text{umbral}} \text{ y en_recarga}_{i-1} = 0,\\
\text{en_recarga}_i = 0, & \text{si tiempo_en_recarga}_{i-1} \geq \Delta t_{\text{recarga}}.
\end{cases}$$

Cuando $\text{en\_recarga}_i = 1$, se detiene el desplazamiento ($v_i^* = 0$) y se selecciona la estación más cercana:

$$j^* = \arg\min_j \big\|\text{posiciones}_i - \mathbf{x}_{\text{est},\,j}\big\|,$$

donde $\{\mathbf{x}_{\text{est},\,j}\}$ son coordenadas fijas de estaciones de recarga. Se acumula:

$$\text{tiempo\_en\_recarga}_i = \text{tiempo\_en\_recarga}_{i-1} + 1.$$

Al cumplirse $\text{tiempo\_en\_recarga}_i \geq \Delta t_{\text{recarga}}$ (e.g., $60\,\mathrm{s}$), entonces:

$$\begin{aligned}
&\text{SOC}_i \leftarrow E_{\max},\\
&\text{en_recarga}_i \leftarrow 0,\\
&v_i^* \text{ retoma su perfil original.}
\end{aligned}$$

En el código, este interruptor binario se implementa con:

```
if SOC[i-1] < umbral_energia and not en_recarga:
    en_recarga = True
    tiempo_en_recarga = 0
    # seleccionar estación más cercana según posiciones
elif en_recarga:
    tiempo_en_recarga += 1
    if tiempo_en_recarga >= tiempo_recarga:
        en_recarga = False
        SOC[i] = E_max
```

Actualización del Estado de Carga y Lógica Binaria de Recarga

La lógica de recarga se implementa como se describió en la sección anterior, con umbral E_{umbral}. En el código:

```
if State_bater < umbral_energia and not en_recarga:
    # Elegir estación más cercana o siguiente en lista
    en_recarga = True
    tiempo_en_recarga = 0
    ruta_actual = i
    speeds[i] = 0
    slopes[i] = 0
elif en_recarga:
    tiempo_en_recarga += 1
    if tiempo_en_recarga >= tiempo_recarga:
        en_recarga = False
        tiempo_en_recarga = 0
        State_bater = E_max
        # Mantener velocidades y pendientes originales desde ruta_actual
```

Cuando $\text{en\_recarga}_i = 1$, se mantiene SOC_i constante (no hay descarga) y la moto no avanza ($v_i^* = 0$). Una vez transcurrido $\Delta t_{\text{recarga}}$, se retorna la SOC a $E_{\max}$ y se reanuda el perfil.

Simulación Basada en Agentes (`AgentBasedModel.py`)

La extensión a múltiples unidades en paralelo reutiliza el mismo formalismo de `ModeloMoto1.py`, encapsulado en la clase `Moto`. A continuación se detallan los aspectos matemáticos de la implementación y las funciones binarias asociadas a la interacción entre agentes.

Definición de la Clase `Moto`

Cada agente `Moto` almacena:

$$\begin{aligned}
&\text{posición } (x_i, y_i), \quad v_i, \quad \text{SOC}_i, \quad \text{en_recarga}_i, \quad \text{tiempo_en_recarga}_i, \\
&\{v_i\},\ \{\theta_i\}, \quad \{\mathbf{x}_{\text{est},\,j}\}_{j=1}^{N}, \quad \text{nombre del agente}, \quad \text{distancia_total}, \quad \text{histórico_SOC}.
\end{aligned}$$

El método `avanzar_paso()` implementa para cada agente las mismas fórmulas discretas del modelo de partícula:

$$
\begin{aligned}
v_i^* &= \frac{v_i}{3.6}, \quad \theta_i^* = \theta_i \, \frac{\pi}{180},\\
F_{\text{aero},\,i} &= \tfrac{1}{2}\,\rho\,A\,C_d\,(v_i^*)^2,\\
F_{\text{rodadura},\,i} &= m\,g\,C_{rr}\,\cos(\theta_i^*),\\
F_{g,\,i} &= m\,g\,\sin(\theta_i^*),\\
\Delta v_i &= v_i^* - v_{i-1}^*, \quad (\text{con } v_{-1}^* = 0),\\
F_{\text{inercia},\,i} &= m\,\frac{\Delta v_i}{1\,\text{s}},\\
F_{\text{total},\,i} &= F_{\text{aero},\,i} + F_{\text{rodadura},\,i} + F_{g,\,i} + F_{\text{inercia},\,i},\\
P_{m,\,i} &= F_{\text{total},\,i}\,v_i^*, \quad P_{\text{eléctrica},\,i} = \frac{P_{m,\,i}}{\eta_{\text{tren}}}, \quad E_{\text{eléctrica},\,i} = \frac{P_{\text{eléctrica},\,i}}{3600},\\
\text{SOC}_i &= \text{SOC}_{i-1} - E_{\text{eléctrica},\,i}.
\end{aligned}
$$

Si $\text{SOC}_i < E_{\text{umbral}}$ y $\text{en\_recarga}_{i-1} = 0$, entonces:

$$\text{en\_recarga}_i = 1, \quad v_i^* = 0, \quad j^* = \arg\min_j \big\| \text{posiciones}_i - \mathbf{x}_{\text{est},\,j} \big\|.$$

Mientras $\text{en\_recarga}_i = 1$:

$$\text{tiempo\_en\_recarga}_i = \text{tiempo\_en\_recarga}_{i-1} + 1, \quad \text{SOC}_i = \text{SOC}_{i-1}, \quad v_i^* = 0.$$

Al cumplirse $\text{tiempo\_en\_recarga}_i \geq \Delta t_{\text{recarga}}$, se restaura:

$$\text{SOC}_i = E_{\max}, \quad \text{en\_recarga}_i = 0, \quad (\text{vuelve a la ruta}).$$

Esta lógica binaria se ejecuta de forma independiente para cada agente, salvo que se añada decisión de cola cuando varias motos compiten por la misma estación:

$$\text{cola}_j : \text{ FIFO entre agentes que llegan simultáneamente a estación } j.$$

Scheduler y Recolección de Datos

Se define un `SimpleScheduler` que itera sobre todos los agentes `Moto` en cada paso de simulación discreta. El modelo `MotoModel` encapsula:

$$\texttt{MotoModel}\big(\{\texttt{Moto}_k\}_{k=1}^N,\ \{\mathbf{x}_{\text{est},\,j}\}_{j=1}^N\big),$$

con un método `step()` que llama a `avanzar_paso()` para cada agente. Se utiliza un `DataCollector` (implícito en atributos de cada agente) para almacenar:

$$\big\{\, \text{SOC}_k(t),\ v_k(t),\ (x_k(t), y_k(t)),\ \text{eventos de recarga} \big\}_{k=1}^N$$

en cada paso t. Luego, se gráfica la evolución de $\text{SOC}_k(t)$ usando Matplotlib, con colores diferenciados para cada agente:

$$\{\text{rojo, azul, verde, } \ldots\}.$$