线性倒立摆模型 (LIP)Matlab 建模

王　芮*
2015010445

1　基本原理

Kajita 书 ***Introduction to Humanoid Robotics*** 第 130 页给出了 LIP 的基本流程。现将基本流程与原理总结一下。澄清几个书中没有说清楚，或说得不明白的条件，以及我自己的理解：

- 不变量：每走一步，花的时间是一定的，即 T_{sup}
- walk parameters s_x, s_y 的含义是前后两只支撑脚在“每一步”前后的相对位移，而非同一只脚的前后位移
- 核心问题是判断出下一步支撑脚要放在哪里。如何选择要放的地方：使得走了由这只脚支撑的这一步后，重心 CoM 尽可能地贴近规划的路线
- walk parameters 预先判断了脚的期望位置，从而帮助我们求出每一步期望的 CoM 的位置，并且保证了规划的 CoM 路线是用脚走出来的
- 之所以需要进行支撑足摆放位置修正，是因为实际中机器人没办法像规划的一样走路。还需要满足 CoM 轨迹的连续性和一阶导数连续性

首先，将书中步骤简化为模拟程序的步骤：

在循环外部，先进行初始化设定，例如初始参数的传递，初始 CoM 以及 foot place 坐标的设定。之后进入循环，遵照 walk parameter 规划步态。

对第 n 次循环：

1. 此时各变量处于第 n 步的起始状态。由机器人当前状态 $x_i, \dot{x}_i$ 绘制持续时间为 T_{sup} 的曲线，并记录到达这一步终点时的坐标 x 和速度 $\dot{x}$
2. 此时第 n 步已经走完，需要规划下一步（第 $n+1$ 步）的路线，先求出在路径规划中下一步脚的理想位置
3. 求出理想情况下，下一步结束时，CoM 相对于支撑足的坐标
4. 由上面两步求出下一步结束时质心 CoM 的理想位置
5. 求出下一步支撑足的实际摆放位置，使得下一步结束时，CoM 的坐标与理想坐标的偏差尽量小

如果需要转弯，就只需要修改质心的位置。在本书中，s_x, s_y 均为非转弯时的脚步，需要自己旋转。实际上，只要给出一条（比较合理的）质心运动曲线，就可以跳过上述 2、3 两步，仍然可以求出相应的落脚点。

2　直线行走

2.1　代码说明

源代码见附录。

源代码中用到的主要变量名称简要说明（比较冗长，没有看懂源代码可以参考此处）：

*wangrui15@mails.tsinghua.edu.cn

Table 1: Definition of Variables Used in the Code

Variable name	Symbol	Meaning
zc	z_c	CoM 所在平面在 z 轴的截距，平地上可以认为就是 CoM 的高度
Tsup	T_{sup}	机器人每一步的时间
sx, sy	s_x, s_y	用户给机器人设计好的期望步态（理想路线）
xi, yi	x_i, y_i	每一步初始的质心坐标
vxi, vyi	$\dot{x}_i, \dot{y}_i$	每一步的初始速度
px, py	p_x^*, p_y^*	实际计算得的落脚点
a, b	a, b	用户设定的参数，书公式 (4.58) 中代价函数的参数，用于调整实际落脚点中质心速度和位置这两个考虑因素所占的比重
n	n	表示第几步
g	g	重力加速度
Tc, C, S, D	T_c, C, S, D	化简公式所用的参数
px0, py0	p_x, p_y	理想的落脚点
xbar, ybar	$\bar{x}, \bar{y}$	理想情况下一步末，质心 CoM 对落脚点的相对坐标
vxbar, vybar	$\dot{\bar{x}}, \dot{\bar{y}}$	理想情况下一步末，质心相对落脚点速度（等于质心在地面坐标系中的速度）
xd, yd, vxd, vyd	x^d, y^d, v_x^d, v_y^d	理想的质心坐标及速度

2.2 代码细节：启动和终止

2.2.1 启动

书 P130 的流程框内提到‘set initial position of CoM and initial foot placement’，但细思极恐的是，这两个值在后续的图片中都没有给读者。启动的时候，假如把 CoM 和 foot place x_0, y_0 均放在一个点如 $(0, 0)$，会导致初速度永远为 0。这一点由书公式 (4.54) 和 (4.55) 可以看出。因此需要我们发挥想象力。有两种可能的方法能让机器人动起来：

