哈 尔 滨 工 业 大 学

硕 士 学 位 论 文 中 期 报 告

基于视觉的经颅磁治疗导航系统研究

院　　（系）　机电工程与自动化学院

学　　科　机械工程

导　　师　王昕

研　究　生　张庆培

学　　号　**16S153587**

中期报告日期　**2018 年 3 月 14 日**

研究生院制

二〇一八年三月

目 录

1 主要研究内容及工作进度

1.1 课题的主要研究内容

基于视觉的经颅磁治疗导航系统主要功能是在病人治疗过程中识别头部位姿，辅助机械臂加持经颅磁治疗头到达患者头部最佳刺激点，实现精准治疗。而且在治疗过程中，该导航系统还应能够识别到哪些空间位置障碍物，使经颅磁治疗头自动避开病人身体和医疗器械。所以，本课题主要研究内容为以下几个方面：

（1）障碍物三维坐标获取及病人头部的三维重建

经颅磁刺激康复机器人辅助治疗系统主要功能是使 TMS 治疗头末端能精确达到病人头部病需要刺激点的位置，在这过程中我们不仅要知道 TMS 治疗头和病人头部位姿，我们还应该知道机器人工作空间的障碍物信息。即系统应该能使 TMS 治疗头自动绕开病人，避免伤害到患者。如果我们能得到机器人工作空间物体的三维坐标信息，并将病人头部三维重建出来，就可以在路径规划时躲开障碍物。另外，如果我们能建出患者头颅的三维模型，对头颅的真实空间和图像空间的配准也有一定的帮助，后期医生可以直接在 PC 端患者头颅模型上直接指出刺激点，使机械臂加持 TMS 治疗头自动到达最佳刺激点，实现自动治疗。

（2）病人脸部特征点定位

在经颅磁治疗过程中，患者没有打麻药，和正常人是一样的，所以头部不会固定在一个位置。这样的话即使一开始经颅磁治疗头准确到达了病灶点对应的外部刺激点，在治疗过程中，这种相对位置也会改变，导致刺激位置不准确，使得经颅磁治疗效果不大。所以，在经颅磁治疗过程中，导航系统应该能识别出病人的头部位置。而人体的头部最明显的特征体现在脸部，假如我们能获得病人脸部的某些特征点的空间位置，我们便很容易能得到病人的头部位置，这样就能让机械臂加持经颅磁治疗头实时跟踪病灶点，达到精准最佳治疗。所以这部分的主要功能就是实时的对病人脸部的特征点进行三维定位。

（3）导航系统整体方案设计

这部分的主要研究内容一是对整个导航系统各个部分的空间位置进行布置，使得我们既可以利用相机准确的得到病人的头部位姿和机器人工作空间障碍物信息，又不占用太大的空间，还应该保证病人头部所在位置是机械臂的可达空间；二是对病人的整个治疗过程进行分析和确定。

（4）导航系统和机械臂联合实验研究

这部分主要是通过实验的方法来改进整个导航系统的整体设计方案，优化三维重建算法以及脸部位姿识别算法，并结合机械臂进行联合实验，进一步提高整体的实用性，使整个系统能获得理想的导航效果。

1.2 课题进度情况

本课题现在已经完成了机器人工作空间障碍物三维坐标的获取、基于视觉的三维重建算法的研究和头部特征部位位姿识别算法的研究等，并将算法在工程中得到了实现，但是目前来说，精度还不是很高，算法还需进一步改进。中期之后主要是结合实验对三维重建算法以及脸部定位算法进行改进提高整合，并进一步确定整个导航系统的具体方案，联合机械臂进行整体实验。表 1 是本课题的进度和完成情况。

表 1 课题进展及完成情况

时间	课题进展与预期目标	完成情况
2017.09.01—2017.09.22	收集资料，阅读文献，确定研究内容与研究方案，完成开题报告，准备开题答辩	完成
2017.09.23—2017.10.31	学习图像相关知识，并熟悉 kinect2 相机在 ubuntu 上的使用方法；学习机械臂相关知识	完成
2017.11.01—2017.11.15	对相机进行标定，并对相机和机械臂进行手眼标定，得到相机和机械臂的相对位姿	完成
2017.11.16—2017.12.31	学习常见的三维重建算法，并在工程中进行实现	完成
2018.01.03—2018.02.28	学习利用机器学习方面的知识对头部进行定位，并把算法应用到工程中	部分完成
2018.03.01—2018.04.20	对三维重建和头部特征部位定位算法进行改进，并进行相关实验，确定整体方案具体实现细节	未完成
2018.04.21——2018.06.21	将实验期间取得的相关成果进行整理，并发表一篇学术小论文	未完成
2018.06.22——2018.09.31	整理研究成果，撰写、修改、完善硕士学位论文	未完成
2018.09.01——2018.12.20	准备硕士学位论文答辩	未完成

2 目前已完成的主要研究工作及结果

2.1 标定

首先，在图像测量以及机器视觉应用中，为确定空间物体表面某点的三维几何位置与其在图像中对应点之间的相互关系，必须建立相机成像的几何模型，这些几何模型参数就是相机参数。在大多数条件下这些参数必须通过实验与计算才能得到，这个求解参数的过程就称之为相机标定；其次，我们利用相机测量得到的空间物体的位置信息是相对相机坐标系来说的，但是我们的整个治疗系统的执行机构是机械臂，所以我们必须要知道相机和机械臂的空间位姿关系。基于以上两点原因，我们进行了相机的标定以及手眼标定工作。

2.1.1 相机的标定

在本课题中，使用的是 Kinect2 作为视觉传感器对机器人工作空间及病人头部进行三维重建。图 2-1 是 Kinect2 的实物图，它由一个普通的 RGB 相机和一个红外相机组成，RGB 相机可以得到一个真实场景的色彩信息，而红外相机可以得到相机视野中每个点的深度。通常来说，我们需要对两个相机都进行标定，但是在查阅众多资料发现对于红外相机获得的深度信息一般标定效果不是很明显，所以我们只对 Kinect2 的 RGB 相机进行了标定。对于 Kinect2 的 RGB 相机，我

图 2-1 Kinect2 相机实物图

们采用用得比较多的针孔模型进行标定。

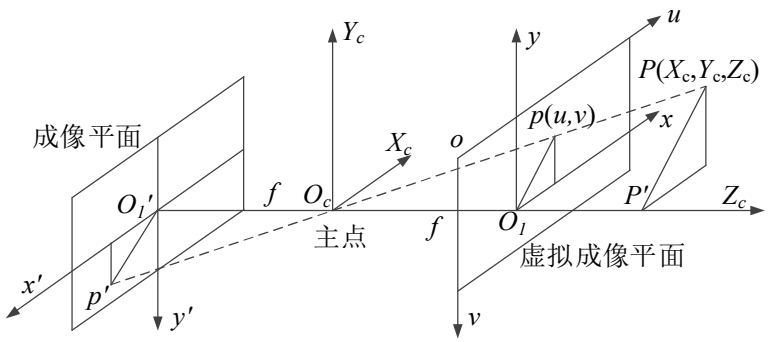

图 2-2 针孔相机成像模型

图 2-2 为针孔相机成像模型，$x'-y'$为相机的成像平面，$O_1'O_c$为焦距f，$X_cO_cY_cZ_c$组成相机坐标系，$X_wO_wY_wZ_w$为世界坐标系。令$x-y$为虚拟成像平面，也称图像平面坐标系，单位为 mm，且满足$O_1O_c=O_1'O_c=f$；将处于同一平面的$u-v$坐标系称为图像像素坐标系，$O_1(u_0,v_0)$为该平面中心点，单位为 pixel。假设三维点P在$X_wO_wY_wZ_w$下的坐标为(X_w,Y_w,Z_w)，在$X_cO_cY_cZ_c$下的坐标为(X_c,Y_c,Z_c)，在$x-y$下的投影点为$p(x,y)$，在$u-v$下的投影点为$p(u,v)$，因此可得(x,y)与(u,v)之间的齐次变换关系为：

$$\begin{cases} u=\dfrac{x}{dx}+u_0 \\ v=\dfrac{y}{dy}+v_0 \end{cases} \Rightarrow \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}=\begin{bmatrix} 1/dx & 0 & u_0 \\ 0 & 1/dy & v_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2-1}$$

