

ROS机器人学习——麦克纳姆轮运动学解算

摘要

本文详细讲解了麦克纳姆轮的工作原理、安装方式，并提供了逆运动学模型的公式，展示了如何通过底盘运动解算轮子速度，包括正反运动学模型的代码实现。适合理解机器人运动学原理的读者阅读。

摘要生成于 [C知道](#)，由 DeepSeek-R1 满血版支持，[前往体验](#) >

麦克纳姆轮运动学解算

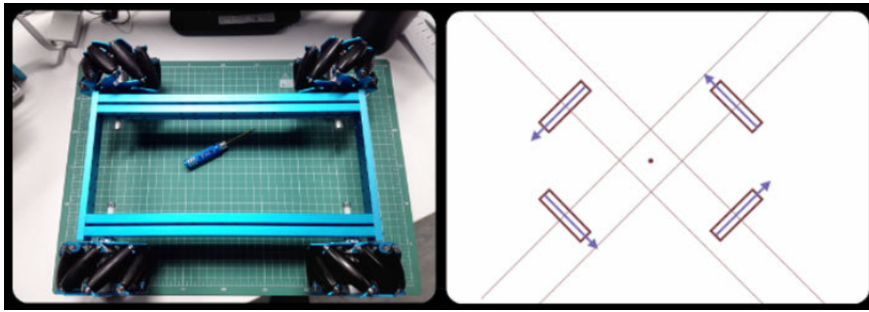
一、麦克纳姆轮介绍

了解过Robomaster的同学都知道，RM战车所用的轮子均为麦克纳姆轮，这种轮子安装方式与普通轮子无异，可安装于平行轴上，但是麦克纳姆轮可以实现全向移动，即**前后运动、水平移动**为以上优点，许多工业上的全向移动平台都会应用这种轮子。缺点也有，就是不耐磨，需要定期更换。

麦克纳姆轮由两部分组成：**轮毂和辊子**，轮毂为轮子的主体，辊子为轮毂周围的类似椭球体的小轮子，轮毂和辊子都有自己的轴，且轮毂轴与辊子轴夹角为45°（可以为其他角度但45°角最



麦轮的安装方式也有讲究，虽然都是同轴安装，但与普通轮子不同，麦轮分为左旋和右旋两种，在一个四轮底盘上需要用两个左旋和两个右旋。安装方式为O型，如图所示：



左图为安装后你看到的样子，右图为四个轮子与地面接触的轍子围成的形状，也就是“O形”

