1. 机器人主界面（**GTDecorationRobotTool**）

此源文件中主要做了各种主界面布局和相关控件属性的设置，具体包含界面上的登录按钮（状态更新显示），基本设置按钮（自带弹出菜单，菜单项中含各类功能设置按钮），最小化和退出按钮，自动运行按钮（状态更新显示），方案演示窗口和模块控制窗口的互相切换，单击鼠标拖动主窗口等各功能。以下对各功能实现的关键处进行详细解析：

1. 去掉主窗口的标题栏，并设置背景透明，控件不透明。具体可参考Qt中QWidget对用户界面的基本属性设置。

this->setWindowFlags(Qt::FramelessWindowHint);

this->setAttribute(Qt::WA\_TranslucentBackground, true);

1. 将工作线程中的信号量与主界面的槽函数关联起来，用于主界面或其它窗口界面上的各显示控件，实时更新显示工作线程中的数据信息。

此处以一简单例子作为模板，如下所示。Qt::UniqueConnection 表明若当前信号与槽已经连接过了，则不再连接。

此处添加Qt::UniqueConnection是为了防止重复连接。具体可参考Qt中关于信号和槽的相关资料。

QObject::connect(ui.page\_ModularControl->m\_pThread\_Chasis, SIGNAL(Chasis\_Info\_Output()), ui.page\_FunctionalDemo, SLOT(on\_Chasis\_Info\_Output()), Qt::UniqueConnection);

1. 方案演示窗口与模块控制窗口的互相切换。

此功能实现依靠Qt中的分页控件QStackedWidget，根据鼠标按下的按钮，设置分页控件显示对应的页。

具体例子如下所示，当按下**方案演示选择按钮**时，就设置分页控件显示方案演示窗口，设置相应的界面属性，并设置**方案演示选择按钮**为按下状态，**模块控制选择按钮**为弹起状态。

void GTDecorationRobotTool::on\_toolButton\_Select\_FunctionalDemo\_pressed()

{

ui.stackedWidget->setCurrentIndex(0);

ui.stackedWidget->currentWidget()->setAttribute(Qt::WA\_TranslucentBackground, true);

ui.toolButton\_Select\_FunctionalDemo->SetButtonChecked(true);

ui.toolButton\_Select\_ModularControl->SetButtonChecked(false);

}

1. 构造函数
2. 构造函数
3. 构造函数
4. 通信设置（QGTCommunicationSet）

此源文件中主要集成了装修机器人系统各模块的通信设置（串口号、波特率、网口Ip地址、网口端口号等），如底盘驱动模块、升降平台模块、机械臂模块、手柄控制模块、激光雷达模块、激光测距模块、Ros系统模块等。

1. 界面属性设置

此处通过设定窗口模式为Qt::WindowModal来设置当前窗口阻塞父窗口。

通过设定窗口属性为Qt::WA\_DeleteOnClose来设置当前窗口在关闭时自动销毁。

具体可参考QWidget中对用户界面的基本属性设置。

this->setWindowFlags(Qt::FramelessWindowHint | Qt::Dialog);

this->setWindowModality(Qt::WindowModal);

this->setAttribute(Qt::WA\_DeleteOnClose, true);

this->setAttribute(Qt::WA\_StyledBackground, true);

1. 设置当前界面上的控件的显示状态

判断当前登录状态是否为开发者状态，若为开发者状态，即表明为可调式状态，则此时显示各模块通信端口的选择控件（checkBox）；否则隐藏各模块通信端口的选择控件（checkBox）。具体实现如下所示：

if (g\_pGlobalUnit->m\_nLoginState == LS\_Developer)

{

ui.checkBox\_IsSelect\_Chasis\_Connect->show();

ui.checkBox\_IsSelect\_Robot\_Connect->show();

ui.checkBox\_IsSelect\_Ros\_Connect->show();

ui.checkBox\_IsSelect\_LiftPlat\_Connect->show();

ui.checkBox\_IsSelect\_JoyHandle\_Connect->show();

ui.checkBox\_IsSelect\_Chasis\_Connect->setChecked(g\_pGlobalUnit->m\_bIsSelect\_Chasis\_Connect);

ui.checkBox\_IsSelect\_Robot\_Connect->setChecked(g\_pGlobalUnit->m\_bIsSelect\_Robot\_Connect);

ui.checkBox\_IsSelect\_Ros\_Connect->setChecked(g\_pGlobalUnit->m\_bIsSelect\_Ros\_Connect);

ui.checkBox\_IsSelect\_Lidar\_Connect->setChecked(g\_pGlobalUnit->m\_bIsSelect\_Lidar\_Connect);

ui.checkBox\_IsSelect\_LaserMeasure\_Connect->setChecked(g\_pGlobalUnit->m\_bIsSelect\_LaserMeasure\_Connect);

ui.checkBox\_IsSelect\_LiftPlat\_Connect->setChecked(g\_pGlobalUnit->m\_bIsSelect\_LiftPlat\_Connect);

ui.checkBox\_IsSelect\_JoyHandle\_Connect->setChecked(g\_pGlobalUnit->m\_bIsSelect\_JoyHandle\_Connect);