式中dx、dy分别为像素点在x、y轴方向上的物理尺寸，单位 mm。

在图 2-2 中，根据ΔO_cO_1p与$\Delta O_cP'P$的相似关系，可得点P在$X_cO_cY_cZ_c$坐标系下与在xO_1y坐标系下的齐次变换关系为：

$$\begin{cases} x=\dfrac{f}{Z_c}\times X_c \\ y=\dfrac{f}{Z_c}\times Y_c \end{cases} \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}=\frac{1}{Z_c}\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2-2}$$

由于$X_cO_cY_cZ_c$与$X_wO_wY_wZ_w$不重合，二者的关系通常由一个3×3的旋转矩阵$\boldsymbol{R}$和一个3×1平移向量$\boldsymbol{T}$进行描述，转换为齐次坐标关系为：

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}=\begin{bmatrix} \boldsymbol{R} & \boldsymbol{T} \\ \boldsymbol{O}^T & 1 \end{bmatrix}\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}=\boldsymbol{M}_2\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2-3}$$

通过连乘式(2-1)~(2-3)中的各个矩阵，可得：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}=\frac{1}{Z_c}\begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}\boldsymbol{M}_2\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}=\frac{1}{Z_c}\boldsymbol{M}_1\boldsymbol{M}_2\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2-4}$$

式中 f_x、f_y——分别为镜头在u，v轴方向上的焦距，$f_x=\dfrac{f}{dx}$，$f_y=\dfrac{f}{dy}$；$\boldsymbol{M}_1$、$\boldsymbol{M}_2$分别为相机的内参矩阵和外参矩阵。

上面的分析是理想的针孔相机模型，但是在实际中由于受镜头机械组装误差，以及镜头加工精度的影响，会使得拍摄的照片产径向畸变和切向畸变，使在图像

上得到的感知点的位置(u,v)发生变化。针对这两种畸变，引入径向畸变和切向径向畸变系数k_1,k_2,k_3和切向畸变系数p_1,p_2，令(u_d,v_d)为在像素坐标系下感知点的位置，(u,v)为理想的位置，则他们满足：

$$\begin{bmatrix} u \\ v \end{bmatrix} = (1+k_1r^2+k_2r^4+k_3r^6)\begin{bmatrix} u_d \\ v_d \end{bmatrix} + \begin{bmatrix} 2p_1u_dv_d + p_2(r^2+2u_d^2) \\ 2p_2u_dv_d + p_1(r^2+2v_d^2) \end{bmatrix} \tag{2-5}$$

图 2-3 matlab 提取的角点

	1	2	3	4
1	532.5010	0	0	
2	0	532.3318	0	
3	327.9292	253.0929	1	
4				

-0.0029	-0.0042

0.0465	-0.1640	0.0391

图 2-4 相机内参和畸变系数

然后用 Kinect2 拍摄 25 张标定板照片，利用 matlab 工具箱提取标定板中的角点信息，如图 2-3 所示，当获得了足够多的 3D/2D 点信息后即可利用式 2-4 和 2-5 计算出相机内参矩阵 M1 和畸变系数，最终得到的相机内参和畸变系数如图 2-4 所示。

2.1.2 手眼标定

这部分主要是为了得到相机坐标系和机械臂基坐标系的空间位置关系。

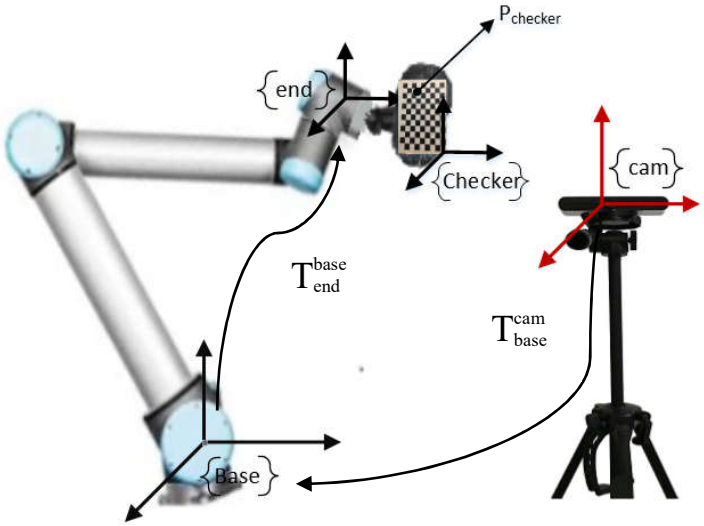

图 2-5 手眼标定空间位置示意图

如图 2-5 所示为相机和机械臂在空间中的相对位置示意图，我们将标定板刚性的固定在机械臂末端的经颅磁治疗头上，分别在标定板、机械臂末端、机械臂

底座和相机处建立四个坐标系：Checker、end、Base 和 cam。现考虑标定板 Checker 坐标系下的一个角点 $P_{checker}$，根据机器人学的知识我们可以得到如下式子：

$$P_{cam} = T_{ins}\ T_{base}^{cam}\ T_{end}^{base}\ T_{checker}^{end}\ P_{checker} \tag{2-6}$$

式中 $P_{checker}$ 是标定板坐标系下的一个点，$T_{checker}^{end}$ 是标定板相对于机械臂末端坐标系的空间变换矩阵，T_{end}^{base} 是机械臂末端坐标系相对于机械臂基坐标系的空间变换矩阵，T_{base}^{cam} 是机械臂基坐标系相对于相机坐标系的空间变换矩阵，T_{ins} 是相机的内参矩阵。在这个式子中，T_{end}^{base} 可以通过读取机械臂的状态直接得到，$P_{checker}$ 是我们直接定义的，P_{cam} 是标定板上的角点在相机的像素坐标系下的坐标，我们可以利用 opencv 的角点检测函数直接得到，而 T_{ins} 可以通过事先标定相机得到。这样在整个等式中的未知量只有两个空间旋转矩阵，而每个空间旋转矩阵的自由度是 6，所以我们在这个等式中存在 12 个自由的未知量，一个空间点 $P_{checker}$ 可以列出 3 个方程，还不能解出相应的未知量，所以我们定义多个角点，移动机械臂末端，拍摄标定板的多个位姿，我们就可以列出多余 12 个的方程组，然后利用 matlab 的优化函数 fmincon 去解出最优的解来最小化误差。

