# 课程7 四足机器人运动学逆解可视化

课程概述

在这个教程中，我们将学习机器人的运动学逆解。

关于课程

按照规划，本课程需要花费大约2小时来完成，目标受众为四足机器人开发爱好者。
note
本课程是基于Mangdang的Mini Pupper系列产品搭建的。
本课程涉及的源代码托管在Github，你可以点击此处访问。

学习目标

在完成这个教程后，你将能够：

1. 了解逆运动学的有无解、有无多解情况。
2. 了解运动学逆解的求解。
3. 熟悉逆运动学中求解的几何法和代数法。
4. 熟悉单腿舵机的简单校准。
5. 掌握可视化逆向运动学计算结果的方法。

课程细节

在这个课程中，我们将学习机器人运动学中的逆向运动学。
逆向运动学则是探究已知工具坐标系$\{H\}$的位置和姿态或$^{World}P$，求解满足要求的θ_i的问题。

准备材料

在开始这个教程之前，你需要准备以下材料：

- 一台安装了Python的PC电脑
- 安装了绘图所需的依赖

引言

在这个教程中，我们将学习运动学逆解的多解情况和求解方法。

整体步骤

这个教程将分为几个任务：

1. 拼装一条mini pupper的腿部
2. 学习逆运动学基础知识
3. 运行程序，观察各种情况

请按照教程中的步骤完成每个任务。

任务1：拼装一条mini pupper的腿部

请参照mini pupper docs中的腿部组装办法，拼装一条mini pupper的腿部。
参考链接：mini pupper docs

任务2：学习逆运动学基础知识

1.什么是逆向运动学 Inverse Kinematics

正向运动学探究的是已知关节角θ_i，求解工具坐标系$\{H\}$或${}^{World}P$的问题。
而逆向运动学则是探究已知工具坐标系$\{H\}$的位置和姿态或${}^{World}P$，求解满足要求的θ_i的问题。
运动学方程解的有无定义了**工作空间**，有解则表示机械臂能到达这个目标点，无解则表示机械臂无法到达这个点，这个目标点位于工作空间之外。
基本的逆运动学可以看做是给定操作臂末端执行器的位置和姿态，计算所有可达给定位置和姿态的关节角的问题，可以认为是机器人位姿从笛卡尔空间到关节空间的“定位”映射。

2.什么是封闭解和数值解

逆运动学不像正运动学那么容易，逆运动学是非线性的，难以找到封闭解，有时候无解，有时候有有多解的问题，这种非线性的超越方程组，没有规矩的、统一、通用的解法，解法分为封闭解法和数值解法。
封闭解是由数学公式的推导得出，对于任意自变量均能求出对应的因变量，计算量可能相对较多，精度高。
数值解则是可以由离散查表或者是插值一类的方法去模拟最终情况，计算量相对较小，精度相对较差。
因此，我们需要根据实际情况来考虑逆向运动学的解和解的情况。

2.解存在吗?

在逆向运动学（IK）中，我们可以通过给定点相对于世界坐标系（Frame World）的坐标来解算出机器人的手臂关节应该旋转的角度θ。
假如给定的一个位置是在很远的地方，机器人的手臂完全够不着，那么求解是没有意义的，因此我们将会用到工作空间来描述机器人可触达的区域。

工作空间

工作空间是手臂末端所能到达的位置范围。指定的目标点必须在工作空间内，逆向运动学求解才有意义。
为了进一步描述工作空间，可以用以下常见的这两种工作空间的表示：
可达工作空间 Reachable workspace
是手臂能用一种或以上的姿态能够到达的位置范围。可达目标坐标系可以描述这个Frame相对于世界坐标系的位置，而一系列的可达目标坐标系的集合构成了可达工作空间。
灵巧工作空间 Dexterous workspace
是手臂末端用任何姿态都能够到达的位置，条件相当苛刻，比如平面2DOFs的RR机械臂模型中L1=L2的摆臂的圆心，在这个模型中，仅此一点是灵巧工作空间。灵巧工作空间是可达工作空间的子集。

3.是否有多个解?

在求解运动学方程时常常会遇到不只一个解的情况。比如平面中具有三个旋转关节的机器人手臂，对于同一个点P，这三个旋转关节可以有不同的位形，在不同的位型下，手臂末端的执行器的可达位置和姿态可以是相同的。
图片：三连杆操作臂多解图

解的选取

对于解的选取有一些基本原则：

1. 速度最快
2. 能耗最低
3. 避开障碍物
4. 在关节允许活动的范围限制内

4.如何求解?