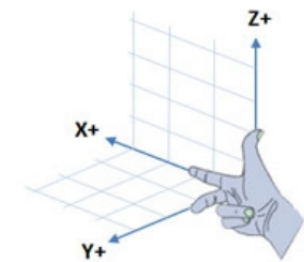
这里的O形指的是与地面接触的轍子围成的形状噢，不要再问为什么左图看起来是个X了

二、麦克纳姆轮运动学模型

1. 基础知识

1.1 坐标系

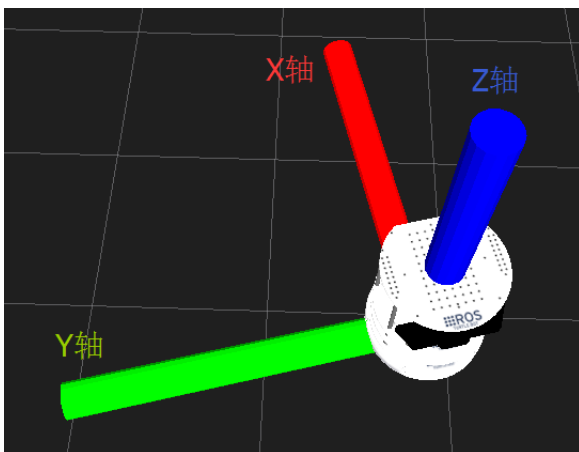
在ROS机器人 中，坐标系使用右手定义



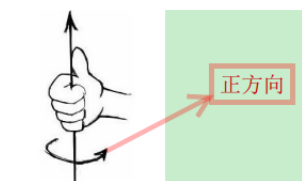
对于ROS机器人，如果以它为坐标系的原点，那么

- x轴：前方
- y轴：左方
- z轴：上方

如图所示：



除此之外，对于旋转运动，也使用右手定义：



根据右手定义，围绕 z轴正旋转 是 逆时针旋转

1.2 测量单位

ROS使用公制：

- 线速度: m/s
- 角速度: rad/s

1.3 轮子序号定义

左前1 右前2

左后3 右后4

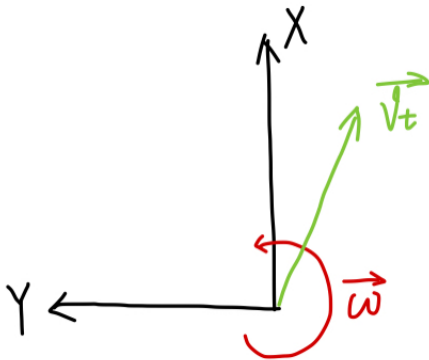
2. 逆运动学解析

逆运动学模型 (inverse kinematic model) 得到的公式可以根据底盘的运动状态解算出四个轮子的速度。

2.1 底盘运动的分解

刚体在平面内的运动可以分解为三个独立分量: X轴平动、Y轴平动、yaw 轴自转。底盘的运动也可以分解为三个量:

如下图所示:

