A1中的roll角计算和控制

前言

A1的代码当中，以下两部分代码很关键但也比较让人费解：

Attitude_Estimation.cpp

```
void WB6_Base::roll_estimate(void)
{
    float cos = cosf(leg_angle),
          sin = sinf(leg_angle);
            //
    float t0 = 2 * (attitude.q.w * attitude.q.y + attitude.q.x * attitude.q.z),//1/2s{\theta}+sin
          t1 = 1 - 2 * (attitude.q.y * attitude.q.y + attitude.q.z * attitude.q.z),
          t2 = 2 * (attitude.q.w * attitude.q.x - attitude.q.y * attitude.q.z),
          t3 = 2 * (attitude.q.w * attitude.q.z + attitude.q.x * attitude.q.y);
    //sin = 1,cos = 0    t1t2 + t0t3          cos = 1,sin = 0   t0t3 + t1t2 = tan(r)*t4
    this->roll = -(sin * t0 - cos * t1) * (cos * t2 - sin * t3) + (cos * t0 + sin * t1) * (sin * t2 + cos * t3);
    //t4 = 2x^2 + 2y^2  - 1     (2wy-xz)
    float t4 = (sin * 2 * (attitude.q.w * attitude.q.y - attitude.q.x * attitude.q.z) - cos * (1 - 2 * (attitude.q.x * attitude.q.x + attitude.q.y * attitude.q.y)));

    roll_err = (-roll - set_roll * t4) / sqrtf(set_roll * set_roll + 1); //sine value of err angle, linearize to err angle
    ground_tilt = atanf(-roll / t4) - atanf((legL.len - legR.len) / param.core.WHEEL_DISTANCE);

    float d_angle = cos * attitude.rotation[0] - sin * attitude.rotation[2];
    d_roll_err = set_d_roll - d_angle;

    //force arm estimation
    float tan_theta, cos_theta;
    if (this->offground)
    {
        tan_theta = 0;
        cos_theta = 1;
    }
```

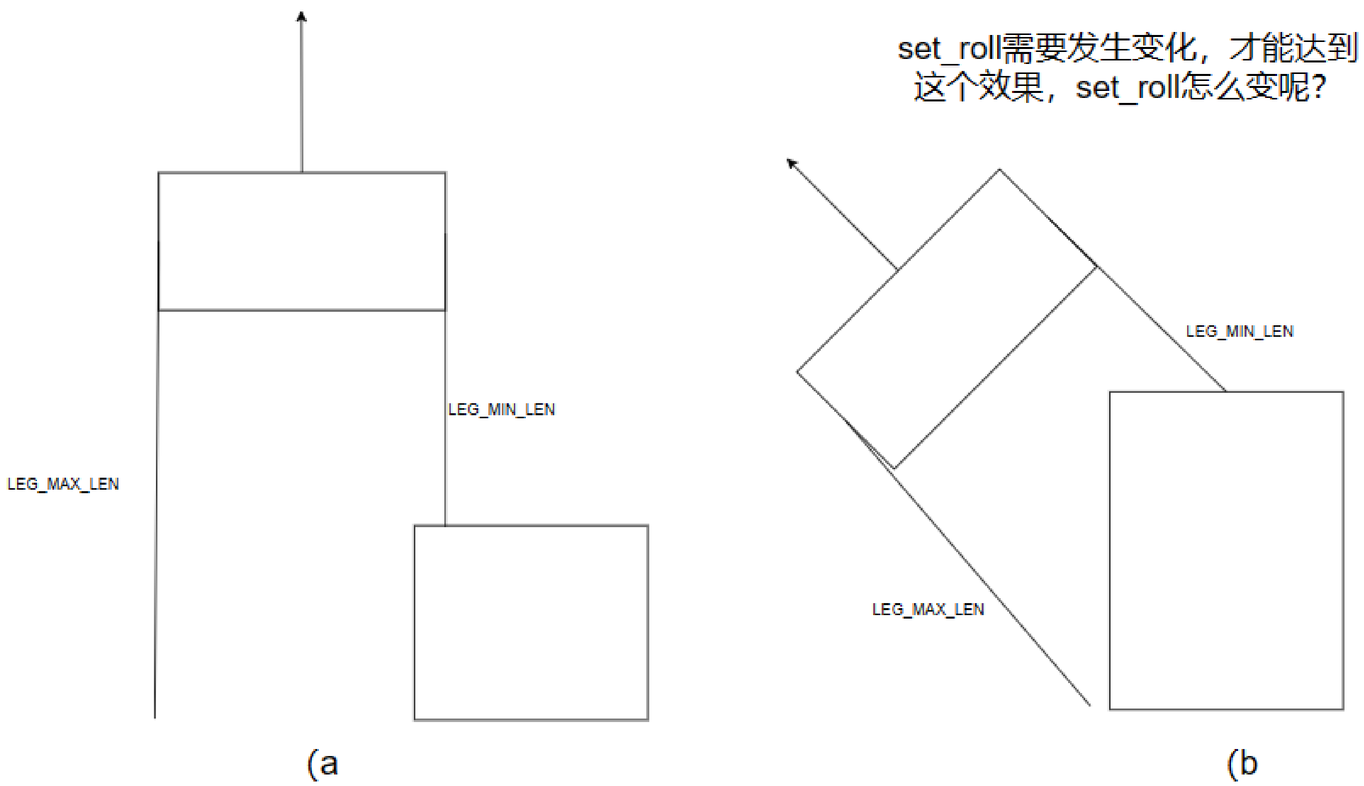
```
    else
    {
        float curve_roll = -(this->vel_forward * this->vel_yaw) / GRAVITY;
        float curve_roll_err = (-roll - curve_roll * t4) / sqrtf(curve_roll * 
curve_roll + 1);

        tan_theta = tanf(curve_roll_err);
        cos_theta = 1 / sqrtf(1 + tan_theta * tan_theta);
    }

    float temp_l = (param.core.WHEEL_DISTANCE / 2 + legL.len * tan_theta),
          temp_r = (param.core.WHEEL_DISTANCE / 2 - legR.len * tan_theta);

    if (temp_l < 0.02f)
        temp_l = 0.02f;
    if (temp_r < 0.02f)
        temp_r = 0.02f;

    legL.force_arm = temp_l * cos_theta;
    legR.force_arm = temp_r * cos_theta;

    float leg_force_arm_psc = legL.force_arm / legR.force_arm;//force arm
    legL.load_psc = 1 / (1 + leg_force_arm_psc);//L1 = / (L1 + L2)
    legR.load_psc = leg_force_arm_psc / (1 + leg_force_arm_psc);
}
```