- 给机器人一个初速度
- 让机器人的质心和脚的位置错开来，得到初始加速度

很显然，在丑陋的源代码中，之所以还传递了参数 vxi, vyi，就是当时为了测试第一种方法留下的遗产。但它最后被抛弃了，因为很快我们可以发现这种方法需要反复手动调参，溯洄从之，道阻且长，很难获得准确的最佳初速度。

而第二种方法则便捷高效，也是我的代码中采用的方法，可以看出是借用了后面寻找 p_x^* 的方法来寻找最佳错开位移。唯一的缺点是不符合常理。

这种方法可以认为一开始机器人有些倾斜，然后以此时它的质心作为原点。但事实上，我们不可能在每次机器人运动开始时都要推它一把，这是很没有尊严的。我们可以想象是机器人有一个极小的初速度，移动到所需要的位置时速度仍小到可以忽略，虽然有些牵强，但其实我们只是需要让

机器人动起来，所以不妨就这样说服自己。这也是代码第 24 行所做的事情。

2.2.2 终止

机器人行走的“终止”可以定义为速度为 0，但 $t = T_{sup}$ 的时候速度为 0，并不能保证最后脚也落到稳定的位置。要实现最终稳定的站立，还要求终止时的落脚点应该在支撑脚，也就是最后一步的 foot place 附近。考虑到实际中机器人的脚是有大小的，只需要速度降为很小，同时 CoM 和支持点相距不太远即可实现机器人的停驻。

如图1，如果按照给出的 s_x, s_y，模拟后我们只能得到不完整的步态曲线，因为最后一步的质心目的地是位于中间而不是回到落脚点上方。因此，我们可以为 s_x, s_y 各添加一列，而这添加的一步机器人实际是不需要走动的，因此均为 0，在代码中 15 行可以看见这一改动。

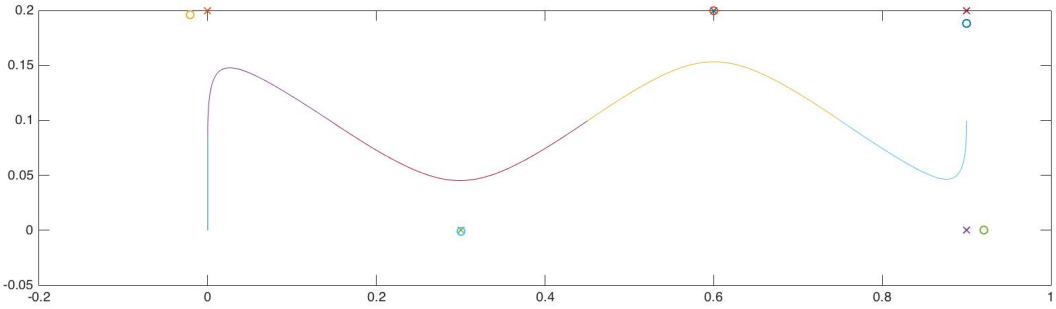

Figure 1: Straight Walking without Coming to a Stop

为此我们可以修改程序，把最后的 a,b 调整为 1,1（或者其他合理的比例，使得最后的 $(x^d - x_f^{(n)})^2, (\dot{x}^d - \dot{x}_f^{(n)})^2$ 都不太大）。在源程序第 51 行后添加：

```
1        if n == size(sx, 2) - 1 % if time has come to calculate the last step
2          a = 1;
3          b = 1;
4          D = a * (C - 1)^2 + b * (S / Tc)^2;
5        end
```

图2显示了这样改动过后的模拟曲线，可以看见，最后质心成功回到了落脚点。

当然，也可以根据实际需求任意修改这一行的代码中的 a,b. 例如，如果令 $a = 0, b > 0$, 则相当于在 cost function 中让速度尽可能接近 0，而实际上速度确实可以降为 0。