求解操作臂运动学方程是非线性的问题，非线性方程组没有通用的求解算法，算法需要针对机器人手臂的模型来制定。如果某一算法可以解出与已知位姿相关的全部关节变量，那么这个机器人手臂就是可解的。
从$Frame_{object}$到$Frame_{World}$的变换矩阵${}_{o}^{W}T$中的转动部分和平移部分可以提取出含未知数的16个数字。
其中的旋转矩阵被xyz相互垂直、xyz为单位向量这六个条件限制到只有三个自由度，其中的位置矢量分量的三个方程有三个自由度，共有6个限制条件，6个自由度，这些方程为非线性超越方程，求解不易。
对于六旋转关节的机械臂，存在解析解（封闭解）的充分条件是相邻的三关节的转轴交于一点。

$$ {}_6^0T = \begin{bmatrix} {}_6^0R_{3x3} & {}^0P_{6\ ORG\ 3x1} \\ 0\ 0\ 0 & 1 \end{bmatrix}_{4x4} = \begin{bmatrix} \hat{X}_6\cdot\hat{X}_0 & \hat{Y}_6\cdot\hat{X}_0 & \hat{Z}_6\cdot\hat{X}_0 & {}_6^0P_{Xorg} \\ \hat{X}_6\cdot\hat{Y}_0 & \hat{Y}_6\cdot\hat{Y}_0 & \hat{Z}_6\cdot\hat{Y}_0 & {}_6^0P_{Yorg} \\ \hat{X}_6\cdot\hat{Z}_0 & \hat{Y}_6\cdot\hat{Z}_0 & \hat{Z}_6\cdot\hat{Z}_0 & {}_6^0P_{Zorg} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

对于基于解析形式的解法，常见的求解方法有几何法和代数法。两种方法相似，求解过程不同。

几何法

几何法求解机械臂的逆运动学问题时，常常需要将空间几何参数转化为平面几何的问题。在$\alpha_i = 0$或$\pm 90^\circ$时几何法会非常容易，应用平面几何常见的公式及角度转换即可求出θ_i的值。

$$ x^2 + y^2 = l_1^2 + l_2^2 - 2l_1l_2(\pi - \theta_2) \tag{余弦定理} $$

$$ Cos\theta_2 = \frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2} \tag{变形1} $$

$$ Cos\psi = \frac{(x^2+y^2)\ \ + l_1^2 - l_2^2}{2l_1\sqrt{x^2+y^2}} \tag{变形2} $$

$$ \theta_1 = \begin{cases} atan2(y,x) + \psi & \theta_2 < 0 \\ atan2(y,x) - \psi & \theta_2 > 0 \end{cases} $$

在计算完θ_2和θ_1后，根据图的几何角度关系，又可算得θ_3，即成功反解运动学的各θ_n

math.atan2()方法

math.atan2()方法是双变量反正切公式，可以计算给定y,x值的反正切值，也就是以弧度形式表达的该段终点与起点连线斜率线的一个角度值。
atan2()优于atan()，因为可以计算x2-x1=0的情况。
参考链接：Python math.atan2(y,x)
计算空间中缺少的自由度

代数法

应用连杆参数（α_{i-1},a_{i-1},θ_i,d_i），通过运动学正解（FK）可以求得机械臂的运动学方程，表现形式为变换矩阵${}_3^0T$。因此，目标点的位置是由手臂末端坐标系相对基坐标系来定的，当研究对象为平面机械臂时，只需要知道x,y,ϕ即可确定目标点位置。
ϕ是 末端杆在平面内的姿态角
将已知的${}_3^0T$与新建立的${}_{object}^{world}T$取等，即可获得对应位置的值相等。

$$ \begin{bmatrix} c_{123} & -s_{123} & 0.0 & l_1c_1 + l_2c_{12} \\ s_{123} & c_{123} & 0.0 & l_1s_1 + l_2s_{12} \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_\phi & -s_\phi & 0.0 & x \\ s_\phi & c_\phi & 0.0 & y \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