}

else

{

ui.checkBox\_IsSelect\_Chasis\_Connect->hide();

ui.checkBox\_IsSelect\_Robot\_Connect->hide();

ui.checkBox\_IsSelect\_Ros\_Connect->hide();

ui.checkBox\_IsSelect\_Lidar\_Connect->hide();

ui.checkBox\_IsSelect\_LaserMeasure\_Connect->hide();

ui.checkBox\_IsSelect\_LiftPlat\_Connect->hide();

ui.checkBox\_IsSelect\_JoyHandle\_Connect->hide();

g\_pGlobalUnit->m\_bIsSelect\_Chasis\_Connect = true;

g\_pGlobalUnit->m\_bIsSelect\_Robot\_Connect = true;

g\_pGlobalUnit->m\_bIsSelect\_Ros\_Connect = true;

g\_pGlobalUnit->m\_bIsSelect\_Lidar\_Connect = true;

g\_pGlobalUnit->m\_bIsSelect\_LaserMeasure\_Connect = true;

g\_pGlobalUnit->m\_bIsSelect\_LiftPlat\_Connect = true;

g\_pGlobalUnit->m\_bIsSelect\_JoyHandle\_Connect = true;

}

1. 各模块选择控件（checkBox）触发

判断系统的连接状态，若系统当前已经连接上各个已经选择的模块，则当前不可更改通信设置界面上的各选择控件的状态。此处以一个模块的选择控件举例说明

if (g\_pGlobalUnit->m\_bIsConnect)

{

QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, tr("Warning"),

"Can't select when system Connected, please disconnect first",

QMessageBox::NoButton | QMessageBox::Ok | QMessageBox::NoButton);

ui.checkBox\_IsSelect\_Chasis\_Connect->setChecked(g\_pGlobalUnit->m\_bIsSelect\_Chasis\_Connect);

return;

}

g\_pGlobalUnit->m\_bIsSelect\_Chasis\_Connect = bIsCheck;

1. 各模块通信端口号等的设置

在界面上操作对应的端口号、波特率、Ip地址等控件后，点击界面上的确定按钮，则会先对各控件值进行校验，校验通过后，则会将各控件值赋值给全局类中各个模块的通信参数变量。基本设置按钮的弹出菜单项中的其它设置界面的操作，也基本上与通信设置界面一致。具体例子如下所示：

void QGTCommunicationSet::on\_toolButton\_Communication\_Ok\_pressed()

{

if ((false == ui.widget\_Chasis\_IpAddress->isValidIpAddr()) || (true == ui.lineEdit\_Chasis\_Port->text().isEmpty()) ||

(false == ui.widget\_Robot\_IpAddress->isValidIpAddr()) || (true == ui.lineEdit\_Robot\_Port->text().isEmpty()) ||

(false == ui.widget\_Ros\_IpAddress->isValidIpAddr()) || (true == ui.lineEdit\_Ros\_Port->text().isEmpty()))

{

QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, tr("Warning"),

"Invalid ip address, Please input valid ip address first",

QMessageBox::NoButton | QMessageBox::Ok | QMessageBox::NoButton);

return;

}

g\_pGlobalUnit->m\_strChasis\_IpAddr = ui.widget\_Chasis\_IpAddress->GetIpText();

g\_pGlobalUnit->m\_strChasis\_Port = ui.lineEdit\_Chasis\_Port->text();

g\_pGlobalUnit->m\_strRobot\_IpAddr = ui.widget\_Robot\_IpAddress->GetIpText();

g\_pGlobalUnit->m\_strRobot\_Port = ui.lineEdit\_Robot\_Port->text();

g\_pGlobalUnit->m\_strRos\_IpAddr = ui.widget\_Ros\_IpAddress->GetIpText();

g\_pGlobalUnit->m\_strRos\_Port = ui.lineEdit\_Ros\_Port->text();

g\_pGlobalUnit->m\_strLidar\_Port = ui.comboBox\_Lidar\_Port->currentText();

g\_pGlobalUnit->m\_iLidar\_Baudrate = ui.comboBox\_Lidar\_Baudrate->currentText().toInt();

g\_pGlobalUnit->m\_iLaserMeasure\_Port = ui.comboBox\_LaserMeasure\_Port->currentIndex() + 1;

g\_pGlobalUnit->m\_iLaserMeasure\_Baudrate = ui.comboBox\_LaserMeasure\_Baudrate->currentText().toInt();

g\_pGlobalUnit->m\_iLiftPlat\_PortNum = ui.comboBox\_LiftPlatform\_Port->currentIndex() + 1;

g\_pGlobalUnit->m\_iLiftPlat\_Baudrate = ui.comboBox\_LiftPlatform\_Baudrate->currentText().toInt();

this->close();

}

1. 造函数
2. 模块控制窗口

此界面主要集成了各个系统模块的具体操作控制窗口，包含底盘控制模块、升降平台控制模块、机械臂控制模块和手柄控制模块等，同时包含了各个系统模块的工作线程。

通过选择按下当前界面上方的各个按钮，来显示不同模块窗口并进行相应的控制操作。

1. 模块选择

根据当前鼠标按下的选择模块按钮，设置当前界面上的分页控件显示与之对应的模块窗口。此时设置当前选择模块按钮为按下状态，并设置其它选择模块按钮为弹起状态。具体实现如下所示：

void QGTModularControlPage::on\_toolButton\_SelectModule\_Chasis\_pressed()

{

ui.stackedWidget->setCurrentIndex(0);

ui.stackedWidget->currentWidget()->setAttribute(Qt::WA\_TranslucentBackground, true);

ui.toolButton\_SelectModule\_Chasis->SetButtonChecked(true);

ui.toolButton\_SelectModule\_LiftPlat->SetButtonChecked(false);

ui.toolButton\_SelectModule\_Robot->SetButtonChecked(false);

ui.toolButton\_SelectModule\_JoyHandle->SetButtonChecked(false);

ui.line\_Module\_Chasis->hide();

ui.line\_Module\_LiftPlat->show();

ui.line\_Module\_Robot->show();

ui.line\_Module\_JoyHandle->show();