这样做的好处是不需要为机器人的“终止”行为进行单独的配置，终止的这一步和中间走路的每一步都是等效的。

实践中，最终是否能够实现稳定到达站立地点？在 3.3 节中我另外进行了一次实验验证，令人欣慰的是，得出了肯定的结论。

2.3 源代码

实现直线步行的完整代码如下
为了保护读者视力，还是放在目录中吧:)

2.4 运行样例

2.4.1 样例 1：直线行走

在 MatLab 的 command window 输入如下命令：

```
>> sx = [0.0, 0.3, 0.3, 0.3, 0];
>> sy = [0.2, 0.2, 0.2, 0.2, 0.2];
>> LIP(0.8, 0.8, sx, sy, 0, 0, 0, 0, 0, 0, 10, 1);
```

会得到图2所示的图像。其中，圆圈表示实际落脚点，叉表示理想落脚点。

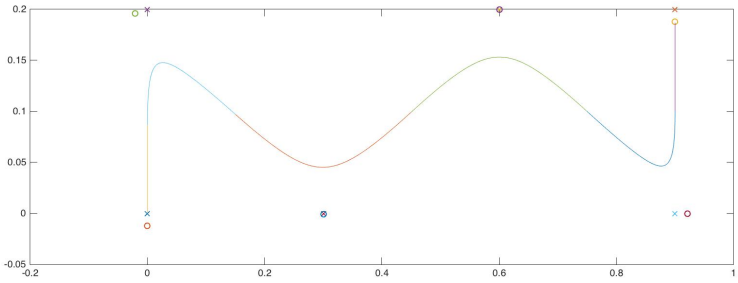

Figure 2: 直线行走

2.4.2 样例 2：走对角线

说明：侧行 (side walk, diagonal walk) 的含义是运动方向与机器人朝向有一个角度，即通过非转向 (turn) 的方式实现整体运动轨迹的倾斜。

在 MatLab 的 command window 输入如下命令：

```
>> sx = [0.0, 0.3, 0.3, 0.3, 0];
>> sy = [0.2, 0.3, 0.1, 0.3, 0.2];
>> LIP(0.8, 0.8, sx, sy, 0, 0, 0, 0, 0, 0, 10, 1);
```

可以得到图3所示的结果。

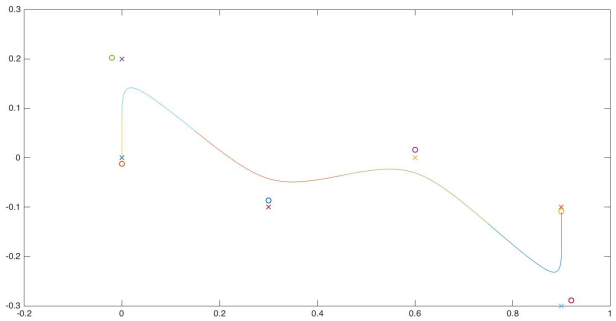

Figure 3: 侧行

3 行走中转向

3.1 注意点

行走中转向和第一问区别不大。只需要在计算理想质心坐标时进行旋转操作。

在做转向问题时，需要注意几点：

- 在第一步和最后一步，是没有转向的，这两步都垂直于机器人面向的方向
- Kajita 书 P133 公式 (4.62) 是错误的，即速度不需要旋转变换。因为参考系没有改变，始终是地面参考系，要保证速度的连续性并不需要改变速度，这是又一大坑。

3.2 源代码

这一篇代码和直线行走相差不大，是在直线行走的代码上直接加上旋转矩阵修改完成的。为继续保护读者视力，将源代码放在了附录中。

3.3 代码说明

鉴于在直线行走中已经详细讨论过启动和终止的情形，在这里就直接给出最优方案。进行实验发现，采用上文提到的启动和终止方案，仍使用原来的数据输入，（书 P143 转弯部分的），发现最终速度达到 $v_x = 0, v_y = 0.0044(4mm/s)$ 也就是说最终速度是非常接近于 0 的，而且由后面的模拟结果可以看出，最后质心也很靠近支持点，这证明之前的终止方法是可行的。

