3D 3D扣扣
WWW.3DKOUKOU.COM

# Delta （rostock 型)3d 打印机算法解读及调试步骤

本资料来自 QQ：鸭嘴兽分享

关注 3D 扣扣网（www.3dkoukou.com）

## 一、基础知识

1)三角函数 sin、cos 这个是理解 rostock 计算过程的基础知识。

2)笛卡尔空间坐标转换/线性代数（如果不懂也没关系，在把所有的外界条件全部设置为理想情况下，这个笛卡尔空间坐标转换可以不用精通的)。

3)marlin 程序的结构逻辑结构(不懂的小白们可以先学一下 arduino 的基础教程先)

## 二、Marlin 程序解读

我们这里不是介绍 marlin 的整个 loop（）函数的流程，只解读 delta 机型的核心部分。对于 marlin 来说，delta 机型和非 delta 机型区别在与 delta 多了一个笛卡尔坐标转换的函数

Marlin 的 loop()主体流程

```
Void loop ()
{
Get_command() ;  //从 sd 卡或者串口获取 gcode
Process_command(); //解析 gcode 并且执行代码
Manage_heater();//控制机器的喷头和热床的温度
Manage_inactivity();//
checkHitEndstops();//检查 endstop 的状态
Lcd_update(); //更新 lcd 上面的信息
}
```

在这个过程中 process_command()是控制的核心，process_command()就是一个巨大的 case 结构，这里讲讲 G1 命令的大致逻辑（G1 直线插补命令）：

```
Process_command()
{
Case 0: //g0->g1
```

```
Case 1 :
       {
     if(Stopped == false) {
        get_coordinates(); // 获取当前的坐标，这里是指打印件的三维坐标，不是 delta 的 xyz 电机的坐标哦！普通结构的打印机则是一样的。
          #ifdef FWRETRACT
            if(autoretract_enabled)
            if( !(code_seen('X') || code_seen('Y') || code_seen('Z')) && code_seen('E')) { //获取 命令中 xyze 轴的参数
            Float echange=destination[E_AXIS]-current_position[E_AXIS]; //这里是算最小回抽值的，如果移动距离小于最小回抽值就不回抽了。这里是一个辅助功能。
              if((echange<-MIN_RETRACT && !retracted) || (echange>MIN_RETRACT && retracted)) { //move appears to be an attempt to retract or recover
                  current_position[E_AXIS] = destination[E_AXIS]; //hide the slicer-generated retract/recover from calculations
                  plan_set_e_position(current_position[E_AXIS]); //AND from the planner
                  retract(!retracted);
                  return;
              }
            }
          #endif //FWRETRACT
        prepare_move(); //执行移动命令

        return;
       }
}
```

从上面的代码来看呢，对于运动类的 Gcode，marlin 会在 process_command()函数中获取 xyze 各轴的参数后算出目标坐标（destination[_AXIS]），也会使用 get_coordinates()来获取当前坐标（current_position[E_AXIS]）（坐标是打印件的三维坐标），当我们知道了目标坐标和当前坐标以后，空间中移动的距离就可以算出来了，接下来 marlin 就使用 perpare_move()来控制电机啦。

接下来介绍 prepare_move()这个函数。先上代码先（精简了一下），只看关键的部分就是 delta 和普通结构的代码，先介绍 plan_buffer_line() 这个函数的作用的把坐标数组 current_position[i] 、 destination[i] 放到一个内存的一个缓存区里面，然后控制电机转多少圈这样一个作用的，具体代码可以自己去看，在一旦进

入这个函数以后，delta 和普通机型的代码都是一样的，也就是说 delta 和普通结构的电机控制其实是一样的。

Difference[i]数组 ：用来储存目标坐标和当前坐标之间的距离的，（这里是包含了 xyze 轴的数组）

Destination[i]数组：目标坐标的数值，是从 process_command()函数中 G1 读取 XYZE 参数获取的。

Current_position[i]数组：当前坐标的数值，是从 G1 命令中 get_coordinates()传递过来的。如果是 3 个轴都归零的情况下，current_position 就是储存三个坐标原点，如果开始运动了，这里的值就是上一个 prepare_move()循环执行后上一次的 destination[i]的值。（这个下面会有看到赋值语句）

