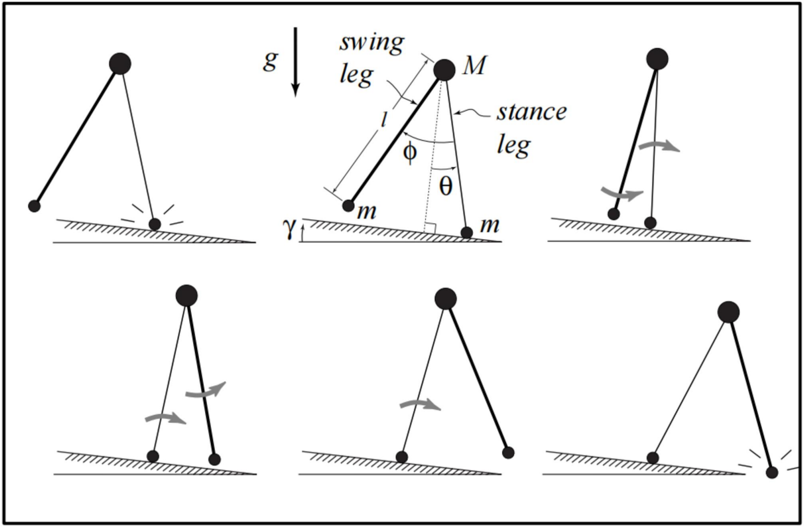
用最简平面二足模型
近似 Rigid Ramp Walker 模型的理论尝试

林海轩

复旦大学物理学系

前言

二足模型概览



为什么要研究如此“简单”的模型？

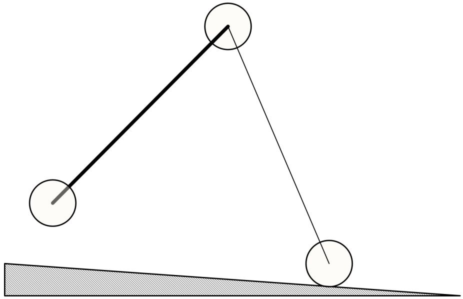
相较于完全自创的复杂模型, 往已经被前人研究过的简单模型中添加修正后, 如果能得到可以解释实验所得的结果, 则计算量更小, 过程更清晰, 结果更令人信服.

为什么要研究二足模型而不是四足模型?

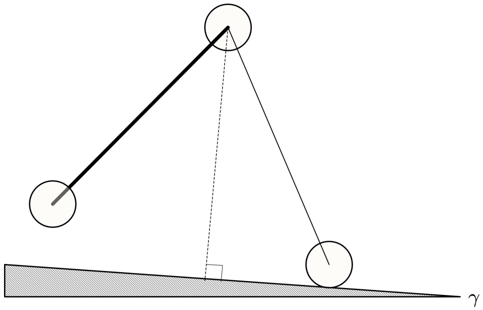
一方面, 根据阅读文献, 似乎没有找到四足模型完全理论化的方程.
另一方面, 根据我本人最初的想法, 这个二足模型通过添加阻尼项, 劲度系数项, 左右半身耦合项, 可以很大程度解释四足的情况, 抑或是把最后的结果直接与四足实验数据作差求得两模型的差异, 为进一步的研究提供方向.
同时, 二足模型的研究方法可以为四足模型的研究提供思路, 见另一篇 ppt.

