DobotDemoForMagician

说明文档

TN01010101 V1.0.0 Date: 2017/03/30

深圳市越疆科技有限公司

修订历史

版本	日期	原因
V1.0.0	2017/03/30	创建文档

目 录

1. 序言.....1
2. 通用桌面系统.....2
2.1 Dobot 动态链接库.....2
2.1.1 编译.....2
2.1.2 使用.....2
2.2 Java Demo.....2
2.2.1 工程说明.....2
2.2.2 Java API.....3
2.2.3 代码说明.....3
2.3 MFC Demo.....6
2.3.1 工程说明.....6
2.3.2 代码说明.....6
2.4 C# Demo.....9
2.4.1 工程说明.....9
2.4.1 C# API.....10
2.4.2 代码说明.....10
2.5 VB Demo.....13
2.5.1 工程说明.....13
2.5.1 VB API.....13
2.5.2 代码说明.....13
2.6 Qt Demo.....14
2.6.1 工程说明.....14
2.6.2 代码说明.....15
2.7 Multi-Control Demo.....18
2.7.1 工程说明.....18
2.7.2 代码说明.....18
2.8 Python Demo.....21
2.8.1 工程说明.....21
2.8.2 Python API.....21
2.8.3 代码说明.....21
3. 嵌入式系统.....24
3.1 注意事项.....24
3.2 STM32 Demo.....24
3.2.1 硬件说明.....24
3.2.2 工程说明.....25
3.2.3 代码说明.....25
3.3 Arduino Demo.....27
3.3.1 硬件说明.....27
3.3.2 工程说明.....28
3.3.3 代码说明.....29
3.4 iOS Demo.....32

3.4.1 工程说明.....32
3.4.2 代码说明.....33
3.5 Android Demo.....35
3.5.1 工程说明.....35
3.5.2 代码说明.....36

1. 序言

本说明文档旨为协助 Dobot Magician 二次开发者熟悉 Dobot 常用 API 和快速搭建开发环境。分别对多种语言，多种框架，多种系统的二次开发环境搭建和 Demo 代码进行说明。

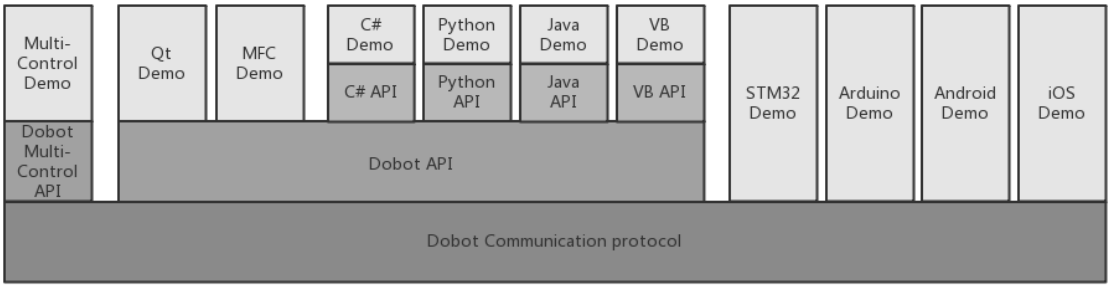

图 1 Demo 结构

2. 通用桌面系统

对于通用桌面系统，我们已经向 Dobot 二次开发者提供了动态链接库。二次开发者只要直接调用动态链接库即可以控制 Dobot，而不需进行通讯协议相关的开发工作。

2.1 Dobot 动态链接库

在 DobotDll 文件夹下可以找到动态链接库的源码和预编译文件。其中，源码使用 Qt 5.6 开发环境。另外，文件夹中可以找到 Windows32 位、Windows64 位、Linux 和 Mac 四种平台的对应动态链接库。

2.1.1 编译

https://download.qt.io/archive/qt/5.6/5.6.0/

开发者请在上述网站中下载适合自己系统的 Qt 版本并安装。

另外要注意的是，如果项目需要配合 PyQt 库，请使用带 MSVC 编译器的的 Qt 版本并用 MSVC 编译 Dobot 动态链接库。

2.1.2 使用

- Windows 系统中，将动态链接库所在目录添加到系统 Path 环境变量。
- Linux 系统中，在~/.bash_profile 文件最后添加下列语句，重启电脑。

程序 1 Linux 添加环境变量语句

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:DOBOT_LIB_PATH
```

- Mac 系统中，在~/.bash_profile 文件最后添加下列语句，重启电脑。

程序 2 mac 添加环境变量语句

```
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH: DOBOT_LIB_PATH
```

2.2 Java Demo

2.2.1 工程说明

配置环境：导入 jna,用于 Java 直接访问本地动态库。

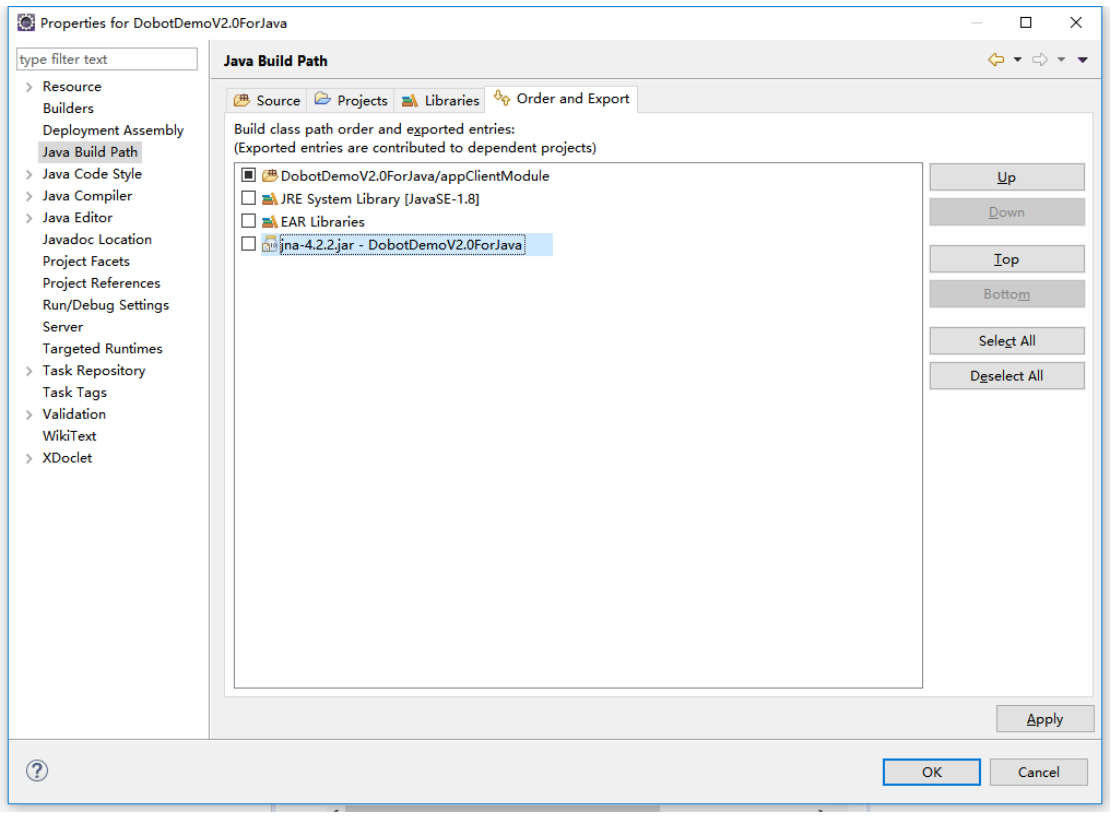

图 2 环境配置

2.2.2 Java API

DobotDll.java 是 Dobot 的 Java API 文件，将 Dobot 动态链接进行了二次封装。其中，加载动态链接库的代码如下：

程序 3 加载动态链接库

```
DobotDll instance = (DobotDll) Native.loadLibrary("DobotDll",  DobotDll.class);
```

代码中“DobotDll”是 Windows 系统中动态链接库的文件名，用户需要根据不同的系统环境修改正确的动态链接库文件名。

2.2.3 代码说明

1 连接机械臂，并且判断是否连接成功；

程序 4 连接机械臂，并且判断是否连接成功

```
IntByReference ib = new IntByReference();
DobotResult ret = DobotResult.values()[
                                       DobotDll.instance.ConnectDobot(
                                       (char)0,  115200)
                                      ];
// 开始连接
if ( ret == DobotResult.DobotConnect_NotFound ||
     ret == DobotResult.DobotConnect_Occupied )
{
```

```
    Msg("Connect error, code:" + ret.name());
    return;
}
Msg("connect success code:" + ret.name());
```

2 设置夹具中心距长度；

程序 5 设置夹具中心距长度

```
EndEffectorParams endEffectorParams = new EndEffectorParams();
endEffectorParams.xBias = 71.6f;
endEffectorParams.yBias = 0;
endEffectorParams.zBias = 0;
DobotDll.instance.SetEndEffectorParams(endEffectorParams, false, ib);
```

3 设置关节点动速度；

程序 6 设置关节速度

```
JOGJointParams jogJointParams = new JOGJointParams();
for(int i = 0; i < 4; i++) {
    jogJointParams.velocity[i] = 200;
    jogJointParams.acceleration[i] = 200;
}
DobotDll.instance.SetJOGJointParams(jogJointParams, false, ib);
```

4 设置坐标轴点动速度；

程序 7 设置坐标轴点动速度

```
JOGCoordinateParams jogCoordinateParams = new JOGCoordinateParams();
for(int i = 0; i < 4; i++) {
    jogCoordinateParams.velocity[i] = 200;
    jogCoordinateParams.acceleration[i] = 200;
}
DobotDll.instance.SetJOGCoordinateParams(jogCoordinateParams, false, ib);
```

5 设置示教再现速度与加速度的百分比，如无设置，默认为 50；

程序 8 设置示教再现速度与加速度的百分比