3.4 运动样例

3.4.1 样例 1：书上案例

在 MatLab 的 command window 输入如下命令：

```
>> sx = [0.0, 0.25, 0.25, 0.25, 0];
>> sy = [0.2, 0.3, 0.2, 0.2, 0.2];
>> theta = [0 20 40 60 60]
>> LIP(0.8, 0.8, sx, sy, 0, 0, 0, 0, 0, 0, 10, 1, theta);
```

可以得到图4所示的结果。注意，在这类问题中，最后为了使得机器人能够回到驻足点，最后一

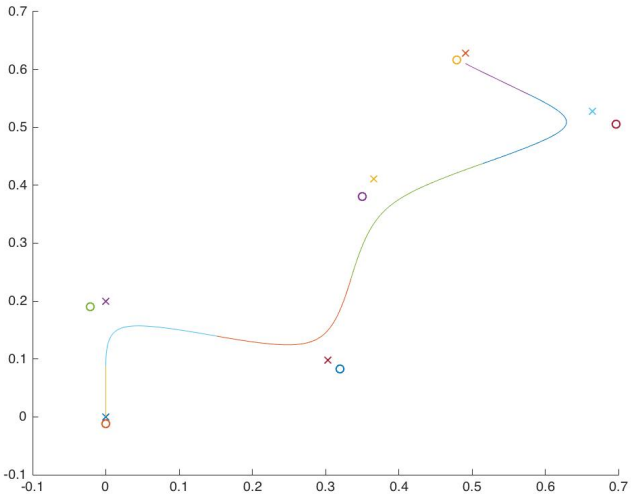

Figure 4: 每走一步转 20 度

次转向的角度设置为与先前相同。

3.4.2 样例 2：circle

在 Matlab 的 command window 输入如下命令：

```
>> sx = linspace(0.3,0.3,18);
>> sx = [0,sx,0];
>> sy=linspace(0.2,0.2,20);
>> theta = 0:20:360;
>> theta = [theta, 360];
```

至此，构建了能够实现圆周运动的基本参数。
再输入：

```
>> LIP(0.8, 0.8, sx, sy, 0, 0, 0, 0, 0, 0, 10, 1, theta);
```

即可实现图5。可以看出，机器人走的圆还是非常圆的。

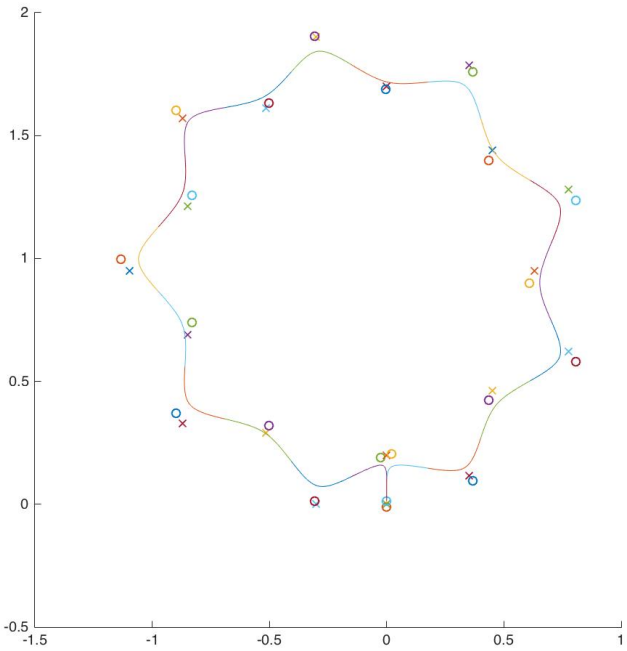

Figure 5: Circling with 20 degrees' turn per step

4 后续讨论

修改输入后，评估一下发现，机器人走得是否理想，对不同转向角度而言这个模型还是比较鲁棒的，但对速度变化比较敏感。

书中给出的几个参数即腿高 0.8，走路每步 $0.8s$ 设计得比较好，机器人按照这个参数走得比较好。但是，如果速度快一点，例如把 T_{sup} 降到 0.3，机器人走得就比较夸张了，在现实中腿是不太可能伸到那么长的。如果把速度放慢，可以发现，越慢和设定路线贴的越接近，甚至完全成为折线