Delta[i]数组：delta 打印机的 xyz 三个电机要移动的距离

```
void prepare_move()
{

#ifdef DELTA // 设置机子是 delta 机型（rostock）
  float difference[NUM_AXIS]; //定义目标距离，用于转换坐标用的过渡变量
  for (int8_t i=0; i < NUM_AXIS; i++) {
    difference[i] = destination[i] - current_position[i];
  } //计算三维坐标的距离值
//***开始计算笛卡尔距离 并且暴力直线插值来减少运算量***//
  float cartesian_mm = sqrt(sq(difference[X_AXIS]) +
                            sq(difference[Y_AXIS]) +
                            sq(difference[Z_AXIS]));
  if (cartesian_mm < 0.000001) { cartesian_mm = abs(difference[E_AXIS]); }
  if (cartesian_mm < 0.000001) { return; }
  float seconds = 6000 * cartesian_mm / feedrate / feedmultiply;
  int steps = max(1, int(delta_segments_per_second * seconds));
  for (int s = 1; s <= steps; s++) {
    float fraction = float(s) / float(steps);//直线插值
    for(int8_t i=0; i < NUM_AXIS; i++) {
      destination[i] = current_position[i] + difference[i] * fraction;
    }
```

```
//***结束计算笛卡尔距离 并且暴力直线插值来减少运算量***//
    calculate_delta(destination);//将打印件的三维坐标转换为 xyz 电机轴的运动量
    plan_buffer_line(delta[X_AXIS], delta[Y_AXIS], delta[Z_AXIS],
                     destination[E_AXIS], feedrate*feedmultiply/60/100.0,
                     active_extruder);
  }
#endif // DELTA
。。。。。。。。。。。。
#if ! (defined DELTA || defined SCARA)
  // Do not use feedmultiply for E or Z only moves
  if( (current_position[X_AXIS] == destination [X_AXIS]) && (current_position[Y_AXIS]
== destination [Y_AXIS])) {
      plan_buffer_line(destination[X_AXIS], destination[Y_AXIS], destination[Z_AXIS],
destination[E_AXIS], feedrate/60, active_extruder); //直接将 destination[i]的值发送去
运动缓存里面
  }
  else {
    plan_buffer_line(destination[X_AXIS], destination[Y_AXIS], destination[Z_AXIS],
destination[E_AXIS], feedrate*feedmultiply/60/100.0, active_extruder);
  }
#endif // !(DELTA || SCARA)

  for(int8_t i=0; i < NUM_AXIS; i++) {
    current_position[i] = destination[i]; //更新当前坐标的值为刚执行的目标坐标值
  }
}
```

小结：对于普通结构来说，G1 每次将新读取 gcode 代码参数传递给 prepare_move()函数中 destination[i]数组以后，prepare_move()就会将其传递到 plan_buffer_line（）进行电机的运动。而 delta 结构呢，就相对复杂一点，G1 命令读取了 gcode 代码参数后也是传递到 prepare_move()函数中 destination[i]，然后 marlin 要计算目标坐标与当前坐标的笛卡尔距离，然后通过固定时间间隔的方式来将笛卡尔距离分成若干个小直线，通过这样的方式来就减少 cpu 的浮点预算量，然后再通过 calculate_delta[i]函数来将简化后的 destination[i]换算成三个电机的运动坐标，并传递到 delta[i]中，接下来就是 plan_buffer_line（）了。