假设、约定与符号

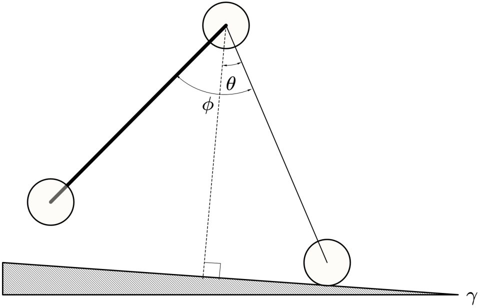
卡通视图以及一些假设与符号



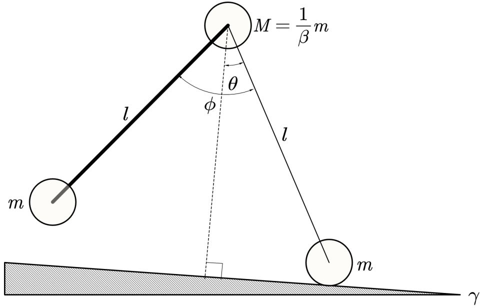
- 刚体假设:
  顶部用一个小球代表身体, 底部用两个小球代表足, 足和身体之间用相同长度的刚体杆连接
- 站立 (stance) 足与摆动 (swift) 足约定:
  用细直线代表与地面接触的足, 用加粗直线代表正在摆动的足
- 理想关节与掠滑过假设:
  关节是无阻尼无恢复系数的, 摆动足滑过地面的时候视为无摩擦



过身体做一条斜面的垂线, 设斜面倾斜角是 γ.



如图设定两个角参数



设腿长为 l, 足质量为 m, 身体质量为 M, 且

$$\beta = \frac{m}{M}$$

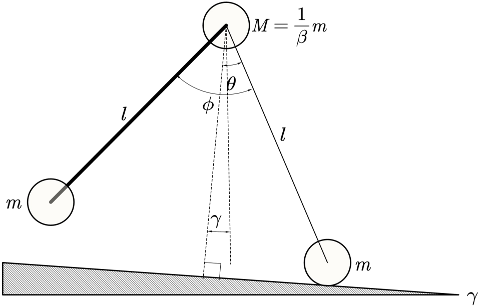
方程推导

设 $\mathcal{L} = \mathcal{L}(q, \dot{q}, t)$ 是 Lagrange 量, 其中 q 是广义坐标, 运算 $\dot{x} = \dfrac{\mathrm{d}x}{\mathrm{d}t}$ 表示对时间求一次导数, $S[q] = \int_{t_1}^{t_2} \mathcal{L} \mathrm{d}t$ 是作用量, 当作用量取得极值 ($\delta S = 0$) 时, 有

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = 0$$

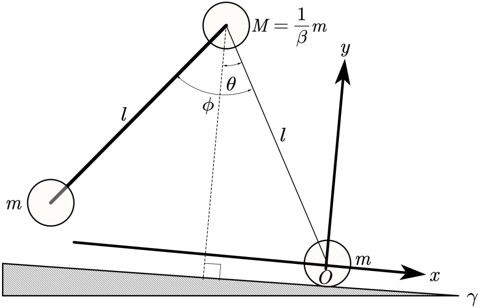
我们要考察的是连杆模型, 显然, 杆的端点的力是不好分析的, 采用此方程能更方便地描述系统.

势能



以站立足为势能零点，可以写出系统的势能

$$V = Mgl\cos(\theta - \gamma) + mgl(\cos(\theta - \gamma) - \cos(\phi - \theta + \gamma))$$



写出身体与摆动足的坐标

$$\boldsymbol{M} = \begin{pmatrix} -l\sin\theta \\ l\cos\theta \end{pmatrix} \qquad \boldsymbol{m} = \begin{pmatrix} -l\left(\sin\theta + \sin\left(\phi - \theta\right)\right) \\ l\left(\cos\theta - \cos\left(\phi - \theta\right)\right) \end{pmatrix}$$

对坐标求导取模长就得到了速率

$$v_M = l\dot{\theta} \qquad v_m = l\sqrt{\dot{\theta}^2 + \left(\dot{\phi} - \dot{\theta}\right)^2 + \cos\phi\left(\dot{\phi} - \dot{\theta}\right)}$$