Planner.cpp

```
void WB6_Planner::roll_cmd(const float input, const float dt)
{
    const float LEG_MAX_DIFF_ANGLE = atanf((param.core.LEG_MAX_LEN - 
param.core.LEG_MIN_LEN)/param.core.WHEEL_DISTANCE);

    if(!robot.balance || robot.mode  == WB6_Base::TRANSFORM_UP)
    {
        robot.set_roll   = robot.ground_tilt;
        robot.set_d_roll = 0;

        return;
    }

    float cmd_forward = cmd.forward.d_val * param.core.WHEEL_RADIUS,
          cmd_rotate  = cmd.yaw.d_val;
    float result = -(cmd_forward * cmd_rotate)/GRAVITY;
    deadzone(result, 0.1f);
```

```
    result = 0;
    robot.roll_curve = result;
    float Cmd   = input + robot.roll_curve,
          d_Cmd = 0;

    float max_cmd = tanf(robot.ground_tilt + LEG_MAX_DIFF_ANGLE),
          min_cmd = tanf(robot.ground_tilt - LEG_MAX_DIFF_ANGLE);
    if(max_cmd >  param.core.MAX_ROLL) max_cmd =  param.core.MAX_ROLL;
    if(min_cmd < -param.core.MAX_ROLL) min_cmd = -param.core.MAX_ROLL;

    if(Cmd < min_cmd)
    {
        Cmd   = min_cmd;
        d_Cmd = 0;
    }
    else if(Cmd > max_cmd)
    {
        Cmd   = max_cmd;
        d_Cmd = 0;
    }

    float maxRoll = robot.set_roll + param.core.MAX_ROLL_RATE*dt,
          minRoll = robot.set_roll - param.core.MAX_ROLL_RATE*dt;

    if(Cmd > maxRoll)
    {
        robot.set_roll   = maxRoll;
        robot.set_d_roll = 0;
    }
    else if(Cmd < minRoll)
    {
        robot.set_roll   = minRoll;
        robot.set_d_roll = 0;
    }
    else
    {
        robot.set_roll   = Cmd;
        robot.set_d_roll = d_Cmd;
    }
}