最后，来看看 calculate_delta（）函数，这个函数的主要用途是将打印件的

3D 3D扣扣 WWW.3DKOUKOU.COM

三维坐标转换为三个垂直的电机轴的运动坐标哦。注意：新的 marlin 支持 SCARA 结构的 delta，那里也有个 calculate_delta（）的函数，不过那个跟 rostock 有点差异。下面介绍 rostock。

```
void calculate_delta(float cartesian[3])
{
  delta[X_AXIS] = sqrt(delta_diagonal_rod_2
                       - sq(delta_tower1_x-cartesian[X_AXIS])
                       - sq(delta_tower1_y-cartesian[Y_AXIS])
                       ) + cartesian[Z_AXIS];
  delta[Y_AXIS] = sqrt(delta_diagonal_rod_2
                       - sq(delta_tower2_x-cartesian[X_AXIS])
                       - sq(delta_tower2_y-cartesian[Y_AXIS])
                       ) + cartesian[Z_AXIS];
  delta[Z_AXIS] = sqrt(delta_diagonal_rod_2
                       - sq(delta_tower3_x-cartesian[X_AXIS])
                       - sq(delta_tower3_y-cartesian[Y_AXIS])
                       ) + cartesian[Z_AXIS];
  /*
  SERIAL_ECHOPGM("cartesian x="); SERIAL_ECHO(cartesian[X_AXIS]);
  SERIAL_ECHOPGM(" y="); SERIAL_ECHO(cartesian[Y_AXIS]);
  SERIAL_ECHOPGM(" z="); SERIAL_ECHOLN(cartesian[Z_AXIS]);

  SERIAL_ECHOPGM("delta x="); SERIAL_ECHO(delta[X_AXIS]);
  SERIAL_ECHOPGM(" y="); SERIAL_ECHO(delta[Y_AXIS]);
  SERIAL_ECHOPGM(" z="); SERIAL_ECHOLN(delta[Z_AXIS]);
  */
}
```

代码很简单 delta[i]是指电机轴的运动坐标，cartesian[i]是指打印件的三维坐标，从上面的程序来看就是从 prepare_move()中经过插值简化的 destination[i]。随便看一个轴的换算

```
  delta[X_AXIS] = sqrt(delta_diagonal_rod_2
                       - sq(delta_tower1_x-cartesian[X_AXIS])
                       - sq(delta_tower1_y-cartesian[Y_AXIS])
                       ) + cartesian[Z_AXIS];
```

delta_diagonal_rod_2 是推杆长的平方

3D 3D扣扣
WWW.3DKOUKOU.COM

delta_tower1_x 是左前柱的 x 坐标值，是由 radius 这个参数算出来的

delta_tower1_y 是左前柱的 y 坐标值，是由 radius 这个参数算出来的

具体怎么算就看下面这个函数

```
void recalc_delta_settings(float radius, float diagonal_rod)
{
    delta_tower1_x= -SIN_60*radius; // front left tower
    delta_tower1_y= -COS_60*radius;
    delta_tower2_x=  SIN_60*radius; // front right tower
    delta_tower2_y= -COS_60*radius;
    delta_tower3_x= 0.0;                    // back middle tower
    delta_tower3_y= radius;
    delta_diagonal_rod_2= sq(diagonal_rod);
}
```

总结一下 marlin 的 delta 机型参数有哪些？
推杆的长度、电机轴上滑块的宽度、喷头支架的宽度，还有三个电机的圆半径。

```
//===================================================================
//========================Delta Settings =============================
//===================================================================
// Enable DELTA kinematics and most of the default configuration for Deltas
#define DELTA

// Make delta curves from many straight lines (linear interpolation).
// This is a trade-off between visible corners (not enough segments)
// and processor overload (too many expensive sqrt calls).
#define DELTA_SEGMENTS_PER_SECOND 200

// NOTE NB all values for DELTA_* values MUST be floating point, so always have a decimal
point in them

// Center-to-center distance of the holes in the diagonal push rods.
#define DELTA_DIAGONAL_ROD 250.0 // mm  //杆长

// Horizontal offset from middle of printer to smooth rod center.
#define DELTA_SMOOTH_ROD_OFFSET 175.0 // mm  //电机轴的圆半径

// Horizontal offset of the universal joints on the end effector.
#define DELTA_EFFECTOR_OFFSET 33.0 // mm  // 装喷嘴的平台的中心到杆连接处的距离

// Horizontal offset of the universal joints on the carriages.
#define DELTA_CARRIAGE_OFFSET 18.0 // mm  //电机轴滑块的距离
```

```
// Effective horizontal distance bridged by diagonal push rods.
#define DELTA_RADIUS
(DELTA_SMOOTH_ROD_OFFSET-DELTA_EFFECTOR_OFFSET-DELTA_CARRIAGE_OFFSET)
```

