1 程序设计

1.1分析

①上位机App需要完成蓝牙连接的功能，确保能够与下位机实现通信。

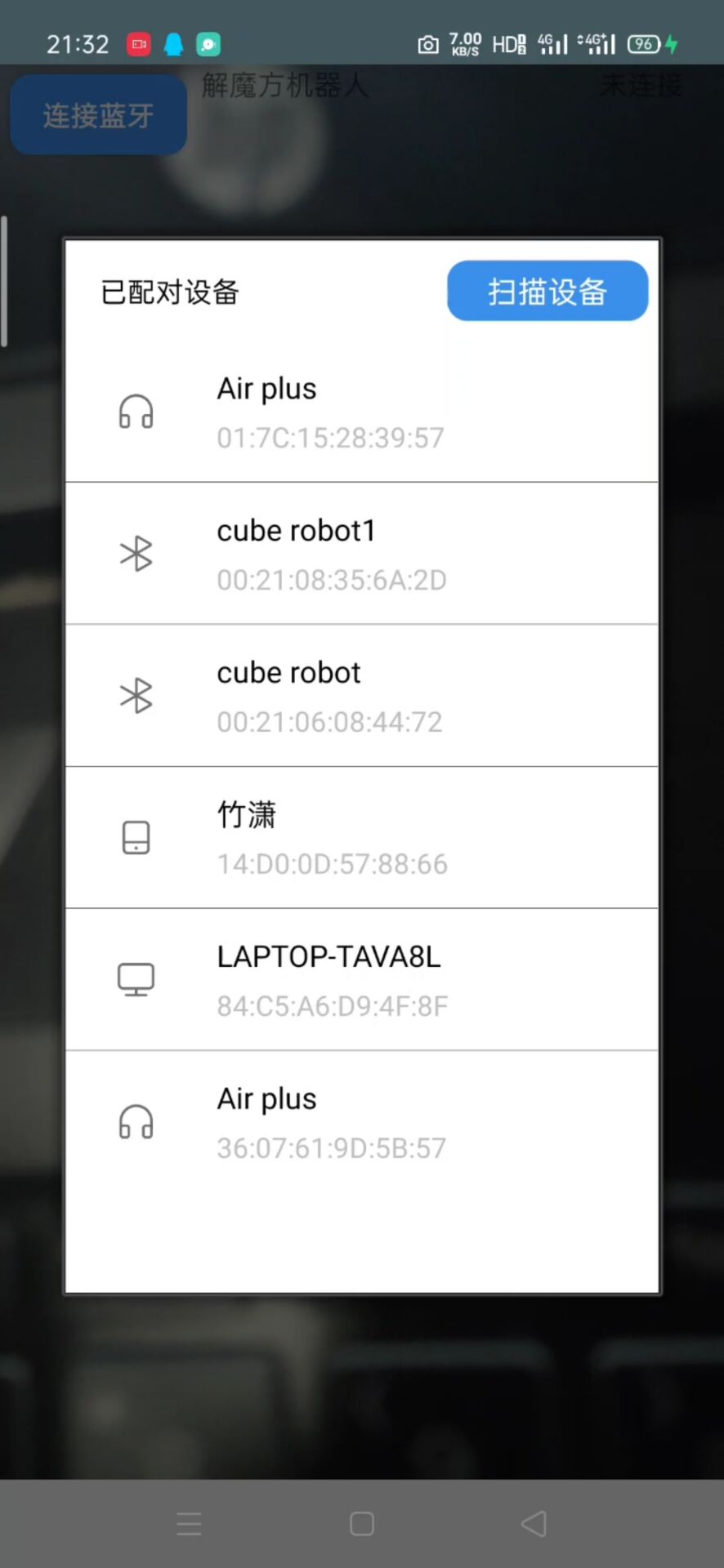
②魔方还原最重要的是颜色识别是否成功，因此需要优化算法来提高识别成功率。

③魔方还原需要还原算法，选择一个合适的算法至关重要

1.2设计

程序主要包括四部分，APP UI界面，蓝牙通信模块，Kociemba算法还原，颜色识别模块。

1.2.1UI界面



1.2.2蓝牙连接

①蓝牙相关的应用程序接口：

|  |  |
| --- | --- |
| BluetoothAdapter | 本地蓝牙设备的适配类，全部的蓝牙操作都要通过该类完毕 |
| BluetoothDevice | 蓝牙设备类。代表了蓝牙通讯过程中的远端设备 |
| BluetoothServerSocket | 蓝牙设备服务端，类似ServerSocket |
| BluetoothSocket | 蓝牙设备client，类似Socket |

