Раздел 5: движение робота

Введение

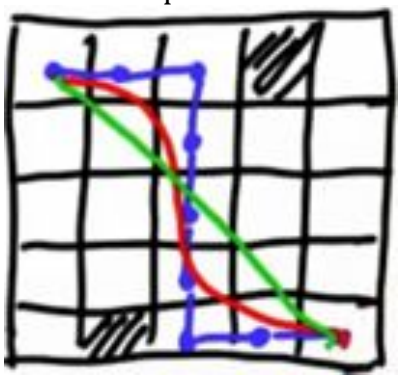
Добро пожаловать в раздел 5
В этом разделе вы узнаете об управлении движением робота. В предыдущем разделе вы узнали, как планировать траекторию робота, а теперь, вы узнаете о том, как превратить эти траектории в реальные команды роботу.

В частности, вы узнаете, как сглаживать полученные траектории.
А также вы узнаете, об управлении роботом, в частности, о ПИД-регуляторе. И, как обычно, вы получите возможность запрограммировать эти удивительные вещи.

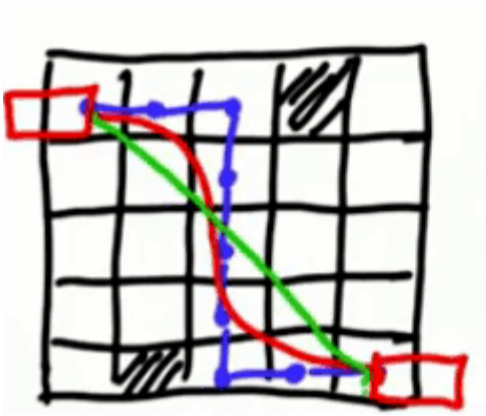
Q-1: движение робота

Возможно, вы помните, что вы изучали планирование в дискретном мире и планы, которые вы находили выглядели так:

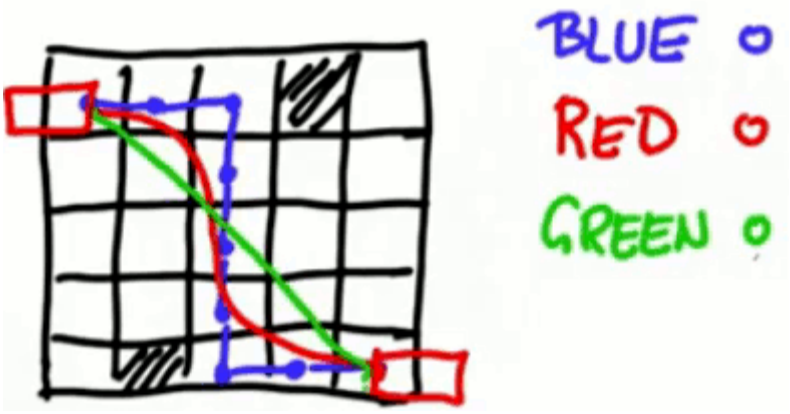
Эта траектория имеет много недостатков. Вы же не хотите, чтобы робот ехал прямо, затем разворачивался на 90 градусов, двигался снова вперед и так далее.
Автомобиль, например, даже не может это сделать. Робот будет двигаться по этому маршруту очень медленно возле углов. Лучшая траектория будет выглядеть как показано красной линией на следующем рисунке.



Это гораздо более плавный путь.
В крайнем случае, вы можете генерировать путь, показанный зеленой линией.



Возникает вопрос, как бы вы хотели изменить траекторию, показанную синим: чтобы она стала похожей на траекторию, показанную красным или зеленым? Это вопрос, который проверяет вашу интуицию, это не математически точный вопрос.



Ответ Ключ к Q-1 движение робота

Вы должны на самом деле предпочитать красную траекторию. И причиной этого является то, что если выбрать зеленую траекторию, робот не сможет выполнить 45-градусный поворот на месте. Зеленый путь короче, но требуются два 45-градусных поворота на месте, недостижимых для робота. Вы должны найти гладкий путь, который может выполняться роботом.

Q-2: алгоритм сглаживания

В уроке планирования мы задавали траектории в виде последовательности точек в 2-мерной сетке:

Для целей сглаживания будем называть каждую точку $\mathbf{x_i}$, от $\mathbf{x_0}$ до $\mathbf{x_{N-1}}$. Каждый $\mathbf{x}$ – это 2-мерная координата, но это должно быть безразлично для сглаживания в общем случае – будь то в 1D, 2D или 3D.
Алгоритм сглаживания:

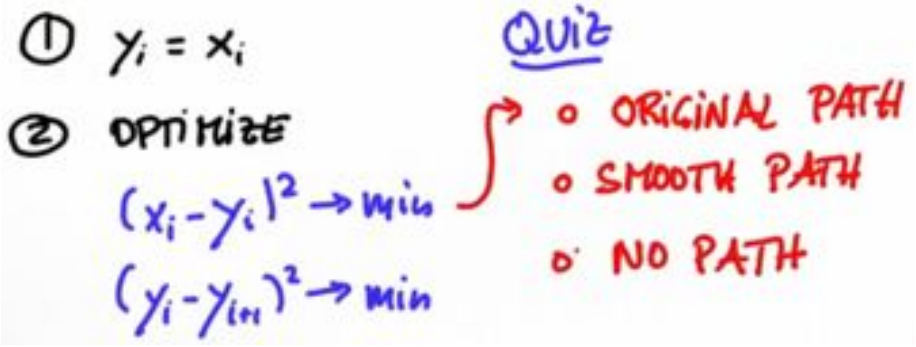
1. Первоначально, создать переменные y_i, которые являются такими же, как x_i. Напомним, что у нас есть негладкая траектория, точки которой нашел планировщик: $\mathbf{y_i = x_i}$
2. Далее оптимизируем траекторию по двум критериям, минимизируя два следующих выражения:

$\mathbf{(x_i - y_i)^2 \rightarrow min}$

$\mathbf{(y_i - y_{i+1})^2 \rightarrow min}$

Первое выражение минимизирует расстояние между исходной точкой и точкой сглаженной траектории, а второе – минимизирует расстояние между двумя соседними точками сглаженной траектории.

Если вы примените только первый критерий, и забудете о втором, то, что вы получите?



Ответ Ключ к Q-2 алгоритм сглаживания

И ответ таков: если вы просто оптимизировали первое выражение, вы получаете оригинальный путь и причина очевидна. Значение этого выражения уже есть 0 в силу исходной настройки, где вы установили $\mathbf{y_i = x_i}$. Таким образом, его нельзя сделать меньше. И вы получаете оригинальный путь.

Q-3: алгоритм сглаживания 2

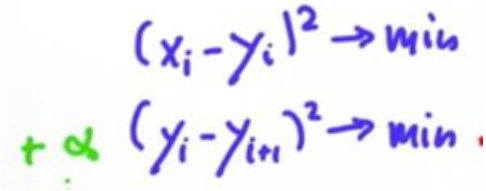
Следующий вопрос аналогичен, но касается второго выражения, так что забудьте о первом. Что произойдет, если оптимизировать только второе выражение?

Ответ Ключ к Q-3 A-3: алгоритм сглаживания

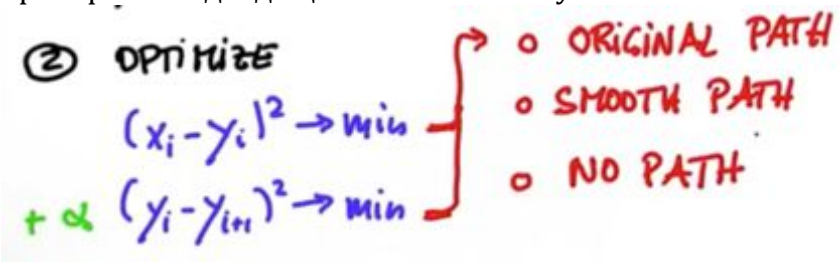
И ответ: вы не получите никакого пути. Этот критерий требует, что y_i был похож на y_{i+1} как только возможно, и лучший способ свести его к минимуму – иметь все точки с одинаковыми координатами, что означает, что вы получите одну точку и путь будет отсутствовать вообще.

Q-4:алгоритм сглаживания 3

Очевидно, что эти два критерия находятся в конфликте друг с другом. На самом деле то, что вы должны делать – это минимизировать оба критерия одновременно, причем можно делать это с каким-то весом α, которым вы можете играть в коде.



Чем больше α, тем более гладкая траектория. Чем меньше α, тем ближе сглаженный путь к исходному. Предположим, что вы оптимизировали одновременно оба критерия с подходящим α. Что вы получаете?

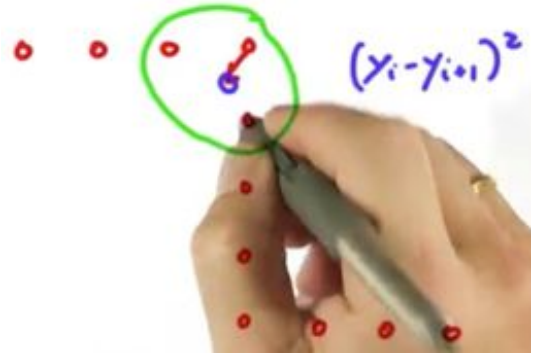


Ответ Ключ к Q-4 A-4: алгоритм сглаживания 3

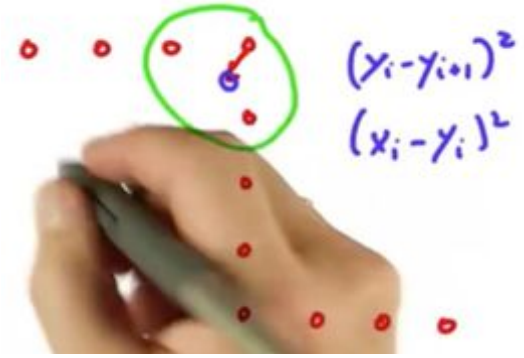
И ответ: вы получаете гладкий путь. Чтобы понять, почему это так, проведем симуляцию оптимизации.
Предположим, вы имеете решение задачи планирования, как на следующей картинке, и вы запустите алгоритм оптимизации.