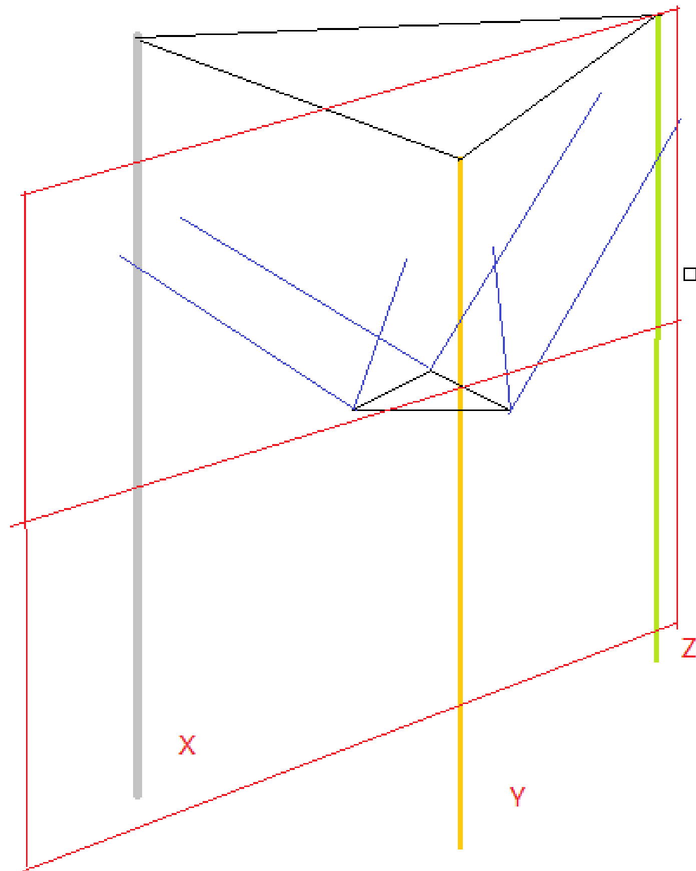
通过上述的参数可以算出一个 DELTA_RADIUS ,这个 delta_radius 就是上面“delta_tower1_x 是左前柱的 x 坐标值，是由 radius 这个参数算出来的 ”里面的 radius 了。