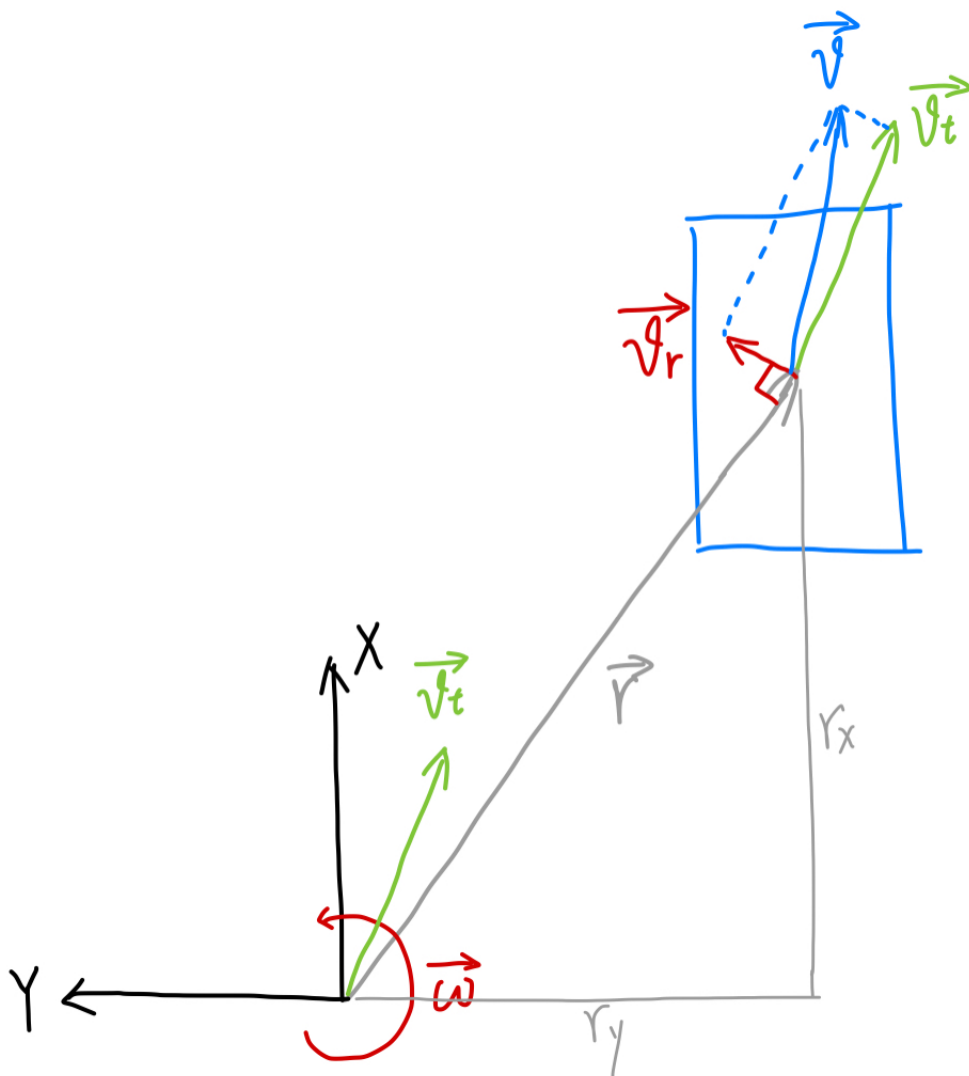


- v_{tx} 表示 X 轴运动的速度, 即前后方向, 定义向前为正;
- v_{ty} 表示 Y 轴运动的速度, 即左右方向, 定义向左为正;
- $\vec{\omega}$ 表示 yaw 轴自转的角速度, 定义逆时针为正。

2.2 计算轮子轴心位置的速度

如下图所示, 以右前轮为例, 蓝色的方框代表轮子, 定义以下变量:

- $\vec{r}$ 为从底盘中心指向轮子轴心的矢量;
- $\vec{v}$ 为轮子轴心的速度矢量;
- $\vec{v}_t$ 为轮子轴心沿垂直于 $\vec{r}$ 的方向 (即切线方向) 的速度分量;



可以计算出：

KaTeX parse error: No such environment: align* at position 8: \begin{align*} \overrightarrow{v} = \overrightarrow{v_t} + \overrightarrow{v_r} \end{align*}

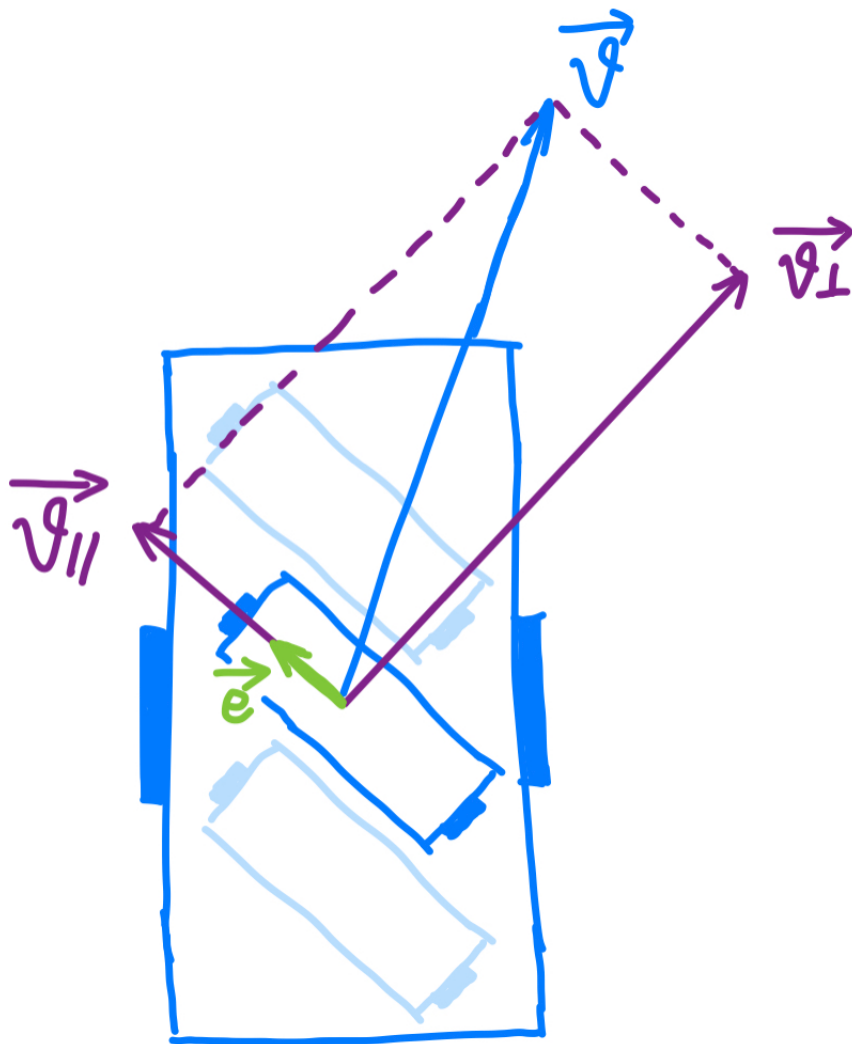
将 $\vec{r}$ 分解为 r_x 和 r_y ，分别计算轮子轴心在X、Y轴的速度分量：

$$\begin{cases} v_x = v_{tx} + \omega \cdot r_y \\ v_y = v_{ty} + \omega \cdot r_x \end{cases}$$