Рассмотрим участок первого поворота, обведенный зеленым.

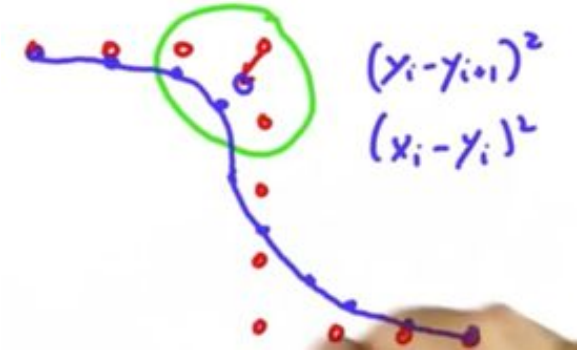
Перенеся угловую точку в направлении влево и вниз (красная стрелка) к новой позиции (синие точки) и возможно, путем сдвига других точек в других направлениях, вы можете уменьшить второй критерий для двух пар точек, включающих угловую.



Тем не менее, это делается, за счет первого критерия, поскольку теперь появится сдвиг значения y от первоначального x.

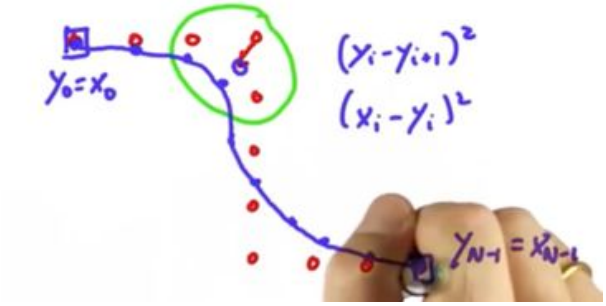


В зависимости от веса, вы можете приходим к следующему:



И для этой новой траектории увеличивается значение $(x_i - y_i)^2$, потому что вы переместите точки от исходных положений, но также существенно уменьшается расстояние между точками и, следовательно, снижается ошибка $(y_i - y_{i+1})^2$. Если вы хотите, чтобы начальная и конечная точки не изменялись, вы просто исключаете их из оптимизации. В самом деле, в этом упражнении вы не должны

изменять эти точки. Будут всегда верны выражения: $y_0 = x_0$ и $y_{N-1} = x_{N-1}$, предполагая что у вас есть N точек и оптимизация применяется только к промежуточным точкам.



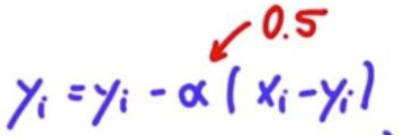
Q-5: сглаживание траектории

Как можно оптимизировать эти два критерия? Вы можете использовать градиентный спуск, то есть для каждого шага по времени вы делаете маленький шаг в направлении минимизации ошибки. Выражение для первой цели оптимизации $(x_i - y_i)^2 \rightarrow min$ следующее:

$y_i = y_i + \alpha(x_i - y_i).$

При итерации, вы записываете в yi старое значение yi с прибавкой члена, пропорционального отклонению yi от xi , взвешенных с весом α.

Установим этот α = 0.5.



Второй критерий можно реализовать двигая старую переменную y в направлении y_{i+1} и от y_i :

$y_i = y_i + \beta*(y_{i+1} - y_i)$

Но еще более эффективная реализация выглядит следующим образом:

$yi = y_i + \beta*(y_{i+1} + y_{i-1} - 2*y_i)$

Это объединяет шаг влево с шагом вправо, учитывая, что каждый yi появляется дважды в оптимизации $(y_i - y_{i+1})^2 \rightarrow min$. Следовательно можно записать это в одном выражении, где вы записываете, чтобы y_i было как можно ближе к y_{i-1} и одновременно как можно ближе к y_{i+1}.

Реализация.

Установим бета в 0,1.

Напоминание - вы не должны применять оптимизацию к первой или последней точке.

Полный код программы:

```
# ------------
# User Instructions
#
# Define a function smooth that takes a path as its input
# (with optional parameters for weight_data, weight_smooth)
# and returns a smooth path.
#
# Smoothing should be implemented by iteratively updating
# each entry in newpath until some desired level of accuracy
# is reached. The update should be done according to the
```

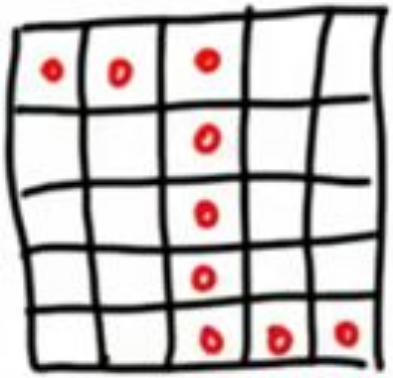
```
# gradient descent equations given in the previous video:
#
# Note that you do not need to use the tolerance parameter
# shown in the video.
#
# If your function isn't submitting it is possible that the
# runtime is too long. Try sacrificing accuracy for speed.
# -----------
from math import *
# Don't modify path inside your function.
path = [[0, 0],
        [0, 1],
        [0, 2],
        [1, 2],
        [2, 2],
        [3, 2],
        [4, 2],
        [4, 3],
        [4, 4]]
# ------------------------------------------------
# smooth coordinates
#
def smooth(path, weight_data = 0.5, weight_smooth = 0.1):
    # Make a deep copy of path into newpath
    newpath = [[0 for row in range(len(path[0]))] for col in range(len(path))]
    for i in range(len(path)):
        for j in range(len(path[0])):
            newpath[i][j] = path[i][j]
    #### ENTER CODE BELOW THIS LINE ###
    tolerance  = 0.00001
    change = tolerance
    while (change >= tolerance):
        change = 0.0
        for i in range(1,len(path)-1):
            for j in range(len(path[0])):
                aux = newpath[i][j]
                newpath[i][j] = newpath[i][j] + weight_data * ( path[i][j] - newpath[i][j])
                newpath[i][j] = newpath[i][j] + weight_smooth * (newpath[i-1][j] + newpath[i+1][j] -
(2.0 * newpath[i][j]))
                change += abs(aux - newpath[i][j])

    return newpath # Leave this line for the grader!


# feel free to leave this and the following lines if you want to print.
newpath = smooth(path)
# thank you - EnTerr - for posting this on our discussion forum
for i in range(len(path)):
    print '['+ ', '.join('%.3f'%x for x in path[i]) +'] -> ['+ ', '.join('%.3f'%x for x in newpath[i])
+']'
```

Комментарии

```
path = [[0, 0],
        [0, 1],
        [0, 2],
        [1, 2],
```

```
        [2, 2],
        [3, 2],
        [4, 2],
        [4, 3],
        [4, 4]]
```

Здесь задана траектория, начинающаяся в [0, 0] и заканчивающаяся в [4, 4].
Если вы посмотрите внимательно, то путь идет направо, затем вниз, а затем еще раз направо.

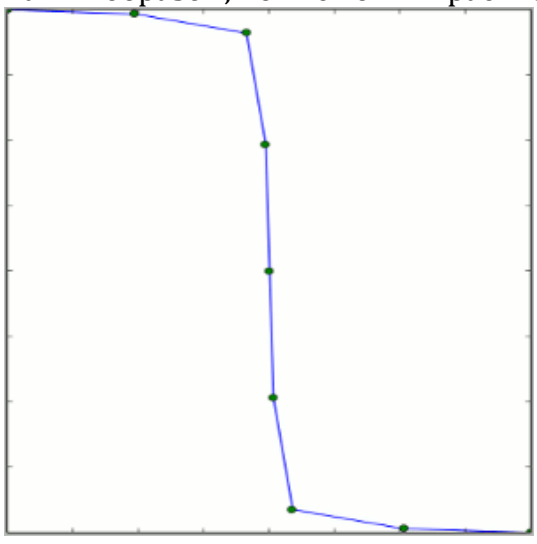


Нужно реализовать функцию сглаживания, которая принимает в качестве входных параметров траекторию и два весовых коэффициента.

```
def smooth(path, weight_data = 0.5, weight_smooth = 0.1):
```

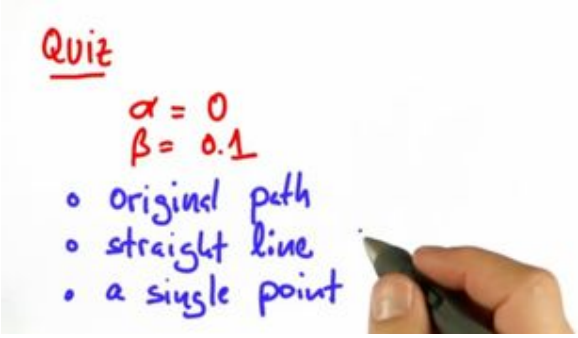
Функция создает новый путь - NewPath - которая состоит из y[i], полученных из точек заданной траектории с применением записанных выше итеративных формул.

