```
 1 import turtle
 2 import math
 3 import time
 4 import random
 5
 6 simu = turtle.getscreen()
 7 simu.title = ("PID Simulator")
 8 turtle.hideturtle()
 9
10
11
12 class Motor() :
13     def __init__(self, color, _entraxe):
14
15         self.entraxe = _entraxe
16         self.motorR = turtle.Turtle()
17         self.motorR.speed(6)
18         self.motorR.penup()
19         self.motorR.goto(0, self.entraxe/2)
20         self.motorR.pendown()
21         self.motorR.color(color)
22
23         self.motorL = turtle.Turtle()
24         self.motorL.speed(6)
25         self.motorL.penup()
26         self.motorL.goto(0, - self.entraxe/2)
27         self.motorL.pendown()
28         self.motorL.color(color)
29
30         self.theta = 0
31         self.l = 0
32         self.alpha = 0
33         self.x0 = 0
34         self.y0 = 0
35         self.center = (0,0)
36
37         self.theta_error = 0
38         self.somme_theta_error = 0
39         self.delta_theta_error = 0
40         self.last_theta_error = 0
41
42         self.l_error = 0
43         self.somme_l_error = 0
44         self.delta_l_error = 0
45         self.last_l_error = 0
```

```
46
47          self.l_order = 0
48          self.theta_order = 0
49
50          self.uMax = 255
51          self.u0 = 0
52          self.diametre = 0.8
53
54          self.vCroisiere = 0.3
55          self.acc = 0.279
56          self.l_evo = 0
57          self.m = 0
58
59
60          self.wCroisiere = 0
61          self.theta_pp = 0
62          self.theta_evo = 0
63          self.t_total1 = 0
64          self.t_total2 = 0
65
66          self.p = 0
67
68          self.speed_t = 0
69          self.speed_theta = 0
70          self.random = 0
71
72          self.tf = 0
73          self.ti = time.time()
74
75          self.last_speed_theta = 0
76          self.last_speed_l = 0
77          self.delta_speed_l = 0
78          self.delta_speed_theta = 0
79
80
81
82
83
84
85      def odometry(self):
86          self.center = ((self.motorR.pos()[0] + self.motorL
    .pos()[0])/2, (self.motorR.pos()[1] + self.motorL.pos()[1]
    )/2)
87          self.alpha = self.motorL.heading()
88          if self.alpha > 180 :
```

```
 89             self.alpha = self.alpha - 360
 90
 91     def consigne(self, x_cible, y_cible):
 92         self.odometry()
 93         xR, yR = self.motorR.position()
 94         xL, yL = self.motorL.position()
 95         #print("posR :", xR, yR)
 96         #print("posL :", xL, yL)
 97         x = (xR + xL)/2
 98         y = (yR + yL)/2
 99         self.center = (x, y)
100         self.x0 = self.center[0]
101         self.y0 = self.center[1]
102         print('center:', self.center)
103         self.l = ((x_cible - self.x0)**2 + (y_cible -
    self.y0)**2)**(1/2)
104         self.theta = (math.atan2(y_cible - self.y0,
    x_cible - self.x0))*(180/math.pi) - self.motorL.heading()
105         if self.theta > 180 :
106             self.theta -= 360
107         print('thetaOK:', self.theta)
108         print('l:', self.l)
109
110
111     def staticharact(self, u1, v1, u2, v2): #construction
     courbe tension vitesse
112         v1 = v1/1000
113         v2 = v2/1000
114         self.theta_pp = (self.acc / (self.entraxe / 2)) *
     (180 / math.pi) * 100
115         self.m = ((v1 - v2)/(u1 - u2))
116         self.p = v1 - (u1*self.m)
117         self.u0 = -(self.p/self.m)
118         self.wCroisiere = 41
119         self.vCroisiere = self.uMax*self.m + self.p
120         #print(self.theta_pp)