其他三个轮子同理

2.3 计算与地面接触的辊子速度

由2.2算得的轮子轴心速度，可以分解为沿辊子轴方向的 $\vec{v}_{\parallel}$ 和垂直辊子轴方向的 $\vec{v}_{\perp}$ ，如图所示



其中 $\vec{v}_\perp$ 用于让辊子空转，可以忽略

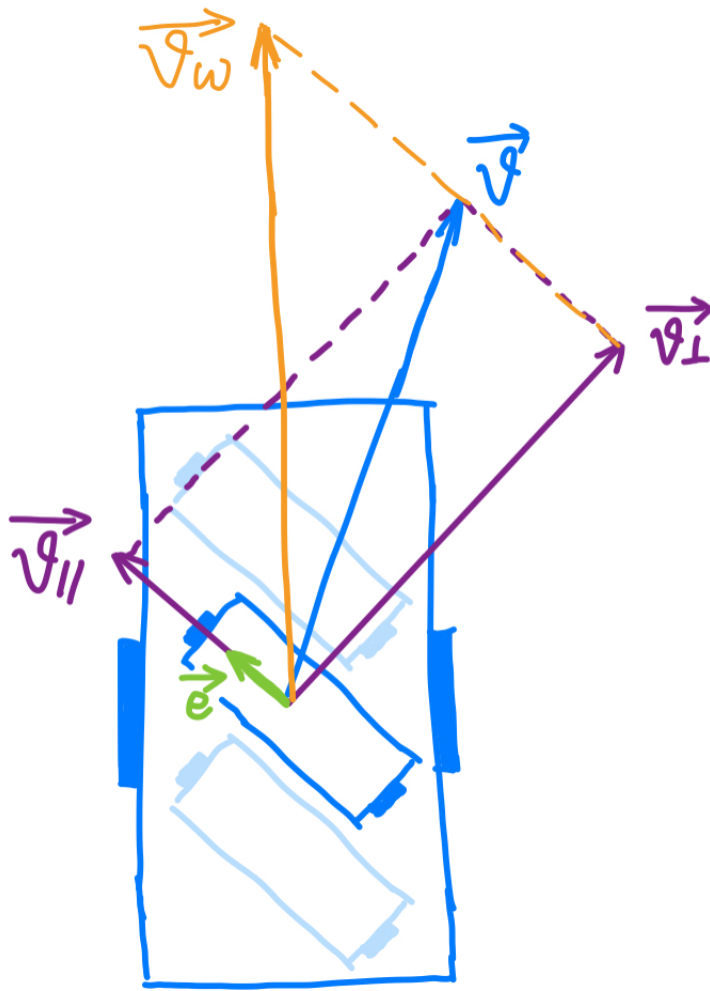
定义一个沿辊子方向的单位矢量 $\hat{e}$ ，对于右前轮来说， $\hat{e} = \frac{1}{\sqrt{2}} \cdot \hat{i} + \frac{1}{\sqrt{2}} \cdot \hat{j}$

则沿轴线的速度为 $\vec{v}$ 在 $\hat{e}$ 方向的投影：

KaTeX parse error: No such environment: align* at position 8: \begin{align*}\overrightarrow{v}

2.4 计算轮子的转速（和地面接触点的线速度）

如图所示，轮子转速为 v_w



由于辊子与轮轴呈45°角，则 v_w 可求得：

KaTeX parse error: No such environment: align* at position 8: \begin{align*} v_w &= \frac{v_{...}