```
JOGCommonParams jogCommonParams = new JOGCommonParams();
jogCommonParams.velocityRatio = 50;
jogCommonParams.accelerationRatio = 50;
DobotDll.instance.SetJOGCommonParams(jogCommonParams, false, ib);
```

6 设置示教再现的关节运动参数；

程序 9 设置示教再现的关节运动参数

```
PTPJointParams ptpJointParams = new PTPJointParams();
for(int i = 0; i < 4; i++) {
    ptpJointParams.velocity[i] = 200;
    ptpJointParams.acceleration[i] = 200;
}
```

```
DobotDll.instance.SetPTPJointParams(ptpJointParams, false, ib);
```

7 设置示教再现的坐标系运动参数；

程序 10 设置示教再现的坐标系运动参数

```
PTPCoordinateParams ptpCoordinateParams = new PTPCoordinateParams();
ptpCoordinateParams.xyzVelocity = 200;
ptpCoordinateParams.xyzAcceleration = 200;
ptpCoordinateParams.rVelocity = 200;
ptpCoordinateParams.rAcceleration = 200;
DobotDll.instance.SetPTPCoordinateParams(ptpCoordinateParams, false, ib);
```

8 设置示教再现中 JUMP 运动模式时的参数；

程序 11 设置示教再现中 JUMP 运动模式时的参数

```
PTPJumpParams ptpJumpParams = new PTPJumpParams();
ptpJumpParams.jumpHeight = 20;
ptpJumpParams.zLimit = 180;
DobotDll.instance.SetPTPJumpParams(ptpJumpParams, false, ib);
```

9 获取机械臂的姿态信息；

程序 12 获取机械臂的姿态信息

```
Pose pose = new Pose();
DobotDll.instance.GetPose(pose);
Msg( "joint1Angle="+pose.jointAngle[0]+"  "
    + "joint2Angle="+pose.jointAngle[1]+"  "
    + "joint3Angle="+pose.jointAngle[2]+"  "
    + "joint4Angle="+pose.jointAngle[3]+"  "
    + "x="+pose.x+"  "
    + "y="+pose.y+"  "
    + "z="+pose.z+"  "
    + "r="+pose.r+"  ");
```

10 示教再现（两点之间来回摆动）。

程序 13 示教再现

```
while(true)
{
    try{
        PTPCmd ptpCmd = new PTPCmd();
        ptpCmd.ptpMode = 0;
        ptpCmd.x = 260;
        ptpCmd.y = 0;
        ptpCmd.z = 50;
        ptpCmd.r = 0;
        DobotDll.instance.SetPTPCmd(ptpCmd, true, ib);
        //Thread.sleep(200);
```

```
        ptpCmd.ptpMode = 0;
        ptpCmd.x = 220;
        ptpCmd.y = 0;
        ptpCmd.z = 80;
        ptpCmd.r = 0;
        DobotDll.instance.SetPTPCmd(ptpCmd, true, ib);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

2.3 MFC Demo

2.3.1 工程说明

三个功能模块依次为机械臂点动（JOG），获取姿态信息（Pose），示教再现（PTP）。

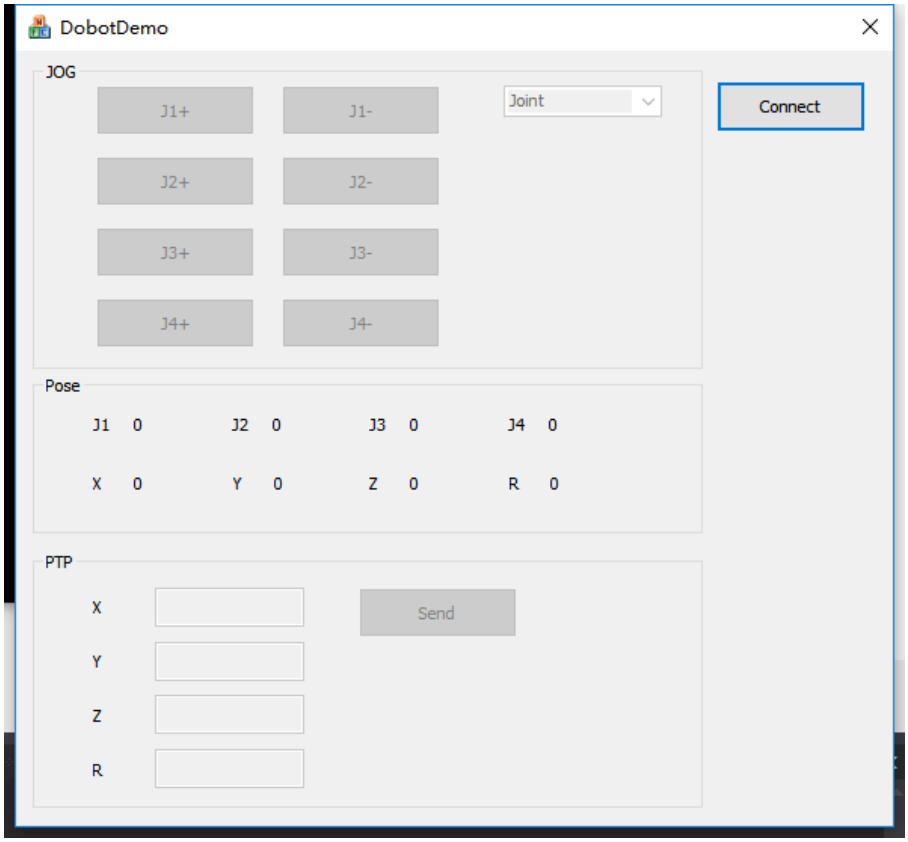

图 3 MFC Demo 界面

2.3.2 代码说明

1 连接机械臂，并且判断机械臂是否成功；

程序 14 连接机械臂

```
if (!m_bConnectStatus) {
    if (ConnectDobot(0, 115200) != DobotConnect_NoError) {
        ::AfxMessageBox(L"Cannot connect Dobot!");
```

```
        return;
    }
```

2 获取机械臂序列号；

程序 15 获取机械臂序列号

```
char deviceSN[64];
GetDeviceSN(deviceSN, sizeof(deviceSN));
```

3 获取机械臂名字；

程序 16 获取机械臂名字

```
char deviceName[64];
GetDeviceName(deviceName, sizeof(deviceName));
```

4 获取机械臂版本信息；

程序 17 获取机械臂版本信息

```
uint8_t majorVersion, minorVersion, revision;
GetDeviceVersion(&majorVersion, &minorVersion, &revision);
```

5 设置夹具中心距长度；

程序 18 设置夹具中心长度

```
EndEffectorParams endEffectorParams;
memset(&endEffectorParams, 0, sizeof(EndEffectorParams));
endEffectorParams.xBias = 71.6f;
SetEndEffectorParams(&endEffectorParams, false, NULL);
```

6 设置关节点动参数；

程序 19 设置关节点动参数

```
JOGJointParams jogJointParams;
for (uint32_t i = 0; i < 4; i++) {
    jogJointParams.velocity[i] = 200;
    jogJointParams.acceleration[i] = 200;
}
SetJOGJointParams(&jogJointParams, false, NULL);
```

7 设置坐标轴点动参数；

程序 20 设置坐标轴点动参数

```
JOGCoordinateParams jogCoordinateParams;
for (uint32_t i = 0; i < 4; i++) {
    jogCoordinateParams.velocity[i] = 200;
    jogCoordinateParams.acceleration[i] = 200;
}
SetJOGCoordinateParams(&jogCoordinateParams, false, NULL);
```

8 设置示教再现中的速度与加速度百分比，如无设置，默认为 50；

程序 21 设置示教参数

```
JOGCommonParams jogCommonParams;
jogCommonParams.velocityRatio = 50;
jogCommonParams.accelerationRatio = 50;
SetJOGCommonParams(&jogCommonParams, false, NULL);
```

9 设置示教再现的关节运动参数；

程序 22 设置示教关节运动参数

```
PTPJointParams ptpJointParams;
for (uint32_t i = 0; i < 4; i++) {
    ptpJointParams.velocity[i] = 200;
    ptpJointParams.acceleration[i] = 200;
}
SetPTPJointParams(&ptpJointParams, false, NULL);
```

10 设置示教再现的坐标轴运动参数；

程序 23 设置示教轴运动参数

```
PTPCoordinateParams ptpCoordinateParams;
ptpCoordinateParams.xyzVelocity = 200;
ptpCoordinateParams.xyzAcceleration = 200;
ptpCoordinateParams.rVelocity = 200;
ptpCoordinateParams.rAcceleration = 200;
SetPTPCoordinateParams(&ptpCoordinateParams, false, NULL);
```

11 设置示教再现中的 Jump 运动模式参数；

程序 24 设置示教 Jump 参数

```
PTPJumpParams ptpJumpParams;
ptpJumpParams.jumpHeight = 10;
ptpJumpParams.zLimit = 150;
SetPTPJumpParams(&ptpJumpParams, false, NULL);
```

12 机械臂点动（JOG）；

程序 25 机械点动

```
JOGCmd jogCmd;
jogCmd.isJoint = m_JOGMode.GetCurSel() == 0;
jogCmd.cmd = i + 1;
SetJOGCmd(&jogCmd, false, NULL);
```

13 获取机械臂姿态信息（Pose）；

程序 26 获取状态信息

```
Pose pose;
if (GetPose(&pose) != DobotCommunicate_NoError) {
    break;
}
CString str;
```

```
str.Format(L"%1.3f", pose.jointAngle[0]);
m_StaticJ1.SetWindowText(str);
str.Format(L"%1.3f", pose.jointAngle[1]);
m_StaticJ2.SetWindowText(str);
str.Format(L"%1.3f", pose.jointAngle[2]);
m_StaticJ3.SetWindowText(str);
str.Format(L"%1.3f", pose.jointAngle[3]);
m_StaticJ4.SetWindowText(str);
str.Format(L"%1.3f", pose.x);
m_StaticX.SetWindowText(str);
str.Format(L"%1.3f", pose.y);
m_StaticY.SetWindowText(str);
str.Format(L"%1.3f", pose.z);
m_StaticZ.SetWindowText(str);
str.Format(L"%1.3f", pose.r);
m_StaticR.SetWindowText(str);
```