利用三角函数和角公式

$$ Sin_{1\pm 2} = Sin_1Cos_2 \pm Cos_1Sin_2 $$

$$ Cos_{1\pm 2} = Cos_1Cos_2 \mp Sin_1Sin_2 $$

可得

$$ Cos\theta_2 = \frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2} $$

此式在$1 \geq Cos\theta_2 \geq -1$时有解

假设目标点在工作空间内，又有

$$Sin\theta_2 = \pm\sqrt{1-c^2}$$

应用几何法中提到的math.Atan2()求解θ_2，利用θ_2再去对其他θ_n求解，具体方法参考教材，本处仅作代数法引入。
通俗的来说，就是确认θ_n的$Sin\theta_n$和$Cos\theta_n$，再利用双变量反正切公式math.Atan2()求θ_n

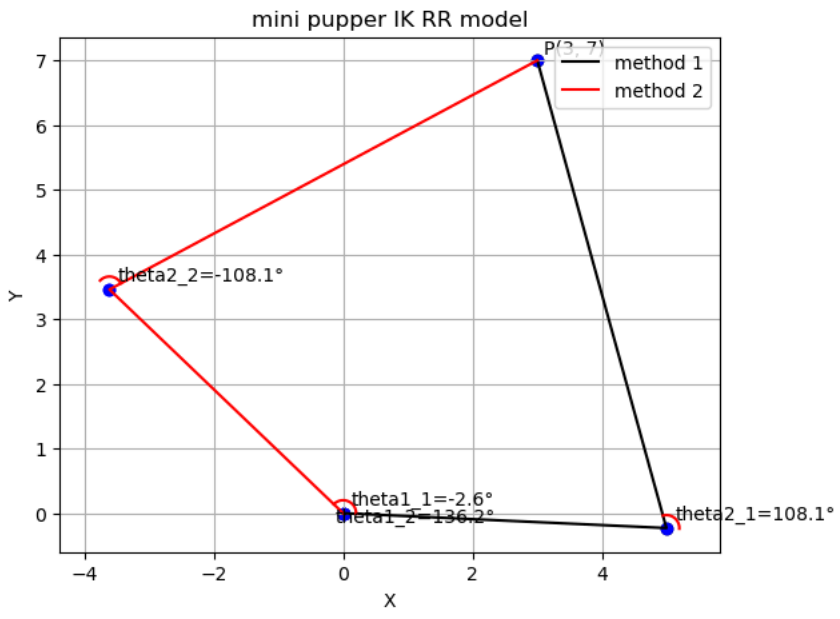
任务3：运行程序

程序1：逆运动学的多解与求解

运行程序，观察运动学逆解的多解情况，观察程序中运动学逆解的求解方法。

```
sudo python rr_IK.py
# Example values: 3 7
```

```python
#!/usr/bin/python
# coding:utf-8
# rr_IK.py
# Inverse kinematics IK
# The simplified single leg of mini pupper can be regarded as an RR robot arm on the same plane, visualize the robot arm, and calculate the rot
import matplotlib.pyplot as plt # import matplotlib
import numpy as np # import numpy
from math import degrees, sin, cos


# Geometry method: end point coordinates to joint angles
def position_2_theta(x, y, l1, l2):
    """
    Kinematics inverse solution Convert the input endpoint coordinates into corresponding joint angles
    :param x: point p coordinate x value
    :param y: point p coordinate y value
    :param l1: arm length
    :param l2: forearm length
    :return: joint angle 1 value 1 joint angle 1 value 2 joint angle 2 value 1 joint angle 2 value 1
    """
    cos2 = (x ** 2 + y ** 2 - l1 ** 2 - l2 ** 2) / (2 * l1 * l2)
    # print(cos2)
    sin2_1 = np. sqrt(1 - cos2 ** 2)
    sin2_2 = -sin2_1
    # print(sin2_1)
    # print("sin2 has two values, they are sin2_1=%f, sin2_2=%f" % (sin2_1, sin2_2)) # If you consider the joint situation, you can only take a
    theta2_1 = np.arctan2(sin2_1, cos2)
    theta2_2 = np.arctan2(sin2_2, cos2)
    phi_1 = np.arctan2(l2 * sin2_1, l1 + l2 * cos2)
    phi_2 = np.arctan2(l2 * sin2_2, l1 + l2 * cos2)
    theta1_1 = np.arctan2(y, x) - phi_1
    theta1_2 = np.arctan2(y, x) - phi_2
    # print(degrees(theta1_1), degrees(theta1_2), degrees(theta2_1), degrees(theta2_2))
    return theta1_1, theta1_2, theta2_1, theta2_2


def preprocess_drawing_data(theta1, theta2, l1, l2):
    """
    Process angle data and convert it to a drawing format adapted to matplotlib
    :param theta1: angle data 1
    :param theta2: angle data 2
    :param l1: bar length 1
    :param l2: bar length 2
    :return: x-coordinate list and corresponding y-coordinate list of drawing data
    """
    xs = [0]
    ys = [0]
    # Calculate x1 y1 and x2 y2 respectively
    x1 = l1 * cos(theta1)
    y1 = l1 * sin(theta1)
    x2 = x1 + l2 * cos(theta1 + theta2)