}

1. 工作线程的启动和关闭

判断传入参数的值是否为true，并通过判断每个模块是否已经连接，从而设置相应模块的工作线程开启与否。具体实现如下所示：

void QGTModularControlPage::Set\_Thread\_ModularControl\_Enable(bool bIsBeginWork)

{

if (bIsBeginWork)

{

if (g\_pGlobalUnit->IsChasisConnect())

{

m\_pThread\_Chasis->m\_bIsExit = false;

m\_pThread\_Chasis->start();

}

if (g\_pGlobalUnit->IsLiftPlatConnect())

{

m\_pThread\_LiftPlat->m\_bIsExit = false;

m\_pThread\_LiftPlat->start();

}

if (g\_pGlobalUnit->IsRobotConnect())

{

m\_pThread\_Robot->m\_bIsExit = false;

m\_pThread\_Robot->start();

}

if (g\_pGlobalUnit->IsRosConnect())

{

m\_pThread\_Ros->m\_bIsExit = false;

m\_pThread\_Ros->start();

}

if (g\_pGlobalUnit->IsLidarConnect())

{

m\_pThread\_Lidar->m\_bIsExit = false;

m\_pThread\_Lidar->start();

}

if (g\_pGlobalUnit->IsLaserMeasureConnect())

{

m\_pThread\_LaserMeasure->m\_bIsExit = false;

m\_pThread\_LaserMeasure->start();

}

if (g\_pGlobalUnit->IsJoyHandleConnect())

{

m\_pThread\_JoyHandle->m\_bIsExit = false;

m\_pThread\_JoyHandle->start();

}

}

else

{

if (g\_pGlobalUnit->IsChasisConnect())

{

m\_pThread\_Chasis->m\_bIsExit = true;

m\_pThread\_Chasis->quit();

}

if (g\_pGlobalUnit->IsLiftPlatConnect())

{

m\_pThread\_LiftPlat->m\_bIsExit = true;

m\_pThread\_LiftPlat->quit();

}

if (g\_pGlobalUnit->IsRobotConnect())

{

m\_pThread\_Robot->m\_bIsExit = true;

m\_pThread\_Robot->quit();

}

if (g\_pGlobalUnit->IsRosConnect())

{

m\_pThread\_Ros->m\_bIsExit = true;

m\_pThread\_Ros->quit();

}

if (g\_pGlobalUnit->IsLidarConnect())

{

m\_pThread\_Lidar->m\_bIsExit = true;

m\_pThread\_Lidar->quit();

}

if (g\_pGlobalUnit->IsLaserMeasureConnect())

{

m\_pThread\_LaserMeasure->m\_bIsExit = true;

m\_pThread\_LaserMeasure->quit();

}

if (g\_pGlobalUnit->IsJoyHandleConnect())

{

m\_pThread\_JoyHandle->m\_bIsExit = true;

m\_pThread\_JoyHandle->quit();

}

}

}

1. 构造函
2. 构造函
3. 构造函
4. 方案演示窗口

此界面主要集成了各个演示方案的窗口，包含A（循线）、B（循线、结构光检测、打磨）、C（机械臂画线）、D（贴瓷砖）、E（循线、升降平台运动、机械臂喷漆）、F（循线、升降平台运动、Ros系统喷漆）等几个演示方案，。通过选择按下当前界面上方的各个按钮，来显示不同方案演示窗口。

此界面有两个工作线程，分别为激光线检测线程和演示方案工作线程。

1. 演示方案窗口选择

根据当前鼠标按下的选择方案按钮，设置当前界面上的分页控件显示与之对应的演示方案窗口。此时设置当前选择方案按钮为按下状态，并设置其它选择方案按钮为弹起状态。具体实现如下所示：

void QGTFunctionalDemoPage::on\_toolButton\_SelectDemoPlan\_A\_pressed()

{

ui.stackedWidget->setCurrentIndex(0);

ui.stackedWidget->currentWidget()->setAttribute(Qt::WA\_TranslucentBackground, true);

ui.toolButton\_SelectDemoPlan\_A->SetButtonChecked(true);

ui.toolButton\_SelectDemoPlan\_B->SetButtonChecked(false);

ui.toolButton\_SelectDemoPlan\_C->SetButtonChecked(false);

ui.toolButton\_SelectDemoPlan\_D->SetButtonChecked(false);

ui.toolButton\_SelectDemoPlan\_E->SetButtonChecked(false);

ui.toolButton\_SelectDemoPlan\_F->SetButtonChecked(false);

ui.line\_SelectDemoPlan\_A->hide();

ui.line\_SelectDemoPlan\_B->show();

ui.line\_SelectDemoPlan\_C->show();

ui.line\_SelectDemoPlan\_D->show();

ui.line\_SelectDemoPlan\_E->show();

ui.line\_SelectDemoPlan\_F->show();

}

1. 演示方案的工作线程使能

根据判断传入参数值是否为true，设置演示方案工作线程的开启与否。具体实现如下：

void QGTFunctionalDemoPage::Set\_Thread\_DemoPlan\_Enable(bool bIsBeginWork)

{

if (bIsBeginWork)

{

m\_pThread\_DemoPlan->m\_bIsExit = false;

m\_pThread\_DemoPlan->start();

}

else

{

m\_pThread\_DemoPlan->m\_bIsExit = true;

m\_pThread\_DemoPlan->quit();

}

}

1. 自动运行按钮使能

根据判断传入参数值是否为true，再通过判断当前选择的方案演示序号，设置激光线检测线程的开启与否，以及相应演示方案的各个参数初始值。具体实现如下所示：

void QGTFunctionalDemoPage::Set\_AutoRun\_DemoPlan\_Enable(bool bIsRunning)

{

if (bIsRunning)