Результат работы функции должен выглядеть следующим образом:
[0.000, 0.000] -> [0.000, 0.000]
[0.000, 1.000] -> [0.029, 0.971]
[0.000, 2.000] -> [0.176, 1.824]
[1.000, 2.000] -> [1.029, 1.971]
[2.000, 2.000] -> [2.000, 2.000]
[3.000, 2.000] -> [2.971, 2.029]
[4.000, 2.000] -> [3.824, 2.176]
[4.000, 3.000] -> [3.971, 3.029]
[4.000, 4.000] -> [4.000, 4.000]
Таким образом, новые точки траектории выглядят следующим образом:



Q-6: Нулевой вес данных

Остановитесь на минутку, чтобы подумать о том, как весовые параметры **α** (или weight_data) и **β** (или weight_smooth) влияют на сглаживание траектории. Каков ожидаемый результат сглаживания траектории если мы зададим **α равной** нулю? Это сложный вопрос. Помните, что в данной конкретной реализации вы не изменяете первую и последнюю точки пути. Не стесняйтесь проверить результат с помощью кода из предыдущего вопроса.



Итак, результат:
а) Исходный путь
б) Прямая линия
с) Одна точка

Ответ Ключ к Q-6 A-6: Нулевой вес данных

И ответ б, прямая линия. Не одна точка, как вы видели раньше (A-3). Причина этого: на этот раз мы не меняем первую и последнюю точки пути, поэтому наилучшее решение минимизации критерия сглаживания представляет собой прямую линию. Если бы мы не зафиксировали эти две точки, то результат действительно будет одной точкой.
Очевидно, прямая линия делает все расстояния между точками наименее возможными, самый короткий путь между двумя точками – прямая.
Теперь, если вы делаете противоположное и установить бета равной нулю, результат - оригинальная траектория. Попробуйте эти два варианта в коде.

ПИД-регулятор

В следующем разделе поговорим о ПИД-регуляторе.
Теория управления это огромное поле, и в настоящее время, вы узнаете основы реализации ПИД-регулятора, не слишком заботясь о теории, лишь нащупывая как он работает.

Автомобиль Google на самом деле использует версию этого контроллера (описанную в статье http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.7175&rep=rep1&type=pdf), которая немного более приспособлена к особенностям автомобиля, но основная идея остается той же. Вы получите суть того, что значит управление автомобилем.

Q-7: ПИД-регулятор.

Рассмотрим автомобиль с управляемым передним мостом и двумя неуправляемыми задними колесами. Вы хотите, чтобы автомобиль двигался вдоль черной линии.

Эта траектория может рассматриваться как часть выходных данных алгоритма сглаживания, который мы только что обсуждали. Предположим, что автомобиль имеет фиксированную поступательную скорости, но у нас есть возможность задавать угол поворота автомобиля.

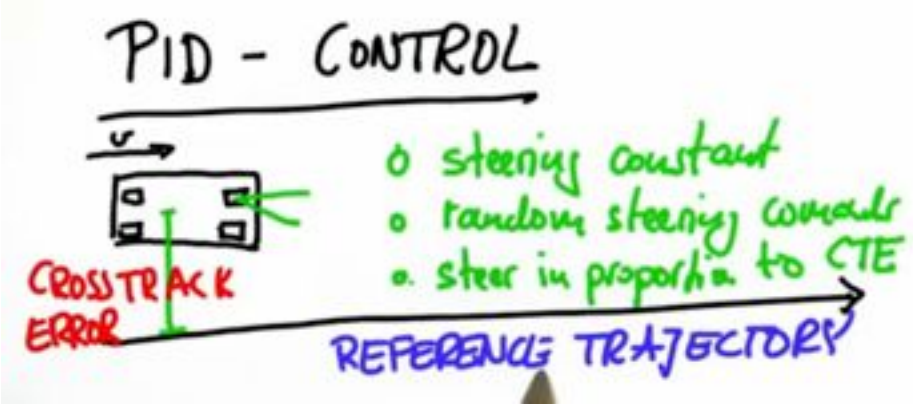
Как управлять рулевыми колесами, чтобы автомобиль полностью повторял эталонную траекторию?

Какой из предложенных вариантов лучше всего подходит для управления автомобилем?

а) Постоянная уставка поворота

б) Случайные уставки

с) Уставка поворота пропорциональна траекторное ошибке, которая определяется как расстояние между заданной траекторией и положением автомобиля.



Ответ Ключ к Q-7 A-7: ПИД-регулирование

И ответ С, вы должны управлять пропорционально ошибке. Чем больше ошибка, тем больше вы должны поворачивать к целевой траектории. Вы можете представить, как это работает – чем ближе вы подходите к траектории, вы будете поворачивать все медленнее и медленнее, и вы попадете на траекторию!

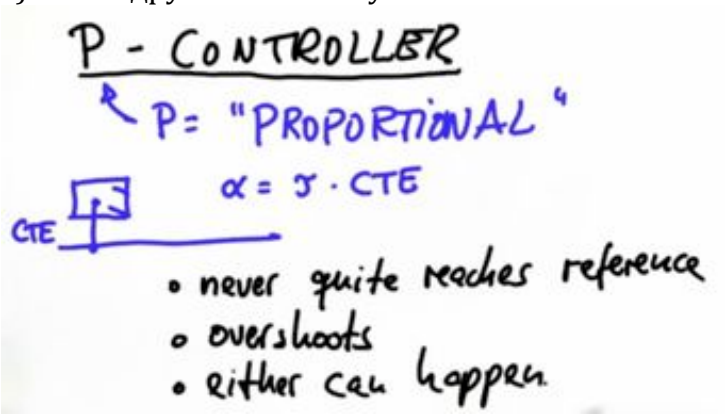
Легко понять, что другие два варианта действительно плохи. Постоянная уставка управления привела бы к движению по кругу, но не по прямой линии. Случайные управления, по существу, приведут к случайному блужданию (http://en.wikipedia.org/wiki/Random_walk), что не очень хорошая идея.

Пропорциональное управление

То, что вы только что узнали, называется контроллером P, где P означает пропорционально. Это потому что вы управляете пропорционально траекторной ошибке.

Q-8: Пропорциональное управление

Предположим, вы будете управлять пропорционально траекторной ошибке. Это означает, что угол поворота пропорционален с неким коэффициентом **τ** траекторной ошибке. Что будет с машиной?
а) Она никогда не достигает эталонной траектории
б) Она перескочит через траекторию
с) И то и другое может случиться



Ответ Ключ к Q-8 A-8: Пропорциональное управление

И ответ б, автомобиль проскакивает эталонную траекторию.
Когда уставка поворота задана в ноль (движение прямо), автомобиль направлен в сторону от траектории. Это происходит независимо от того, каким будет тау. Это означает, что применительно к автомобилю, П-регулятор будет действовать как на картинке ниже, будет выброс и колебания.

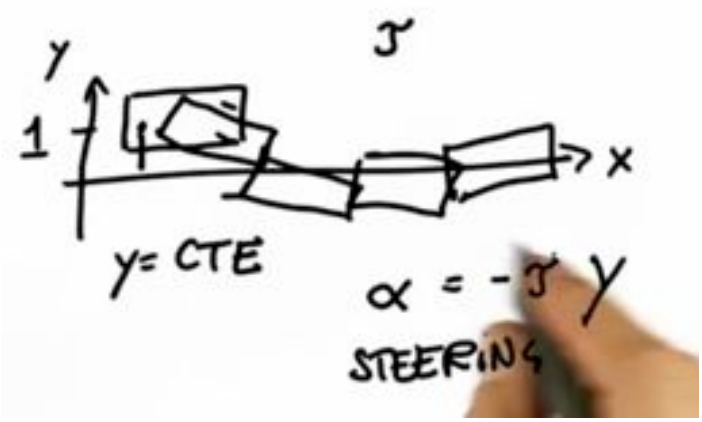
Такое поведение называется незначительно стабильной (или часто просто устойчиво) в литературе. Это колебание может быть приемлемым, если оно достаточно мало для конкретного применения.

Q-9: Реализация П-регулятора

Реализуем П-регулятор, описанный в предыдущем разделе.

Вам дан класс робота с функциями которого вы уже знакомы - **`init, set, set_noise, move`** и т.д. Добавлено поле steering_drift, но оно не используется в настоящее время.

Нужно реализовать функцию run, которая принимает коэффициент пропорционального управления **`т (или param)`**. Инициализируем робота в положении (0.0, 1.0, 0.0), со скоростью 1,0 и выполним в общей сложности 100 шагов моделирования. Это означает, что вы должны рассчитать управление поворотом рулевого колеса и смоделировать движения для каждого из этих шагов.
На рисунке ниже показано, как ожидается, что произойдет.



Принимая, что задаваемая траектория – это ось X, можно определить траекторную ошибку как Y-координату робота. Распечатайте координаты робота и выход контроллера для каждого шага.