```

以上代码是有所关联的，本质上要解决的问题是合理规划6 Dof轮足下腿长和给定roll角。具体来说是想要解决以下这样的问题：



基础知识回顾

四元数、rpy角和旋转矩阵

首先还是明确下基本的概念：我们控制的时候关注的是rpy角(对应下列旋转矩阵的角r p y)，或者说**世界坐标系下先后**绕x、y、z三个轴的旋转角，或者说**机身坐标系下**先后绕z,y,x三个轴的旋转角（ZYX欧拉角）。

查阅维基百科可知，ZYX的旋转矩阵为：

$$Z_1Y_2X_3 = \begin{bmatrix} c_y c_p & c_r s_p s_y - c_r s_y & s_y s_r + c_y c_r s_p \\ c_p s_y & c_y c_r + s_y s_p s_r & c_r s_y s_p - c_y s_r \\ -s_p & c_p s_r & c_p c_r \end{bmatrix}$$

四元数转旋转矩阵：

$$Z_1Y_2X_3 = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y-q_wq_z) & 2(q_xq_z+q_wq_y) \\ 2(q_xq_y+q_wq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z-q_wq_x) \\ 2(q_xq_z-q_yq_w) & 2(q_yq_z+q_wq_x) & 1-2(q_x^2+q_y^2) \end{bmatrix}$$