（比如 $T_{sup} = 2s$ 时）。同时，频率如果想要上去，就必须把高度降下来，小的机器人可以走的频率很高。但机器人小了，腿也伸不了设定得那么长了，所以其实也不太可行。总之（就这个单一的模型而言）LIP 注定速度快不起来，不知是否有更高级的模型。

此算法由于简化了问题情形，所以仍有一些问题：

- 实际问题中，启动时需要考虑如何运动？我猜想是会靠自身电机产生一个初速度。不过不清楚大小如何设定。
- 实际中要把机器人等效成一个质点是有点困难的。
- 要把机器人的质心高度固定怎么做到的，比较困惑。

附录:
直线行走的代码:(函数,需放在脚本文件中)

```
function LIP(zc, Tsup, sx, sy, xi, yi, vxi, vyi, px, py, a, b)
% zc, Tsup: z-axis intersection; support time
% sx, sy: walk parameters(vectors)
% xi, yi: initial CoM coordinates
% px, py: initial foot placement
% vxi, vyi: initial CoM velocity
% a, b: params of costfunc
%% Initialization
n = 0;
g = 9.8;

sx = [sx, 0];
sy = [sy, 0]; % the step after the last step should be zero

px0 = px;
py0 = py; % desired foot placement

Tc = sqrt(zc / g);
C = cosh(Tsup / Tc);
S = sinh(Tsup / Tc);
D = a * (C - 1)^2 + b * (S / Tc)^2;

hold on
cal_foot_place(0); % set the initial placement of foot
%% carry out the steps indicated by walk parameters sx, sy
while n < size(sx, 2) % sx was expanded by one element previously
    %% plot the n-th step trajectory
    t = 0;
    dt = Tsup / 100;
    xt = xi; vxt = vxi; yt = yi; vyt = vyi;

    for i = 0 : 100
        xt = [xt, (xi - px) * cosh(t / Tc) +...
             Tc * vxi * sinh(t / Tc) + px];
        vxt = [vxt, (xi - px) / Tc * sinh(t / Tc) +...
              vxi * cosh(t / Tc)];
        yt = [yt, (yi - py) * cosh(t / Tc) +...
             Tc * vyi * sinh(t / Tc) + py];
        vyt = [vyt, (yi - py) / Tc * sinh(t / Tc) +...
              vyi * cosh(t / Tc)];
        t = t + dt;
    end
    plot(xt, yt);
    hold on
    %% update xi, yi, vxi, vyi for the next step
    vxi = vxt(end); vyi = vyt(end);
    xi = xt(end); yi = yt(end);
    %% update n, the order number of the step
```

```
    n = n + 1;

    if n < size(sx, 2)
        cal_foot_place(n); % calculate the actual foot place for
           step n
    end
end
%% calculate the foot place
function cal_foot_place(n) % nested function
    %% calculate the desired foot place during the n-th step
    if n ~= 0
        px0 = px0 + sx(n);
        py0 = py0 - (-1)^n * sy(n);
    end
    %% calculate the coordinate (xbar, ybar)
    xbar = sx(n + 1) / 2;
    ybar = (-1)^n * sy(n + 1) / 2;
    vxbar = (C + 1)/(Tc * S) * xbar;
    vybar = (C - 1)/(Tc * S) * ybar;
    %% target state of CoM, of the n-th step
    xd = px0 + xbar;
    yd = py0 + ybar;
    vxd = vxbar;
    vyd = vybar;
    %% update px, py to be the real foot place in step n
    % a,b are parameters for cost func
    px = - a * (C - 1) / D * (xd - C * xi - Tc * S * vxi)...
        - b * S / (Tc * D) * (vxd - S / Tc * xi - C * vxi);
    py = - a * (C - 1) / D * (yd - C * yi - Tc * S * vyi)...
        - b * S / (Tc * D) * (vyd - S / Tc * yi - C * vyi);
    plot(px0, py0, 'x');
    plot(px, py, 'o');
end
end
```