    y2 = y1 + l2 * sin(theta1 + theta2)
    xs.append(x1)
    xs.append(x2)
    ys.append(y1)
    ys.append(y2)
    return xs, ys


def annotate_angle(x0, y0, rad1, rad2, name, inverse=False):
    """
    draw angles for two lines
    :param x0: x coordinate of the center of the circle
    :param y0: x coordinate of the center of the circle
    :param rad1: starting angle
    :param rad2: end angle
    :param name: role name
    :param inverse: used to solve the overlapping problem of point 1
    :return: None
    """
    theta = np.linspace(rad1, rad2, 100)  # 0~rad
    r = 0.2  # circle radius
    x1 = r * np.cos(theta) + x0
    y1 = r * np.sin(theta) + y0
    plt.plot(x1, y1, color='red')
    plt.scatter(x0, y0, color='blue')
    degree = degrees((rad2 - rad1))
    if inverse:
        plt.annotate("%s=%.1f°" % (name, degree), [x0, y0], [x0 - r / 1.5, y0 - r / 1.5])
    else:
        plt.annotate("%s=%.1f°" % (name, degree), [x0, y0], [x0 + r / 1.5, y0 + r / 1.5])


# Joint information
# Arm length: 5 cm Arm length: 7.5 cm
link_length = [5, 7.5]  # in cm
# input end position
position_pre = input("Please enter the x-coordinate and y-coordinate of the end, separated by spaces:")
position = [float(n) for n in position_pre. split()]
print(position)

# Compute and preprocess plot data
joints_angles = position_2_theta(position[0], position[1], link_length[0], link_length[1])
# print(joints_angles)
figure1 = preprocess_drawing_data(joints_angles[0], joints_angles[2], link_length[0], link_length[1])
figure2 = preprocess_drawing_data(joints_angles[1], joints_angles[3], link_length[0], link_length[1])
# print(figure1)
# print(figure2)

# drawing
fig, ax = plt.subplots()  # build image
plt. axis("equal")
ax. grid()
plt.plot(figure1[0], figure1[1], color='black', label='method 1')
plt.scatter(figure1[0], figure1[1], color='black')
plt.plot(figure2[0], figure2[1], color='red', label='method 2')
plt.scatter(figure2[0], figure2[1], color='blue')
ax.set(xlabel='X', ylabel='Y', title='mini pupper IK RR model')
plt. legend()
# Annotation
annotate_angle(figure1[0][0], figure1[1][0], 0, joints_angles[0], "theta1_1")
annotate_angle(figure1[0][1], figure1[1][1], joints_angles[0], joints_angles[2]+joints_angles[0], "theta2_1")
annotate_angle(figure2[0][0], figure2[1][0], 0, joints_angles[1], "theta1_2", inverse=True)
annotate_angle(figure2[0][1], figure2[1][1], joints_angles[1], joints_angles[3]+joints_angles[1], "theta2_2")
plt.annotate("P(%d, %d)" % (position[0], position[1]), [figure1[0][2], figure1[1][2]], [figure1[0][2] + 0.1, figure1[1][2] + 0.1])
plt.tight_layout()
plt. show()
```