得到 Lagrange 量

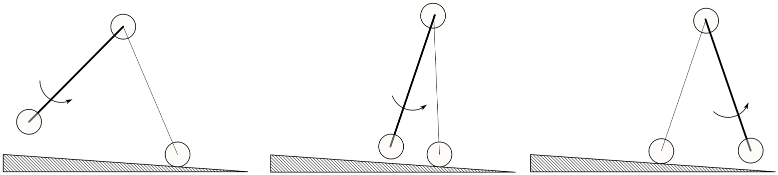
$$\mathcal{L} = T - V = \frac{1}{2}mv_m^2 + \frac{1}{2}Mv_M^2 - V$$

代入关于两个广义坐标的 Euler-Lagrange 方程

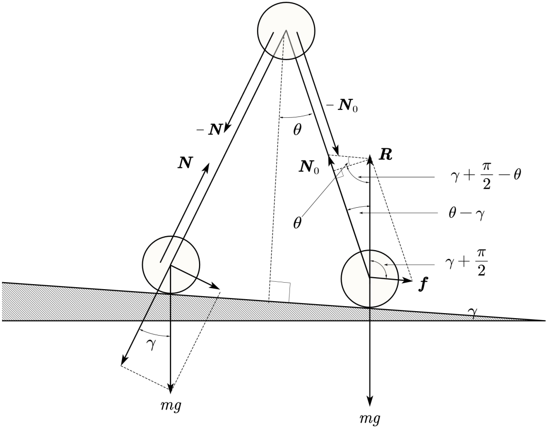
$$\begin{pmatrix} \dfrac{\mathrm{d}}{\mathrm{d}t}\dfrac{\partial}{\partial\dot{\theta}} - \dfrac{\partial}{\partial\theta} \\ \dfrac{\mathrm{d}}{\mathrm{d}t}\dfrac{\partial}{\partial\dot{\phi}} - \dfrac{\partial}{\partial\phi} \end{pmatrix}\mathcal{L} = 0$$

写开来得到

$$
\begin{pmatrix} 1+2\beta\left(1-\cos\phi\right) & -\beta\left(1-\cos\phi\right) \\ \beta\left(1-\cos\phi\right) & -\beta \end{pmatrix} \begin{pmatrix} \ddot{\theta} \\ \ddot{\phi} \end{pmatrix}
+ \begin{pmatrix} -\beta\sin\phi\left(\dot{\phi}^2 - 2\dot{\theta}\dot{\phi}\right) \\ \beta\dot{\theta}\sin\phi \end{pmatrix}
+ \begin{pmatrix} \frac{\beta g}{l}\left[\sin\left(\theta-\phi-\gamma\right) - \sin\left(\theta-\gamma\right)\right] - \frac{g}{l}\sin\left(\theta-\gamma\right) \\ \frac{\beta g}{l}\sin\left(\theta-\phi-\gamma\right) \end{pmatrix} = 0
$$



回顾一个基础的假设: 摆动足会无摩擦地滑过斜面



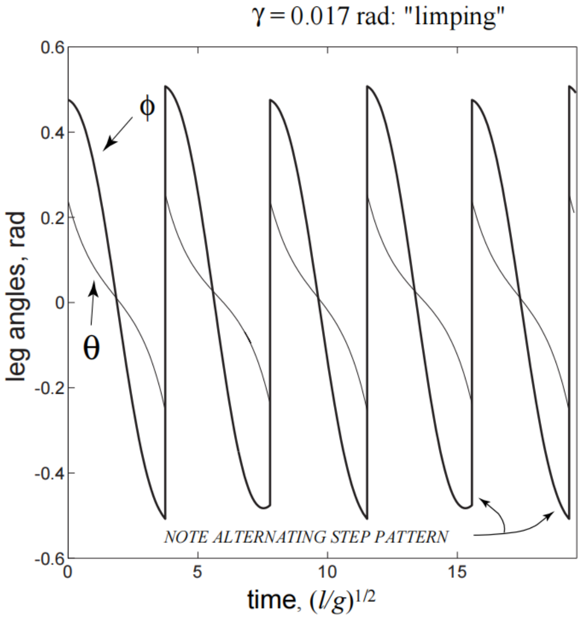
考虑到脚触地, 两个足的定义相交换, '+' 表示触地后, '-' 表示触地前, 当 $\phi-2\theta=0$ 时, θ 反号.