将2.2求出的 $\begin{cases} v_x = v_{tx} + \omega \cdot r_y \\ v_y = v_{ty} + \omega \cdot r_x \end{cases}$ 带入上式，可求出此轮的转速：

$$v_w = v_{tx} + v_{ty} + \omega(r_x + r_y)$$

结合以上四个步骤，可以根据底盘运动状态解算出四个轮子的转速：

$$\begin{cases} v_{w1} = v_{tx} - v_{ty} - \omega(r_x + r_y) \\ v_{w2} = v_{tx} + v_{ty} + \omega(r_x + r_y) \\ v_{w3} = v_{tx} + v_{ty} - \omega(r_x + r_y) \\ v_{w4} = v_{tx} - v_{ty} + \omega(r_x + r_y) \end{cases}$$

以上方程组就是O形麦轮底盘的逆运动学模型。

2.5 代码实现

```
c
1 //参数宏定义
2 #define ENCODER_RESOLUTION 1440.0 //编码器分辨率，轮子转一圈，编码器产生的脉冲数
3 #define WHEEL_DIAMETER 0.058 //轮子直径,单位：米
4 #define D_X 0.18 //底盘Y轴上两轮中心的间距
5 #define D_Y 0.25 //底盘X轴上两轮中心的间距
6 #define PID_RATE 50 //PID调节PWM值的频率
7
8 double pulse_per_meter = 0;
9 float rx_plus_ry_cali = 0.3;
10 double angular_correction_factor = 1.0;
11 double linear_correction_factor = 1.0;
12 double angular_correction_factor = 1.0;
13
14 /**
15  * @函数作用：运动学解析参数初始化
16  */
17 void Kinematics_Init(void)
```

```

18 {
19     // 轮子转动一圈，移动的距离为轮子的周长WHEEL_DIAMETER*3.1415926，编码器产生的脉冲信号为ENCODER_RESOLUTION。则电机编码器转一圈产生的脉冲信号除以轮子周长可得轮子前进1m的距离，
20     pulse_per_meter = (float)(ENCODER_RESOLUTION/(WHEEL_DIAMETER*3.1415926))/linear_correction_factor;
21
22     float r_x = D_X/2;
23     float r_y = D_Y/2;
24     rx_plus_ry_cal = (r_x + r_y)/angular_correction_factor;
25 }
26
27 /**
28  * @函数作用: 逆向运动学解析，底盘三轴速度-->轮子速度
29  * @输入: 机器人三轴速度 m/s
30  * @输出: 电机应达到的目标速度（一个PID控制周期内，电机编码器计数值的变化）
31  */
32 void Kinematics_Inverse(int16_t* input, int16_t* output)
33 {
34     float v_tx = (float)input[0];
35     float v_ty = (float)input[1];
36     float omega = (float)input[2];
37     static float v_w[4] = {0};
38
39     v_w[0] = v_tx - v_ty - (r_x + r_y)*omega;
40     v_w[1] = v_tx + v_ty + (r_x + r_y)*omega;
41     v_w[2] = v_tx + v_ty - (r_x + r_y)*omega;
42     v_w[3] = v_tx - v_ty + (r_x + r_y)*omega;
43
44     // 计算一个PID控制周期内，电机编码器计数值的变化
45     output[0] = (int16_t)(v_w[0] * pulse_per_meter/PID_RATE);
46     output[1] = (int16_t)(v_w[1] * pulse_per_meter/PID_RATE);
47     output[2] = (int16_t)(v_w[2] * pulse_per_meter/PID_RATE);
48     output[3] = (int16_t)(v_w[3] * pulse_per_meter/PID_RATE);
49 }

```

收起 ^

3. 正运动学解析

3.1 正运动学模型

正运动学模型（forward kinematic model）让我们可以通过四个轮子的速度，计算出底盘的运动状态。可以直接根据逆运动学模型中的三个方程解出来，比如：

$$\begin{cases} v_{tx} = \frac{v_4 + v_3}{2} \\ v_{ty} = \frac{v_3 - v_1}{2} \\ \omega = \frac{v_2 - v_3}{2(r_x + r_y)} \end{cases}$$