Ответ Ключ к Q-9 A-9: Реализация П-регулятора

Ниже исходный код предлагаемой реализации.

```
# -----------
# User Instructions
#
# Implement a P controller by running 100 iterations
# of robot motion. The steering angle should be set
# by the parameter tau so that:
#
# steering = -tau * crosstrack_error
#
# Note that tau is called "param" in the function
# run to be completed below.
#
# Your code should print output that looks like
# the output shown in the video. That is, at each step:
# print myrobot, steering
#
# Only modify code at the bottom!
# ------------

from math import *
import random



# --------------------------------------------------
#
# this is the robot class
#
```

```
class robot:

    # --------
    # init:
    #    creates robot and initializes location/orientation to 0, 0, 0
    #

    def __init__(self, length = 20.0):
        self.x = 0.0
        self.y = 0.0
        self.orientation = 0.0
        self.length = length
        self.steering_noise = 0.0
        self.distance_noise = 0.0
        self.steering_drift = 0.0

    # --------
    # set:
    # sets a robot coordinate
    #

    def set(self, new_x, new_y, new_orientation):

        self.x = float(new_x)
        self.y = float(new_y)
        self.orientation = float(new_orientation) % (2.0 * pi)


    # --------
    # set_noise:
    # sets the noise parameters
    #

    def set_noise(self, new_s_noise, new_d_noise):
        # makes it possible to change the noise parameters
        # this is often useful in particle filters
        self.steering_noise = float(new_s_noise)
        self.distance_noise = float(new_d_noise)

    # --------
    # set_steering_drift:
    # sets the systematical steering drift parameter
    #

    def set_steering_drift(self, drift):
        self.steering_drift = drift

    # --------
    # move:
    #    steering = front wheel steering angle, limited by max_steering_angle
    #    distance = total distance driven, most be non-negative

    def move(self, steering, distance,
             tolerance = 0.001, max_steering_angle = pi / 4.0):

        if steering > max_steering_angle:
```

```
            steering = max_steering_angle
        if steering < -max_steering_angle:
            steering = -max_steering_angle
        if distance < 0.0:
            distance = 0.0


        # make a new copy
        res = robot()
        res.length         = self.length
        res.steering_noise = self.steering_noise
        res.distance_noise = self.distance_noise
        res.steering_drift = self.steering_drift

        # apply noise
        steering2 = random.gauss(steering, self.steering_noise)
        distance2 = random.gauss(distance, self.distance_noise)

        # apply steering drift
        steering2 += self.steering_drift

        # Execute motion
        turn = tan(steering2) * distance2 / res.length

        if abs(turn) < tolerance:

            # approximate by straight line motion

            res.x = self.x + (distance2 * cos(self.orientation))
            res.y = self.y + (distance2 * sin(self.orientation))
            res.orientation = (self.orientation + turn) % (2.0 * pi)

        else:

            # approximate bicycle model for motion

            radius = distance2 / turn
            cx = self.x - (sin(self.orientation) * radius)
            cy = self.y + (cos(self.orientation) * radius)
            res.orientation = (self.orientation + turn) % (2.0 * pi)
            res.x = cx + (sin(res.orientation) * radius)
            res.y = cy - (cos(res.orientation) * radius)

        return res

    def __repr__(self):
        return '[x=%.5f y=%.5f orient=%.5f]'  % (self.x, self.y, self.orientation)

############## ADD / MODIFY CODE BELOW ####################

# ------------------------------------------------------------------------
#
# run - does a single control run


def run(param):
```

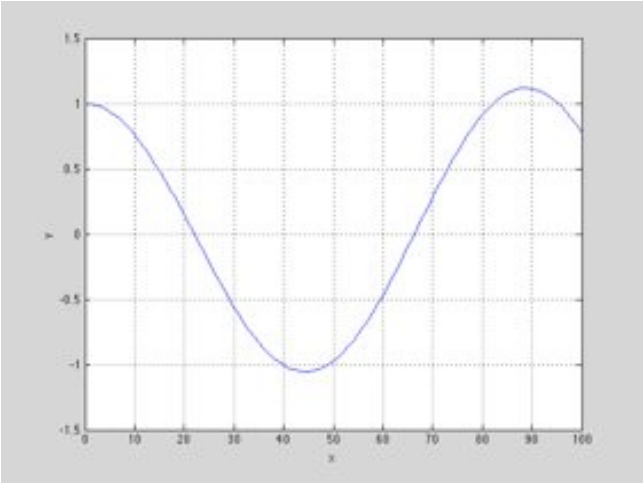
```
    myrobot = robot()
    myrobot.set(0.0, 1.0, 0.0)
    speed = 1.0 # motion distance is equal to speed (we assume time = 1)
    N = 100
    myrobot = robot()
    myrobot.set(0.0, 1.0, 0.0)
    speed = 1.0
    N = 100
    for i in range (N):
        crosstrack_error = myrobot.y
        steer = -param * crosstrack_error
        myrobot = myrobot.move(steer, speed)
        print myrobot, steer

run(0.1) # call function with parameter tau of 0.1 and print results
```

Программа проходит по всем шагам симуляции. Сначала она получает crosstrack_error как Y-координату робота, затем применяет закон управления, чтобы получить угол поворота руля, и, наконец, применяет соответствующие команды перемещения для данного шага.

Результаты работы программы:

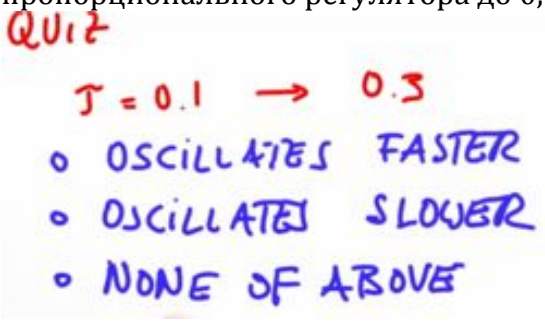
[x=1.00000 y=0.99749 orient=6.27817] -0.1
[x=1.99997 y=0.98997 orient=6.27316] -0.0997491638459
[x=2.99989 y=0.97747 orient=6.26820] -0.0989972950612
[x=3.99973 y=0.96003 orient=6.26330] -0.0977469440459
[x=4.99948 y=0.93774 orient=6.25848] -0.0960031957433
[x=5.99912 y=0.91068 orient=6.25378] -0.0937736475381
[x=6.99861 y=0.87900 orient=6.24921] -0.0910683732206
[x=7.99796 y=0.84283 orient=6.24481] -0.0878998733516
[x=8.99714 y=0.80235 orient=6.24058] -0.0842830124796
[x=9.99614 y=0.75775 orient=6.23656] -0.0802349437746
[x=10.99497 y=0.70925 orient=6.23277] -0.0757750217292
[x=11.99360 y=0.65707 orient=6.22921] -0.0709247036578
[x=12.99206 y=0.60149 orient=6.22592] -0.0657074407797
[x=13.99033 y=0.54275 orient=6.22291] -0.060148559709
[x=14.98843 y=0.48116 orient=6.22020] -0.0542751351986
[x=15.98637 y=0.41701 orient=6.21779] -0.0481158549857
[x=16.98416 y=0.35062 orient=6.21570] -0.0417008775783
[x=17.98183 y=0.28231 orient=6.21395] -0.0350616837984
[x=18.97938 y=0.21242 orient=6.21254] -0.0282309228647
[x=19.97685 y=0.14130 orient=6.21147] -0.0212422537586
[x=20.97428 y=0.06965 orient=6.21077] -0.0141301825775
[x=21.97166 y=-0.00270 orient=6.21042] -0.0069651329429
[x=22.96901 y=-0.07541 orient=6.21043] 0.00027038891367
[x=23.96637 y=-0.14810 orient=6.21081] 0.00754064527652
[x=24.96375 y=-0.22041 orient=6.21155] 0.0148095532718
[x=25.96122 y=-0.29143 orient=6.21265] 0.0220408565504
[x=26.95879 y=-0.36118 orient=6.21411] 0.0291433273808
[x=27.95647 y=-0.42930 orient=6.21592] 0.0361181253838
[x=28.95428 y=-0.49545 orient=6.21806] 0.0429300977929
[x=29.95224 y=-0.55929 orient=6.22054] 0.0495447901321
[x=30.95036 y=-0.62049 orient=6.22334] 0.0559286166539
[x=31.94866 y=-0.67875 orient=6.22645] 0.0620490283003
[x=32.94715 y=-0.73376 orient=6.22985] 0.0678746774675
[x=33.94582 y=-0.78523 orient=6.23352] 0.0733755787953
[x=34.94468 y=-0.83291 orient=6.23746] 0.0785232651524
[x=35.94373 y=-0.87654 orient=6.24163] 0.0832909379354
[x=36.94296 y=-0.91588 orient=6.24603] 0.0876536107578
[x=37.94235 y=-0.95074 orient=6.25062] 0.0915882455706
[x=38.94189 y=-0.98092 orient=6.25539] 0.0950738802388
[x=39.94157 y=-1.00625 orient=6.26031] 0.0980917465938
[x=40.94136 y=-1.02661 orient=6.26535] 0.100625377997
[x=41.94124 y=-1.04186 orient=6.27051] 0.102660705491
[x=42.94119 y=-1.05193 orient=6.27573] 0.104186141662
[x=43.94118 y=-1.05674 orient=6.28101] 0.105192651434
[x=44.94118 y=-1.05626 orient=0.00313] 0.105673809091
[x=45.94116 y=-1.05048 orient=0.00843] 0.105625840959
[x=46.94110 y=-1.03941 orient=0.01370] 0.105047653308
[x=47.94096 y=-1.02310 orient=0.01892] 0.103940845177
[x=48.94073 y=-1.00161 orient=0.02405] 0.102309705972
[x=49.94038 y=-0.97505 orient=0.02908] 0.100161197868
[x=50.93988 y=-0.94353 orient=0.03397] 0.0975049231631
[x=51.93922 y=-0.90720 orient=0.03870] 0.0943530769232
[x=52.93838 y=-0.86624 orient=0.04325] 0.0907203853695
[x=53.93735 y=-0.82084 orient=0.04759] 0.0866240305954
[x=54.93611 y=-0.77121 orient=0.05170] 0.0820835623127
[x=55.93467 y=-0.71760 orient=0.05557] 0.0771207974092
[x=56.93303 y=-0.66026 orient=0.05916] 0.0717597081754
[x=57.93118 y=-0.59948 orient=0.06247] 0.0660263001012
[x=58.92913 y=-0.53556 orient=0.06547] 0.0599484801696
[x=59.92690 y=-0.46880 orient=0.06815] 0.0535559165834
[x=60.92450 y=-0.39953 orient=0.07050] 0.0468798908438
[x=61.92195 y=-0.32810 orient=0.07249] 0.0399531430778
[x=62.91926 y=-0.25485 orient=0.07414] 0.0328097114672
[x=63.91646 y=-0.18014 orient=0.07541] 0.025484766586
[x=64.91362 y=-0.10481 orient=0.07631] 0.018014441401
[x=65.91071 y=-0.02857 orient=0.07684] 0.0104805695803
[x=66.90776 y=0.04819 orient=0.07698] 0.00285687488975
[x=67.90480 y=0.12509 orient=0.07674] -0.00481907100619
[x=68.90186 y=0.20175 orient=0.07611] -0.0125092590929
[x=69.89900 y=0.27729 orient=0.07510] -0.0201754227701
[x=70.89623 y=0.35163 orient=0.07372] -0.0277289188223
[x=71.89358 y=0.42440 orient=0.07196] -0.035162965946
[x=72.89107 y=0.49524 orient=0.06983] -0.0424401550807
[x=73.88872 y=0.56378 orient=0.06736] -0.0495237370919
[x=74.88654 y=0.62967 orient=0.06453] -0.0563778032893
[x=75.88456 y=0.69259 orient=0.06138] -0.0629674638147
[x=76.88278 y=0.75220 orient=0.05791] -0.06925902318
[x=77.88121 y=0.80820 orient=0.05414] -0.0752201521712
[x=78.87986 y=0.86030 orient=0.05009] -0.0808200552658
[x=79.87871 y=0.90822 orient=0.04578] -0.0860296326455
[x=80.87776 y=0.95171 orient=0.04123] -0.0908216358291
[x=81.87700 y=0.99054 orient=0.03646] -0.0951708158967
[x=82.87643 y=1.02451 orient=0.03149] -0.0990540632447
[x=83.87601 y=1.05342 orient=0.02635] -0.102450537793
[x=84.87572 y=1.07712 orient=0.02106] -0.105341788565
[x=85.87555 y=1.09547 orient=0.01565] -0.107711861598
[x=86.87547 y=1.10838 orient=0.01015] -0.109547395169
[x=87.87544 y=1.11575 orient=0.00459] -0.110837701437
[x=88.87544 y=1.11754 orient=6.28217] -0.111574833648
[x=89.87543 y=1.11372 orient=6.27656] -0.111753638231