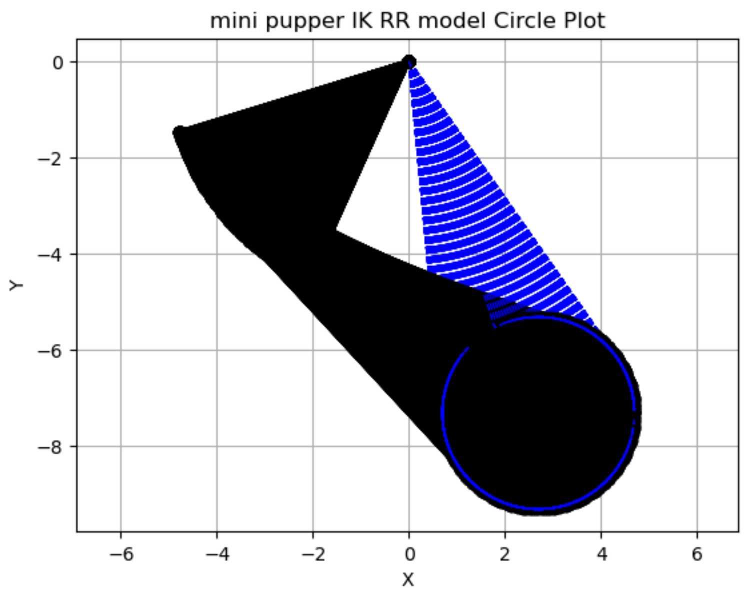


图片1：rr_IK

程序2：逆运动学可视化

观察程序，通过圆轨迹的运动学逆解来观察mini pupper腿部的运动。

```python
#!/usr/bin/python
# coding:utf-8
# rr_IK_circle.py
# Inverse kinematics IK
# The simplified single leg of mini pupper can be regarded as an RR-like robotic arm on the same plane, and the inverse kinematics of the quad
import matplotlib.pyplot as plt # import matplotlib
import numpy as np # import numpy
from math import degrees, radians, sin, cos
import matplotlib.animation as animation


# Geometry method: end point coordinates to joint angles
def position_2_theta(x, y, l1, l2):
    """
    Kinematics inverse solution Convert the input endpoint coordinates into corresponding joint angles
    :param x: point p coordinate x value
    :param y: point p coordinate y value
    :param l1: arm length
    :param l2: forearm length
    :return: joint angle 1 value 1 joint angle 1 value 2 joint angle 2 value 1 joint angle 2 value 1
    """
    cos2 = (x ** 2 + y ** 2 - l1 ** 2 - l2 ** 2) / (2 * l1 * l2)
    sin2_1 = np. sqrt(1 - cos2 ** 2)
    sin2_2 = -sin2_1
    theta2_1 = np.arctan2(sin2_1, cos2)
    theta2_2 = np.arctan2(sin2_2, cos2)
    phi_1 = np.arctan2(l2 * sin2_1, l1 + l2 * cos2)
    phi_2 = np.arctan2(l2 * sin2_2, l1 + l2 * cos2)
    theta1_1 = np.arctan2(y, x) - phi_1
    theta1_2 = np.arctan2(y, x) - phi_2
    return theta1_1, theta1_2, theta2_1, theta2_2


def preprocess_drawing_data(theta1, theta2, l1, l2):
    """
    Process angle data and convert it to a drawing format adapted to matplotlib
    :param theta1: angle data 1
    :param theta2: angle data 2
    :param l1: bar length 1
    :param l2: bar length 2
    :return: x-coordinate list and corresponding y-coordinate list of drawing data
    """
    xs = [0]
    ys = [0]
    # Calculate x1 y1 and x2 y2 respectively
    x1 = l1 * cos(theta1)
    y1 = l1 * sin(theta1)
    x2 = x1 + l2 * cos(theta1 + theta2)
    y2 = y1 + l2 * sin(theta1 + theta2)
    xs.append(x1)
    xs.append(x2)
    ys.append(y1)
    ys.append(y2)
    return xs, ys


def animate_plot(n):
    # Generate circular trajectory
    circle_point = [2.696152422706633, -7.330127018922193] # The center of the circular motion
    position = [0, 0]
    history_position_x = [0]
    history_position_y = [0]
    circle_r = 2
    theta = n * np.pi / 100
    position[0] = circle_point[0] + circle_r * np.cos(theta)
    position[1] = circle_point[1] + circle_r * np.sin(theta)

    # Compute and preprocess plot data
    joints_angles = position_2_theta(position[0], position[1], link_length[0], link_length[1])
    figure1 = preprocess_drawing_data(joints_angles[0], joints_angles[2], link_length[0], link_length[1])

    # Trajectory tracking
    for i in range(0, (n % 200)+1):
        history_theta = ((n % 200) + 1 - i) * np.pi / 100
        history_position_x.append(circle_point[0] + circle_r * np.cos(history_theta))
        history_position_y.append(circle_point[1] + circle_r * np.sin(history_theta))

    # drawing
    p = plt.plot(figure1[0], figure1[1], 'o-', lw=2, color='black')
    p += plt.plot(history_position_x, history_position_y, '--', color='blue', lw=1)
    return p


# Joint information
# Arm length: 5 cm Arm length: 7.5 cm
link_length = [5, 7.5] # in cm

# matplotlib visualization part
fig, ax = plt.subplots() # build image
plt. axis("equal")
plt. grid()
ax.set(xlabel='X', ylabel='Y', title='mini pupper IK RR model Circle Plot')
ani = animation.FuncAnimation(fig, animate_plot, interval=10, blit=True)
plt. show()
```