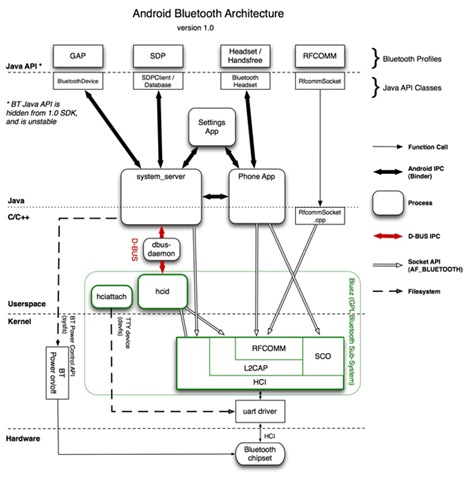


图5 蓝牙代码框架图

②传输数据流程

（1） uart口取得蓝牙模块的数据；

（2） uart口通过ldisc传给hci\_uart；

（3） hci\_uart传给在其上的h4；

（4） h4传给hci层；

（5） hci层传给l2cap层；

（6） l2cap层再传给rfcomm。

③使用BluetoothAdapter：蓝牙适配器实现连接

1.2.3 Kociemba算法还原

Kociemba算法是一个使用较短时间和较少次数还原魔方的算法。Kociemba算法使用的搜索算法是 IDA\* 算法，也就是迭代加深搜索算法，而且两个阶段使用的都是 IDA\* 算法。一般的，广搜是能够找到最短路径的，深搜不能，IDA\*虽然使用的是深搜，但是它有深度限制，也是能够找到最短路径。比如说我搜索深度定为1，没搜到，再定为2重新搜索，依次下去，就能够得到最短路径。但是Kociemba的二阶段算法一次搜索可能得不到最优解，因此需要优化算法，我们可以加深第一阶段的搜索深度，虽然第一阶段的路径变长了，但是第二阶段可能找到更短路径，这样它们合起来可能更优。这样一步步下去，是总能找到最优解的，也就是说Kociemba算法一次搜索可能得不到最优解，但是只要给它时间，是一定能够找到最优解的。但是对于Kociemba并不是拿来寻找最优解的，目的是短时间内找到一组比较优的解。

1.2.4颜色识别模块

在这一部分，为提高识别成功率，主要使用了KNN算法进行颜色分类。

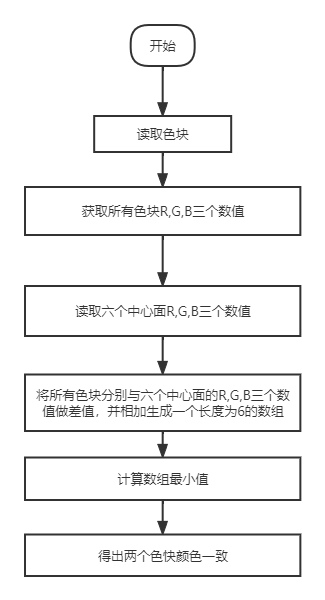


图4 KNN算法流程图

基于对图4算法的了解可知,我们只需将魔方颜色分为与面数相同的类别而不需要判断其具体颜色。对于三阶魔方，只有54个样本，且各类颜色间差异显著，样本十分平衡，所以在选定KNN算法下较易达到颜色识别的目的。

1.3编码

①程序编码选择Java语言，应用熟练，易于维护

②程序风格文档化，使用注释，数据说明规范，说明语句有序

③程序代码易测试、调试，易维护修改

1.4 测试

测试主要包括拍照是否成功，颜色识别是否正确这两个问题。因此上位机主要在这两方面进行测试。

（1）拍照测试

在拍照方面，经过多次测试，拍照没有问题，但信号不稳定时，即蓝牙传输出现问题时，手机接收指令会出错，拍照可能拍不够六张。

|  |  |  |
| --- | --- | --- |
| 测试条件 | 测试次数 | 平均成功率 |
| 相机拍照 | 50 | 99.00 |

表 1 拍照测试

（2）识别测试

在颜色识别部分，可以看到在亮度适宜的正常环境下，颜色识别正确率达到了100%，在黑暗和光照条件下均出现了识别不成功的现象，但是准确率均在90%以上。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试类型 | 测试条件 | 测试次数 | 平均准确率 |
| 颜色识别环境 | 强光照环境 | 100 | 93.00 |
| 颜色识别环境 | 较暗环境 | 100 | 94.00 |
| 颜色识别环境 | 正常环境 | 100 | 100.00 |

表 2 识别测试