14 机械臂示教再现运动（PTP）。

程序 27 示教再现

```
PTPCmd ptpCmd;
ptpCmd.ptpMode = mode;
ptpCmd.x = x;
ptpCmd.y = y;
ptpCmd.z = z;
ptpCmd.r = r;
uint64_t queuedCmdIndex;
do {
    int result = SetPTPCmd(&ptpCmd, true, &queuedCmdIndex);
    if (result == DobotCommunicate_NoError) {
        break;
    }
} while (1);
```

2.4 C# Demo

2.4.1 工程说明

三个功能模块依次为机械臂点动（JOG），获取姿态信息（Pose），示教再现（Playback）。

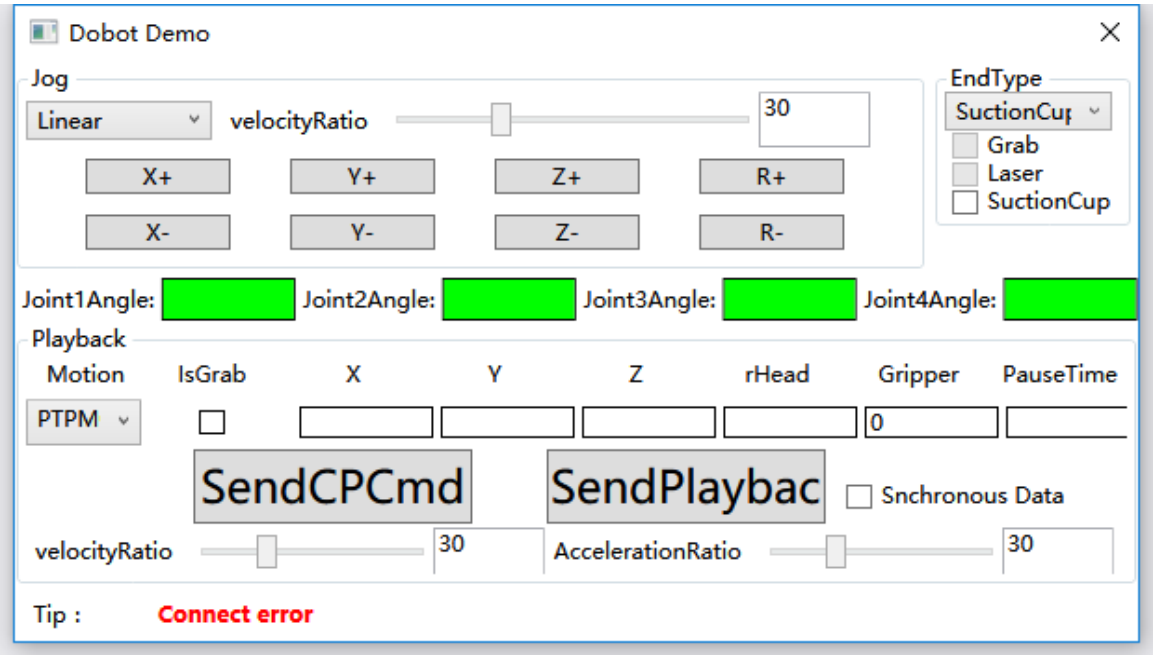

图 4 C# Demo 界面

2.4.1 C# API

DobotDll.cs 与 DobotDllType.cs 是 Dobot 动态链接库的 C# API 文件，将 Dobot 动态链接库进行了二次封装。例如，连接函数（ConnectDobot）：

程序 28 连接函数

```
[DllImport("DobotDll.dll",
          EntryPoint = "ConnectDobot",
          CallingConvention = CallingConvention.Cdecl
)]
public static extern int ConnectDobot(string portName,
                                      int baudrate,
                                      StringBuilder fwType,
                                      StringBuilder version);
```

代码中的“DobotDll.dll”是 Windows 系统中的动态链接库文件名，用户需要根据自身系统修改正确的文件名。

2.4.2 代码说明

1 连接机械臂，并判断是否连接成功；

程序 29 连接机械臂

```
int ret = DobotDll.ConnectDobot("", 115200, fwType, version);
if (ret != (int)DobotConnect.DobotConnect_NoError)
{
    Msg("Connect error", MsgInfoType.Error);
    return;
}
```

2 设置机械臂关节点动参数；

程序 30 设置关节点动参数

```
JOGJointParams jsParam;
jsParam.velocity = new float[] { 200, 200, 200, 200 };
jsParam.acceleration = new float[] { 200, 200, 200, 200 };
DobotDll.SetJOGJointParams(ref jsParam, false, ref cmdIndex);
```

3 设置机械臂点动速度与加速度的百分比；

程序 31 设置点动参数

```
JOGCommonParams jdParam;
jdParam.velocityRatio = 100;
jdParam.accelerationRatio = 100;
DobotDll.SetJOGCommonParams(ref jdParam, false, ref cmdIndex);
```

4 设置示教再现的关节运动参数；

程序 32 设置示教关节运动参数

```
PTPJointParams pbsParam;
pbsParam.velocity = new float[] { 200, 200, 200, 200 };
pbsParam.acceleration = new float[] { 200, 200, 200, 200 };
DobotDll.SetPTPJointParams(ref pbsParam, false, ref cmdIndex);
```

5 设置示教再现的坐标轴运动参数；

程序 33 设置示教坐标运动参数

```
PTPCoordinateParams cpbsParam;
cpbsParam.xyzVelocity = 100;
cpbsParam.xyzAcceleration = 100;
cpbsParam.rVelocity = 100;
cpbsParam.rAcceleration = 100;
DobotDll.SetPTPCoordinateParams(ref cpbsParam, false, ref cmdIndex);
```

6 设置示教再现的 Jump 运动模式参数；

程序 34 设置示教 Jump 运动参数

```
PTPJumpParams pjp;
pjp.jumpHeight = 20;
pjp.zLimit = 100;
DobotDll.SetPTPJumpParams(ref pjp, false, ref cmdIndex);
```

7 设置示教再现速度与加速度的百分比，默认为 50；

程序 35 设置示教参数

```
PTPCommonParams pbdParam;
pbdParam.velocityRatio = 30;
pbdParam.accelerationRatio = 30;
DobotDll.SetPTPCommonParams(ref pbdParam, false, ref cmdIndex);
```

8 机械臂点动（JOG）；

程序 36 点动

```
currentCmd.isJoint = isJoint;
currentCmd.cmd = e.ButtonState == MouseButtonState.Pressed ?
                    (byte)JogCmdType.JogAPPressed :
                    (byte)JogCmdType.JogIdle;
DobotDll.SetJOGCmd(ref currentCmd, false, ref cmdIndex);
```

9 获取机械臂姿态信息（Pose）；

程序 37 获取位姿

```
DobotDll.GetPose(ref pose);
this.Dispatcher.BeginInvoke((Action)delegate()
{
    tbJoint1Angle.Text = pose.jointAngle[0].ToString();
    tbJoint2Angle.Text = pose.jointAngle[1].ToString();
    tbJoint3Angle.Text = pose.jointAngle[2].ToString();
    tbJoint4Angle.Text = pose.jointAngle[3].ToString();
    if (sync.IsChecked == true)
    {
        X.Text = pose.x.ToString();
        Y.Text = pose.y.ToString();
        Z.Text = pose.z.ToString();
        rHead.Text = pose.rHead.ToString();
        pauseTime.Text = "0";
    }
});
```

10 教再现运动（PTP）；

程序 38 示教再现

```
pdbCmd.ptpMode = style;
pdbCmd.x = x;
pdbCmd.y = y;
pdbCmd.z = z;
pdbCmd.rHead = r;
while(true)
{
    int ret = DobotDll.SetPTPCmd(ref pdbCmd, true, ref cmdIndex);
    if (ret == 0)
        break;
}
```

11 获取机械臂报警信息（Alarm）。

程序 39 获取报警信息

```
int ret;
byte[] alarmsState = new byte[32];
UInt32 len = 32;
ret = DobotDll.GetAlarmsState(alarmsState,
                                    ref len,
                                    alarmsState.Length);
```

2.5 VB Demo

2.5.1 工程说明

简单展示了连接机械臂后，用示教再现方式分别运动了两个坐标点（PTP1 与 PTP2）。

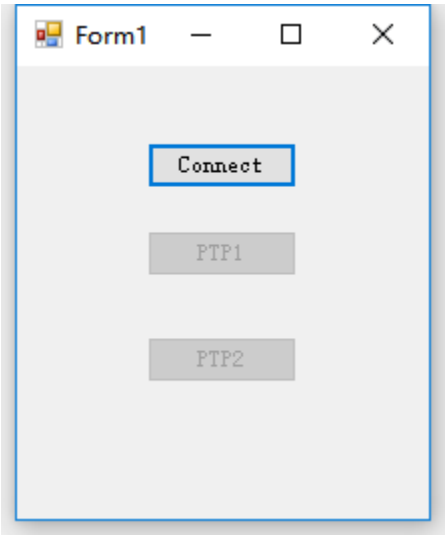

图 5 VB Demo 界面

2.5.1 VB API

DobotDll.vb 与 DobotDllType.vb 是 Dobot 的 VB API 文件，将 Dobot 动态链接库进行了二次封装。例如，连接机械臂函数（ConnectDobot）：

程序 40 连接函数

```
Class DobotDll
<DllImport("DobotDll.dll", CallingConvention:=CallingConvention.Cdecl)>
    Public Shared Function ConnectDobot(ByVal portName As String,
                                        ByVal baudrate As Int32) As Int32
End Function
End Class
```