图片2：rr_IK_circle

程序3：校准舵机

将组装好的单腿各舵机线材，按照程序中提示的接线接入，对舵机2与舵机3进行回零校准。
此程序需要在树莓派电脑上执行

```python
#!/usr/bin/python
# coding:utf-8
# servo_calibrate.py
# By default, all servos return to zero, and then wait for the input angle

import RPi.GPIO as GPIO
import time


def servo_map(before_value, before_range_min, before_range_max, after_range_min, after_range_max):
    """
    Function: Map a range of values to another range of values
    Parameters: a certain value in the original range, the minimum value in the original range, the maximum value in the original range, the m
    Return: the transformed range corresponds to a certain value
    """
    percent = (before_value - before_range_min) / \
        (before_range_max - before_range_min)
    after_value = after_range_min + percent * \
        (after_range_max - after_range_min)
    return after_value


signal_ports = input(
    "Enter the signal port numbers of each servo, separated by spaces, press Enter if there is no input, the default signal port is: 32 33 35\
signal_ports = [int(n) for n in signal_ports. split()]
for i in range(0, len(signal_ports)):
    print("The port corresponding to steering gear %d is %d" %
          (i+1, signal_ports[i]))

GPIO.setmode(GPIO.BOARD)  # Initialize GPIO pin coding mode
servo = [0, 0, 0]
servo_SIG = signal_ports  # PWM signal port
servo_VCC = [2, 4, 1]  # VCC terminal
servo_GND = [30, 34, 39]  # GND terminal
servo_freq = 50  # PWM frequency
servo_width_min = 2.5  # Minimum working pulse width
servo_width_max = 12.5  # maximum working pulse width
GPIO.setmode(GPIO.BOARD)  # Initialize GPIO pin coding mode
for i in range(0, len(servo_SIG)):
    GPIO.setup(servo_SIG[i], GPIO.OUT)
    servo[i] = GPIO.PWM(servo_SIG[i], servo_freq)
    servo[i].start(0)
    # Return to the center position of the servo
    servo[i].ChangeDutyCycle((servo_width_min + servo_width_max) / 2)
print("Initialization back to zero is complete, wait for input after two seconds")
time. sleep(2)

# Specify the position for the servo
try:  # try and except are a fixed combination, used to capture whether the user presses ctrl+C to terminate the program during execution
    while 1:
        angles = input(
            "If you need to change the angle of the servo, please input the angle value of 0°-180° for different servos:\n")
        angles = [int(n) for n in angles. split()]

        for i in range(0, len(angles)):
            dc_trans = servo_map(
                angles[i], 0, 180, servo_width_min, servo_width_max)
            servo[i].ChangeDutyCycle(dc_trans)
            print("The servo %d has turned to %d°" % (i+1, angles[i]))
except KeyboardInterrupt:
    pass
for i in range(0, len(servo_SIG)):
    servo[i].stop()  # stop pwm
GPIO.cleanup()  # Clean up GPIO pins
```