由于这个标定过程需要拍摄多张标定板照片，并同时记录下机械臂末端 TCP 的位姿，而且还需要使用 opencv 检测出标定板角点的像素坐标，并将所有这些数据传送到 matlab 来解出最优解，标定起来需要注意的细节比较多，基于以上原因，就在 linux 上做了一个小的界面程序，使标定过程简单易行，图 2-6 是做的界面，使用该界面时只需要点几次按钮，并同时移动多次机械臂末端位姿就可以直接标定出来相机坐标系和机械臂的基坐标系之间的空间位姿关系。图 2-7 是我们标定出来的机械臂的基坐标系和相机坐标系之间的位姿关系和每个像素的重投影误差均值，重投影误差均值在 1 个像素以内，标定效果还是很不错的。

图 2-6 手眼标定界面程序

```
>> Demo
Starting Arm Calibration
Extracting Chessboards

ans =

    11

Finding Camera Parameters
Running Optimization
Converting to Transformation matricies
Calibration completed in 7.9 seconds with a mean error of 0.285 pixels

Final camera to arm base transform is
    0.9988    0.0270    0.0401   -0.1127
    0.0435   -0.1382   -0.9895    0.3931
   -0.0212    0.9900   -0.1392    1.2685
         0         0         0    1.0000

Final end effector to checkerboard transform is
   -0.9988    0.0494    0.0017    0.2385
    0.0494    0.9986   -0.0189   -0.0789
   -0.0026   -0.0188   -0.9998    0.0436
         0         0         0    1.0000

Final camera matrix is
  527.9520         0  321.7693
         0  528.1056  247.1670
         0         0    1.0000

Final camera radial distortion parameters are
    0.0576   -0.2487    0.1554

   -0.0032   -0.0040
```

图 2-7 手眼标定结果

2.2 障碍物三维坐标获取及头部的三维重建

机械臂加持经颅磁治疗头跟随病人头部病灶点做到随动的过程中要进行路径规划，而进行路径规划的前提是必须知道障碍物的空间三维坐标信息以及病人的头部模型，所以有必要对其工作空间内的物体进行三维坐标获取以及对病人头部进行三维重建。

2.2.1 获得 Kinect2 的 RGB 和深度图像

Kinect 2 是微软公司 2014 年生产的一款深度相机，由于微软公司对 linux 系统的封闭性比较强，Kinect 2 的很多东西包括 SDK 和驱动程序等在 linux 上都不能用，所以在获取图像数据方面，我们只能从网上找别人的驱动包或者自己写驱动。

在 linux 系统上使用 Kinect2，在获得数据方面目前用的比较多的方法主要有两种，一种是使用网上开源的 ros 包 iai_kinect2，如图 2-8 是使用 iai_kinect2 获得的 Kinect2 彩色图像并利用 ros 的 rviz 工具进行了可视化。使用这种方法，我们可以直接运行 launch 文件对 Kinect2 进行标定，并得到 Kinect2 的 RGB 相机和红外相机的内参矩阵和两个相机之间的空间位姿矩阵，除此之外有一个最大的优点就是在获得图片数据方面，我们只需要订阅 ros 的相关话题就可以得到 Kinect2 的 RGB 和深度图像数据，而且他们都是标定好的并都映射到 RGB 相机坐标系下的。但是有一个很大的缺点就是我们必须得遵守 ros 的时钟系统，使用这种方法获得的图像的频率是由 ros 自己决定的，我们可以改，但是很麻烦。从图 2-8 中我们可以看出我们获得的彩色图像的频率是 25Hz 左右，要是再加上处理图片的时间可能满足不了我们的要求。

图 2-8 利用 iai_kinect2 获得 Kinect2 的图像数据

另外一种方法就是利用 libfreenect2 库来自己写驱动程序，libfreenect2 是 OpenKinect 小组开发的一个针对 kinect v2 的非官方驱动，它是开源的，在 linux 系统上我们可以很容易的安装，并可以很自由的使用。当我们在官方驱动不能满足需求的时候就可以使用它。利用 libfreenect2 我们可以获得 kinect2 的彩色图像，深度图像以及红外图像等。利用这种方法我们可以自己控制获得图像数据的时间，而且获得的图像都是原始的，只是可能需要花一些时间去来利用 libfreenect2 库。

经过比较两种方法的优缺点，我们最终选用使用 libfreenect2 来写驱动程序的方法来获得 Kinect2 的 RGB 和深度图像。图 2-9 是我们利用 libfreenect2 获得的 RGB 和深度图像，由图像中的帧率可以看出，我们的获取速度明显比用 ros 包快

图 2-9 利用 libfreenect2 获得的图像数据

很多。

2.2.2 障碍物三维坐标信息的获取

Kinect2 获得的彩色图像和深度图像经过标定之后已经对齐，直接读取像素对应的深度信息便可以得到距离信息，但是要想获得相机坐标系下的三维坐标还需要进行相应的转换。

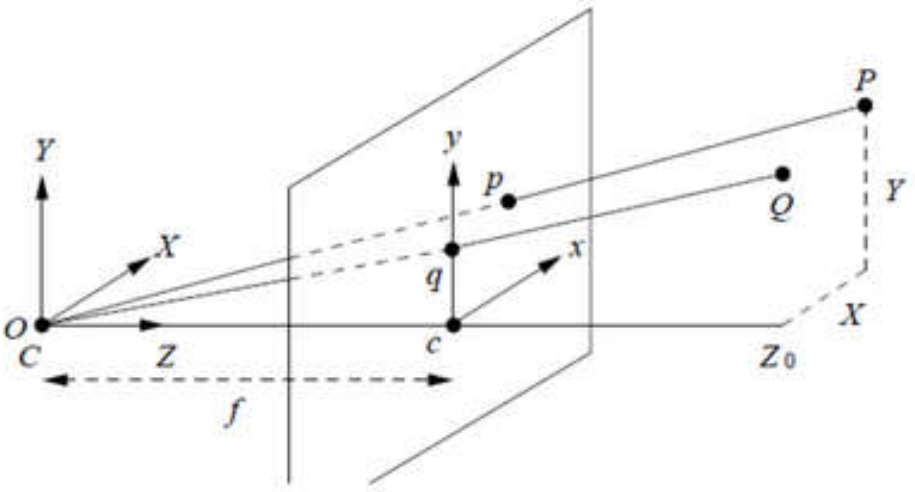

图 2-10 相机成像示意图

图 2-10 是相机的像素坐标系和三维坐标系之间的空间位置关系，现考虑相机的三维坐标系下一点 $P(X,Y,Z)$ 它在相机的像素坐标系下的感应点对应的坐标为 $p(u,v)$，根据理想的小孔成像模型我们知道：

$$\begin{bmatrix} u \\ v \\ d \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \text{即：} \quad \left.\begin{aligned} u &= X * f_x / Z + C_x \\ v &= Y * f_y / Z + C_y \\ d &= Z \end{aligned}\right\} \tag{2-7}$$

根据式 2-7 我们可以得到：

$$\left.\begin{aligned} Z &= d \\ X &= (u - C_x) * Z / f_x \\ Y &= (v - C_y) * Z / f_y \end{aligned}\right\} \tag{2-8}$$

在上述式子中，u 和 v 是每一个点的像素位置，我们可以直接得到，d 是每个点的深度信息，我们可以从深度图像中得到，所以我们很容易就可以得到 RGB 图像中每个点在相机坐标系下的三维信息。图 2-11 是我们测量的标准标定板两个角点之间的距离。在图 2-11 中，我们选取了两个角点，用我们的 Kinect2