代码中的“DobotDll.dll”是 Windows 系统中的动态链接库文件名，用户需要根据自身系统环境修改为正确的文件名。

2.5.2 代码说明

1 连接机械臂；

程序 41 连接

```
result = DobotDll.ConnectDobot("", 115200)
```

```
If result <> 0 Then
    MsgBox("Could not find Dobot or Dobot is occupied!")
    Return
End If
```

2 获取机械臂信息；

程序 42 获取信息

```
DobotDll.GetDeviceName(deviceName, 64)
```

3 机械臂示教再现运动（PTP）；

程序 43 示教再现

```
Dim ptpCmd As PTPCmd
ptpCmd.ptpMode = ptpMode
ptpCmd.x = x
ptpCmd.y = y
ptpCmd.z = z
ptpCmd.r = r
Dim result As Int32
Dim queuedCmdIndex As UInt64
Dim currentQueuedCmdIndex As UInt64
While True
result = DobotDll.SetPTPCmd(ptpCmd, True, queuedCmdIndex)
If result = DobotCommunicate.DobotCommunicate_NoError Then
    Exit While
End If
End While
```

4 机械臂获取姿态信息。

程序 44 获取姿态

```
result = DobotDll.GetPose(pose)
If result <> DobotCommunicate.DobotCommunicate_NoError Then
    Return
End If
Debug.Print(pose.x)
Debug.Print(pose.y)
Debug.Print(pose.z)
Debug.Print(pose.r)
Debug.Print(pose.joint1Angle)
Debug.Print(pose.joint2Angle)
Debug.Print(pose.joint3Angle)
Debug.Print(pose.joint4Angle)
```

2.6 Qt Demo

2.6.1 工程说明

选择 Qt5.0 及以上的版本。当 QtCreator 的编译器为 mingw 时，直接链接 DobotDll.dll 即可，如果编译器 MSVC 时，需要在 DobotDll.dll 所在的库文件夹中放入 DobotDll.lib。三个功能模块依次为机械臂点动（JOG），获取姿态信息（Pose），示教再现（Playback）。

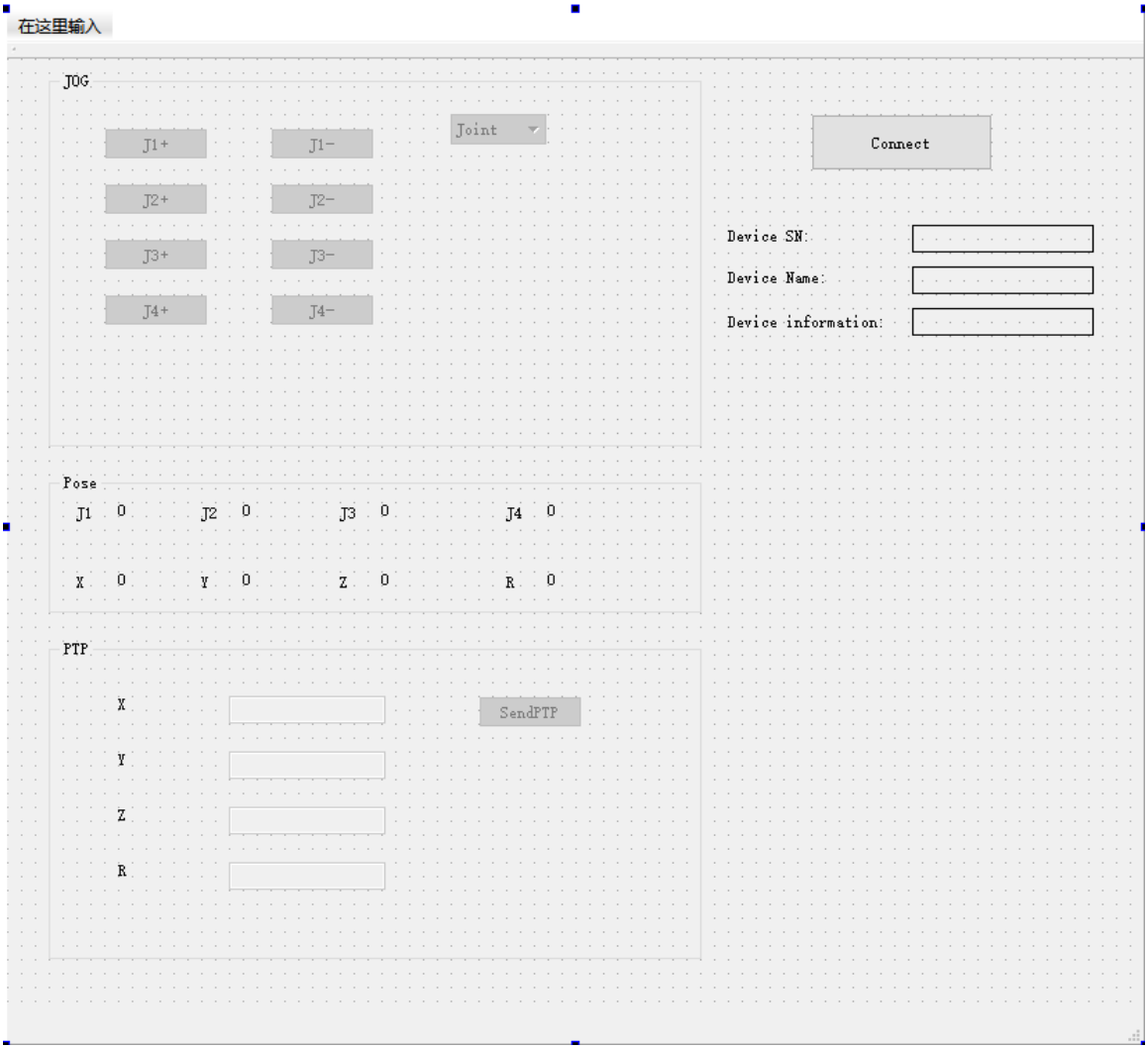

图 6 图 QT Demo 界面

2.6.2 代码说明

1 连接机械臂，并判断是否连接成功；

程序 45 连接

```
if (!connectStatus) {
    if (ConnectDobot(0, 115200) != DobotConnect_NoError) {
        QMessageBox::information(this, tr("error"),
                                 tr("Connect dobot error!!!"),
                                 QMessageBox::Ok);
        return;
    }
}
```

2 获取机械臂序列号信息；

程序 46 获取序列号

```
char deviceSN[64];
GetDeviceSN(deviceSN, sizeof(deviceSN));
ui->deviceSNLabel->setText(deviceSN);
```

3 获取机械臂名字信息；

程序 47 获取名字

```
char deviceName[64];
GetDeviceName(deviceName, sizeof(deviceName));
ui->DeviceNameLabel->setText(deviceName);
```

4 获取机械臂版本信息；

程序 48 获取版本

```
uint8_t majorVersion, minorVersion, revision;
GetDeviceVersion(&majorVersion, &minorVersion, &revision);
ui->DeviceInfoLabel->setText(QString::number(majorVersion) +
                             "." + QString::number(minorVersion) +
                             "." + QString::number(revision));
```

5 设置机械臂夹具中心距参数；

程序 49 获取夹具中心距

```
EndEffectorParams endEffectorParams;
memset(&endEffectorParams, 0, sizeof(endEffectorParams));
endEffectorParams.xBias = 71.6f;
SetEndEffectorParams(&endEffectorParams, false, NULL);
```

6 设置机械臂关节点动的速度与加速度参数；

程序 50 设置关节点动参数

```
JOGJointParams jogJointParams;
for (int i = 0; i < 4; i++) {
    jogJointParams.velocity[i] = 100;
    jogJointParams.acceleration[i] = 100;
}
SetJOGJointParams(&jogJointParams, false, NULL);
```

7 设置机械臂坐标轴点动的速度与加速度参数；

程序 51 设置坐标点动参数

```
JOGCoordinateParams jogCoordinateParams;
for (int i = 0; i < 4; i++) {
    jogCoordinateParams.velocity[i] = 100;
    jogCoordinateParams.acceleration[i] = 100;
}
SetJOGCoordinateParams(&jogCoordinateParams, false, NULL);
```

8 设置机械臂点动速度与加速度的半分比，默认为 50；

程序 52 设置点动参数

```
JOGCommonParams jogCommonParams;
jogCommonParams.velocityRatio = 50;
jogCommonParams.accelerationRatio = 50;
SetJOGCommonParams(&jogCommonParams, false, NULL);
```

9 设置机械臂示教再现的关节运动参数；

程序 53 设置示教关节参数

```
PTPJointParams ptpJointParams;
for (int i = 0; i < 4; i++) {
    ptpJointParams.velocity[i] = 100;
    ptpJointParams.acceleration[i] = 100;
}
SetPTPJointParams(&ptpJointParams, false, NULL);
```

10 设置机械臂示教再现的坐标轴运动参数；

程序 54 设置示教坐标参数

```
PTPCoordinateParams ptpCoordinateParams;
ptpCoordinateParams.xyzVelocity = 100;
ptpCoordinateParams.xyzAcceleration = 100;
ptpCoordinateParams.rVelocity = 100;
ptpCoordinateParams.rAcceleration = 100;
SetPTPCoordinateParams(&ptpCoordinateParams, false, NULL);
```

11 设置机械臂示教再现的 Jump 运动模式的参数；

程序 55 设置示教 Jump 参数

```
PTPJumpParams ptpJumpParams;
ptpJumpParams.jumpHeight = 20;
ptpJumpParams.zLimit = 150;
SetPTPJumpParams(&ptpJumpParams, false, NULL);
```

12 机械臂点动（JOG）；

程序 56 点动

```
JOGCmd jogCmd;
jogCmd.isJoint = ui->teachMode->currentIndex() == 0;
jogCmd.cmd = index + 1;
while (SetJOGCmd(&jogCmd, false, NULL) != DobotCommunicate_NoError)
{…}
```