至此所有有关与 delta 的运动的代码已经通读了一遍。下面就开始分析分析代码和运动的关系了。

## 三、Rostock 运动分析

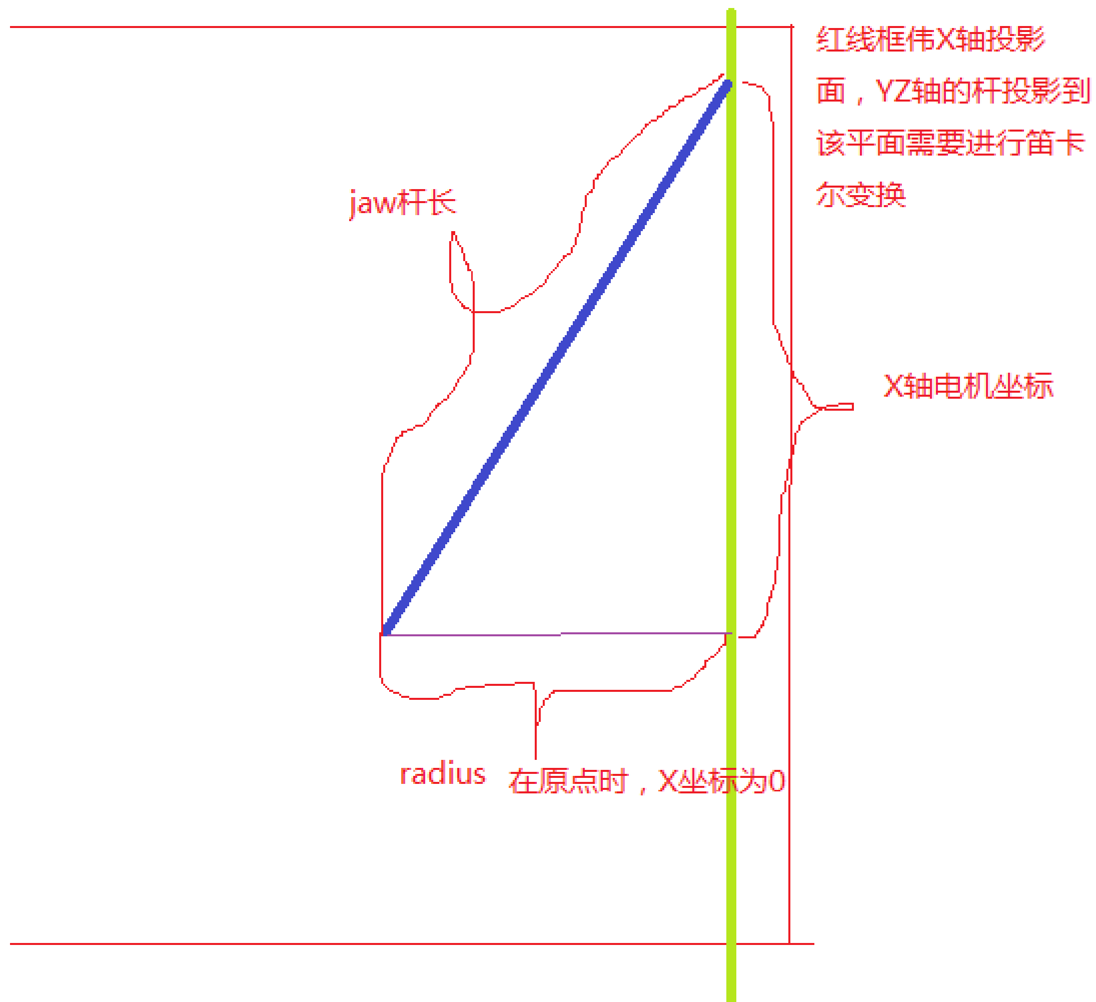
下面是 Rostock 的结构示意图，分析的第一步是简化整个结构，这里就需要将 XY 电机的两个竖轴投影到 Z 轴的平面上，下图中红色线框画出来的就是 z 轴的平面，同时我们可以不考虑 XY 电机的推杆的运动情况，因为可以 XY 电机轴的运动可以通过投影在 z 轴平面上的虚拟轴笛卡尔空间变换转换回去的。

3D扣扣
WWW.3DKOUKOU.COM




投影好了以后接下来把 z 轴放平，那么单独考虑 z 轴情况，这个情况是在坐标原点的 z 电机轴与推杆的情况(简化过程,已经把 z 轴滑块,喷头平台都设定为 0 )。那么,z 电机轴方向便形成了一个三角形,推杆、radius 和 z 轴电机上的电机坐标,这个时候三角函数出来啦！ 推杆 2 = radius 2+ 电机坐标 2 在这三角形中推杆是不变的，另外三角形始终都会是一个直角三角形。