[x=90.87538 y=1.10430 orient=6.27097] -0.111371791205
[x=91.87527 y=1.08931 orient=6.26543] -0.110429818507
[x=92.87506 y=1.06882 orient=6.25996] -0.108931100014
[x=93.87472 y=1.04291 orient=6.25459] -0.106881857196
[x=94.87423 y=1.01171 orient=6.24936] -0.10429112454
[x=95.87357 y=0.97535 orient=6.24428] -0.101170705027
[x=96.87272 y=0.93401 orient=6.23939] -0.0975351101346
[x=97.87165 y=0.88790 orient=6.23471] -0.0934014849732
[x=98.87037 y=0.83721 orient=6.23026] -0.0887895192955
[x=99.86885 y=0.78221 orient=6.22606] -0.0837213452249



Q-10: Колебания

Следующий вопрос, какие последствия будет иметь изменение коэффициента пропорционального регулятора до 0,3?



Ответ Ключ к Q-10 A-10: Колебания

И ответ - автомобиль будет колебаться быстрее. Изучение вывода программы, написанной на предыдущем шаге может показать, что для больших значений коэффициента робот достигает отрицательного значения у быстрее. Фактически, он пересек линию на шаге 13, в то время, как для значения параметра 0,1 при том же шаге значение Y 0.6. Таким образом, для меньших значений коэффициента пропорционального регулятора колебания гораздо медленнее и компенсации ошибки гораздо меньше.

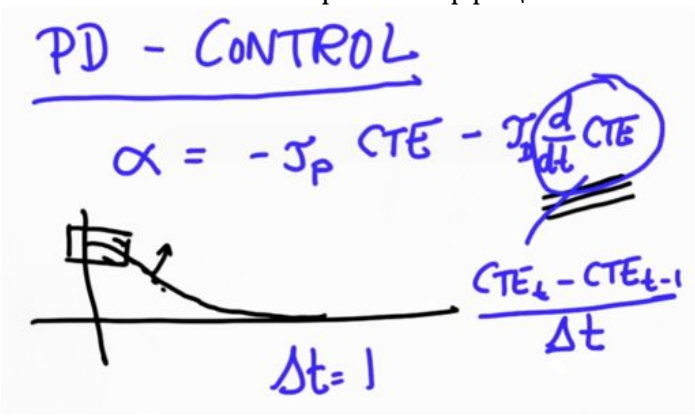
Q-11: ПД-регулятор

Следующий основные вопрос "есть ли способ, чтобы избежать перерегулирования?" Было бы хорошо иметь такой способ, потому что передвигаться в колеблющемся автомобиле опасно. Хитрость, которая позволяет сделать это называется PD-регулятором. В PD-регуляторе уставка угла поворота рулевого колеса связана не только с величиной траекторное ошибки, но и с ее производной по времени. Это

позволяет автомобилю корректно подходить к своей целевой траектории, предполагая соответствующие настройки дифференциального коэффициента. Но как вычислить временную производную ошибки?
В определенное время t производная ошибки рассчитывается как:

$$\frac{CTE_t - CTE_{t-1}}{\Delta t}$$

Для вашего кода Δt предполагается равным 1, так что вы можете опустить знаменатель. Теперь автомобиль контролируется не только пропорционально ошибке, но также пропорционально разнице ошибок текущего и предыдущего шага, с использованием второго коэффициента τ_d .



Реализуйте это!
Новая функция run имеет два параметра - param1 и param2 - которые представляют соответственно коэффициенты пропорциональной и дифференциальной составляющих.
На каждом шаге он вычисляем производную ошибки. Вычисляем угол поворота рулевого колеса не просто пропорционально ошибке, но и пропорционально ее производной.

Полный код программы:

```
# ----------
# User Instructions
#
# Implement a PD controller by running 100 iterations
# of robot motion. The steering angle should be set
# by the parameter tau so that:
#
# steering = -tau_p * CTE - tau_d * diff_CTE
# where differential crosstrack error (diff_CTE)
# is given by CTE(t) - CTE(t-1)
#
# Your code should print output that looks like
# the output shown in the video.
#
# Only modify code at the bottom!
# ------------

from math import *
import random


# ------------------------------------------------
#
# this is the robot class
#

class robot:

    # --------
```

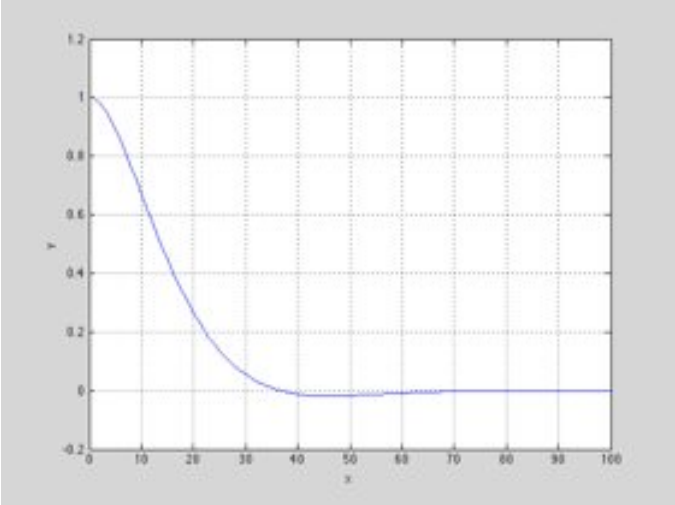
```
    # init:
    #    creates robot and initializes location/orientation to 0, 0, 0
    #

    def __init__(self, length = 20.0):
        self.x = 0.0
        self.y = 0.0
        self.orientation = 0.0
        self.length = length
        self.steering_noise = 0.0
        self.distance_noise = 0.0
        self.steering_drift = 0.0

    # --------
    # set:
    #sets a robot coordinate
    #

    def set(self, new_x, new_y, new_orientation):

        self.x = float(new_x)
        self.y = float(new_y)
        self.orientation = float(new_orientation) % (2.0 * pi)


    # --------
    # set_noise:
    #sets the noise parameters
    #

    def set_noise(self, new_s_noise, new_d_noise):
        # makes it possible to change the noise parameters
        # this is often useful in particle filters
        self.steering_noise = float(new_s_noise)
        self.distance_noise = float(new_d_noise)

    # --------
    # set_steering_drift:
    #sets the systematical steering drift parameter
    #

    def set_steering_drift(self, drift):
        self.steering_drift = drift

    # --------
    # move:
    #    steering = front wheel steering angle, limited by max_steering_angle
    #    distance = total distance driven, most be non-negative

    def move(self, steering, distance,
             tolerance = 0.001, max_steering_angle = pi / 4.0):

        if steering > max_steering_angle:
            steering = max_steering_angle
        if steering < -max_steering_angle:
            steering = -max_steering_angle
        if distance < 0.0:
            distance = 0.0


        # make a new copy
        res = robot()
        res.length         = self.length
        res.steering_noise = self.steering_noise
        res.distance_noise = self.distance_noise
        res.steering_drift = self.steering_drift

        # apply noise
        steering2 = random.gauss(steering, self.steering_noise)
```

```
        distance2 = random.gauss(distance, self.distance_noise)

        # apply steering drift
        steering2 += self.steering_drift

        # Execute motion
        turn = tan(steering2) * distance2 / res.length

        if abs(turn) < tolerance:

            # approximate by straight line motion

            res.x = self.x + (distance2 * cos(self.orientation))
            res.y = self.y + (distance2 * sin(self.orientation))
            res.orientation = (self.orientation + turn) % (2.0 * pi)

        else:

            # approximate bicycle model for motion

            radius = distance2 / turn
            cx = self.x - (sin(self.orientation) * radius)
            cy = self.y + (cos(self.orientation) * radius)
            res.orientation = (self.orientation + turn) % (2.0 * pi)
            res.x = cx + (sin(res.orientation) * radius)
            res.y = cy - (cos(res.orientation) * radius)

        return res

    def __repr__(self):
        return '[x=%.5f y=%.5f orient=%.5f]'  % (self.x, self.y, self.orientation)

############## ADD / MODIFY CODE BELOW ####################

# ------------------------------------------------------------------------
#
# run - does a single control run.


def run(param1, param2):
    myrobot = robot()
    myrobot.set(0.0, 1.0, 0.0)
    speed = 1.0 # motion distance is equal to speed (we assume time = 1)
    N = 100

    crosstrack_error = myrobot.y
    for i in range(N):
        diff_crosstrack_error = myrobot.y - crosstrack_error
        crosstrack_error = myrobot.y
        steer = - param1 * crosstrack_error - param2 * diff_crosstrack_error
        myrobot = myrobot.move(steer, speed)
        print myrobot, steer

# Call your function with parameters of 0.2 and 3.0 and print results
run(0.2, 3.0)
```