{

if ((g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 0) || (g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 1) ||

(g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 4))

{

m\_pThread\_LineFollow\_Read->m\_bIsExit = false;

m\_pThread\_LineFollow\_Read->start();

}

if (g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 0)

{

g\_pGlobalUnit->m\_nAutoRun\_NowStatus = ARS\_LineFollow\_RotateAdjust;

}

else if (g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 1)

{

g\_pGlobalUnit->m\_nAutoRun\_NowStatus = ARS\_LineFollow\_RotateAdjust;

}

else if (g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 2)

{

g\_pGlobalUnit->Calc\_DrawLine\_RectPos();

g\_pGlobalUnit->m\_nAutoRun\_NowStatus = ARS\_DrawLine\_Begin;

}

else if (g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 3)

{

g\_pGlobalUnit->Write\_Debug\_Info("Tiling Start", DIT\_Normal);

g\_pGlobalUnit->Calc\_Tiling\_MaterielAndPaste\_Pos();

g\_pGlobalUnit->m\_nAutoRun\_NowStatus = ARS\_Tiling\_PickMateriel\_Begin;

}

else if (g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 4)

{

g\_pGlobalUnit->m\_iSprayPaint\_Current\_Count\_MovedBy\_Chasis = 0;

g\_pGlobalUnit->m\_iSprayPaint\_Current\_Count\_MovedBy\_LiftPlat = 0;

g\_pGlobalUnit->m\_iSprayPaint\_Current\_Count\_MovedBy\_Robot = 0;

g\_pGlobalUnit->m\_bIs\_SprayPaint\_Robot\_Move = false;

g\_pGlobalUnit->Calc\_SprayPaint\_Param();

g\_pGlobalUnit->Write\_Debug\_Info("SprayPaint Start", DIT\_Normal);

g\_pGlobalUnit->m\_nAutoRun\_NowStatus = ARS\_LineFollow\_RotateAdjust;

}

else if (g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 5)

{

g\_pGlobalUnit->m\_iRos\_SprayPaint\_Current\_Count\_MovedBy\_Chasis = 0;

g\_pGlobalUnit->m\_iRos\_SprayPaint\_Current\_Count\_MovedBy\_LiftPlat = 0;

g\_pGlobalUnit->m\_bIs\_Ros\_Work\_Finished = true;

g\_pGlobalUnit->m\_bIs\_Ros\_SprayPaint\_RosWork = false;

g\_pGlobalUnit->m\_nRos\_Work\_Pos\_Status = RWPS\_Idle;

g\_pGlobalUnit->Calc\_Ros\_SprayPaint\_Param();

g\_pGlobalUnit->Write\_Debug\_Info("Ros SprayPaint Start", DIT\_Normal);

g\_pGlobalUnit->m\_nAutoRun\_NowStatus = ARS\_Ros\_Chasis\_RotateAdjust;

}

}

else

{

g\_pGlobalUnit->Set\_DemPlan\_Stop();

if ((g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 0) || (g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 1) ||

(g\_pGlobalUnit->m\_iDemoPlanSet\_Index == 4))

{

m\_pThread\_LineFollow\_Read->m\_bIsExit = true;

m\_pThread\_LineFollow\_Read->quit();

}

}

}

1. 各个演示方案窗口的数据显示

通过关联各个控制模块的工作线程中的信号与当前演示方案窗口的槽函数实现。而在上述槽函数中，针对每个方案演示窗口，实现不同的数据显示。具体实现如下所示：

void QGTFunctionalDemoPage::on\_Chasis\_Info\_Output()

{

ui.DemoPlan\_A->Update\_ChasisInfo\_DemoPlanA();

ui.DemoPlan\_B->Update\_ChasisInfo\_DemoPlanB();

ui.DemoPlan\_E->Update\_ChasisInfo\_DemoPlanE();

ui.DemoPlan\_F->Update\_Chasis\_Info\_DemoPlanF();

}

void QGTFunctionalDemoPage::on\_Robot\_Info\_Output()

{

ui.DemoPlan\_B->Update\_RobotInfo\_DemoPlanB();

ui.DemoPlan\_C->Update\_RobotInfo\_DemoPlanC();

ui.DemoPlan\_E->Update\_RobotInfo\_DemoPlanE();

}

void QGTFunctionalDemoPage::on\_Lidar\_Info\_Output()

{

ui.DemoPlan\_F->Update\_Lidar\_Info\_DemoPlanF();

}

void QGTFunctionalDemoPage::on\_LaserMeasure\_Info\_Output()

{

ui.DemoPlan\_F->Update\_LaserMeasure\_Info\_DemoPlanF();

}

void QGTFunctionalDemoPage::on\_LineFollow\_Info\_Output()

{

ui.DemoPlan\_A->Update\_LineFollowInfo\_DemoPlanA();

ui.DemoPlan\_B->Update\_LineFollowInfo\_DemoPlanB();

ui.DemoPlan\_E->Update\_LineFollowInfo\_DemoPlanE();

}

1. 构造函
2. 构造函
3. 底盘控制模块

此界面主要是装修机器人底盘控制部分，包含底盘的前进、后退、左移、右移、顺时针自旋转、逆时针自旋转、旋转舵轮至指定角度、连杆机构升降、电磁铁开关等功能。

在界面上有实时的左右舵轮数据（角度、速度、行程）显示，并有相应的动态舵轮图像显示。

各功能按键有与之匹配的底盘工作状态量，按下不同的按钮，即为设置不同的底盘工作状态量。

1. 界面上各功能按键

当鼠标按下当前界面上的功能按键时，会首先判断底盘是否已经连接。若已经连接，则设置全局类中底盘模块的工作状态量为与按键匹配的状态量，从而实现相应的功能。此处以功能按键中的一个为例作为说明。

void QGTChasisModule::on\_toolButton\_Chasis\_Move\_Forward\_pressed()