13 获取机械臂姿态信息（Pose）；

程序 57 获取位姿

```
Pose pose;
while (GetPose(&pose) != DobotCommunicate_NoError) {…}
ui->joint1Label->setText(QString::number(pose.jointAngle[0]));
```

```
ui->joint2Label->setText(QString::number(pose.jointAngle[1]));
ui->joint3Label->setText(QString::number(pose.jointAngle[2]));
ui->joint4Label->setText(QString::number(pose.jointAngle[3]));
ui->xLabel->setText(QString::number(pose.x));
ui->yLabel->setText(QString::number(pose.y));
ui->zLabel->setText(QString::number(pose.z));
ui->rLabel->setText(QString::number(pose.r));
```

14 机械臂示教再现运动（PTP）。

程序 58 示教再现

```
PTPCmd ptpCmd;
ptpCmd.ptpMode = PTPMOVJXYZMode;
ptpCmd.x = ui->xPTPEdit->text().toFloat();
ptpCmd.y = ui->yPTPEdit->text().toFloat();
ptpCmd.z = ui->zPTPEdit->text().toFloat();
ptpCmd.r = ui->rPTPEdit->text().toFloat();
while (SetPTPCmd(&ptpCmd, true, NULL) != DobotCommunicate_NoError)
{…}
```

2.7 Multi-Control Demo

2.7.1 工程说明

此 Demo 的 DobotDll 库为专为多机联动所编译的函数库，与其他 Demo 的函数库不通用，不可混淆。

2.7.2 代码说明

与 QtDemo 的代码几乎一样，但是每个 API 函数都多了一个 dobotId 参数，以此确定连接的机械臂 ID 号，用于多机联动。

1 连接机械臂，动态库将自动返回连接的机械臂的 ID 号，后面对于机械臂的操作，都需要带上 ID 号指定机械臂；

程序 59 连接

```
if (!connectStatus) {
    if (ConnectDobot(ui->lineEdit->text().toLatin1().data(),
                        115200, fwType, version, &dobotId) !=
                        DobotConnect_NoError)
{
        QMessageBox::information(this, tr("error"),
                                        tr("Connect dobot error!!!"),
                                        QMessageBox::Ok);
        return;
    }
}
qDebug() << "dobotId" << dobotId;
```

2 获取机械臂序列号信息；

程序 60 获取序列号

```
char deviceSN[64];
GetDeviceSN(dobotId, deviceSN, sizeof(deviceSN));
ui->deviceSNLabel->setText(deviceSN);
```

3 获取机械臂名字信息；

程序 61 获取名字

```
char deviceName[64];
GetDeviceName(dobotId, deviceName, sizeof(deviceName));
ui->DeviceNameLabel->setText(deviceName);
```

4 获取机械臂版本信息；

程序 62 获取版本

```
uint8_t majorVersion, minorVersion, revision;
GetDeviceVersion(dobotId, &majorVersion, &minorVersion, &revision);
ui->DeviceInfoLabel->setText(QString::number(majorVersion) +
                             "." + QString::number(minorVersion) +
                             "." + QString::number(revision));
```

5 设置机械臂夹具中心距参数；

程序 63 获取中心距

```
EndEffectorParams endEffectorParams;
memset(&endEffectorParams, 0, sizeof(endEffectorParams));
endEffectorParams.xBias = 71.6f;
SetEndEffectorParams(dobotId, &endEffectorParams, false, NULL);
```

6 设置机械臂关节点动的速度与加速度参数；

程序 64 设置关节点动参数

```
JOGJointParams jogJointParams;
for (int i = 0; i < 4; i++) {
    jogJointParams.velocity[i] = 100;
    jogJointParams.acceleration[i] = 100;
}
SetJOGJointParams(dobotId, &jogJointParams, false, NULL);
```

7 设置机械臂坐标轴点动的速度与加速度参数；

程序 65 设置坐标点动参数

```
JOGCoordinateParams jogCoordinateParams;
for (int i = 0; i < 4; i++) {
    jogCoordinateParams.velocity[i] = 100;
    jogCoordinateParams.acceleration[i] = 100;
}
SetJOGCoordinateParams(dobotId, &jogCoordinateParams, false, NULL);
```

8 设置机械臂点动速度与加速度的半分比，默认为 50；

程序 66 设置点动参数

```
JOGCommonParams jogCommonParams;
jogCommonParams.velocityRatio = 50;
jogCommonParams.accelerationRatio = 50;
SetJOGCommonParams(dobotId, &jogCommonParams, false, NULL);
```

9 设置机械臂示教再现的关节运动参数；

程序 67 设置示教关节参数

```
PTPJointParams ptpJointParams;
for (int i = 0; i < 4; i++) {
    ptpJointParams.velocity[i] = 100;
    ptpJointParams.acceleration[i] = 100;
}
SetPTPJointParams(dobotId, &ptpJointParams, false, NULL);
```

10 设置机械臂示教再现的坐标轴运动参数；

程序 68 设置示教坐标参数

```
PTPCoordinateParams ptpCoordinateParams;
ptpCoordinateParams.xyzVelocity = 100;
ptpCoordinateParams.xyzAcceleration = 100;
ptpCoordinateParams.rVelocity = 100;
ptpCoordinateParams.rAcceleration = 100;
SetPTPCoordinateParams(dobotId, &ptpCoordinateParams, false, NULL);
```

11 设置机械臂示教再现的 Jump 运动模式的参数；

程序 69 设置示教 Jump 参数

```
PTPJumpParams ptpJumpParams;
ptpJumpParams.jumpHeight = 20;
ptpJumpParams.zLimit = 150;
SetPTPJumpParams(dobotId, &ptpJumpParams, false, NULL);
```

12 机械臂点动（JOG）；

程序 70 点动

```
JOGCmd jogCmd;
jogCmd.isJoint = ui->teachMode->currentIndex() == 0;
jogCmd.cmd = index + 1;
while (SetJOGCmd(dobotId, &jogCmd, false, NULL) !=
        DobotCommunicate_NoError)
{…}
```

13 获取机械臂姿态信息（Pose）；

程序 71 获取位姿

```
Pose pose;
while (GetPose(dobotId, &pose) != DobotCommunicate_NoError) {
```

```
}
ui->joint1Label->setText(QString::number(pose.jointAngle[0]));
ui->joint2Label->setText(QString::number(pose.jointAngle[1]));
ui->joint3Label->setText(QString::number(pose.jointAngle[2]));
ui->joint4Label->setText(QString::number(pose.jointAngle[3]));
ui->xLabel->setText(QString::number(pose.x));
ui->yLabel->setText(QString::number(pose.y));
ui->zLabel->setText(QString::number(pose.z));
ui->rLabel->setText(QString::number(pose.r));
```

14 机械臂示教再现运动（PTP）。

程序 72 示教再现

```
PTPCmd ptpCmd;
ptpCmd.ptpMode = PTPMOVJXYZMode;
ptpCmd.x = ui->xPTPEdit->text().toFloat();
ptpCmd.y = ui->yPTPEdit->text().toFloat();
ptpCmd.z = ui->zPTPEdit->text().toFloat();
ptpCmd.r = ui->rPTPEdit->text().toFloat();
SetPTPCmd(dobotId, &ptpCmd, true, NULL);
```

2.8 Python Demo

2.8.1 工程说明

Python Demo 有 DobotControl.py，DobotDllType.py 共两个文件。DobotDllType.py 是 Dobot API 的 python 二次封装，DobotControl 是具体的实现文件。

运行 DobotControl.py 前将 Dobot 动态链接库放到 python 的运行目录，或者将 Dobot 动态链接库所在路径添加到系统环境变量。

2.8.2 Python API

DobotDllType.py 是 Dobot 的 Python API 文件，对 Dobot 的动态链接库进行了二次封装。其中加载动态链接库的代码如程序 73。

程序 73 加载动态链接库

```
def load():
    if platform.system() == "Windows":
        return CDLL("DobotDll.dll",  RTLD_GLOBAL)
    elif platform.system() == "Darwin" :
        return CDLL("libDobotDll.dylib",  RTLD_GLOBAL)
    elif platform.system() == "Linux":
        return cdll.loadLibrary("libDobotDll.so")
```

注意：为保证正确加载动态链接库，请务必将 Dobot 动态链接库目录添加到系统环境变量中，详情见 2.1.2。

2.8.3 代码说明

本例程中所有运动 API 均使用队列模式。

1 加载动态链接库，获得库对象（api）；后续所有 python API 调用时都使用到该对象；

程序 74 加载动态链接库

```
api = dType.load()
```

2 连接 Dobot，并打印连接信息，连接成功才处理相关代码；

程序 75 连接 Dobot

```
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])
if (state == dType.DobotConnect.DobotConnect_NoError):
    #Dobot 交互代码
dType.DisconnectDobot(api)
```

3 队列控制(清空队列，开始队列，停止队列)；

程序 76 队列控制

```
dType.SetQueuedCmdClear(api)
dType.SetQueuedCmdStartExec(api)
#将运动指令下发到队列中
dType.SetQueuedCmdStopExec(api)
```

4 设置运动参数；

程序 77 设置运动参数

```
dType.SetHOMEParams(api, 200, 200, 200, 200, isQueued = 1)
dType.SetPTPJointParams(api, 200, 200, 200, 200,
                            200, 200, 200, 200, isQueued = 1)
dType.SetPTPCommonParams(api, 100, 100, isQueued = 1)
```

5 将来回 PTP 运动指令下发到队列中，并获取最后条指令 index；

程序 78 PTP 运动

```
for i in range(0, 5):
        if i % 2 == 0:
            offset = 50
        else:
            offset = -50
        lastIndex = dType.SetPTPCmd(api,
                                    dType.PTPMode.PTPMOVLXYZMode,
                                    200 + offset,
                                    offset,
                                    offset,
                                    offset,
                                    isQueued = 1)[0]
```