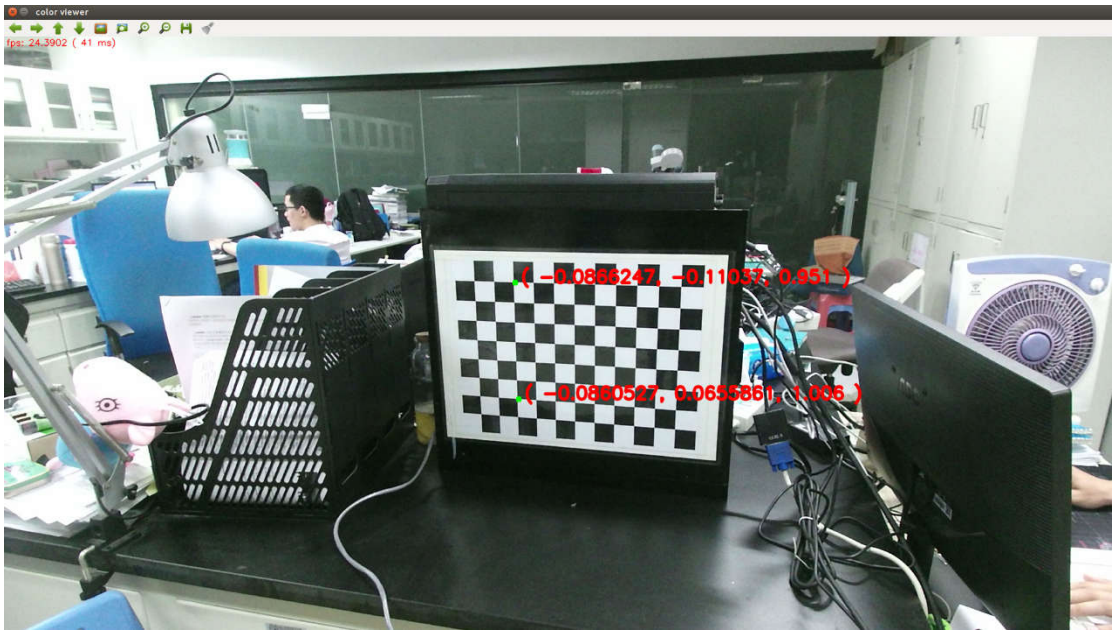

图 2-11 机器人工作空间物体的三维坐标信息

测出来的他们在相机坐标系下的坐标分别是(-0.0866247, -0.11037, 0.951) 和(-0.0860527, 0.0655861, 1.006),则这两个点之间的距离我们可以算出来是 0.18435 m，而我们的标定板每个方格的长度是 0.03m，这两个点之间的相距 6 个方格，所以他们的实际距离是 0.18m 和我们用相机测出来的相差是 4 个 mm，又做了多组测试，计算出来的误差均在 5 个 mm 之内，能满足本课题的要求。

2.2.3 三维重建算法的研究和实现

受相机视野的限制，我们使用单张照片得到的只是物体的一个局部信息，要想得整个物体的信息，我们必须需要移动相机或者移动物体，移动物体通常需要去分割出目标，而移动相机相对来说没那么麻烦，所以本课题是采用移动

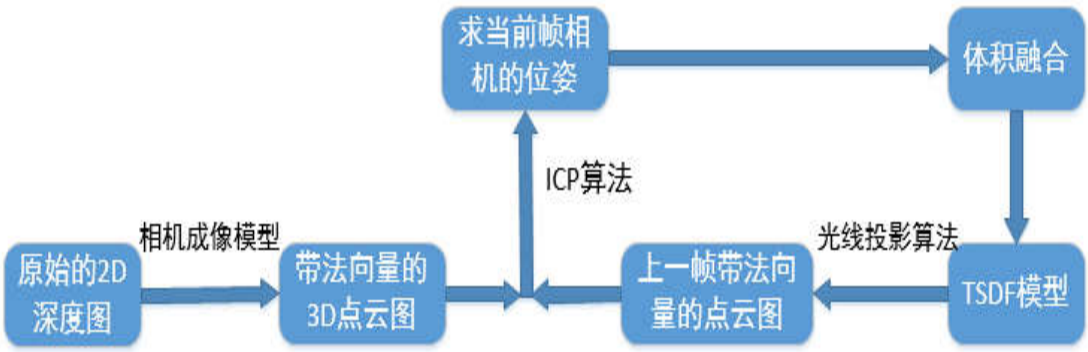

图 2-12 三维重建流程图

相机的方法来对物体进行三维重建。在算法方面采用的是 ICP 算法来求解两帧图像之间的位姿，并结合 GPU 来加速，如图 2-12 是整个三维重建的流程图。整个三维重建流程主要分为 4 部分，分别是：2D 深度图像转换 3D 点云并求得每一点的法向量；利用 ICP 算法求当前帧相机位姿；根据相机的位姿将点云数据融合到场景的 TSDF 模型中；光线投影算法求当前视角下新的带法向量的点云。下面将详细介绍每部分。

(1) 2D 深度图像转换 3D 点云并求得每一点的法向量

这部分主要是将从 Kinect2 获得的深度图像转换为相机坐标系下的空间点云，其原理和上一节障碍物识别的原理类似，都是利用相机成像模型将像素坐标系下的每个点转换为相机坐标系下的三维点，不同的是我们现在使用的是深度相机，还多了一个对每个点求取法向量的步骤。对于法向量的求取，我们采用的策略是在目标点周围找两个点，和目标点一共三个点形成两个向量，两个向量叉乘出来的向量就是我们要求的向量，原理比较简单，所以就不再详细赘述了。如图 2-13 是从 2D 深度图中得到的 3D 点云。