{

if (false == g\_pGlobalUnit->IsChasisConnect())

{

QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, tr("Warning"),

"Chasis disconnect now, please connect first",

QMessageBox::NoButton | QMessageBox::Ok | QMessageBox::NoButton);

return;

}

g\_pGlobalUnit->m\_nChasis\_NowType = CMT\_NORMAL\_Forward;

}

1. 底盘数据实时更新显示

将底盘工作线程中的信号与当前控制界面的槽函数关联，实时输出工作线程中底盘的详细数据，如左右舵轮的角度、速度和行程等，并通过当前控制界面上的左右舵轮的图像控件。实时更新显示舵轮的状态。具体实现如下所示：

void QGTChasisModule::Update\_ActualInfo\_Chasis()

{

QString strSpeed\_Left = QString::number(g\_pGlobalUnit->m\_nChasisInfo.iSpeed\_LeftWheel);

QString strSpeed\_Right = QString::number(g\_pGlobalUnit->m\_nChasisInfo.iSpeed\_RightWheel);

double dwAngle\_Left = (double)g\_pGlobalUnit->m\_nChasisInfo.iAngle\_LeftWheel / (double)10;

double dwAngle\_Right = (double)g\_pGlobalUnit->m\_nChasisInfo.iAngle\_RightWheel / (double)10;

QString strAngle\_Left = QString::number(dwAngle\_Left, 'f', 1);

QString strAngle\_Right = QString::number(dwAngle\_Right, 'f', 1);

QString strDis\_Left = QString::number(g\_pGlobalUnit->m\_nChasisInfo.iDis\_LeftWheel);

QString strDis\_Right = QString::number(g\_pGlobalUnit->m\_nChasisInfo.iDis\_RightWheel);

ui.label\_Chasis\_LeftWheel\_Speed->setText(strSpeed\_Left);

ui.label\_Chasis\_RightWheel\_Speed->setText(strSpeed\_Right);

ui.label\_Chasis\_LeftWheel\_Angle->setText(strAngle\_Left);

ui.label\_Chasis\_RightWheel\_Angle->setText(strAngle\_Right);

ui.label\_Chasis\_LeftWheel\_Dis->setText(strDis\_Left);

ui.label\_Chasis\_RightWheel\_Dis->setText(strDis\_Right);

ui.label\_Chasis\_LeftWheel->UpdateChasisWheel(pix\_LeftWheel, dwAngle\_Left);

ui.label\_Chasis\_RightWheel->UpdateChasisWheel(pix\_RightWheel, dwAngle\_Right);

}

1. 构造函
2. 构造函
3. 构造函
4. 构造函
5. 构造函
6. 升降平台控制

此界面主要是装修机器人升降平台控制部分，包含升降平台的上升、下降、停止、电机转速设置等功能。

在界面上有实时的升降平台电机数据显示。

各功能按键有与之匹配的升降平台工作状态量，鼠标按下不同的按钮，即为设置不同的升降平台工作状态量。

1. 界面上各功能按键

当鼠标按下当前界面上的功能按键时，会首先判断升降平台是否已经连接。若已经连接，则设置全局类中升降平台模块的工作状态量为与按键匹配的状态量，从而实现相应的功能。此处以功能按键中的一个为例作为说明。

void QGTLiftPlatModule::on\_toolButton\_LiftPlat\_MoveUp\_pressed()

{

if (false == g\_pGlobalUnit->IsLiftPlatConnect())

{

QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, tr("Warning"),

"LiftPlat disconnect now, please connect first",

QMessageBox::NoButton | QMessageBox::Ok | QMessageBox::NoButton);

return;

}

g\_pGlobalUnit->m\_nLiftPlat\_NowStatus = LPS\_Up;

}

1. 升降平台数据实时更新显示

将升降平台工作线程中的信号与当前控制界面的槽函数关联，实时输出工作线程中升降平台的详细数据，如电机使能状态、电机当前转动的步数、电机当前转动的总步数、电机的电压、电机的电流、电机的温度、电机的输出脉宽、电机的目标转速、电机的加速度、电机的起始转速、电机当前转速等。具体实现如下所示：

void QGTLiftPlatModule::Update\_ActualInfo\_LiftPlat\_Normal()

{

QString strModbus\_enable = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iModbus\_enable, 10);

QString strEN\_enable = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iEN\_enable % 2, 10);

QString strPU = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iPU, 10);

QString strPU\_all = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iPU\_all, 10);

QString strTarget\_speed = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iTarget\_speed, 10);

QString strAcceleration = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iAcceleration, 10);

QString strSpeed\_start = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iSpeed\_start, 10);

QString strSystem\_DC\_current = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iSystem\_DC\_current / (double)2000, 'f', 1);

QString strSystem\_voltage = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iSystem\_voltage / (double)327, 'f', 1);

QString strSystem\_temperature = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iSystem\_temperature, 10);

QString strSystem\_pwm = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iSystem\_pwm / (double)327, 'f', 1);

QString strSpeed\_actual = QString::number(g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iSpeed\_actual / (double)10, 'f', 1);

if (g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iModbus\_enable == 0)

{

ui.label\_Modbus\_Status\_explain->setText(QString::fromLocal8Bit("禁止"));

}

else

{

ui.label\_Modbus\_Status\_explain->setText(QString::fromLocal8Bit("使能"));

}

if (g\_pGlobalUnit->m\_nLiftPlat\_MotorInfo.iEN\_enable == 0)