6 等待最后一条运动指令完成。

程序 79 等待

```
while lastIndex > dType.GetQueuedCmdCurrentIndex(api)[0]:
    dType.dSleep(100)
```

3. 嵌入式系统

对于嵌入式系统，用户需要根据 Dobot 通讯协议，进行通讯相关的开发工作。

3.1 注意事项

DobotMagician 扩展接口的电平信号是 3.3V，耐压可达 5V。注意使用 AD 功能时不能接入超过 3.3V 的电压，其余的端口可以接入 5V 信号（包括串口接口）。在使用除 STM32 和 Arduino 以外的芯片进行二次开发时，需要注意电平兼容。

3.2 STM32 Demo

3.2.1 硬件说明

本 Demo 是基于 STM32F103VET6 芯片编写，因此使用本程序时需要用户自行配备一块 STM32F103VET6 开发板。（使用其他型号的芯片需要自行移植）

通讯的端口使用了机械臂扩展 10P 接口，接口的类型是 FC-10P，接口的定义如下图 7 所示。我们需要用 RX、TX、GND 这三个端口，如图 8 机械臂与开发板连接 RX->TX1，TX->RX1，GND->GND。

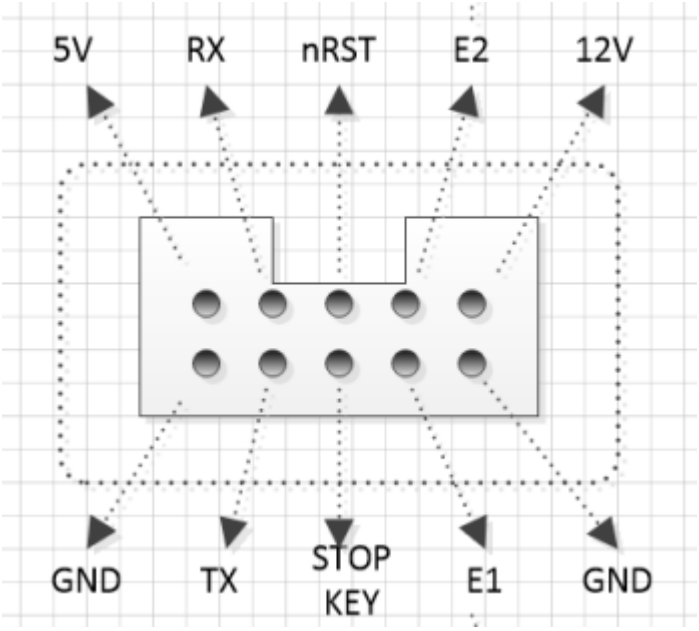

图 7 机械臂的扩展接口定义

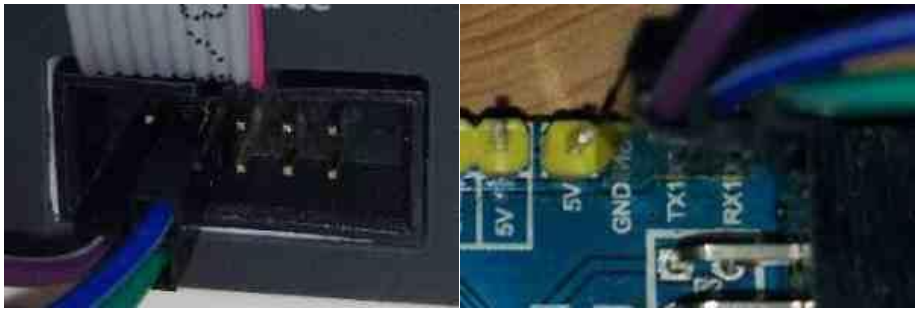

图 8 机械臂与开发板连接

3.2.2 工程说明

本工程使用的编译器软件为 KEIL（4 或 5），DFP 版本为 2.0。

1 通讯

在 DobotMagician 的通讯协议里面有详细的说明，在此仅简单略作说明。每一帧数据包括以下几个内容：包头两个 0xAA，参数长度（取值为 2+N Params 的长度），命令编号 ID，Ctrl 位（包括 RW 读写标记以及 isQueued 队列标记），命令参数 Params（不定长 N），检验和 CheckSum。

表 1 通讯协议格式

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+N	XX	1/0	1/0	N（Byte）	Payload Checksum

DobotMagician 的命令模式有两种，一种是队列指令，一种是立即指令。顾名思义队列命令是按照队列顺序来执行命令的，立即指令是不管当前的状态如何都会立即执行的。在使用队列命令时，机械臂会立即返回一个当前命令对应的索引编号，通过反复查询比较正在执行的命令的索引编号与之前返回的索引编号，可判断队列命令是否执行完成。

2 文件结构

工程文件包含：app、dirver、core、stlib、stm32f10x、complatform 等 5 个目录。其中 app 文件目录存放了主要的命令和 main 函数，这两个文件是用户主要修改使用的地方。driver 文件目录存放了硬件的驱动文件这里主要涉及芯片的串口配置以及时钟配置。Core 文件目录存放了 M3 的核心文件一般不需要改动。Stlib 和 stm32f10x 存放的都是库文件也是不需要修改的。Complatform 文件目录存放的主要是协议的处理文件，用户最好不要进行改动。

3.2.3 代码说明

1 ProtocolProcess 函数说明；

Demo 的发送和接收命令都是保存在 Ringbuffer 里面，通过 ProtocolProcess 函数来处理命令数据。待发送的指令会操作外设发送，外设接收的数据会保存在 Ringbuffer 里面。

2 Common 命令解析；

用户涉及到主要修改的文件是 main.cpp 文件和 common.cpp 文件。Main.cpp 是主函数文件，common 是命令处理文件。以一个典型的 PTP 命令来说明，SetPTPCmd 函数传入的一个 PTPCmd 的结构体数据，队列标记，以及索引编号（暂未使用，主要用来记录当前下发的命令的序列编号）。

程序 80 SetPTPCmd 接口

```
int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued,
                uint64_t *queuedCmdIndex)
{
    Message tempMessage;
```

```
    memset(&tempMessage, 0, sizeof(Message));
    tempMessage.id = ProtocolPTPCmd;
    tempMessage.rw = true;
    tempMessage.isQueued = isQueued;
    tempMessage.paramsLen = sizeof(PTPCmd);
    memcpy(tempMessage.params, (uint8_t *)ptpCmd,
           tempMessage.paramsLen);
    MessageWrite(&gUART4ProtocolHandler, &tempMessage);
    (*queuedCmdIndex)++;
    return true;
}
```

结合表 1 和程序 80 中的 common 命令需要输入的数据都是 Payload 里面的 id、rw、isQueued、params，还有 length（params 的字节长度）。在 SetPTPCmd 里传入的 PTPCmd 包含以上的参数，结构体定义在 common.h 的头文件中。

现有版本中已经提供了 13 条命令，能完成基本的运动控制，需要更多的功能需要参看 DobotMagician 的通讯协议来添加命令实现高级功能。

3 命令发送和接收；

在 Protocol 文件里面，程序会查询发送缓冲区是否为空，不为空就会把串口 4 的发送中断使能，进入串口 4 的中断程序发送命令。Demo 的串口 4 是中断接收，接收的数据会放在接收缓冲区。通过 MessageRead(ProtocolHandler *protocolHandler, Message *message)来读取缓冲区的数据，数据会放在 Message 结构体的变量中。

程序 81 Message 结构体

```
typedef struct tagPTPCmd {
    uint8_t ptpMode;
    float x;
    float y;
    float z;
    float r;
}PTPCmd;
```

4 main 函数。

Main 函数里面主要实现了机械臂前后 100mm 往复运动的功能，如要修改运动的位置直接修改 gPTPCmd 里面的坐标参数。如要实现高级的运动需要结合通讯协议增加命令来实现。

程序 82 main 函数

```
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    SystickInit();          //初始化时钟
    Uart1Init(115200);      //串口 1 初始化，波特率为 115200
    Uart4Init(115200);      //串口 4 初始化，波特率为 115200
    InitRAM();              //初始化运动参数
    ProtocolInit();         //初始化协议
```

```
    //配置笛卡尔坐标的运动参数
    SetPTPCoordinateParams(&gPTPCoordinateParams,
                                  true,
                                &gQueuedCmdIndex);
    //配置 PTP 运动参数比值（0-100）
    SetPTPCommonParams(&gPTPCommonParams,
                             true,
                           &gQueuedCmdIndex);
    printf("\r\n======Enetr demo application======\r\n");
    for(; ;)
    {
        static uint32_t timer = gSystick;
        static uint32_t count = 0;
        if(gSystick - timer > 3000)         //延时 3S
        {
            timer = gSystick;
            count++;
            if(count & 0x01)
            {
                //配置 PTP 运动 X 坐标+100
                gPTPCmd.x += 100;
                //运动 PTP 运动，坐标为 gPTPCmd 结构体里面的坐标
                SetPTPCmd(&gPTPCmd,
                             true,
                           &gQueuedCmdIndex);
            }
            else
            {
                //配置 PTP 运动 X 坐标-100
                gPTPCmd.x -= 100;
                //运动 PTP 运动，坐标为 gPTPCmd 结构体里面的坐标
                SetPTPCmd(&gPTPCmd,
                             true,
                           &gQueuedCmdIndex);
            }
        }
        ProtocolProcess();
    }
}
```

3.3 Arduino Demo

3.3.1 硬件说明

本 Demo 是基于 ArduinoMega2560 芯片编写，因此使用本程序时需要用户自行配备一块 arduinoMega2560 开发板。（理论上其他型号的 Arduino 也可运行但未经测试需要用户自行移