121         #print('vC:', self.vCroisiere)
122         #print('wC:', self.wCroisiere)
123         #print('u0', self.u0)
124         #print('m:', self.m)
125         #print('p:', self.p)
126
127     def profile_l(self,t):
128         l_acc = (self.vCroisiere**2)/(2*self.acc)
129         #print('l_acc:', l_acc)
```

```
130         if (self.l/2) <= l_acc:
131             t_bang = (self.l/(self.acc))**(1/2)
132             self.t_total1 = 2*t_bang
133             if t <= t_bang:
134                 self.l_evo = (self.acc/2)*(t**2)
135             else:
136                 self.l_evo = -(self.acc/2)*((t - t_bang)
    **2) + (t_bang*self.acc)*(t - t_bang) + (self.l/2)
137             #print('t bang:', t_bang)
138 
139 
140         else :
141             l_croisiere = self.l - ((self.vCroisiere ** 2
    ) / self.acc)
142             t_bang = self.vCroisiere/self.acc
143             self.t_total1 = (2*t_bang) + (l_croisiere/
    self.vCroisiere)
144             t_croisiere = l_croisiere/self.vCroisiere
145 
146             if t < t_bang:
147                 self.l_evo = (self.acc/2)*(t**2)
148 
149             if (t > t_bang) and (t < self.t_total1 -
    t_bang):
150                 self.l_evo = self.vCroisiere*(t - t_bang)
     + l_acc
151 
152             if t > (self.t_total1 - t_bang):
153                 self.l_evo = -(self.acc/2)*((t - (t_bang
    + t_croisiere))**2) + self.vCroisiere*(t - (t_bang +
    t_croisiere)) + l_croisiere + l_acc
154         print('l evo:', self.l_evo)
155         #print('total1:', self.t_total1)
156 
157     def profile_theta(self,t):
158         #print('thetapp:', self.theta_pp)
159         #print('wcroisiere:', self.wCroisiere)
160         theta_acc = (self.wCroisiere**2) / (2*self.
    theta_pp)
161         #print('theta acc:', theta_acc)
162         t_croisiere = 0
163         if abs(self.theta/2) <= theta_acc:
164             if self.theta > 0:
165                 t_bang = (abs(self.theta) / self.theta_pp
    ) ** (1/2)
```

```
166                 #print('theta:', self.theta)
167                 #print('thetapp:', self.theta_pp)
168                 #print('t bang:', t_bang)
169                 self.t_total2 = 2*t_bang
170                 if t <= t_bang:
171                     self.theta_evo = (self.theta_pp / 2)
    * (t**2)
172                 else:
173                     self.theta_evo = -(self.theta_pp / 2)
     * ((t - t_bang) ** 2) + (t_bang * self.theta_pp) * (t -
    t_bang) + (self.theta / 2)
174                 #print('t bang:', t_bang)
175             else:
176                 t_bang = (abs(self.theta) / self.theta_pp
    ) ** (1/2)
177                 #print('theta:', self.theta)
178                 #print('thetapp:', self.theta_pp)
179                 #print('t bang:', t_bang)
180                 self.t_total2 = 2*t_bang
181                 if t <= t_bang:
182                     self.theta_evo = -(self.theta_pp / 2)
     * (t**2)
183                 else:
184                     self.theta_evo = (self.theta_pp / 2)
    * ((t - t_bang) ** 2) + (t_bang * -self.theta_pp) * (t -
    t_bang) + (self.theta / 2)
185                 #print('t bang:', t_bang)
186
187         else:
188             theta_croisiere = abs(self.theta) - ((self.
    wCroisiere ** 2) / self.theta_pp)
189             t_bang = self.wCroisiere / self.theta_pp
190             self.t_total2 = (2 * t_bang) + (
    theta_croisiere / self.wCroisiere)
191             t_croisiere = theta_croisiere / self.