{

ui.label\_En\_Status\_explain->setText(QString::fromLocal8Bit("禁止"));

}

else

{

ui.label\_En\_Status\_explain->setText(QString::fromLocal8Bit("使能"));

}

ui.label\_Modbus\_Status->setText(strModbus\_enable);

ui.label\_En\_Status->setText(strEN\_enable);

ui.label\_Pu\_Step->setText(strPU);

ui.label\_Pu\_AllStep->setText(strPU\_all);

ui.label\_Target\_Speed->setText(strTarget\_speed);

ui.label\_Acceleration->setText(strAcceleration);

ui.label\_Speed\_start->setText(strSpeed\_start);

ui.label\_Motor\_current->setText(strSystem\_DC\_current);

ui.label\_Motor\_voltage->setText(strSystem\_voltage);

ui.label\_Motor\_voltage->setText(strSystem\_temperature);

ui.label\_Motor\_PWM->setText(strSystem\_pwm);

ui.label\_Motor\_Speed\_actual->setText(strSpeed\_actual);

}

1. 升降平台其它数据信息实时更新显示

通过在升降平台上增加拉线编码器，可获取升降平台的实时高度信息。升降平台使用的拉线编码器，实则由底盘模块控制。其数据信息，也是通过读取底盘模块的数据获取。具体实现如下所示：

void QGTLiftPlatModule::Update\_ActualInfo\_LiftPlat\_Other()

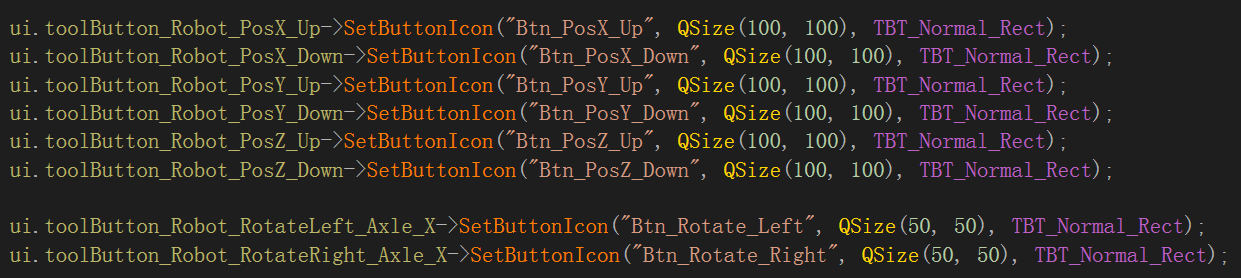
{

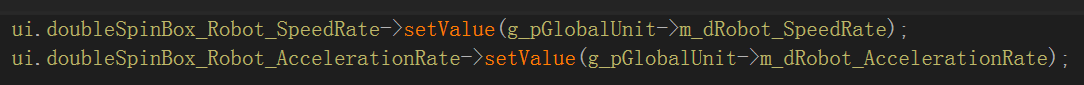
QString strHeight\_actual = QString::number(g\_pGlobalUnit->m\_nChasisInfo.iHeight\_LiftPlat);

ui.label\_Plat\_Height->setText(strHeight\_actual);

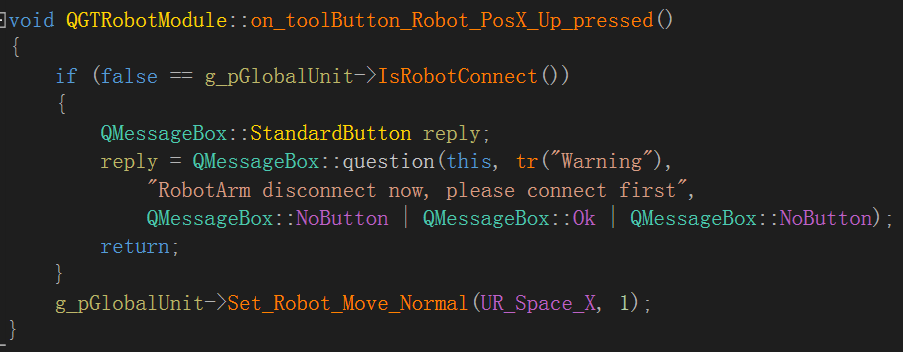
}

1. 构造函
2. 构造函
3. 机械臂控制界面类（QGTRobotModule）
4. 构造函数

界面上定义了多个QGTToolButton按钮对象，在当前构造函数中，调用QGTToolButton按钮类的SetButtonIcon函数，将各个按钮初始化。

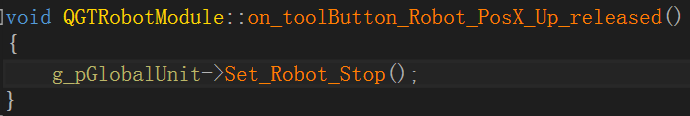
将配置文件中读取的机械臂速率（m\_dRobot\_SpeedRate）和机械臂加速率（m\_dRobot\_AccelerationRate）的值赋值给界面上对应的SpinBox控件框。