植）

通讯的端口使用了机械臂扩展 10P 接口，接口的类型是 FC-10P，接口的定义如下图 7 所示。我们需要用 RX、TX、GND 这三个端口，如机械臂与开发板连接 RX->TX1，TX->RX1，GND->GND。

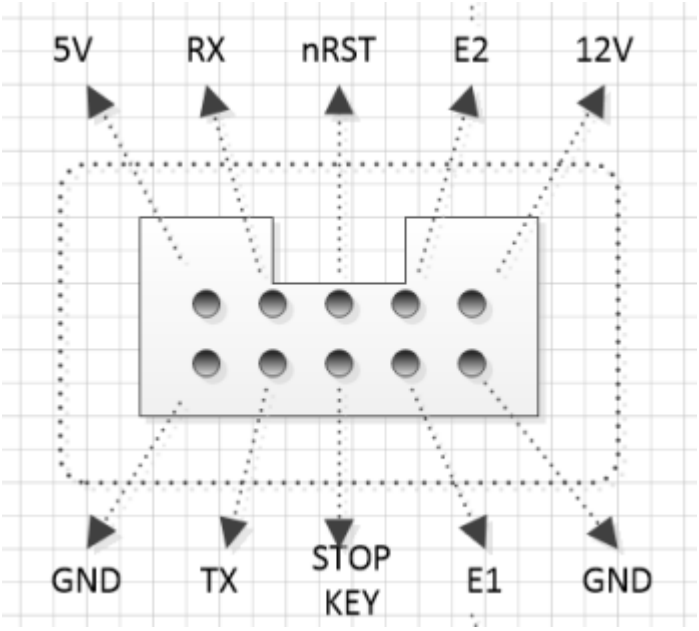

图 9 机械臂的扩展接口定义

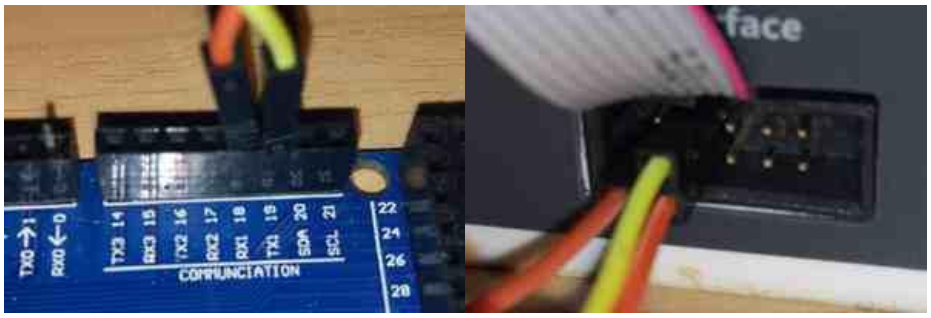

图 10 机械臂与开发板连接

3.3.2 工程说明

本工程使用的编译器软件为 Arduino 1.8.1。

1 通讯；

在 DobotMagician 的通讯协议里面有详细的说明，在此仅简单略作说明。每一帧数据包括以下几个内容：包头两个 0XAA，参数长度（取值为 2+N Params 的长度），命令编号 ID，Ctrl 位（包括 RW 读写标记以及 isQueued 队列标记），命令参数 Params（不定长 N），检验和 CheckSum。

表 2 通讯协议格式

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		

0xAA 0xAA	2+N	XX	1/0	1/0	N（bite）	Payload Checksum

DobotMagician 的命令模式有两种，一种是队列指令，一种是立即指令。顾名思义队列命令是按照队列顺序来执行命令的，立即指令是不管当前的状态如何都会立即执行的。在使用队列命令时，机械臂会立即返回一个当前命令对应的索引编号，然后通过命令去查询当前索引编号是否完成此命令。

2 文件结构。

工程文件主要包含两部分：一是协议层处理文件包括：以 Protocol、Message、Packet 这三个开头的文件；二是应用程序实现文件包括：Common、DobotDemo 文件。此外 FlexTimer2 是 Arduino 的扩展库实现定时器的功能。

3.3.3 代码说明

1 ProtocolProcess 函数说明；

Demo 的发送和接收命令都是保存在 Ringbuffer 里面，通过 ProtocolProcess 函数来处理命令数据。待发送的指令会操作外设发送，外设接收的数据会保存在 Ringbuffer 里面。

2 Common 命令解析；

用户涉及到主要修改的文件是 DobotDemo.ino 文件和 common.cpp 文件。DobotDemo.ino 是主函数文件，common 是命令处理文件。在 Common 文件中以一个典型的 PTP 命令来说明，SetPTPCmd 函数传入的一个 PTPCmd 的结构体数据，队列标记，以及索引编号（暂未使用，主要用来记录当前下发的命令的序列编号）。

程序 83 SetPTPCmd 示例

```
int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex)
{
    Message tempMessage;

    memset(&tempMessage, 0, sizeof(Message));
    tempMessage.id = ProtocolPTPCmd;
    tempMessage.rw = true;
    tempMessage.isQueued = isQueued;
    tempMessage.paramsLen = sizeof(PTPCmd);
    memcpy(tempMessage.params, (uint8_t *)ptpCmd, tempMessage.paramsLen);
    MessageWrite(&gUART4ProtocolHandler, &tempMessage);
    (*queuedCmdIndex)++;
    return true;
}
```

结合表 2 和程序 83 中的 common 命令需要输入的数据都是 Payload 里面的 id、rw、isQueued、params，还有 length（params 的字节长度）。在 SetPTPCmd 里传入的 PTPCmd 包含以上的参数，结构体定义在 common.h 的头文件中。

现有版本中已经提供了 13 条命令，能完成基本的运动控制，需要更多的功能需要参看 DobotMagician 的通讯协议来添加命令实现高级功能。

3 命令发送和接收；

在 Protocol 文件里面，程序会查询发送缓冲区是否为空，不为空就会把从缓冲区中取出数据往串口 1 发送。Demo 的串口 1 是中断接收，接收的数据会放在接收缓冲区。通过 MessageRead(ProtocolHandler *protocolHandler, Message *message)来读取缓冲区的数据，数据会放在 Message 结构体的变量中。

程序 84 Message 结构体

```
typedef struct tagPTPCmd {
uint8_t ptpMode;
    float x;
    float y;
    float z;
    float r;
}PTPCmd;
```

4 配置函数说明；

4.1 Setup 函数主要配置硬件初始化。

程序 85 setup 函数

```
void setup() {
    Serial.begin(115200);              //启动串口 0，波特率为 115200
    Serial1.begin(115200);             //启动串口 1，波特率为 115200
    printf_begin();                    //配置 Printf，定向输出到串口 0
    //Set Timer Interrupt
    FlexiTimer2::set(100,Serialread); //配置定时中断，每 100ms 执行 Serialread 函数
    FlexiTimer2::start();              //启动定时器
}
```

4.2 Serialread 函数读取串口 1 的数据存放到接收缓冲区中。

程序 86 Serialread()函数

```
void Serialread()
{
  while(Serial1.available()) {                //判断串口 1 是否有数据
        uint8_t data = Serial1.read();        //读取 8bit 数据存放到 data
        if (RingBufferIsFull(
                                &gSerialProtocolHandler.rxRawByteQueue)
                                == false) {
              //RingBuffer 有空余空间则会保存数据
              RingBufferEnqueue(
                                &gSerialProtocolHandler.rxRawByteQueue,
                                &data);
        }
  }
}
```

4.3 Serial_putc(char c, struct __file *)和printf_begin(void)主要实现Printf功能。

4.4 InitRAM(void)主要实现配置默认的运动参数。

5 主循环函数。

主循环函数里面主要实现了机械臂前后100mm往复运动的功能，如要修改运动的位置直接修改gPTPCmd里面的坐标参数。如要实现高级的运动需要结合通讯协议增加命令来实现。

程序 87 main函数

```
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    SystickInit();      //初始化时钟
    Uart1Init(115200); //串口1初始化，波特率为115200
    Uart4Init(115200); //串口4初始化，波特率为115200
    InitRAM();          //初始化运动参数
    ProtocolInit();     //初始化协议

    //配置笛卡尔坐标的运动参数
    SetPTPCoordinateParams(&gPTPCoordinateParams,
                                true,
                               &gQueuedCmdIndex);
    //配置PTP运动参数比值（0-100）
    SetPTPCommonParams(&gPTPCommonParams, true, &gQueuedCmdIndex);

    printf("\r\n======Enetr demo application======\r\n");
    for(; ;)
    {
        static uint32_t timer = gSystick;
        static uint32_t count = 0;
        if(gSystick - timer > 3000)         //延时3S
        {
            timer = gSystick;
            count++;
            if(count & 0x01)
            {
                gPTPCmd.x += 100;           //配置PTP运动X坐标+100

                //运动PTP运动，坐标为gPTPCmd结构体里面的坐标
                SetPTPCmd(&gPTPCmd,
                            true,
                           &gQueuedCmdIndex);
            }
            else
```

```
            {
                //配置 PTP 运动 X 坐标-100
                gPTPCmd.x -= 100;
                //运动 PTP 运动，坐标为 gPTPCmd 结构体里面的坐标
                SetPTPCmd(&gPTPCmd, true, &gQueuedCmdIndex);
            }
        }
        ProtocolProcess();//Protocol 进程
    }
}
```