程序4：观察运动学逆解的实际运行

运动学逆解的实际运行会受到非常多因素的干扰，例如校准情况、杆间的测量误差、信号线材的传输波动、树莓派本身的计算能力。
值得一提的是，千机千面，舵机的校准每个人遇到的情况不同，对于单腿的舵机，在3中提到的校准程序只能帮助你发现简单的安装错误，如果需要实际校准，需要运行整机的可视化校准代码。

```python
#!/usr/bin/python
# coding:utf-8
# rr_IK_circle_synchronous.py
# Kinematics inverse solution to draw a circle, synchronize the image display of the control terminal and the hardware movement
import matplotlib.pyplot as plt  # import matplotlib
import numpy as np  # import numpy
from math import degrees, sin, cos
import matplotlib.animation as animation
import time
import RPi.GPIO as GPIO


# Geometry method: end point coordinates to joint angles
def position_2_theta(x, y, l1, l2):
    """
    Kinematics inverse solution Convert the input endpoint coordinates into corresponding joint angles
    :param x: point p coordinate x value
    :param y: point p coordinate y value
    :param l1: arm length
    :param l2: forearm length
    :return: joint angle 1 value 1 joint angle 1 value 2 joint angle 2 value 1 joint angle 2 value 1
    """
    cos2 = (x ** 2 + y ** 2 - l1 ** 2 - l2 ** 2) / (2 * l1 * l2)
    sin2_1 = np. sqrt(1 - cos2 ** 2)
    sin2_2 = -sin2_1
    theta2_1 = np.arctan2(sin2_1, cos2)
    theta2_2 = np.arctan2(sin2_2, cos2)
    phi_1 = np.arctan2(l2 * sin2_1, l1 + l2 * cos2)
    phi_2 = np.arctan2(l2 * sin2_2, l1 + l2 * cos2)
    theta1_1 = np.arctan2(y, x) - phi_1
    theta1_2 = np.arctan2(y, x) - phi_2
    return theta1_1, theta1_2, theta2_1, theta2_2


def servo_map(before_value, before_range_min, before_range_max, after_range_min, after_range_max):
    """
    Function: Map a range of values to another range of values
    Parameters: a certain value in the original range, the minimum value in the original range, the maximum value in the original range, the m
    Return: the transformed range corresponds to a certain value
    """
    percent = (before_value - before_range_min) / \
        (before_range_max - before_range_min)
    after_value = after_range_min + percent * \
        (after_range_max - after_range_min)
    return after_value


def theta_to_servo_degree(theta, servo_number, relation_list, config_calibration_value=None):
    """
    Convert the angle of the rod to the angle of the steering gear
    :param theta: member angle in radians
    :param servo_number: servo number
    :param relation_list: Servo relationship mapping table
    :param config_calibration_value:
    :return: servo angle in angle system
    """
    if config_calibration_value is None:
        config_calibration_value = [0, 0, 0]
    theta = degrees(theta)
    servo_degree = 0
    if servo_number == 1:
        # print("servo1")
        servo_degree = 0  # here needs to be modified according to servo 1
    elif servo_number == 2:
        # print("servo2")
        servo_degree = theta + relation_list[1] + config_calibration_value[1]
    elif servo_number == 3:
        # print("servo3")
        servo_degree = theta + relation_list[2] + config_calibration_value[2]
    else:
        # print("ERROR: theta_to_servo_degree")
        servo_degree = 0
    return servo_degree


def servo_control(servo_number, degree):
    """
    Use the angle value to control the corresponding angle of the motor output
    :return:
    """
    dc_trans = servo_map(degree, 0, 180, servo_width_min, servo_width_max)
    servo[servo_number - 1].ChangeDutyCycle(dc_trans)
    print("Servo %d has turned to %f°" % (servo_number, degree))


def circle_point_generate(center_point, radius, frame):
    """
     Enter the center of the circle [x0, y0], the radius r, and the count c, and return the coordinates [x, y] of a single point on the circum
    :param center_point: circle center [x0,y0]
    :param radius: radius
    :param frame: the number of sample points for segmentation
    :return: A list of two arrays composed of the coordinates [x,y] of a single point on the circumference
    """
    theta = np.linspace(0, 2 * np.pi, frame)
    xs = center_point[0] + radius * np.cos(theta)
    ys = center_point[1] + radius * np.sin(theta)
    return xs, ys


def preprocess_drawing_data(theta1, theta2, l1, l2):
    """
    Process angle data and convert it to a drawing format adapted to matplotlib
    :param theta1: angle data 1