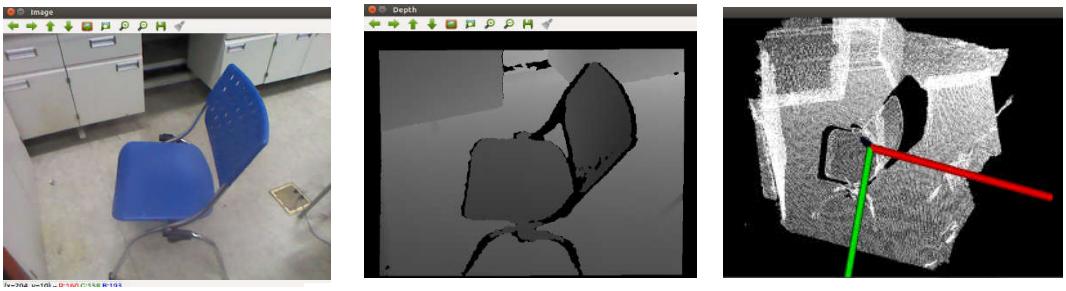

图 2-13 深度图中获得的点云

(2) 利用 ICP 算法求当前帧相机位姿

ICP 算法用于两帧点云配准，主要通过最小化误差，求取最优的位姿变换矩阵，如图 2-14 是它的基本流程 。

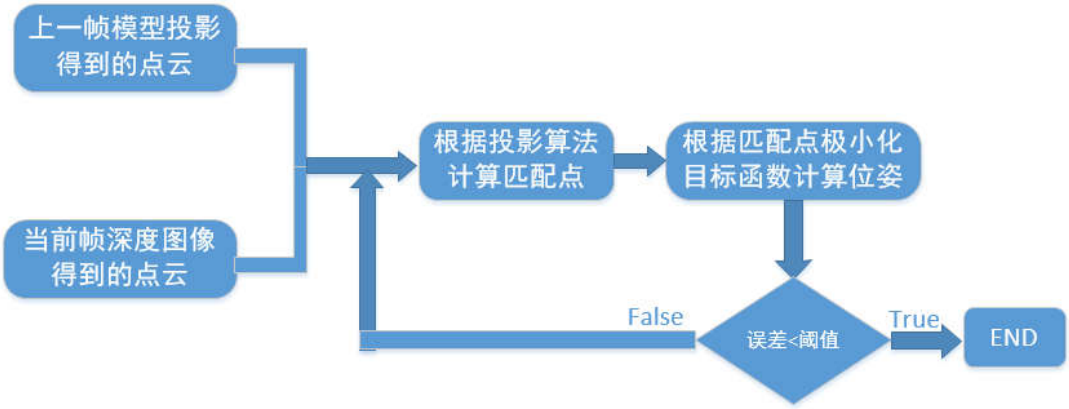

图 2-14 ICP 算法计算相机位姿流程图

当我们通过上一步得到当前帧深度图像的 3D 点云后，假设其中一个点为 P_i，根据投影算法我们可以计算其在模型投影中的对应点像素，如 2-9 所示：

$$p(u,v,1) = K\ M\ P_i \tag{2-9}$$

式中 M 是当前帧点云和从模型投影出来的上一帧之间的相对位姿矩阵，初始化为单位矩阵，K 是相机的内参矩阵，$p(u,v,1)$ 是 P_i 在上一帧中对应的像素点坐标，这样我们通过投影算法就得到了当前帧中一个点 P_i 在上一帧中对用的像素点 $p(u,v,1)$，然后我们通过查询上一帧的 $p(u,v,1)$ 像素对应的点的三维坐标信息得到一个空间点 Q_i，这样我们就找到了 P_i 的对应点 Q_i。

当我们找到一帧图像中所有的 P_i 和其对应点 Q_i 后就可以计算这两幅点云之间的距离误差之和 E：

$$E = \sum_{i=1}^{k} ((RP_i + t - Q_i) \bullet n_i)^2 \tag{2-10}$$

式中 R 和 t 共同组成 2-9 中的 M，n_i 是 Q_i 处的法向量，假如我们的 R 和 t 是没有误差的两帧之间的相对位姿，那我们得到的误差 E 应该是很小的几乎为 0，这样我们就可以通过最小化误差 E 来求出一个最优的 R 和 t 作为我们要求的帧与帧之间的位姿矩阵，求解相对位姿问题就转化成了一个求解最优解问题。

在 2-10 中，R,t 是待求解的 pose，在每次迭代时实际是值很小的 ΔR 和 Δt，而且，关于 R 只有三个自由度，我们用 $r = (r_x, r_y, r_z)$ 表示，平移向量 t 表示为 $t = (t_x, t_y, t_z)$，旋转矩阵 R 是非线性的，而目标函数也是非线性，这里将 R 线性化。当相邻两帧之间位姿变化较小时，可以有以下近似：$\sin(\theta) = \theta, \cos(\theta) = 1$，在三个方向上 R 的旋转角为 $r = (r_x, r_y, r_z)$，对于 $RP_i + t \approx P_i + r \times P_i + t$，并且 $r \times P_i \bullet n_i = r \bullet (P_i \times n_i)$，由以上近似，目标函数可以写为：

$$E \approx \sum_{i=1}^{k} ((P_i - Q_i) \bullet n_i + r \bullet (P_i \times n_i) + t \bullet n_i)^2 \tag{2-11}$$

以上目标函数对 6 个维度位姿参数求导并且令导数为 0，可以得到 Ax+b=0，A 的表达式为：

$$A = \sum_{i=1}^{k} \begin{pmatrix} P_i \\ n_i \end{pmatrix} \begin{pmatrix} (P_i \times n_i)^T & n_i^T \end{pmatrix} \tag{2-12}$$

b 的表达式为：

$$\mathrm{b} = \sum_{i=1}^{k} \begin{pmatrix} (P_i \times n_i)^T & n_i \end{pmatrix}^2 \bullet ((P_i - Q_i)^T \bullet n_i) \tag{2-13}$$

未知参量 x 表达式为 $x = (r_x, r_y, r_z, t_x, t_y, t_z)$，上式对于单点求和的形式可以用 GPU 做并行计算，累加的结果传到 CPU，在 CPU 中求解 $Ax + b = 0$，得到帧与帧之间的位姿变换矩阵。

在这里我们需要说一下 GPU 并行计算的知识，GPU 是图形处理单元的简称，在 linux 系统上我们主要是利用 cuda 来对 GPU 进行编程。在平常的程序中，假

如我们没有利用 GPU，只是在 CPU 上进行程序的测试，程序一般都是在串行的执行，有的 CPU 处理能力高一点可能会开辟多个线程，如果程序不大还可以，但是如果程序比较大的时候，CPU 会运算不过来而出现崩溃。而 GPU 是在物理上就存在多个内核，我们在 GPU 上运行程序可以同时开辟上百个甚至上千个线程，这样一来我们在处理程序时会使时间缩短成百上千倍，从而提高了效率，而对于三维重建，我们必须得要求实时性，所以有必要借用 GPU 的知识来缩短整个程序的周期。

(3)将点云数据融合到场景的 TSDF 模型中

根据上一步计算得到的相机位姿，将点云数据融合到全局 TSDF 模型中。TSDF 模型是在三维重建领域用的比较多的模型，如图 2-15 是它的模型和二维投影示意图。它的主要思想是将待重建的三维场景划分成很多网格，网格中的数值

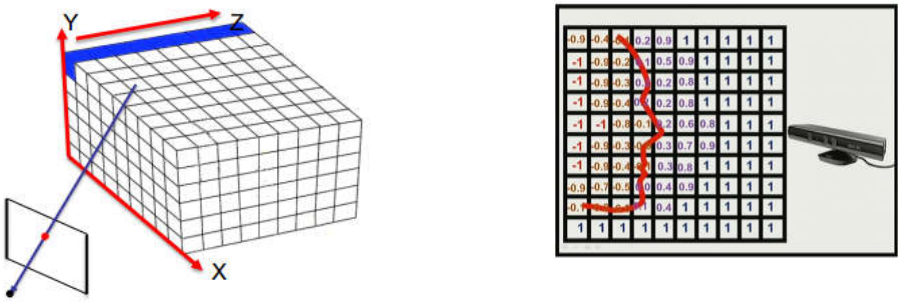

图 2-15　TSDF 模型示意图

代表离重建场景表面的距离，网格中从正到负的穿越点代表重建的表面，我们将每一帧深度图像都融合到这个网格模型中，最后找到的表面就是我们将重建的物体，接下来将详细介绍我们怎么利用每一帧点云数据更新网格模型。