对旋转矩阵、rpy、四元数，三者的表达式和相互关系是确定，受顺序影响的是欧拉角，因为旋转顺序有12种组合。

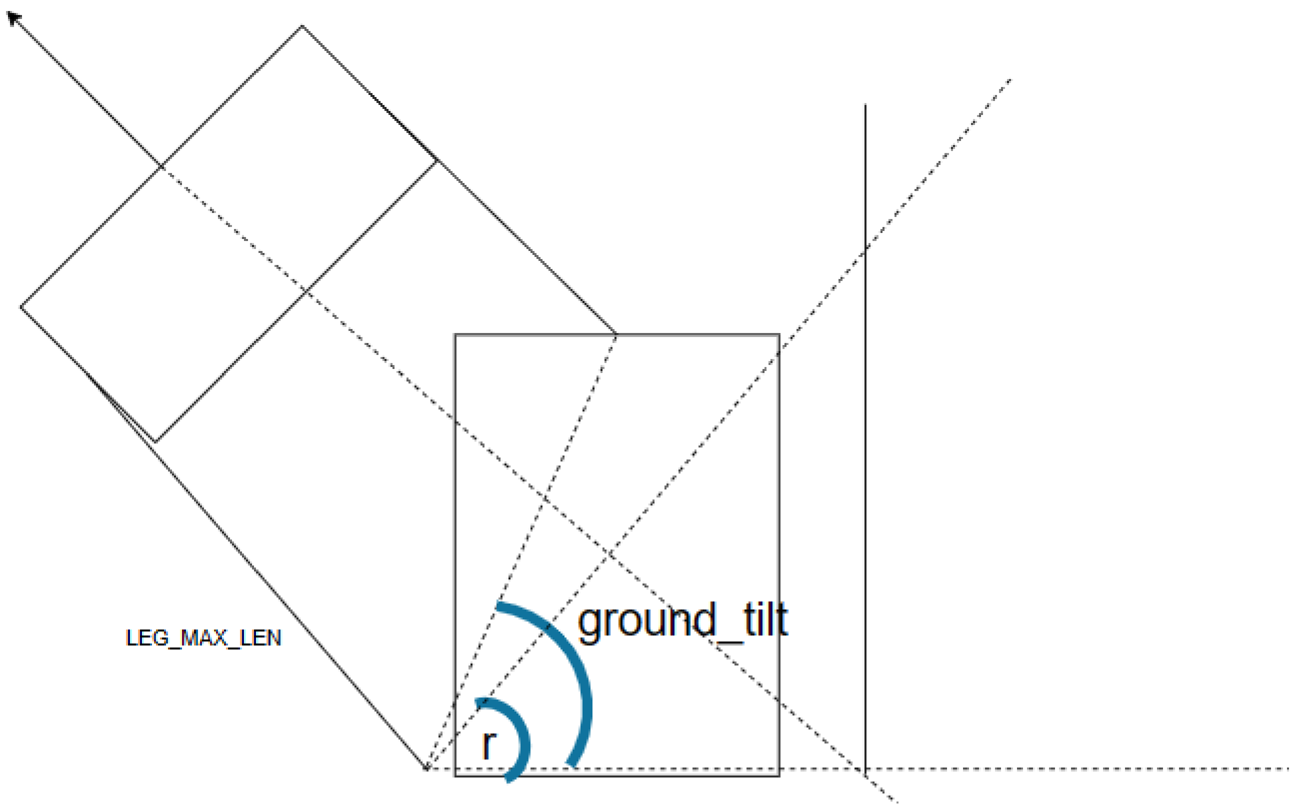
Proper Euler angles	Tait-Bryan angles
$X_1Z_2X_3 = \begin{bmatrix} c_2 & -c_3 s_2 & s_2 s_3 \\ c_1 s_2 & c_1 c_2 c_3 - s_1 s_3 & -c_3 s_1 - c_1 c_2 s_3 \\ s_1 s_2 & c_1 s_3 + c_2 c_3 s_1 & c_1 c_3 - c_2 s_1 s_3 \end{bmatrix}$	$X_1Z_2Y_3 = \begin{bmatrix} c_2 c_3 & -s_2 & c_2 s_3 \\ s_1 s_3 + c_1 c_3 s_2 & c_1 c_2 & c_1 s_2 s_3 - c_3 s_1 \\ c_3 s_1 s_2 - c_1 s_3 & c_2 s_1 & c_1 c_3 + s_1 s_2 s_3 \end{bmatrix}$
$X_1Y_2X_3 = \begin{bmatrix} c_2 & s_2 s_3 & c_3 s_2 \\ s_1 s_2 & c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_2 c_3 s_1 \\ -c_1 s_2 & c_3 s_1 + c_1 c_2 s_3 & c_1 c_2 c_3 - s_1 s_3 \end{bmatrix}$	$X_1Y_2Z_3 = \begin{bmatrix} c_2 c_3 & -c_2 s_3 & s_2 \\ c_1 s_3 + c_3 s_1 s_2 & c_1 c_3 - s_1 s_2 s_3 & -c_2 s_1 \\ s_1 s_3 - c_1 c_3 s_2 & c_3 s_1 + c_1 s_2 s_3 & c_1 c_2 \end{bmatrix}$
$Y_1X_2Y_3 = \begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & s_1 s_2 & c_1 s_3 + c_2 c_3 s_1 \\ s_2 s_3 & c_2 & -c_3 s_2 \\ -c_3 s_1 - c_1 c_2 s_3 & c_1 s_2 & c_1 c_2 c_3 - s_1 s_3 \end{bmatrix}$	$Y_1X_2Z_3 = \begin{bmatrix} c_1 c_3 + s_1 s_2 s_3 & c_3 s_1 s_2 - c_1 s_3 & c_2 s_1 \\ c_2 s_3 & c_2 c_3 & -s_2 \\ c_1 s_2 s_3 - c_3 s_1 & c_1 c_3 s_2 + s_1 s_3 & c_1 c_2 \end{bmatrix}$
$Y_1Z_2Y_3 = \begin{bmatrix} c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 & c_3 s_1 + c_1 c_2 s_3 \\ c_3 s_2 & c_2 & s_2 s_3 \\ -c_1 s_3 - c_2 c_3 s_1 & s_1 s_2 & c_1 c_3 - c_2 s_1 s_3 \end{bmatrix}$	$Y_1Z_2X_3 = \begin{bmatrix} c_1 c_2 & s_1 s_3 - c_1 c_3 s_2 & c_3 s_1 + c_1 s_2 s_3 \\ s_2 & c_2 c_3 & -c_2 s_3 \\ -c_2 s_1 & c_1 s_3 + c_3 s_1 s_2 & c_1 c_3 - s_1 s_2 s_3 \end{bmatrix}$
$Z_1Y_2Z_3 = \begin{bmatrix} c_1 c_2 c_3 - s_1 s_3 & -c_3 s_1 - c_1 c_2 s_3 & c_1 s_2 \\ c_1 s_3 + c_2 c_3 s_1 & c_1 c_3 - c_2 s_1 s_3 & s_1 s_2 \\ -c_3 s_2 & s_2 s_3 & c_2 \end{bmatrix}$	$Z_1Y_2X_3 = \begin{bmatrix} c_1 c_2 & c_1 s_2 s_3 - c_3 s_1 & s_1 s_3 + c_1 c_3 s_2 \\ c_2 s_1 & c_1 c_3 + s_1 s_2 s_3 & c_3 s_1 s_2 - c_1 s_3 \\ -s_2 & c_2 s_3 & c_2 c_3 \end{bmatrix}$
$Z_1X_2Z_3 = \begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_2 c_3 s_1 & s_1 s_2 \\ c_3 s_1 + c_1 c_2 s_3 & c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 \\ s_2 s_3 & c_3 s_2 & c_2 \end{bmatrix}$	$Z_1X_2Y_3 = \begin{bmatrix} c_1 c_3 - s_1 s_2 s_3 & -c_2 s_1 & c_1 s_3 + c_3 s_1 s_2 \\ c_3 s_1 + c_1 s_2 s_3 & c_1 c_2 & s_1 s_3 - c_1 c_3 s_2 \\ -c_2 s_3 & s_2 & c_2 c_3 \end{bmatrix}$