$$\begin{pmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{pmatrix}^{+} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & \cos 2\theta & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & \cos 2\theta\,(1-\cos 2\theta) & 0 & 0 \end{pmatrix}^{-} \begin{pmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{pmatrix}^{-}$$

不考虑定义互换

$$\begin{pmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{pmatrix}^{+} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos^2 2\theta & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & -\cos 2\theta\,(1-\cos 2\theta) & 0 & 0 \end{pmatrix}^{-} \begin{pmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{pmatrix}^{-}$$

模型的数值解图像


γ = 0.017 rad: "limping"
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
leg angles, rad
ϕ
θ
NOTE ALTERNATING STEP PATTERN
0
5
10
15
time, (l/g)^1/2

Python 代码实现

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
```

1. numpy 库提供了数学里的基本函数, 矩阵运算等
2. solve_ivp 是一个常微分方程求解器, 相比于其他求解器它有 2 个优点:
   1. solve_ivp 可以求解刚性微分方程, 即允许出现突变的地方
   2. solve_ivp 有监测事件的功能, 当沿着时间序列对微分方程积分的时候, 如果事件函数取得了零点, 则可以触发一次事件, 对相应的量进行突变调整或改变需要被积分的微分方程, 在本情形中对应摆动腿触地（地有瞬时冲量）, 即一次步态的结束, 对应的方程是两只脚定义的呼唤
3. matplotlib 库用以画图, 将求得的数值解化为图像验收

匀加速直线运动 $\dot{x} = kt$

```
k = 1
x_0 = [0]

def acceleration_equation(t, x, k):
    x_dot = k*t
    return x_dot
```

常量k

初值被规定必须都放在一个列表中

这里只有一个待积分变量所以列表里只有一个初值

规定第一个形参必须是自变量（一般是时间t）

第二个形参是待积分的变量列表

这里只有一个待积分变量位移x

返回的是一次微分后的结果

```python
t_span = (0, 10)
t_eval = np.linspace(t_span[0], t_span[1], 100)
```

积分到第10s

需要采集的数据点：10s 内等距的100个时间点

第一个形参是要积分的方程　第二个形参是时间长度　第五个形参是被采样的时间点

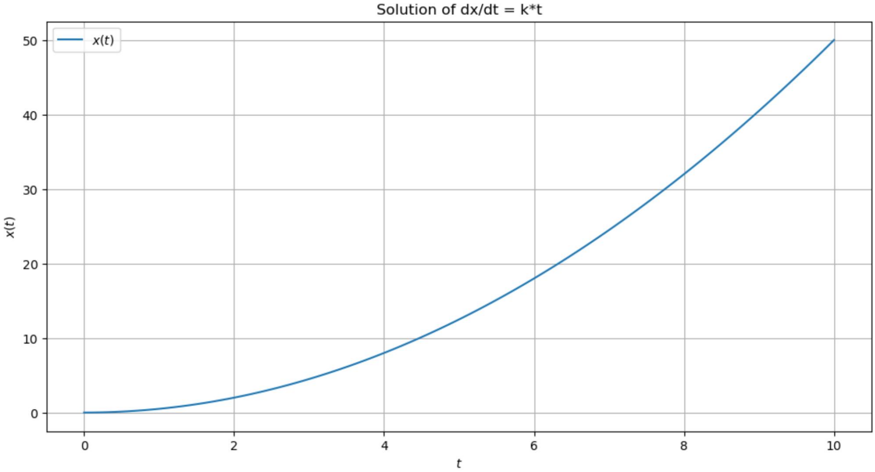
```
sol = solve_ivp(acceleration_equation, t_span, x_0, args=(k,), t_eval=t_eval)
```

第三个形参是初值　第四个形参是参数取值