Результаты работы программы

[x=0.99998 y=0.99493 orient=6.27305] -0.2
[x=1.99987 y=0.98015 orient=6.26376] -0.183783335986
[x=2.99960 y=0.95690 orient=6.25611] -0.151684277587
[x=3.99914 y=0.92678 orient=6.25000] -0.12163374202
[x=4.99851 y=0.89122 orient=6.24524] -0.0949841761938
[x=5.99772 y=0.85149 orient=6.24165] -0.0715681419784
[x=6.99680 y=0.80869 orient=6.23910] -0.051108951395
[x=7.99579 y=0.76378 orient=6.23743] -0.0333438382448
[x=8.99475 y=0.71804 orient=6.23653] -0.0180299179573
[x=9.99366 y=0.67140 orient=6.23621] -0.0063823756277
[x=10.99255 y=0.62444 orient=6.23649] 0.0056480269145
[x=11.99146 y=0.57776 orient=6.23729] 0.0159966108536
[x=12.99044 y=0.53249 orient=6.23851] 0.0244861716904
[x=13.98947 y=0.48856 orient=6.23998] 0.0293078245412
[x=14.98858 y=0.44622 orient=6.24168] 0.0340612290127
[x=15.98775 y=0.40568 orient=6.24357] 0.0377796340959
[x=16.98701 y=0.36709 orient=6.24560] 0.0405030903054
[x=17.98634 y=0.33057 orient=6.24772] 0.0423513955459
[x=18.98575 y=0.29619 orient=6.24989] 0.0434418365105
[x=19.98523 y=0.26400 orient=6.25209] 0.0438821838886
[x=20.98478 y=0.23401 orient=6.25428] 0.0437704382338
[x=21.98440 y=0.20618 orient=6.25644] 0.0431952196343
[x=22.98407 y=0.18049 orient=6.25855] 0.042236248297
[x=23.98379 y=0.15689 orient=6.26060] 0.0409648571941
[x=24.98355 y=0.13529 orient=6.26257] 0.0394445221994
[x=25.98336 y=0.11562 orient=6.26446] 0.037731399894

[x=26.98320 y=0.09780 orient=6.26626] 0.0358748647636
[x=27.98307 y=0.08172 orient=6.26795] 0.0339180388448
[x=28.98297 y=0.06728 orient=6.26955] 0.0318983082123
[x=29.98288 y=0.05439 orient=6.27104] 0.0298478218925
[x=30.98282 y=0.04294 orient=6.27243] 0.0277939699035
[x=31.98277 y=0.03283 orient=6.27372] 0.0257598380325
[x=32.98273 y=0.02396 orient=6.27491] 0.0237646377632
[x=33.98270 y=0.01623 orient=6.27600] 0.0218241103973
[x=34.98267 y=0.00904 orient=6.27700] 0.0199509049412
[x=35.98265 y=0.00285 orient=6.27798] 0.0197511764223
[x=36.98264 y=-0.00235 orient=6.27888] 0.017995972518
[x=37.98263 y=-0.00665 orient=6.27969] 0.0160731865752
[x=38.98262 y=-0.01015 orient=6.28040] 0.0142337803659
[x=39.98262 y=-0.01293 orient=6.28103] 0.0125221191971
[x=40.98262 y=-0.01509 orient=6.28157] 0.0109440778786
[x=41.98262 y=-0.01671 orient=6.28205] 0.00949759959244
[x=42.98262 y=-0.01784 orient=6.28246] 0.00817841235712
[x=43.98261 y=-0.01857 orient=6.28281] 0.0069812392289
[x=44.98261 y=-0.01895 orient=6.28310] 0.00590017294341
[x=45.98261 y=-0.01904 orient=0.00016] 0.00492887907439
[x=46.98261 y=-0.01887 orient=0.00036] 0.00406074933967
[x=47.98261 y=-0.01851 orient=0.00053] 0.00328902875384
[x=48.98261 y=-0.01798 orient=0.00066] 0.00260692257194
[x=49.98261 y=-0.01732 orient=0.00076] 0.00200768567185
[x=50.98261 y=-0.01656 orient=0.00083] 0.00148469633156
[x=51.98261 y=-0.01573 orient=0.00089] 0.0010315161355
[x=52.98261 y=-0.01484 orient=0.00092] 0.000641937606777
[x=53.98261 y=-0.01392 orient=0.00093] 0.000310021041455
[x=54.98261 y=-0.01299 orient=0.00094] 3.01219046731e-05
[x=55.98261 y=-0.01205 orient=0.00092] -0.000203089964004
[x=56.98261 y=-0.01113 orient=0.00091] -0.000394618197549
[x=57.98261 y=-0.01022 orient=0.00088] -0.000549133766727
[x=58.98261 y=-0.00934 orient=0.00084] -0.000670973929299
[x=59.98261 y=-0.00850 orient=0.00081] -0.0007641454232
[x=60.98261 y=-0.00769 orient=0.00076] -0.000832331152652
[x=61.98261 y=-0.00693 orient=0.00072] -0.000878899699409
[x=62.98261 y=-0.00621 orient=0.00068] -0.000906917068816
[x=63.98261 y=-0.00554 orient=0.00063] -0.000919160152108
[x=64.98261 y=-0.00491 orient=0.00058] -0.000918131452456
[x=65.98261 y=-0.00432 orient=0.00054] -0.000906074682825
[x=66.98261 y=-0.00378 orient=0.00049] -0.000884990898843
[x=67.98261 y=-0.00329 orient=0.00045] -0.000856654879978
[x=68.98261 y=-0.00284 orient=0.00041] -0.000822631517429
[x=69.98261 y=-0.00243 orient=0.00037] -0.000784292007663
[x=70.98261 y=-0.00206 orient=0.00033] -0.000742829686703
[x=71.98261 y=-0.00173 orient=0.00030] -0.000699275372362
[x=72.98261 y=-0.00143 orient=0.00027] -0.000654512109966
[x=73.98261 y=-0.00116 orient=0.00024] -0.000609289241937
[x=74.98261 y=-0.00093 orient=0.00021] -0.000564235743242
[x=75.98261 y=-0.00072 orient=0.00018] -0.00051987278344
[x=76.98261 y=-0.00054 orient=0.00016] -0.000476625492079
[x=77.98261 y=-0.00038 orient=0.00014] -0.000434833917838
[x=78.98261 y=-0.00025 orient=0.00012] -0.000394763183253
[x=79.98261 y=-0.00013 orient=0.00010] -0.000356612846377
[x=80.98261 y=-0.00003 orient=0.00008] -0.000320525488486
[x=81.98261 y=0.00005 orient=0.00007] -0.000286594553179
[x=82.98261 y=0.00012 orient=0.00005] -0.000254871467083
[x=83.98261 y=0.00017 orient=0.00004] -0.000225372076076
[x=84.98261 y=0.00021 orient=0.00003] -0.000198082433563
[x=85.98261 y=0.00025 orient=0.00002] -0.00017296397913
[x=86.98261 y=0.00027 orient=0.00002] -0.000149958146879
[x=87.98261 y=0.00029 orient=0.00001] -0.000128990443104
[x=88.98261 y=0.00030 orient=0.00001] -0.00010997403277
[x=89.98261 y=0.00031 orient=0.00000] -9.28128736236e-05
[x=90.98261 y=0.00031 orient=6.28318] -7.74044357341e-05
[x=91.98261 y=0.00030 orient=6.28318] -6.36420430037e-05
[x=92.98261 y=0.00030 orient=6.28318] -5.14168716151e-05
[x=93.98261 y=0.00029 orient=6.28317] -4.0619638716e-05
[x=94.98261 y=0.00028 orient=6.28317] -3.11420128129e-05
[x=95.98261 y=0.00026 orient=6.28317] -2.28777754606e-05
[x=96.98261 y=0.00025 orient=6.28317] -1.5723761868e-05
[x=97.98261 y=0.00024 orient=6.28317] -9.58060612564e-06
[x=98.98261 y=0.00022 orient=6.28317] -4.35331480382e-06
[x=99.98261 y=0.00021 orient=6.28317] 4.83092176815e-08



Последовательность значений сходится гораздо более мягко к нулю. Это не было достигнуто с пропорциональным контроллером.