三角函数知识回顾

1. sin(x) $\approx$ x
2. sin(x) = $sin(x) \; = \; \frac{tax(x)}{\sqrt{(1 + tan^2(x))}}$

A1中的实现逻辑

ground_tilt计算



ground_tilt本质就是基座的r角再加上两腿落差角度。即

$$ground_{tilt} = r + atan(\frac{legR.len - legL.len}{WheelDiastance})$$

$$max_{groundtilt} = atan(\frac{max_{len} - min_{len}}{WheelDistance})$$

因此，当ground_tilt > max_ground_tilt （代码中为LEG_MAX_DIFF_ANGLE），只需让给定的roll为:

ground_tilt - max_ground_tilt

即可，代码中为：

```
robot.ground_tilt - LEG_MAX_DIFF_ANGLE
```

roll_cmd()中变量的定义

简单的结论：

1. this->roll = tan(r) * t4
2. set_roll : tan(r)
3. roll_err： 可能是 sin(r_err) ≈ r_err
4. 源代码中的t0 t1 t2 t3 t4 在ZYX旋转矩阵中的含义：

```
void toRotationMatrix(float R[3][3])//ZYX sequence
{ //          t1                                               t0
    R[0][0] = 1 - 2*(y*y + z*z); R[0][1] = 2*(x*y - w*z);     R[0][2] = 2*(x*z + w*y);
  //          t3                                                -t2
    R[1][0] = 2*(x*y + w*z);     R[1][1] = 1 - 2*(x*x + z*z); R[1][2] = 2*(y*z - w*x);
  //      -t4(sin = 1)                                        -t4(cos=1)
    R[2][0] = 2*(x*z - w*y);     R[2][1] = 2*(y*z + w*x);     R[2][2] = 1 - 2*(x*x + y*y);
}
```

原作者思路：

可以证明， roll / t4 = tan(r)，在 cos(leg_angle) = cos(0) = 1时成立。

> 当cos = 1，sin = 0 时
>
> this->roll = t1*t2 + t0*t3
>
> t4 = 2*(x*z + w*y)
>
> this->roll / t4 = (t1*t2+t0*t3) / t4
>
> = $\dfrac{(c_y c_p)(c_y s_r - c_r s_y s_p) + (c_p s_y)(s_y s_r + c_y c_r s_p)}{-c_p c_r}$
>
> = $\dfrac{c_y^2 c_p s_r + s_y^2 c_p s_r}{-c_p c_r}$
>
> = $\dfrac{c_y^2 c_p s_r + s_y^2 c_p s_r}{-c_p c_r}$
>
> =tan(r)

无法证明： roll / t4 = tan(r)在 sin=1时 成立。

> 当sin = 1，cos = 0 时
>
> this->roll = t1*t2 + t0*t3
>
> t4 = 2*(x*x + y*y) - 1
>
> r = this->roll / t4 = (t1*t2+t0*t3) / t4
>
> = $\dfrac{(c_y c_p)(c_y s_r - c_r s_y s_p) + (c_p s_y)(s_y s_r + c_y c_r s_p)}{s_p}$
>
> = $\dfrac{c_y^2 c_p s_r + s_y^2 c_p s_r}{s_p}$
>
> = $\dfrac{c_p s_r}{s_p}$
>
> = $cot(p) * s_r$
>
> 若p = 90
>
> r = 0

作者此处的做法是为了规避出现万向锁的情况，但从表达式上看是不对的，**应当属于原作者进行了近似，是可以修改和优化的地方。**

在脑袋水平时，该方法效果可以起作用，但如果进行了抬头，使用上应该会跟预期有出入。