首先我们建立这个网格模型的坐标系，我们把这个网格模型的空间坐标系原点建在初始位姿时相机坐标系的原点，并定义待重建场景的大小和对应的网格个数，考虑到我们是要对机器人工作空间的物体进行重建，所以就将待重建的场景大小定义为了 3×3×3 大小，并设置对应的网格个数为 512×512×512，然后对于

$$
\begin{aligned}
&for\ each\ grid\ v^g\ in\ the\ TSDF\ model\\
&\quad v^c \leftarrow T_i^{-1} v_g\\
&\quad (u,v) \leftarrow A\ v_c\\
&\quad if(\ (u,v)\ \ in\ the\ camera\ filed)\\
&\qquad sdf_i \leftarrow \left\| t_i - v^g \right\| - D_i(u,v)\\
&\qquad if(sdf_i > 0)\\
&\qquad\quad tsdf_i \leftarrow \min(1, sdf_i / \max\_truncation)\\
&\qquad else\\
&\qquad\quad tsdf_i \leftarrow \max(1, sdf_i / \max\_truncation)\\
&\quad tsdf^{avg} \leftarrow \frac{tsdf_i^{-1} w_i^{-1} + tsdf_i w^i}{w_{i-1} + w_i}\\
&end
\end{aligned}
$$

```
TsdfVolume::elem_type* vptr = volume.beg(x, y);
for(int i = 0; i < volume.dims.z; ++i, vc += zstep, vptr = volume.zstep(vptr))
{
    float2 coo = proj(vc);

    //#if defined __CUDA_ARCH__ && __CUDA_ARCH__ >= 300
    // this is actually workaround for kepler. it doesn't return 0.f for texture
    // fetches for out-of-border coordinates even for cudaAddressModeBorder mode
    if (coo.x < 0 || coo.y < 0 || coo.x >= dists_size.x || coo.y >= dists_size.y)
        continue;
    //#endif
    float Dp = tex2D(dists_tex, coo.x, coo.y);


    if(Dp == 0 || vc.z <= 0)
      continue;
    if(Dp>6.85)
      continue;

    float sdf = Dp - __fsqrt_rn(dot(vc, vc)); //Dp - norm(v)

    if (sdf >= -volume.trunc_dist)
    {
        float tsdf = fmin(1.f, sdf * tranc_dist_inv);

        //read and unpack
        int weight_prev;
        float tsdf_prev = unpack_tsdf (gmem::LdCs(vptr), weight_prev);

        float tsdf_new = __fdividef(__fmaf_rn(tsdf_prev, weight_prev, tsdf), weight_prev + 1);   //把当前权重设为1 --
        int weight_new = min (weight_prev + 1, volume.max_weight);          //max_weight=64

        //pack and write
        gmem::StCs(pack_tsdf (tsdf_new, weight_new), vptr);
    }
}
```

图 2-16 更新网格模型未伪代码和程序

更新的每一深度帧我们都执行以下操作：

对于 2-15 中左图中的每一个蓝色的(x,y)长方形单元做并行处理，即使用 GPU 每个线程处理一个长方形单元。对于每个长方形单元从前向后逐个处理单元格，根据上一步中我们得到的当前帧相对于上一帧深度图像的位姿我们可以得到当前帧深度图像相对于第一帧深度图像即全局坐标系的相机的位姿T_i，然后我们利用这个位姿将单元格v^g从全局坐标系转换到相机坐标系下得到v^c，再转换到相机的像素坐标系下得到(u,v)。如果(u,v)在相机视野内，就去寻找该像素对应的深度信息$D_i(u,v)$，然后通过比较$D_i(u,v)$和网格的真实距离去就去更新该单元格；如果不在，则不处理该单元格。图 2-16 是它的伪代码的实现和详细程序：

(4) 光线投影算法求当前视角下新的点云

这部分主要是根据当前帧相机的位姿，从 TSDF 模型中投影出来当前视角下能看到的物体表面的点云信息，其思路主要是：从光心出发，连接光心和像素点形成多条光心，穿过 TSDF 模型的从正到负的穿越点即是当前视角下能看到的物体的表面，如图 2-17 是它的原理示意图。当前视角下光线投影得到的点云图和下一帧深度图像进行配准以求解下一帧相机位姿。图 2-18 是对人体在进行三维重建时，使用 opencv 可视化模块进行可视化的当前视视角下的模型投影。

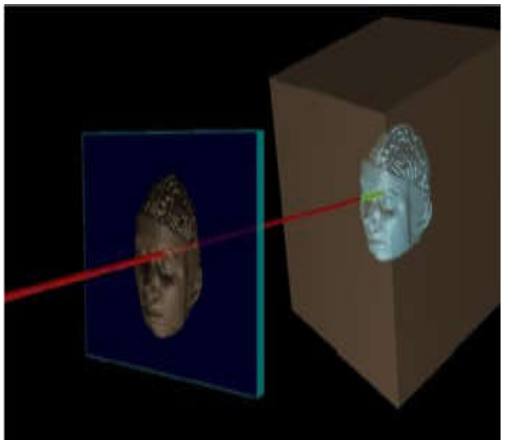

图 2-17 光线投影算法示意图

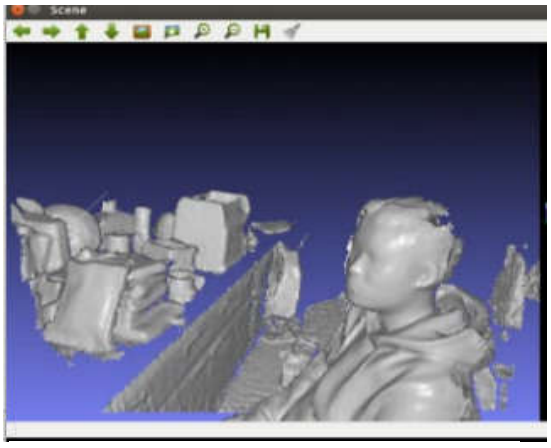

图 2-18 当前视角的模型投影

经过上面这四步之后我们就可以利用 Kinect2 获得的深度数据来重建出物体的表面并获得点云信息，如图 2-19 是对实验室场景三维重建出来的效果图。2-20 是对人体模型重建出来的效果图。