    wCroisiere
192             if self.theta > 0 :
193                 if t < t_bang:
194                     self.theta_evo = (self.theta_pp / 2)
    * (t ** 2)
195
196                 if (t > t_bang) and (t < self.t_total2 -
    t_bang):
197                     self.theta_evo = self.wCroisiere * (t
     - t_bang) + theta_acc
```

```
198
199                 if t > (self.t_total2 - t_bang):
200                     self.theta_evo = -(self.theta_pp / 2)
     * ((t - (t_bang + t_croisiere)) ** 2) + self.wCroisiere
    * (
201                                 t - (t_bang + t_croisiere)) +
     theta_croisiere + theta_acc
202             else:
203                 if t < t_bang:
204                     self.theta_evo = (-self.theta_pp / 2)
     * (t ** 2)
205
206                 if (t > t_bang) and (t < self.t_total2 -
    t_bang):
207                     self.theta_evo = -self.wCroisiere * (
    t - t_bang) - theta_acc
208
209                 if t > (self.t_total2 - t_bang):
210                     self.theta_evo = (self.theta_pp / 2)
    * (
211                                 (t - (t_bang + t_croisiere))
    ** 2) - self.wCroisiere * (
212                                                     t - (t_bang
    + t_croisiere)) - theta_croisiere - theta_acc
213         #print(2*t_bang + t_croisiere)
214         #print('theta evo:', self.theta_evo)
215         #print('total2:', self.t_total2)
216
217     def PIDregulator(self):
218         kp_l = 650
219         ki_l = 0
220         kd_l = 15
221
222         kp_theta = 20
223         ki_theta = 0
224         kd_theta = 0.1
225
226         self.odometry()
227         x = self.center[0]
228         y = self.center[1]
229         if self.motorL.heading() > 180:
230             heading = self.motorL.heading() - 360
231         else :
232             heading = self.motorL.heading()
233         print('heading:', heading)
```

```
234
235
236         self.theta_error = self.theta_evo - heading
237         self.somme_theta_error += self.theta_error
238         self.delta_theta_error = self.theta_error - self.
    last_theta_error
239         self.last_theta_error = self.theta_error
240
241         #print('p:', kp_theta*self.theta_error)
242         #print('i:', ki_theta*self.somme_theta_error)
243         #print('d:', kd_theta*self.delta_theta_error)
244         #print('order:', kp_theta*self.theta_error +
    ki_theta*self.somme_theta_error + kd_theta*self.
    delta_theta_error)
245
246         #print('pd:', kp_l * self.l_error)
247         #print('id:', ki_l * self.somme_l_error)
248         #print('dd:', kd_l * self.delta_l_error)
249         #print('order l:',kp_theta * self.theta_error +
    ki_theta * self.somme_theta_error + kd_theta * self.
    delta_theta_error)
250
251         self.theta_order = kp_theta*self.theta_error +
    ki_theta*self.somme_theta_error + kd_theta*self.
    delta_theta_error
252
253
254         self.l_error = self.l_evo - (((x - self.x0)**2 +
    (y - self.y0)**2)**(1/2))
255         #print('error l:', self.l_error)
256         #print('error theta:', self.theta_error)
257         self.somme_l_error += self.l_error
258         self.delta_l_error = self.l_error - self.
    last_l_error
259         self.last_l_error = self.l_error
260
261         self.l_order = kp_l * self.l_error + ki_l * self.
    somme_l_error + kd_l * self.delta_l_error
262
263         self.l_order += self.u0
264         self.theta_order += self.u0
265
266         if self.l_order > self.uMax:
267             self.l_order = self.uMax
268         if self.l_order < -self.uMax:
```

```
269             self.l_order = -self.uMax
270 
271         if self.theta_order > self.uMax:
272             self.theta_order = self.uMax
273         if self.theta_order < (-self.uMax):
274             self.theta_order = -(self.uMax)
275 
276         print('cmd theta:', self.theta_order)
277         print('cmd l:', self.l_order)
278 
279         self.speed_l = self.m*self.l_order + self.p
280         self.speed_theta = ((self.m*self.theta_order +
    self.p)/((self.entraxe/2)))*(180/math.pi)*20
281         #print('speed theta:', self.speed_theta)
282 
283         self.delta_speed_theta = self.speed_theta - self.
    last_speed_theta
284         self.delta_speed_l = self.speed_l - self.
    last_speed_l
285         #print('last speed old:', self.last_speed_theta)
286 
287         self.last_speed_theta = self.speed_theta
288         self.last_speed_l = self.speed_l
289         #print('last speed:', self.last_speed_theta)
290 
291         #if abs(self.theta_order) < abs(self.u0):
292             #self.speed_theta = 0
293         #else:
294             #self.speed_theta = (((self.acc*self.
    theta_order) + self.p)/(self.entraxe/2))*(180/math.pi)
295         #print('w:', self.speed_theta)
296 
297         #if abs(self.theta_order) < abs(self.u0):
298             #self.speed_theta = 0
299         #else:
300             #self.speed_theta = (((self.acc*self.