3.4 iOS Demo

3.4.1 工程说明

DOBOTKit.framework 是静态库，把静态库加入工程就可以在工程中使用，DOBOTKit 是示例工程，里面展示了基于静态库的简单使用。

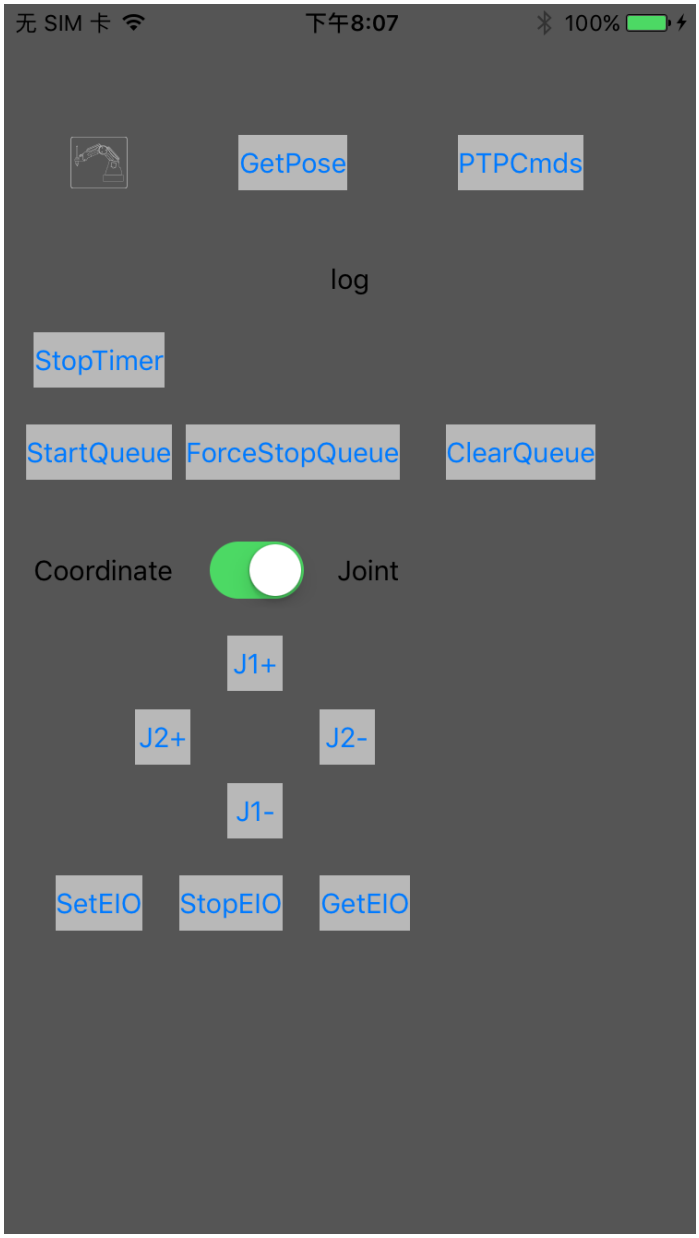

图 11 Demo 界面

3.4.2 代码说明

本例程实现了获取实时位姿功能，开发者可以参照本例程和通信协议文档实现其他功能，iOS 静态库中已经封装好对应的 API。

1 初始化；

使用时，首先初始化 BLEMsgMgr，把当前 VC 作为代理和消息处理者，这个单例类负责处理蓝牙连接、消息收发等。

程序 88 初始化 BLEMsgMgr

```
[BLEMsgMgr sharedMgr].delegate = self;
[[BLEMsgMgr sharedMgr] addMsgHandler:self];
```

把当前 ViewController 加入消息处理者，这样就可以收到消息回调的通知（也可以在闭

包里处理消息回调，后续会有说明）。

2 蓝牙连接和断开蓝牙连接；

程序 89 连接控制

```
if ([[BLEMsgMgr sharedMgr] isConnected]) {
        //断开连接
        [[BLEMsgMgr sharedMgr] disconnect];
    }
    else
    {
       [[BLEMsgMgr sharedMgr]
                    scanDevice:30.0f
                    mode:BLESearchMode_FindAndConnectTheFirst];
    }
```

连接蓝牙时，应用会连接第一个搜索到的机械臂。

3 消息示例，获取实时位姿。

如程序 90，构建一个 Payload 对象，设置相应的参数，调用 BLEMsgMgr 的 sendMsg 方法，就可以通过蓝牙把命令下发到机械臂。

程序 90 下发命令

```
Payload *payload = [[Payload alloc] init];
    [payload cmdGetPose];
    payload.complete = ^(MsgResult result, id msg){
        if (result == MsgResult_Ok) {
            //解析返回的位置信息
            Payload *msgPayload = ((DobotMagicianMsg *)msg).payload;
            Pose p;
            [msgPayload.params getBytes:&p length:sizeof(p)];
            NSString *text = [NSString stringWithFormat:
                              @"Pose:x:%.0f,y:%.0f,z:%.0f,r:%.0f",
                              p.x,p.y,p.z,p.r];
            dispatch_async(dispatch_get_main_queue(), ^{
                _lblLog.text = text;
            });
        }
    };
    [[BLEMsgMgr sharedMgr] sendMsg:payload];
```

如程序 91 实现 MsgHandler 协议,就可以在 handleMsg 这个回调方法里收到机械臂返回的消息。也可以像上面的代码，实现 payload.complete，在闭包里处理消息返回。

程序 91 接收返回数据

```
-(void)handleMsg:(DobotMagicianMsg *)msg
{
```

```
    Payload *payload = msg.payload;
    switch ((int)payload.ID) {
        case ProtocolGetPose:{
            Pose p;
            [payload.params getBytes:&p length:sizeof(p)];
            NSString *text = [NSString stringWithFormat:
                              @"Pose:x:%.0f,y:%.0f,z:%.0f,r:%.0f",
                              p.x,p.y,p.z,p.r];
            //记录当前 pose
            _lblLog.text = text;
        }
            break;
        default:
            break;
    }
}
```

3.5 Android Demo

3.5.1 工程说明

Dobot.jar 是 DobotMagician 的封装库，里面封装了 Android4.3+平台的 BLE 常用操作及 DobotMagician 通信协议的部分封装。使用前只需导入 Dobot.jar 到 Android Studio(或 Eclipse)项目中的 libs 目录便可在工程中引用封装好的 API 对 DobotMagician 进行操作。DobotDemo 是示例工程，简单的演示了调用 Dobot.jar 中 API 的方法步骤。

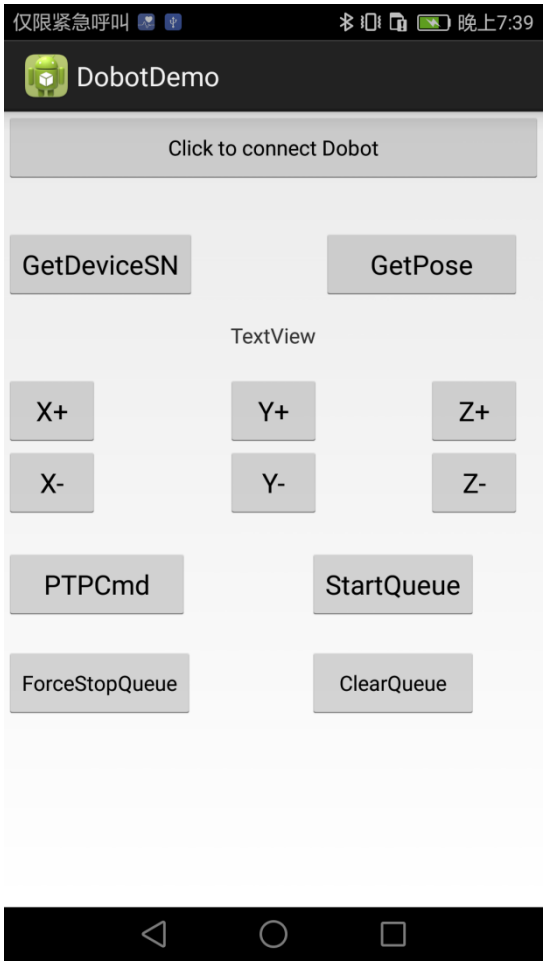

图 12 Andriod Demo 界面

3.5.2 代码说明

本例程实现了获取实时位姿功能，开发者可以参照本例程和通信协议文档实现其他功能，Dobot.jar 库中已经封装好部分对应的 API。

1. 在 Android Project 的 AndroidManifest.xml 文件中添加蓝牙权限；

程序 92 添加蓝牙权限

```
<uses-permission
    android:name="android.permission.BLUETOOTH"/>
<uses-permission
    android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-feature
    android:name="android.hardware.bluetooth_le"
    android:required="true" />
```

2. 创建 Dobot 对象；

程序 93 创建 Dobot 对象

```
//创建时传入 Context 参数，实现 DobotCallbacks()接口
Dobot  myDobot = new  Dobot(this,new DobotCallbacks() {
    @Override
```

```
    public void DobotDisconnected(BluetoothGatt arg0, BluetoothDevice arg1) {
        // TODO Auto-generated method stub
        Log.d("dobot","dobot Disconnected");
        }

    @Override
    public void DobotConnected(BluetoothGatt arg0, BluetoothDevice arg1) {
        // TODO Auto-generated method stub
        Log.d("dobot","dobot connected");
        }

    @Override
    public void DobotConnectTimeOut() {
        // TODO Auto-generated method stub
        Log.d("dobot","dobot connect timeout");
    }
});
```

3. 初始化 Dobot 对象；

程序 94 初始化 Dobot 对象

```
myDobot.initialize();
```

4. 连接手机到 Dobot；

程序 95 连接

```
myDobot.Connect(); //需要断开连接时调用 myDobot.close();
```

5. 调用 API 获取实时位姿。

程序 96 获取位姿

```
myDobot.GetPose(new DataReceiveListener() {
    @Override
    public void OnReceive() {
        // TODO Auto-generated method stub
        TagPose  pose = myDobot.ReadPose();
        float x= pose.getX();
        float y= pose.getY();
        float z= pose.getZ();
        float r= pose.getR();
        Log.d("dobot","X :"+x+"---Y :"+y+"---Z :"+z+"---R :"+r);
       }
});
```

深圳市越疆科技有限公司

邮编：510630

网址：www.dobot.cc

电话：(0755)38730916

地址：深圳市南山区西丽桃源街道塘朗工业区 A 区 8 栋 4 楼