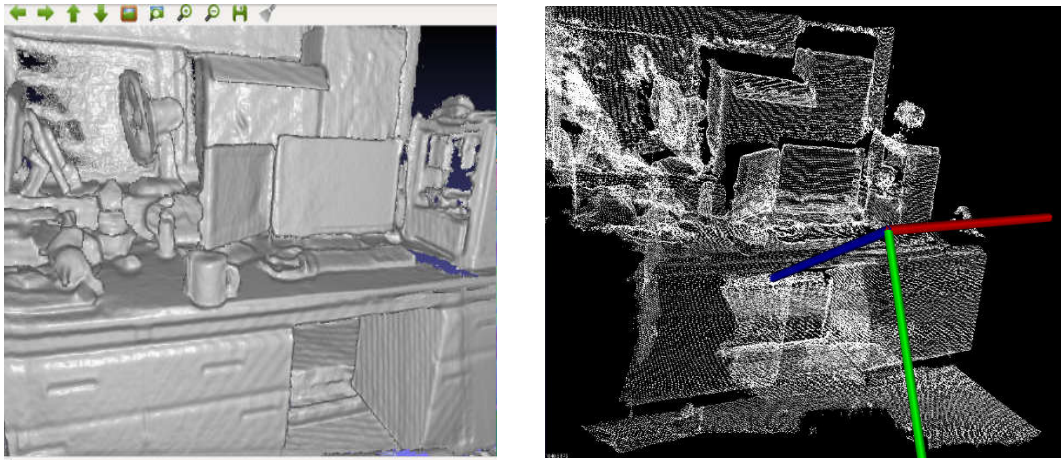

图 2-19 实验室场景重建效果图

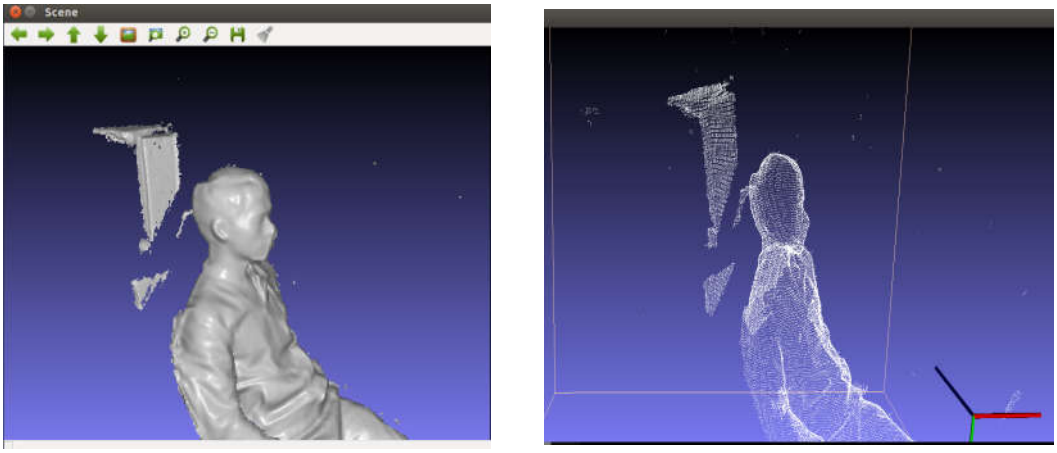

图 2-20 人体重建效果图

2.3 脸部特征点定位算法的研究和简单实现

在介绍本部分的内容之前，首先说明一下为什么要对病人脸部特征点定位算法进行研究。

经过前面的研究我们可以得到病人头部的三维点云模型，并能得到当前帧视角下相机可以看到的病人头部局部的三维坐标信息。但是我们只是得到了头部模型的点云，并不知道点云中哪个是鼻尖，哪些是嘴巴，哪部分是眼睛等。而我们是想得到病人头部的特定部位(主要指脸部特定部位)的一些三维坐标信息，所以这并不是我们想要的。但是好在我们还有 Kinect2 的 RGB 图像可以利用。Kinect2 的 RGB 图像就是普通光学相机的图像，利用机器学习的知识我们可以得到单张人脸图像中人脸的特征点在图像中的像素位置，而 Kinect2 可以实时的对着病人的人脸拍摄，所以我们很容易就可以得到人脸特征点在 RGB 图像中的像素位置。每个 RGB 图像的像素点就对应着三维点云模型中一个点，所以我们便很容易可以得到人脸的特征点在点云中的位置，即脸部特征点在相机坐标系下的三维坐标

信息。所以，对脸部特征点定位算法的研究是很有必要的。然后我们对 Dlib 机器学习库的脸部特征点定位算法进行了初步研究。

Dlib 是一个 C++编写的轻量化工具包，它包含了机器学习算法以及一些用来解决现实复杂问题的工具，可以广泛应用于机器人、嵌入式设备、手机，甚至高性能计算中，在脸部特征点定位方面，它是分两步来完成的，首先是利用 Hog 特征和 SVM 算法检测到图像中的人脸大概区域，然后在这个基础上利用 ERT（ensemble of regression trees）级联回归算法对人脸部的特征点进行定位，下面将详细介绍。

2.3.1 基于 HOG 特征和 SVM 算法的人脸检测

方向梯度直方图（Histogram of Oriented Gradient, HOG）特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。它通过计算和统计图像局部区域的梯度方向直方图来构成特征。HOG 特征结合 SVM 分类器已经被广泛应用于图像识别中，尤其在人脸检测和行人检测中获得了极大的成功。它的主要思想是在一副图像中，局部目标的表象和形状（appearance and shape）能够被梯

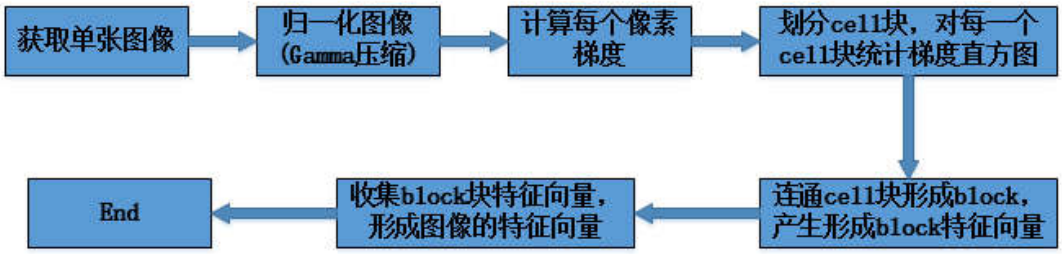

图 2-21　HOG 特征提取流程

度或边缘的方向密度分布很好地描述。图 2-21 是提取图像中 HOG 特征的简单流程图。详细描述如下：

当我们获得一张图像后，为了减少光照因素的影响，首先需要将整个图像进行规范化（归一化）。我们采用 Gamma 压缩的方法对图像进行归一化，压缩处理能够有效地降低图像局部的阴影和光照变化。

然后计算图像每个像素横坐标和纵坐标方向的梯度，并据此计算每个像素的梯度方向和幅值。

第三步是为局部图像区域提供一个编码，同时能够保持对图像中人体对象的姿势和外观的弱敏感性。我们将图像分成若干个 cell，例如每个 cell 为 6*6 个像素。假设我们采用 9 个 bin 的直方图来统计这 6*6 个像素的梯度信息。也就是将 cell 的梯度方向 360 度分成 9 个方向块，如图 2-22 所示。例如：如果某个像素的梯度方向是 20-40 度，直方图第 2 个 bin 的计数就加一，这样，对 cell 内每个像素用梯度方向在直方图中进行加权投影（映射到固定的角度范围），就可以得到这个 cell 的梯度方向直方图了，就是该 cell 对应的 9 维特征向量。像素梯度方向

用到了，那么梯度大小呢？梯度大小就是作为投影的权值的。例如说：这个像素的梯度方向是 20-40 度，然后它的梯度大小是 2 那么直方图第 2 个 bin 的计数就不是加一了，而是加二。

然后我们把各个细胞单元组合成大的、空间上连通的区间（blocks）。这样，一个 block 内所有 cell 的特征向量串联起来便得到该 block 的 HOG 特征。