1. X方向开始正向移动

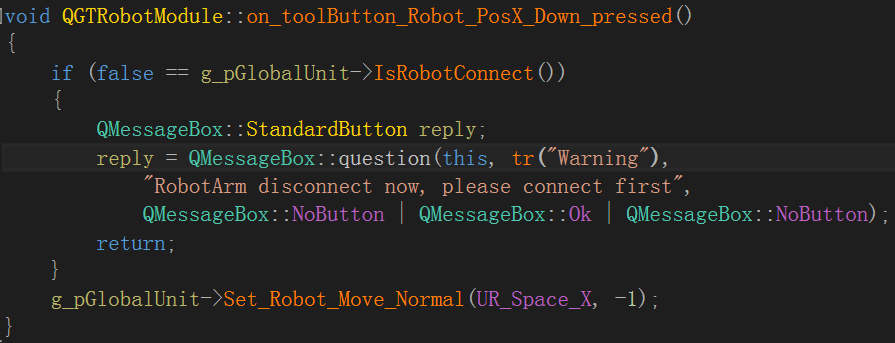
调用全局类中的IsRobotConnect()函数，判断机械臂是否已经连接。若正常连接，才可做相关控制。调用全局类的Set\_Robot\_Move\_Normal函数，使机械臂在基坐标系的X方向开始正向移动。

1. X方向停止正向移动

调用全局类的Set\_Robot\_Stop函数，使机械臂在基坐标系的X方向停止正向移动。

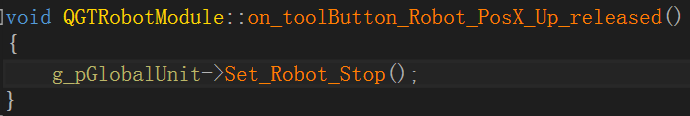


1. X方向开始反向移动

调用全局类中的IsRobotConnect()函数，判断机械臂是否已经连接。若正常连接，才可做相关控制。调用全局类的Set\_Robot\_Move\_Normal函数，使机械臂在基坐标系的X方向开始反向移动。

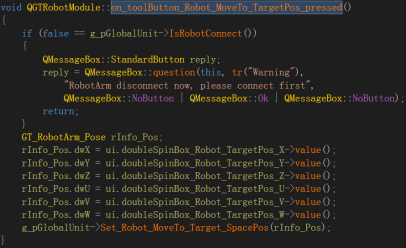
1. X方向停止反向移动

调用全局类的Set\_Robot\_Stop函数，使机械臂在基坐标系的X方向停止反向移动。

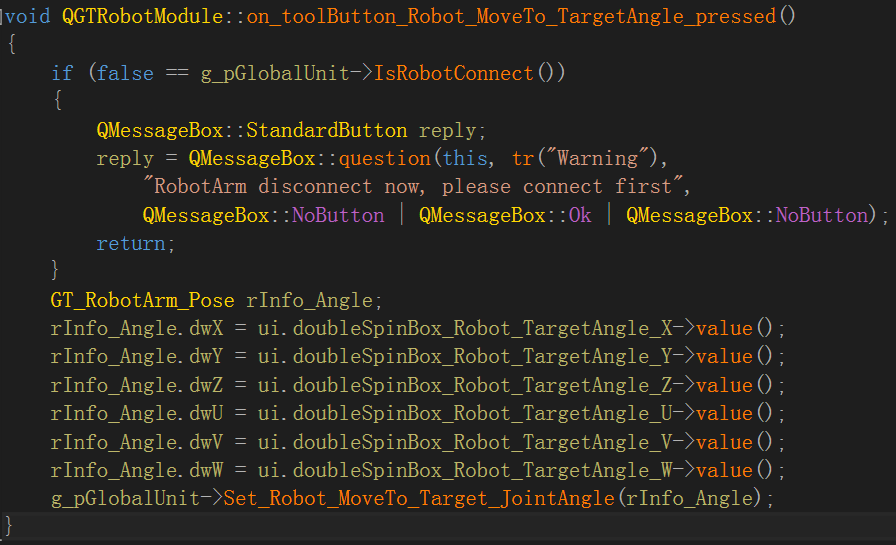


1. 控制机械臂移动到指定坐标位置

定义一个机械臂坐标位姿GT\_RobotArm\_Pose对象rInfo\_Pos，将界面上的指定坐标位置的坐标值赋值给rInfo\_Pos的各参数。调用全局类的Set\_Robot\_MoveTo\_Target\_SpacePos函数，使机械臂位置伺服运动到指定的坐标位置。

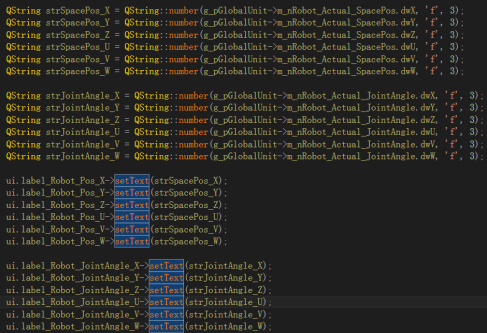


1. 控制机械臂移动到指定的角度位置

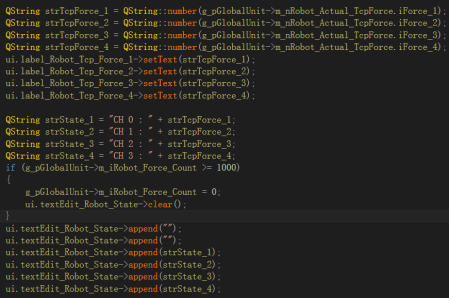
定义一个机械臂坐标位姿GT\_RobotArm\_Pose对象rInfo\_Angle，将界面上的指定坐标位置的坐标值赋值给rInfo\_Angle的各参数。调用全局类的Set\_Robot\_MoveTo\_Target\_SpacePos函数，使机械臂移动到指定的角度位置。

1. 显示机械臂数据

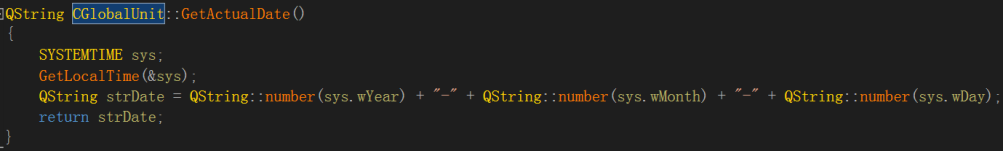
将机械臂的当前坐标姿态和关节角度信息转换成字符串格式，显示在界面上的对应的QLabel控件上。