Q-12: Систематическая ошибка

Систематическая ошибка является общей проблемой в области робототехники. Примером систематической ошибки может быть небольшой поворот колес робота при нулевом управлении. Если управляет человек, это не будет большой проблемой, т.к. он будет поворачивать немного в противоположную сторону. Что же происходит с пропорциональным регулятором? Добавьте эту строку в функцию run:

```
myrobot.set_steering_drift(10.0 / 180.0 * pi) # 10 degrees
```
Этим мы устанавливаем ошибку поворота колес на 10 градусов. Для этого вопроса зададим первый параметр 0,2, а второй 0.0, для того, чтобы отключить дифференциальное звено. Что происходит с роботом, когда колеса неправильно выровнены?
а) Ничего не изменится
б) Это вызовет большую траекторную ошибку

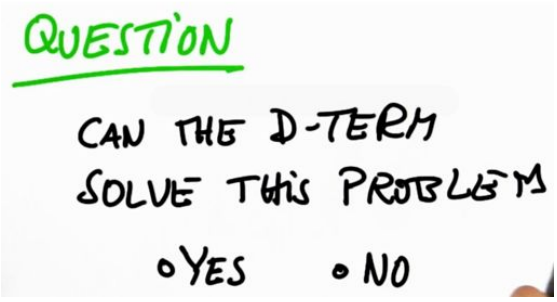
Ответ Ключ к Q-12 A-12: систематическая ошибка

И, конечно, ответ б, это вызывает большие ошибки.

Если вы посмотрите вывод вашей программы, вы обнаружите, что у составляет от 0,7 до 0,9, что является большой ошибкой. Иными словами, робот колеблется с довольно постоянным новым смещением. Хотя смещение было в управлении, оно проявляется как увеличение ошибки по у.

Q-13: Является ли PD-управление достаточным?

Может ли дифференциальная составляющая быть решением проблемы систематической ошибки? Попробуйте использовать программу перед ответом на вопрос.

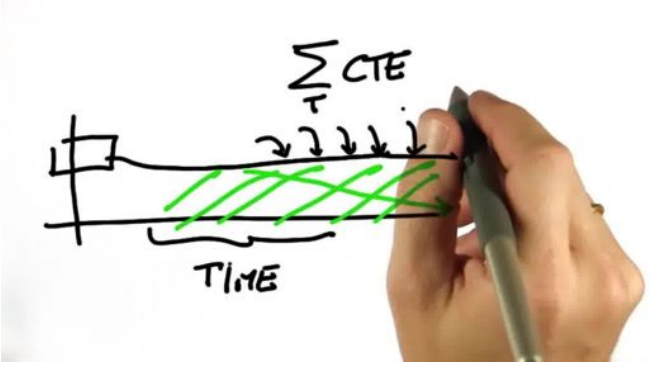


Ответ Ключ к Q-13 A-13: Достаточно ли PD?

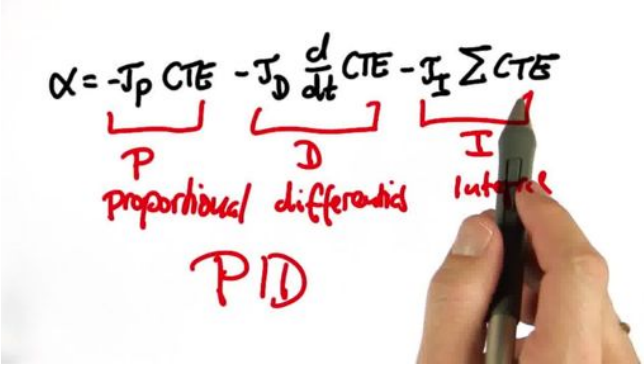
И правильный ответ будет отрицательным. Если вы введете 3.0 для дифференциальной составляющей и запустите программу, вы получите еще большие значения у. Они сходятся к 0,87, но это действительно далеко от ожидаемого 0.0.

Q-14: Реализация PID

Если вы управляете машиной и ваше обычное управление приводит к траектории далеко от цели, и вы заметите, что в течение длительного периода времени вы не приблизились к цели, то вы бы повернете к цели сильнее для компенсации смещения. Вы приспособитесь к систематической ошибке.
Как распознать наличие систематической ошибки? Измерить систематическую ошибку путем суммирования ошибки с течением времени.



Теперь изменим наше управление, чтобы включить эту составляющую.



Чтобы понять, почему это работает, рассмотрим ситуацию с постоянной ошибкой 0,8. В обновленном уравнении компонент ΣCTE будет увеличиваться на 0,8 за каждую единицу времени, что в свою очередь увеличит уставку поворота и в конечном итоге исправит движение робота. Это называется ПИД-регулятором. Он включает в себя пропорциональное, дифференциальное и интегральную составляющие.
Реализуем ПИД-регулятор в коде. Полный код программы:

```
# -----------
# User Instructions
#
# Implement a PID controller by running 100 iterations
# of robot motion. The steering angle should be set
# by the parameter tau so that:
#
# steering = -tau_p * CTE - tau_d * diff_CTE - tau_i * int_CTE
#
# where the integrated crosstrack error (int_CTE) is
# the sum of all the previous crosstrack errors.
# This term works to cancel out steering drift.
#
# Your code should print a list that looks just like
# the list shown in the video.
```

```
#
# Only modify code at the bottom!
# ------------

from math import *
import random


# --------------------------------------------------
#
# this is the robot class
#

class robot:

    # --------
    # init:
    #    creates robot and initializes location/orientation to 0, 0, 0
    #

    def __init__(self, length = 20.0):
        self.x = 0.0
        self.y = 0.0
        self.orientation = 0.0
        self.length = length
        self.steering_noise = 0.0
        self.distance_noise = 0.0
        self.steering_drift = 0.0

    # --------
    # set:
    # sets a robot coordinate
    #

    def set(self, new_x, new_y, new_orientation):

        self.x = float(new_x)
        self.y = float(new_y)
        self.orientation = float(new_orientation) % (2.0 * pi)


    # --------
    # set_noise:
    # sets the noise parameters
    #

    def set_noise(self, new_s_noise, new_d_noise):
        # makes it possible to change the noise parameters
        # this is often useful in particle filters
        self.steering_noise = float(new_s_noise)
        self.distance_noise = float(new_d_noise)

    # --------
    # set_steering_drift:
    # sets the systematical steering drift parameter
    #

    def set_steering_drift(self, drift):
        self.steering_drift = drift

    # --------
    # move:
```

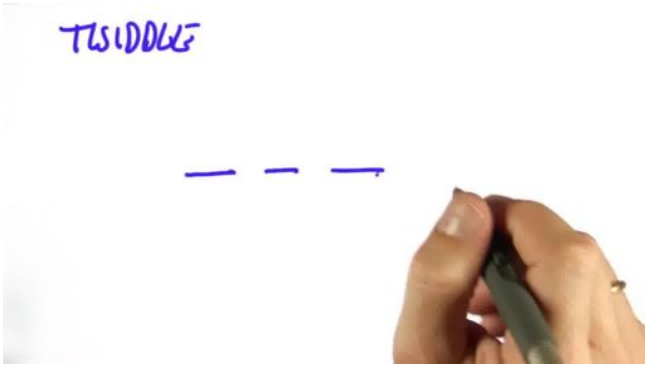
```
    #    steering = front wheel steering angle, limited by max_steering_angle
    #    distance = total distance driven, most be non-negative

    def move(self, steering, distance,
             tolerance = 0.001, max_steering_angle = pi / 4.0):

        if steering > max_steering_angle:
            steering = max_steering_angle
        if steering < -max_steering_angle:
            steering = -max_steering_angle
        if distance < 0.0:
            distance = 0.0


        # make a new copy
        res = robot()
        res.length         = self.length
        res.steering_noise = self.steering_noise
        res.distance_noise = self.distance_noise
        res.steering_drift = self.steering_drift

        # apply noise
        steering2 = random.gauss(steering, self.steering_noise)
        distance2 = random.gauss(distance, self.distance_noise)

        # apply steering drift
        steering2 += self.steering_drift

        # Execute motion
        turn = tan(steering2) * distance2 / res.length

        if abs(turn) < tolerance:

            # approximate by straight line motion

            res.x = self.x + (distance2 * cos(self.orientation))
            res.y = self.y + (distance2 * sin(self.orientation))
            res.orientation = (self.orientation + turn) % (2.0 * pi)

        else:

            # approximate bicycle model for motion

            radius = distance2 / turn
            cx = self.x - (sin(self.orientation) * radius)
            cy = self.y + (cos(self.orientation) * radius)
            res.orientation = (self.orientation + turn) % (2.0 * pi)
            res.x = cx + (sin(res.orientation) * radius)
            res.y = cy - (cos(res.orientation) * radius)

        return res



    def __repr__(self):
        return '[x=%.5f y=%.5f orient=%.5f]'  % (self.x, self.y,
self.orientation)
```

```
############## ADD / MODIFY CODE BELOW ####################

# ------------------------------------------------------------------------
#
# run - does a single control run.


def run(param1, param2, param3):
    myrobot = robot()
    myrobot.set(0.0, 1.0, 0.0)
    speed = 1.0 # motion distance is equal to speed (we assume time = 1)
    N = 100
    myrobot.set_steering_drift(10.0 / 180.0 * pi) # 10 degree bias, this will
be added in by the move function, you do not need to add it below!

    crosstrack_error = myrobot.y
    int_crosstrack_error = 0.0
    for i in range(N):
        diff_crosstrack_error = myrobot.y - crosstrack_error
        crosstrack_error = myrobot.y
        int_crosstrack_error += crosstrack_error
        steer = - param1 * crosstrack_error - param2 * diff_crosstrack_error
- param3 * int_crosstrack_error
        myrobot = myrobot.move(steer, speed)
        print myrobot, steer



# Call your function with parameters of (0.2, 3.0, and 0.004)
run(0.2, 3.0, 0.004)
```