行进中转向的代码:（函数，需放在脚本文件中）

```
function LIP(zc, Tsup, sx, sy, xi, yi, vxi, vyi, px, py, a, b, theta
    ) % a, b: params of costfunc
% zc, Tsup, a, b
% sx, sy are walk parameters, vectors
% xi, yi: initial CoM coordinate
% px, py: initial foot placement
% vxi, vyi: initial CoM velocity
% theta: changing *degrees* (not rad!) per step
%% Initialization
n = 0;
g = 9.8;

sx = [sx, 0];
sy = [sy, 0]; % the step after the last step should be zero
theta = [theta, 0]; % theta(end) can be any value
```

```
15
16  px0 = px;
17  py0 = py; % desired foot placement
18
19  Tc = sqrt(zc / g);
20  C = cosh(Tsup / Tc);
21  S = sinh(Tsup / Tc);
22  D = a * (C - 1)^2 + b * (S / Tc)^2;
23
24  st = sind(theta);
25  ct = cosd(theta);
26
27  function R = Rotate(i)
28      if i == 0 R = eye(2);
29      else R = [ct(i), -st(i); st(i), ct(i)]; % rotation matrix, each
            time rotates theta degrees
30      end
31  end
32
33  hold on
34  cal_foot_place(0);
35  %% generating CoM trajectory
36  while n < size(sx, 2)  % sx was expanded by one element previously
37      %% plot the n-th step trajectory
38      t = 0;
39      dt = Tsup / 100;
40      xt = xi; vxt = vxi; yt = yi; vyt = vyi;
41
42      for i = 0 : 100
43          xt = [xt, (xi - px) * cosh(t / Tc) +...
44                Tc * vxi * sinh(t / Tc) + px];
45          vxt = [vxt, (xi - px) / Tc * sinh(t / Tc) +...
46                vxi * cosh(t / Tc)];
47          yt = [yt, (yi - py) * cosh(t / Tc) +...
48               Tc * vyi * sinh(t / Tc) + py];
49          vyt = [vyt, (yi - py) / Tc * sinh(t / Tc) +...
50                vyi * cosh(t / Tc)];
51          t = t + dt;
52      end
53      plot(xt, yt);
54      hold on
55      %% update xi, yi, vxi, vyi for the next step
56      vxi = vxt(end); vyi = vyt(end);
57      xi = xt(end); yi = yt(end);
58      %% update n, the order number of the step
59      n = n + 1;
60
61      if n < size(sx, 2)
62          cal_foot_place(n); % calculate the actual foot place for
                step n
63      end
```

```
end
%% calculate foot position during the next step
function cal_foot_place(n) % nested function
    %% calculate the desired foot place during the n-th step
    if n ~= 0 % note: rotating
        pxy0 = [px0; py0];
        sxy= [sx(n); -(-1)^n * sy(n)];
        pxy0 = pxy0 + Rotate(n) * sxy;
        px0 = pxy0(1);
        py0 = pxy0(2);
    end
    %% calculate the coordinate (xbar, ybar)
    xybar = [sx(n + 1) / 2; (-1)^n * sy(n + 1) / 2];
    xybar = Rotate(n + 1) * xybar;
    xbar = xybar(1);
    ybar = xybar(2);
    vxybar = [(C + 1)/(Tc * S) * xbar; (C - 1)/(Tc * S) * ybar];
    vxybar = Rotate(0) * vxybar;
    vxbar = vxybar(1);
    vybar = vxybar(2);
    %% target state of CoM, of the n-th step
    xd = px0 + xbar;
    yd = py0 + ybar;
    vxd = vxbar;
    vyd = vybar;
    %% update px, py to be the real foot place in step n
    %a,b are parameters for cost func
    px = - a * (C - 1) / D * (xd - C * xi - Tc * S * vxi)...
         - b * S / (Tc * D) * (vxd - S / Tc * xi - C * vxi);
    py = - a * (C - 1) / D * (yd - C * yi - Tc * S * vyi)...
         - b * S / (Tc * D) * (vyd - S / Tc * yi - C * vyi);
    plot(px0, py0, 'x');
    plot(px, py, 'o');
end
end
```