转换为底盘坐标系下对时间求积分即为里程计变化量

3.2 代码实现

c

AI写代码

```

1 //参数宏定义
2 #define ENCODER_MAX 32767
3 #define ENCODER_MIN -32768
4 #define ENCODER_LOW_WRAP ((ENCODER_MAX - ENCODER_MIN)*0.3+ENCODER_MIN)
5 #define ENCODER_HIGH_WRAP ((ENCODER_MAX - ENCODER_MIN)*0.7+ENCODER_MIN)
6 #define PI 3.1415926
7
8 //变量定义
9 int32_t wheel_turns[4] = {0};
10 int32_t encoder_sum_current[4] = {0};
11
12 /**
13  * @函数功能: 正向运动学解析，轮子编码值->底盘三轴里程计坐标
14  * @输入: 编码器累加值
15  * @输出: 三轴里程计 x y yaw
16  */
17 void Kinematics_Forward(int16_t* input, int16_t* output)
18 {
19     static double dv_w_times_dt[4]; // 轮子瞬时变化量dxw=dvw*dt
20     static double dv_t_times_dt[3]; // 底盘瞬时变化量dxt=dvt*dt
21     static int16_t encoder_sum[4];
22
23     // 将左面轮子编码器累加值乘以-1，以计算前进的距离
24     encoder_sum[0] = -input[0];
25     encoder_sum[1] = input[1];
26     encoder_sum[2] = -input[2];
27     encoder_sum[3] = input[3];
28
29     // 编码器计数溢出处理
30     for(int i=0;i<4;i++)
31     {
32         if(encoder_sum[i] < ENCODER_LOW_WRAP && encoder_sum_current[i] > ENCODER_HIGH_WRAP)
33             wheel_turns[i]++;
34         else if(encoder_sum[i] > ENCODER_HIGH_WRAP && encoder_sum_current[i] < ENCODER_LOW_WRAP)
35             wheel_turns[i]--;
36         else
37             wheel_turns[i]=0;
38     }
39 }

```

```

38 }
39
40 //将编码器数值转化为前进的距离，单位m
41 for(int i=0;i<4;i++)
42 {
43     dv_w_times_dt[i] = 1.0*(encoder_sum[i] + wheel_turns[i]*(ENCODER_MAX-ENCODER_MIN)-encoder_sum_current[i])/pulse_per_meter;
44     encoder_sum_current[i] = encoder_sum[i];
45 }
46
47 //要计算坐标所以变回来
48 dv_w_times_dt[0] = -dv_w_times_dt[0];
49 dv_w_times_dt[1] = dv_w_times_dt[1];
50 dv_w_times_dt[2] = -dv_w_times_dt[2];
51 dv_w_times_dt[3] = dv_w_times_dt[3];
52
53 //计算底盘坐标系(base_Link)下x轴、y轴变化距离m与Yaw轴朝向变化rad 一段时间内的变化量
54 dv_t_times_dt[0] = ( dv_w_times_dt[3] + dv_w_times_dt[2])/2.0;
55 dv_t_times_dt[1] = ( dv_w_times_dt[2] - dv_w_times_dt[0])/2.0;
56 dv_t_times_dt[2] = ( dv_w_times_dt[1] - dv_w_times_dt[2])/(2*wheel_track_cali);
57
58 //积分计算里程计坐标系(odom_frame)下的机器人X,Y,Yaw轴坐标
59 //dx = ( vx*cos(theta) - vy*sin(theta) )*dt
60 //dy = ( vx*sin(theta) + vy*cos(theta) )*dt
61 output[0] += (int16_t)(cos((double)output[2])*dv_t_times_dt[0] - sin((double)output[2])*dv_t_times_dt[1]);
62 output[1] += (int16_t)(sin((double)output[2])*dv_t_times_dt[0] + cos((double)output[2])*dv_t_times_dt[1]);
63 output[2] += (int16_t)(dv_t_times_dt[2]*1000);
64
65 //Yaw轴坐标变化范围控制-2π -> 2π
66 if(output[2] > PI)
67     output[2] -= 2*PI;
68 else if(output[2] < -PI)
69     output[2] += 2*PI;
70
71 //发送机器人X轴y轴Yaw轴瞬时变化量，在ROS端除以时间
72 output[3] = (int16_t)(dv_t_times_dt[0]);
73 output[4] = (int16_t)(dv_t_times_dt[1]);
74 output[5] = (int16_t)(dv_t_times_dt[2]);
75 }

```

收起 ^

参考文献:

【1】 <https://zhuanlan.zhihu.com/p/20282234>

【2】 <https://blog.csdn.net/shixiaolu63/article/details/78496457>