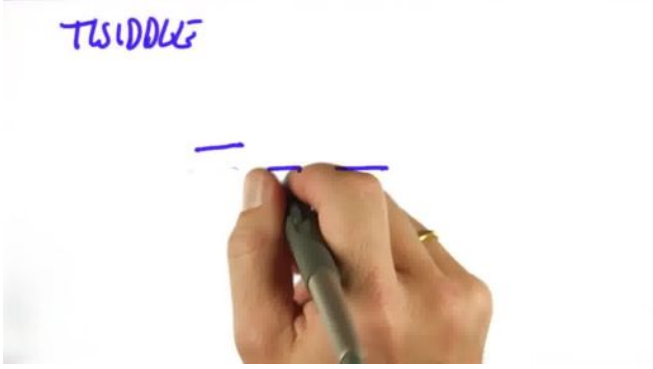
Twiddle

Как определить оптимальные П, И и Д - коэффициенты? Используйте Twiddle-алгоритм чтобы подобрать их!
Начните с вектором параметров p = [0, 0, 0]. Этот вектор представляет лучшее предположение на данный момент. Кроме того, инициализируем вектор dp = [1, 1, 1]. Этот вектор представляет возможные изменения параметров, чтобы применить на следующем шаге.
Запустим симуляцию регулятора с заданными параметрами и выясним, какие из них приводят к наименьшей суммарной ошибке.
Для каждого параметра p, увеличим параметры в соответствии с вектором dp и вычислим ошибку с регулятора с модифицированным вектором параметров. Если эта ошибка меньше, чем минимальная ошибка, найденная до сих пор, попробуем большие значения вектора dp, с коэффициентом 1,1, например. Если нет, попробуем изменить вектор p в обратном направлении путем вычитания соответствующего вектора dp от исходных параметров p для этой итерации. Если это также не производит меньшую ошибку, то задайте dp меньше в 0,9. Продолжайте этот процесс до тех пор, пока сумма элементов вектора dp превышает некоторое заданное пороговое значение.

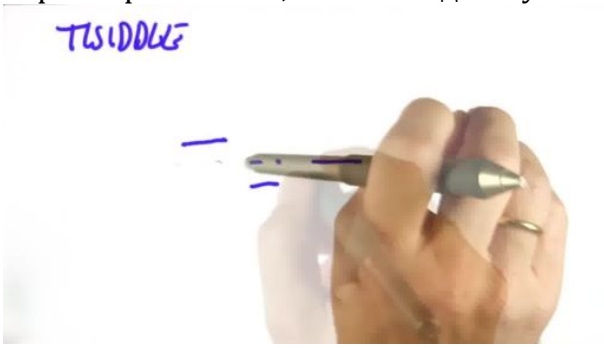
Взгляните на это:
Опять же, начнем с вектором параметров, инициализированным нулями:



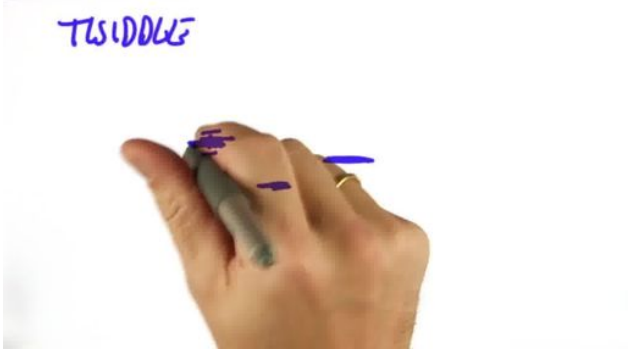
В первой итерации, увеличим первый параметра и посмотрим, как уменьшит ли это ошибку:



Если это произойдет, то переходим ко второму параметру. Для второго параметра, если вы увеличите его, это не улучшит ошибку. Так что теперь, уменьшим второй параметр и скажем, что это ведет к уменьшению ошибки.



Когда вы будете продолжать, возможно, вы обнаружите, что ни увеличение, ни уменьшение ваших параметров не помогает. Уменьшите интервалы зондирования ваших параметров, пока не найдете оптимальные значений параметров.



Реализация Оптимизация параметров

Q-16 (Оптимизация параметров)

Реализуем алгоритм TWIDDLE чтобы найти оптимальные параметры для ПИД-регулятора.
Модифицируем функцию run, чтобы она возвращала среднеквадратическую ошибку после N шагов (всего выполняется 2N шагов).
Реализуем функцию twiddle, которая вызывает функцию run с разными параметрами, подбирая наилучшие, для которых эта среднеквадратическая ошибка минимальна

Полный код программы:

```
# ----------------
# User Instructions
#
# Implement twiddle as shown in the previous two videos.
# Your accumulated error should be very small!
#
# Your twiddle function should RETURN the accumulated
# error. Try adjusting the parameters p and dp to make
# this error as small as possible.
#
# Try to get your error below 1.0e-10 with as few iterations
# as possible (too many iterations will cause a timeout).
# No cheating!
# ------------

from math import *
import random


# --------------------------------------------------
#
# this is the robot class
#

class robot:

    # --------
    # init:
    #    creates robot and initializes location/orientation to 0, 0, 0
    #

    def __init__(self, length = 20.0):
        self.x = 0.0
        self.y = 0.0
        self.orientation = 0.0
        self.length = length
        self.steering_noise = 0.0
        self.distance_noise = 0.0
        self.steering_drift = 0.0

    # --------
    # set:
    # sets a robot coordinate
    #

    def set(self, new_x, new_y, new_orientation):

        self.x = float(new_x)
        self.y = float(new_y)
        self.orientation = float(new_orientation) % (2.0 * pi)
```

```
    # --------
    # set_noise:
    # sets the noise parameters
    #

    def set_noise(self, new_s_noise, new_d_noise):
        # makes it possible to change the noise parameters
        # this is often useful in particle filters
        self.steering_noise = float(new_s_noise)
        self.distance_noise = float(new_d_noise)

    # --------
    # set_steering_drift:
    # sets the systematical steering drift parameter
    #

    def set_steering_drift(self, drift):
        self.steering_drift = drift

    # --------
    # move:
    #    steering = front wheel steering angle, limited by max_steering_angle
    #    distance = total distance driven, most be non-negative

    def move(self, steering, distance,
             tolerance = 0.001, max_steering_angle = pi / 4.0):

        if steering > max_steering_angle:
            steering = max_steering_angle
        if steering < -max_steering_angle:
            steering = -max_steering_angle
        if distance < 0.0:
            distance = 0.0


        # make a new copy
        res = robot()
        res.length         = self.length
        res.steering_noise = self.steering_noise
        res.distance_noise = self.distance_noise
        res.steering_drift = self.steering_drift

        # apply noise
        steering2 = random.gauss(steering, self.steering_noise)
        distance2 = random.gauss(distance, self.distance_noise)

        # apply steering drift
        steering2 += self.steering_drift

        # Execute motion
        turn = tan(steering2) * distance2 / res.length

        if abs(turn) < tolerance:

            # approximate by straight line motion

            res.x = self.x + (distance2 * cos(self.orientation))
            res.y = self.y + (distance2 * sin(self.orientation))
            res.orientation = (self.orientation + turn) % (2.0 * pi)
```

```
        else:

            # approximate bicycle model for motion

            radius = distance2 / turn
            cx = self.x - (sin(self.orientation) * radius)
            cy = self.y + (cos(self.orientation) * radius)
            res.orientation = (self.orientation + turn) % (2.0 * pi)
            res.x = cx + (sin(res.orientation) * radius)
            res.y = cy - (cos(res.orientation) * radius)

        return res




    def __repr__(self):
        return '[x=%.5f y=%.5f orient=%.5f]'  % (self.x, self.y,
self.orientation)


# --------------------------------------------------------------------
#
# run - does a single control run.


def run(params, printflag = False):
    myrobot = robot()
    myrobot.set(0.0, 1.0, 0.0)
    speed = 1.0
    err = 0.0
    int_crosstrack_error = 0.0
    N = 100
    # myrobot.set_noise(0.1, 0.0)
    myrobot.set_steering_drift(10.0 / 180.0 * pi) # 10 degree steering error


    crosstrack_error = myrobot.y


    for i in range(N * 2):

        diff_crosstrack_error = myrobot.y - crosstrack_error
        crosstrack_error = myrobot.y
        int_crosstrack_error += crosstrack_error

        steer = - params[0] * crosstrack_error  \
            - params[1] * diff_crosstrack_error \
            - int_crosstrack_error * params[2]
        myrobot = myrobot.move(steer, speed)
        if i >= N:
            err += (crosstrack_error ** 2)
        if printflag:
            print myrobot, steer
    return err / float(N)


def twiddle(tol = 0.2): #Make this tolerance bigger if you are timing out!
    ############### ADD CODE BELOW ####################

    # -------------
    # Add code here
```

```
    # -------------
    n_params = 3
    dparams = [1.0 for row in range(n_params)]
    params = [0.0 for row in range(n_params)]
    best_error = run(params)
    n=0
    while sum(dparams) > tol:
        for i in range(len(params)):
            params[i] += dparams[i]
            err = run(params)
            if err < best_error:
                best_error = err
                dparams[i] *= 1.1
            else:
                params[i] -= 2.0 * dparams[i]
                err = run(params)
                if err < best_error:
                    best_error = err
                    dparams[i] *= 1.1
                else:
                    params[i] += dparams[i]
                    dparams[i] *= 0.9
        n += 1
        print 'Twiddle #', n, params, ' -> ', best_error

    return run(params)

twiddle(0.01)
```