    theta_order) + self.p)/(self.entraxe/2))*(180/math.pi)
301         print('w:', self.speed_theta)
302         print('v:', self.speed_l)
303 
304 
305 
306     def forreal(self, method):
307         pass
308         if self.error == 0:
```

```
309             self.error = 2
310             # self.error = random.uniform(-0.15,0.15)
311         self.thp += self.error
312         self.random += 1
313         if self.random % 25 == 0:
314             self.error2 = random.uniform(-2, 2)
315         if 0 < self.random % 25 < 5:
316             print('random')
317             self.thp += self.error2
318 
319 
320     def tourne1(self, dt):
321         pass
322         if self.theta < 0 :
323             self.motorR.circle(-self.entraxe/2, -self.
    speed_theta*dt)
324             self.motorL.circle(self.entraxe/2, self.
    speed_theta*dt)
325         else :
326             self.motorR.circle(-self.entraxe / 2, self.
    speed_theta*dt)
327             self.motorL.circle(self.entraxe / 2, -self.
    speed_theta*dt)
328         #print('heading:', self.motorR.heading())
329 
330     def tourne(self, dt):
331         self.motorR.circle(-self.entraxe/2, -self.
    speed_theta*dt)
332         self.motorL.circle(self.entraxe/2, self.
    speed_theta*dt)
333         #print('heading:', self.motorR.heading())
334 
335 
336     def roule(self, dt):
337         self.motorR.fd(self.speed_l*dt)
338         self.motorL.fd(self.speed_l*dt)
339         xR, yR = self.motorR.position()
340         xL, yL = self.motorL.position()
341         x = (xR + xL) / 2
342         y = (yR + yL) / 2
343         center = (x, y)
344         print('center:', center)
345 
346     def runmotor(self, x_cible, y_cible, u1, u2, v1, v2):
347         self.odometry()
```

```
348         self.consigne(x_cible, y_cible)
349         self.staticharact(u1, v1, u2, v2)
350         self.profile_theta(0)
351         self.profile_l(0)
352         dt = 0
353         ti = time.time()
354         dt1 = 0
355         dt2 = 0
356         #print('tt2:', self.t_total2)
357         #print('tt1:', self.t_total1)
358 
359         while dt1 < self.t_total2 or self.theta_error > 0
    .001:
360 
361             tf = time.time()
362             dt = tf - ti
363             tf = ti
364             ti = time.time()
365             dt1 += dt
366             dt2 += dt
367             if dt2 > self.t_total2:
368                 dt2 = self.t_total2
369             self.profile_theta(dt2)
370             self.profile_l(0)
371             self.PIDregulator()
372             self.tourne(dt)
373             self.roule(dt)
374             #print('dt:', dt)
375             #print('dt1:', dt1)
376             #print('dt2:', dt2)
377             #if dt != 0:
378                 #print('var theta:', self.
    delta_speed_theta/dt)
379 
380         dt1 = 0
381         dt2 = 0
382 
383         while dt2 < self.t_total1 or self.l_error > 0.001
    :
384             #print('thevo:', self.theta_evo)
385             tf = time.time()
386             dt = tf - ti
387             tf = ti
388             ti = time.time()
389             dt1 += dt
```

```
390             dt2 += dt
391             if dt1 > self.t_total1:
392                 dt1 = self.t_total1
393             self.profile_l(dt1)
394             self.PIDregulator()
395             self.tourne(dt)
396             self.roule(dt)
397             #print('dt:', dt)
398             #print('dt1:', dt1)
399             #print('dt2:', dt2)
400
401
402
403
404
405
406
407
408
409
410
411
412         #print((self.motorL.pos()[0] + self.motorR.pos()[
    0]) / 2, (self.motorL.pos()[1] + self.motorR.pos()[1]) /
    2)
413
414
415 def main():
416     test1 = Motor("red", 20)
417     test1.odometry()
418     test1.staticharact(90, 119.1, 250, 357.8)
419     test1.consigne(10, 10)
420     test1.runmotor(-50,25,90,250,119.1,357.8)
421
422
423
424
425
426
427
428
429
430
431
432     input()
```

```
433
434 main()
435
436
437
438
```