    :param theta2: angle data 2
    :param l1: bar length 1
    :param l2: bar length 2
    :return: x-coordinate array and y-coordinate array of leg points
    """
    xs = [0]
    ys = [0]
    # Calculate x1 y1 and x2 y2 respectively
    x1 = l1 * cos(theta1)
    y1 = l1 * sin(theta1)
    x2 = x1 + l2 * cos(theta1 + theta2)
    y2 = y1 + l2 * sin(theta1 + theta2)
    xs.append(x1)
    xs.append(x2)
    ys.append(y1)
    ys.append(y2)
    return xs, ys


def animation_update(frame):
    """
    Update animation and sync to hardware motor
    Note: The matplotlib online animation lock frame is at 30fps, and the frame should not be higher than 30
    :param frame:
    :return:
    """
    # Hardware servo motion synchronization
    servo_control(1, servo_degree[0][frame])
    servo_control(2, servo_degree[1][frame])

    # Update circle drawing
    circle_artist.set_xdata(xs[0:frame])  # set x directly
    circle_artist.set_ydata(ys[0:frame])  # set y directly
    # update leg drawing
    leg_artist.set_xdata(leg_data_xs[frame])  # set x directly
    leg_artist.set_ydata(leg_data_ys[frame])  # set y directly
    return circle_artist, leg_artist


# Configuration and initialization
center_point = [1.767767, -8.838835]  # The center of the circular motion
radius = 2  # circle radius
frame = 60  # number of split samples
leg_data_xs = []  # data x of each point of leg
leg_data_ys = []  # Data y of each point of leg
position = [0, 0]  # The position passed to the servo
link_length = [5, 7.5]  # Length of the member in cm
config_degree_relation_list = [+0, +225, +0]
servo = [0, 0, 0]
servo_degree = [[], []]  # Servo data table
servo_SIG = [32, 33]  # PWM signal terminal
servo_VCC = [2, 4, 1]  # VCC terminal
servo_GND = [30, 34, 39]  # GND terminal
servo_freq = 50  # PWM frequency
servo_width_min = 2.5  # Minimum working pulse width
servo_width_max = 12.5  # maximum working pulse width
GPIO.setmode(GPIO.BOARD)  # Initialize GPIO pin coding mode
for i in range(0, len(servo_SIG)):
    GPIO.setup(servo_SIG[i], GPIO.OUT)
    servo[i] = GPIO.PWM(servo_SIG[i], servo_freq)
    servo[i].start(0)
    # Return to the center position of the servo
    servo[i].ChangeDutyCycle((servo_width_min + servo_width_max) / 2)


# circle trajectory generation
xs, ys = circle_point_generate(center_point, radius, frame)
# leg trajectory generation
for i in range(0, frame):
    position[0] = xs[i]
    position[1] = ys[i]
    # Get kinematics inverse solution value
    joints_angles = position_2_theta(
        position[0], position[1], link_length[0], link_length[1])
    # Convert the inverse solution value to the drawing data
    leg_data_pre = preprocess_drawing_data(
        joints_angles[0], joints_angles[2], link_length[0], link_length[1])
    leg_data_xs.append(leg_data_pre[0])
    leg_data_ys.append(leg_data_pre[1])
    # Rod angle to servo angle
    servo_degree[0].append(theta_to_servo_degree(
        joints_angles[0], 2, config_degree_relation_list))
    servo_degree[1].append(theta_to_servo_degree(
        joints_angles[2], 3, config_degree_relation_list))

print("Initialization is complete, wait for operation after 1 second")
time. sleep(1)
# matplotlib visualization part
fig, ax = plt.subplots(figsize=(6, 6))  # create image
plt. axis("equal")
plt. grid()
circle_artist = ax.plot(xs[0], ys[0], '--', lw=2, color='blue')[0]
leg_artist = ax.plot(
    leg_data_xs[0], leg_data_ys[0], 'o-', lw=2, color='black')[0]
ax.set(xlim=[-6, 7], ylim=[-12, 1], xlabel='X', ylabel='Y',
       title='mini pupper IK RR model Circle Plot')
# plt. tick_params(axis="both")
# set animation, interval unit is ms
ani = animation.FuncAnimation(
    fig=fig, func=animation_update, frames=frame, interval=1, blit=True)

plt. show()
plt.clf("all")
for i in range(0, len(servo_SIG)):
    servo[i].stop()  # stop pwm
```

```
GPIO.cleanup()  # Clean up GPIO pins
```

总结

经过本知识点的学习和实验操作，你应该能达到以下水平：

知识点	内容	了解	熟悉	掌握
逆运动学	运动学逆解的有无解、有无多解情况	✓		
逆运动学	运动学逆解的求解	✓		
逆运动学	几何法和代数法		✓	
硬件	单腿舵机的简单校准		✓	
可视化	动态可视化运动学计算结果			✓

背景信息和资源

本教程基于以下资源编写：
参考链接：matplotlib
参考链接：numpy
参考链接：mini pupper docs
参考链接：Python math.atan2(y,x)
版权信息：教材尚未完善，此处预留版权信息处理方式
mini pupper相关内容可访问：https://github.com/mangdangroboticsclub