```
plt.figure(figsize=(12, 6))
plt.plot(sol.t, sol.y[0], label='$x(t)$')
plt.title('Solution of dx/dt = k*t')
plt.xlabel('$t$')
plt.ylabel('$x(t)$')
plt.legend()
plt.grid(True)
plt.show()
```

纵坐标是数值解集中的第一个变量（这里只有一个变量）位移

横坐标为数值解集中的时间t


Solution of dx/dt = k*t
x(t)
50
40
30
20
10
0
0
2
4
6
8
10
t
x(t)

定义常量和初始化条件

下面通过代码来模拟二足模型

```
g = 9.81
l = 1
gamma = 0.02
beta = 0.005

theta_0 = 0.2
theta_dot_0 = 0
phi_0 = 0.4
phi_dot_0 = 0

variable_list_0 = [theta_0, theta_dot_0, phi_0, phi_dot_0]
```

g：重力加速度

l：腿长

γ：斜面倾角

$\beta = \frac{m}{M}$：腿与身体的质量比，小量

θ_0：站立足初张角

$\dot{\theta}_0$：站立足初角速度

ϕ_0：摆动足初张角

$\dot{\phi}_0$：摆动足初角速度

初始化变量打包放进列表

$$\begin{pmatrix} 1+2\beta\left(1-\cos\phi\right) & -\beta\left(1-\cos\phi\right) \\ \beta\left(1-\cos\phi\right) & -\beta \end{pmatrix}\begin{pmatrix} \ddot{\theta} \\ \ddot{\phi} \end{pmatrix} + \begin{pmatrix} -\beta\sin\phi\left(\dot{\phi}^2-2\dot{\theta}\dot{\phi}\right) \\ \beta\dot{\theta}\sin\phi \end{pmatrix} + \begin{pmatrix} \frac{\beta g}{l}\left[\sin\left(\theta-\phi-\gamma\right)-\sin\left(\theta-\gamma\right)\right]-\frac{g}{l}\sin\left(\theta-\gamma\right) \\ \frac{\beta g}{l}\sin\left(\theta-\phi-\gamma\right) \end{pmatrix} = 0$$

```python
def equations(t, variable_list):

    theta, theta_dot, phi, phi_dot = variable_list

    matrix_A = np.array([[1+2*beta*(1-np.cos(phi)), -beta*(1-np.cos(phi))],
                         [beta*(1-np.cos(phi)),     -beta                 ]])

    vector_B = np.array([-beta*np.sin(phi)*(phi_dot**2-2*theta_dot*phi_dot),
                         beta*theta_dot**2*np.sin(phi)                       ])

    vector_C = np.array([beta*g/l*(np.sin(theta-phi-gamma)-np.sin(theta-gamma)-g/l*np.sin(theta-gamma)),
                         beta*g/l*np.sin(theta-phi-gamma)])

    theta_ddot, phi_ddot = np.linalg.solve(matrix_A, -vector_B-vector_C)

    return [theta_dot, theta_ddot, phi_dot, phi_ddot]
```

变量解包 $\theta,\ \dot{\theta},\ \phi,\ \dot{\phi}$

解出 $\ddot{\theta},\ \ddot{\phi}$

以列表的形式返回一次微分后的结果

之后会对它们积分

定义事件：摆动足触地

$$\begin{pmatrix}\theta\\ \dot\theta\\ \phi\\ \dot\phi\end{pmatrix}^+ = \begin{pmatrix}-1 & 0 & 0 & 0\\ 0 & \cos 2\theta & 0 & 0\\ -2 & 0 & 0 & 0\\ 0 & \cos 2\theta\,(1-\cos 2\theta) & 0 & 0\end{pmatrix}^- \begin{pmatrix}\theta\\ \dot\theta\\ \phi\\ \dot\phi\end{pmatrix}^-$$