最后一步就是将检测窗口中所有重叠的块进行 HOG 特征的收集，并将它们结合成最终的特征向量供分类使用，图 2-23 是单张图片的 HOG 描述子示意图。

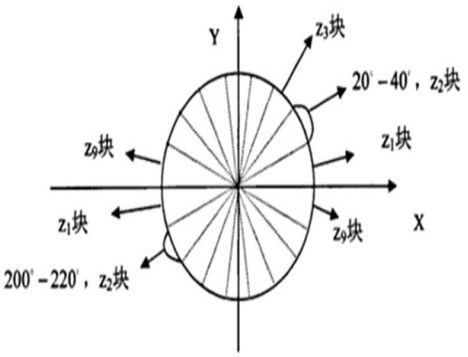

图 2-22 梯度方向划分示意图

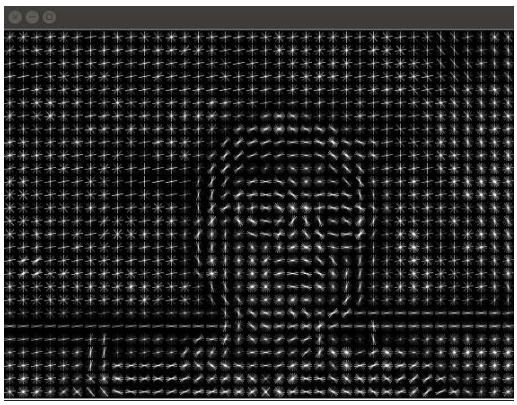

图 2-23 HOG 描述子示意图

有了 HOG 特征我们，然后我们利用经典的 SVM 分类器来判别出，这些窗口的 HOG 特征是否有人脸，有人脸的用矩形框标记起来。HOG 人脸特征及所对

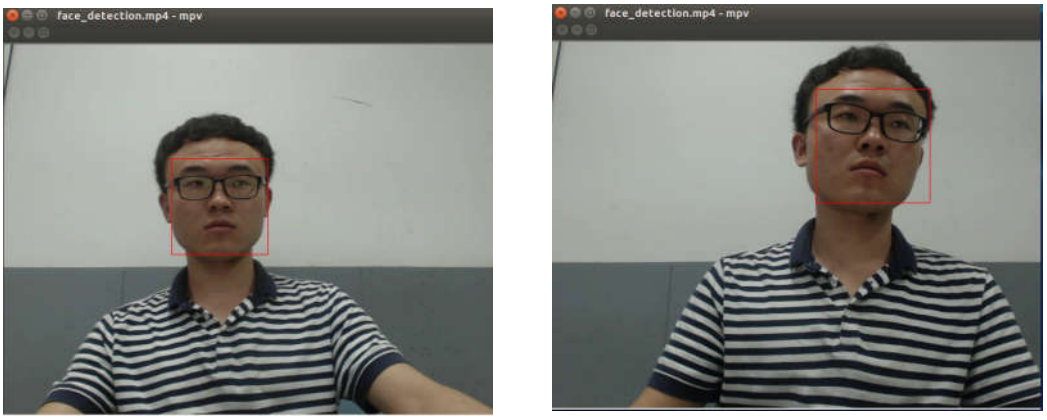

图 2-24 单个人脸检测效果图

应的 SVM 分类器的参数，在 dlib 中已经训练好了，我们需要得到 HOG 特征，然后调用 SVM 即可得到判别结果，图 2-24 是我们对视频中单个人的人脸进行检测的效果图。

2.3.2 基于级联回归算法的人脸特征点定位

人脸特征点定位也叫人脸对齐，就是在上一步已经检测到的人脸的基础上，自动找到人脸上的眼睛鼻子嘴和脸轮廓等标志性特征位置，其最终目的就是在已知的人脸方框上定位其准确的形状。人脸对齐的算法主要分为两大类：基于优化的方法（Optimization-based method）和基于回归的方法（Regression-based method）。

Dlib 库中基于级联回归的算法是属于后者。

级联回归主题思想是把面部特征点定位问题看作是学习一个回归函数 F，以图象 I 作为输入，输出 θ 为特征点的位置（人脸形状）：$\theta = F(I)$。简单的说，级联回归模型可以统一为以下框架：学习多个回归函数 $\{f_1,\ldots,f_{n-1},f_n\}$ 来逼近函数 F。所谓的级联，即当前函数 f_i 的输入依赖于上一级函数 f_{i-1} 的输出 θ_{i-1}，而每一个 f_i 的学习目标都是逼近特征点的真实位置 θ，θ_0 为初始形状。通常情况，f_i 不是直接回归真实位置 θ，而回归当前形状 θ_{i-1} 与真实位置 θ 之间的差：$\triangle\theta_i = \theta_i - \theta_{i-1}$。图 2-25 是我们直接利用 Dlib 库对脸部特征点进行定位的效果图。

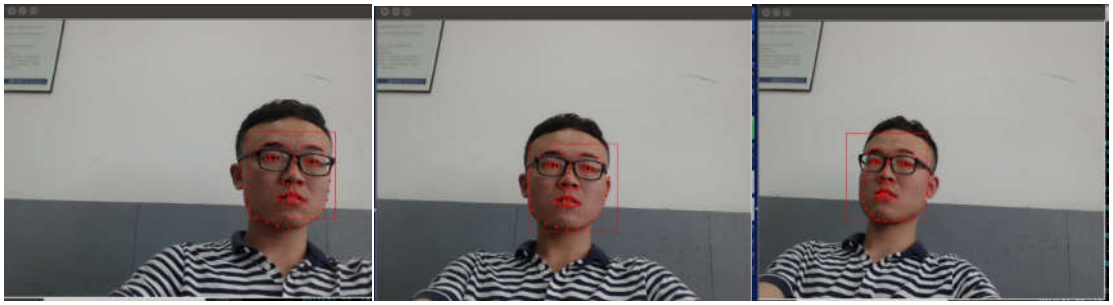

图 2-25 特征点定位效果图

3 后期拟完成的研究工作及进度安排

后期需要完成的工作主要是结合实验改进一下现在的算法，提高算法的鲁棒性和精度，撰写小论文和大论文，具体安排如下所示：

时间	课题进展与预期目标
2018.03.15——2018.03.28	继续学习脸部特征点算法并进行实验
2018.04.01——2018.04.30	对三维重建和脸部特征点定位算法进行改进，并结合机械臂进行联合实验，确定整体方案具体实现细节
2018.05.01——2018.06.21	将实验期间取得的相关成果进行整理，并发表一篇学术小论文
2018.06.22——2018.09.31	整理研究成果，撰写、修改、完善硕士学位论文
2018.10.01——2018.12.20	准备硕士学位论文答辩

4 存在问题及解决方案

4.1 存在问题与困难

(1) 数学功底不够，有些算法方面的理解不是很到位；

(2)算法精度不是很高，急需将提高算法的精度；
(3)现在写的程序都是基于 CmakeList 写的，后期可能还需要移植到 QT 界面中，这个过程中可能会遇到一些问题和麻烦。

4.2 解决方案

针对目前存在的问题和困难，提出以下解决方案：
(1)认真学习一下算法所需数学知识，实在解决不了去咨询专业老师；
(2)多查阅一些其他算法和现在最新的算法，提高整个工程的精度；
(3)学习一下程序移植方面的知识，在实践中解决程序的 bug。

5 如期完成全部论文工作的可能性

现在本课题已经完成了对机器人工作空间障碍物信息的获取，对头部的三维重建算法的研究和实现，以及对头部定位算法的研究并做了简单的实现。后期主要是想结合实验来验证算法的鲁棒性，并对其精度进行提高，对整个代码进行优化，使其适用于本工程，并确定整个系统的具体实现细节，最终做出一套切实可行的针对于本课题的医疗导航系统。按着目前课题的进展情况，是能够按时完成硕士毕业论文的。