3D扣扣
WWW.3DKOUKOU.COM



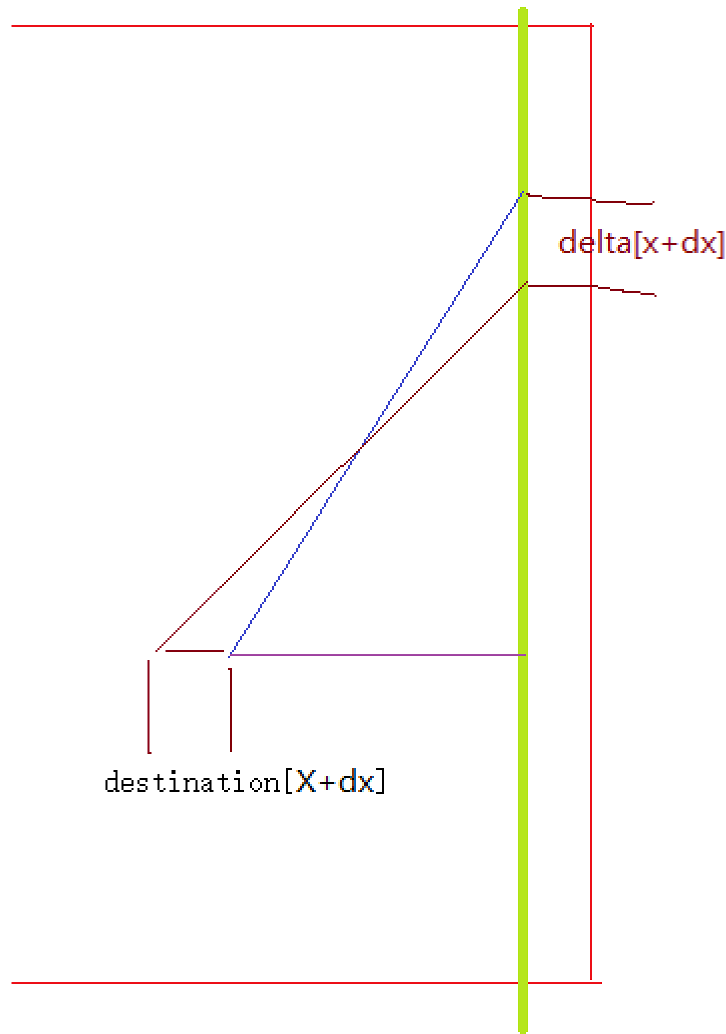

我们假设喷头只在 x 轴上运动，z 轴 y 轴都不动。如果打印件的三维坐标移动 dX 距离，rostock 需要考虑的问题就是怎么讲 dX 转换为 z 轴电机的移动距离了。下图就是用来表示这种情况。由于推杆长度是不变的，那么

推杆 2 = radius 2+ 电机坐标 2  就变成下面这样

推杆 2 = (radius +dX)2+ (电机坐标-dx) 2

Radius 的距离换成 destination[x]，电机坐标换成 delta[x]

推杆 2 = (destination[x +dX])2+ (delta[x-dx]) 2




Marlin 中 calculate_delta（）这个函数其实就是算

推杆 2 = (destination[x +dX])2+ (delta[x-dx]) 2

这个等式明白啦，打印件 X 轴和 Y 轴的运动分析就明白啦。

另外，再看看打印件 z 轴的运动分析，还是看看源代码

```
delta[X_AXIS] = sqrt(delta_diagonal_rod_2
                     - sq(delta_tower1_x-cartesian[X_AXIS])
                     - sq(delta_tower1_y-cartesian[Y_AXIS])
                     ) + cartesian[Z_AXIS];
```

Cartesian[z]是没有在 sqrt 函数里面的，而是直接加在 delta[x]的值上面的。

所以，在调机的时候应该先调 z 轴，这里就是原因。因为在 XYZ 三个轴的坐标中只有 Z 轴是直接通过同步轮和电机脉冲就可以调准的。调准了 z 轴以后再调 XY

轴才是对的。

## 四、调机心得

首先是调机的顺序：

1）选择同步轮可以选择 GT2 20 齿/40 齿 。因为 GT2 是 2mm 齿距，整数齿可以使脉冲数为整数，同时也减少三角函数运算中浮点运算的压力，同时也简化自己的调整步骤。

2）调机时先调 z 轴运动方向的精度，这个是直接用同步轮和脉冲数就可以调好的。

3）z 轴调好以后就调整 x 轴或者 Y 轴的，看看运动的路径是不是呈现平面状态，如果喷头运动路径是弧线这个时候就要调整 radius 了，增加或者减少 radius 的值来调整运动路径是一个平面

4）完成上述步骤以后就可以试打了，这个时候就可以看看 XY 轴的打印误差是多少了。如果杆长等硬件参数都比较准确的话那打印误差不会有多少的。如果 XY 轴有误差的话就要根据误差大小来等量调整 radius 这个变量。对应代码是

```
// Horizontal offset from middle of printer to smooth rod center.
#define DELTA_SMOOTH_ROD_OFFSET 175.0 // mm  //电机轴的圆半径
```

等量修改，比如 X 轴偏大 0.1mm，那么 调整量就是 175.0-0.1 = 174.9 了哦。这样反复几次就基本调好了。