```
def heelstrike_event_and_change_defination(variable_list):

    theta, theta_dot, phi, phi_dot = variable_list

    matrix_Change = np.array([[-1, 0,                                     0, 0],
                              [0,  np.cos(2*theta),                        0, 0],
                              [-2,  0,                                     0, 0],
                              [0,  np.cos(2*theta)*(1-np.cos(2*theta)), 0, 0]])

    variable_matrix = np.array([[theta],
                                [theta_dot],
                                [phi],
                                [phi_dot]])

    result_matrix = np.dot(matrix_Change, variable_matrix)

    theta = result_matrix[0, 0]
    theta_dot = result_matrix[1, 0]
    phi = result_matrix[2, 0]
    phi_dot = result_matrix[3, 0]

    return [theta, theta_dot, phi, phi_dot]
```

变量解包$\theta, \dot\theta, \phi, \dot\phi$

矩阵相乘，覆盖原来的值

事件监测函数

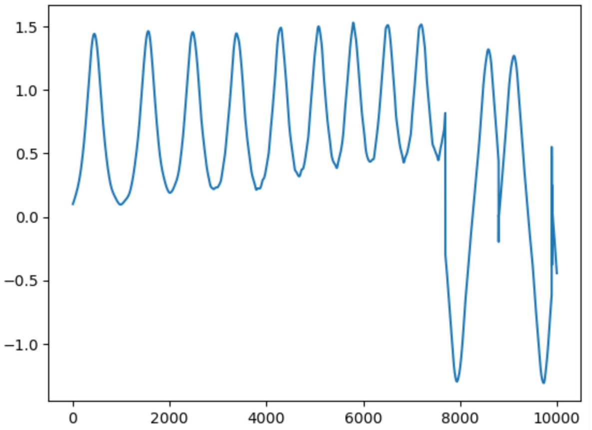
```python
def monitor(t, variable_list):
    theta, theta_dot, phi, phi_dot = variable_list
    return phi - 2*theta
```

事件监测函数, 监测 $\phi - 2\theta$ 的值, 当取得 0 的时候, 行走器与斜面构成等腰三角形, 也就是说摆动足触地了, 触发一次事件

循环主体

```python
while True:

    sol = solve_ivp(kinetic_equation_of_one_step, t_span, variable_list_0, events=monitor, dense_output=True, max_step=0.05)

    t = np.linspace(sol.t[0], sol.t[-1], num=100)
    return_variable = sol.sol(t)

    t_list = np.concatenate((t_list, t))

    theta_list = np.concatenate((theta_list, return_variable[0]))
    theta_dot_list = np.concatenate((theta_dot_list, return_variable[1]))
    phi_list = np.concatenate((phi_list, return_variable[2]))
    phi_dot_list = np.concatenate((phi_dot_list, return_variable[3]))

    current_t = t_list[-1]
    current_variable_list = [theta_list[-1], theta_dot_list[-1], phi_list[-1], phi_dot_list[-1]]

    if sol.status == 1:
        new_theta, new_theta_dot, new_phi, new_phi_dot = heelstrike_event_and_change_defination(t, current_variable_list)
        if current_t == t_span[-1]:
            break
        variable_list_0 = [new_theta, new_theta_dot, new_phi, new_phi_dot]
        t_span[0] = sol.t_events[0][0] + 1e-5
    else:
        break
```

结果



效果不是特别好，突变不明显（过于光滑），解不稳定

后续的改进思路

1. 一个直接的原因是参数设置不当，由于原论文没有详细给出模拟实验参数（初始的角度，速度），所以需要自己摸索
2. 求解器刚性处理不足，需要改进代码
3. 如何迁移这种做法到四足复杂模型中？对于这种简单模型，数值计算都无法令人满意，如何进行复杂模型的运算？

[1]Garcia, Mariano, Anindya Chatterjee, Andy Ruina, and Michael Coleman. 1998. "The Simplest Walking Model: Stability, Complexity, and Scaling." ASME Journal of Biomechanical Engineering. Accepted April 16, 1997; final version February 10, 1998.

[2] 感谢 谢昀城 学长的在代码改进方面指导

END