将读取到的机械臂末端tcp处的压力传感器的值转换成字符串格式，显示在界面上对应的QLabel控件上。

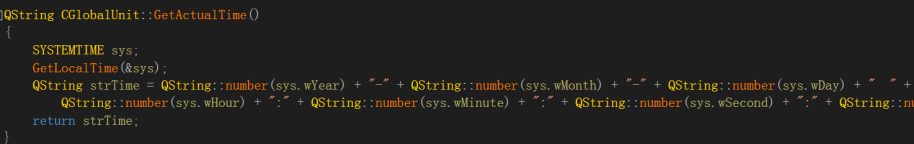


1. 其它
2. 其它
3. 手柄控制模块
4. 全局类（CGlobalUnit）
5. 获取当前日期

定义一个系统时间SYSTEMTIME对象sys，调用windows的API函数GetLocalTime，获取当前系统的时间。将获取的时间的年、月、日转换成字符串格式返回。

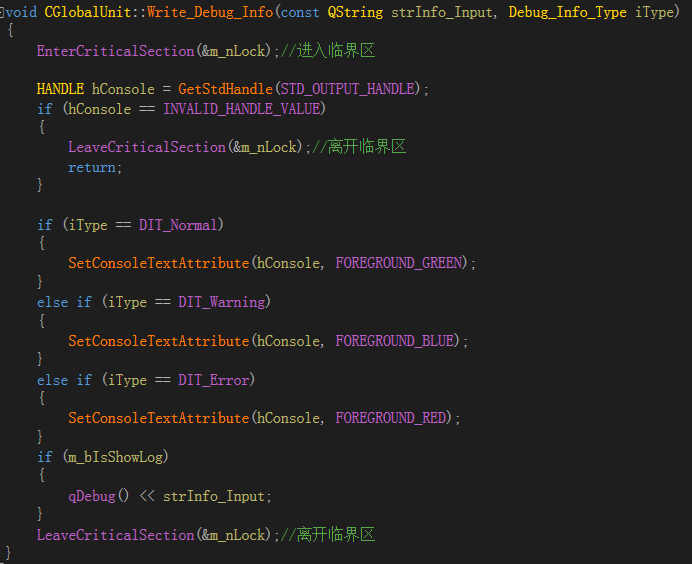
1. 获取当前时间

定义一个系统时间SYSTEMTIME对象sys，调用windows的API函数GetLocalTime，获取当前系统的时间。将获取的时间的年、月、日、时、分、秒转换成字符串格式返回。



1. 输出调试信息

在其它项目程序中有详细说明，此处不再赘述。

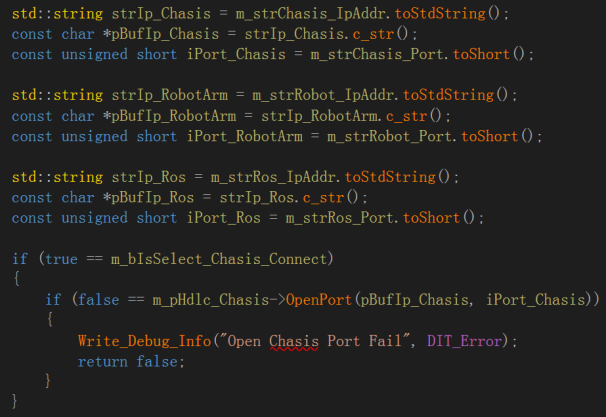


1. 读写配置文件

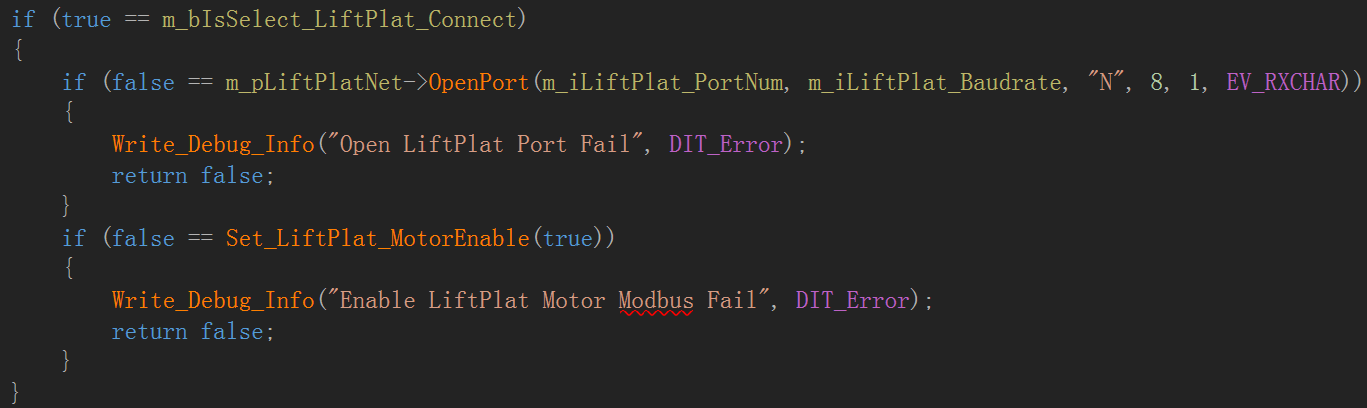
在其它项目程序中有详细说明，此处不再赘述。

1. 设置系统连接到各模块

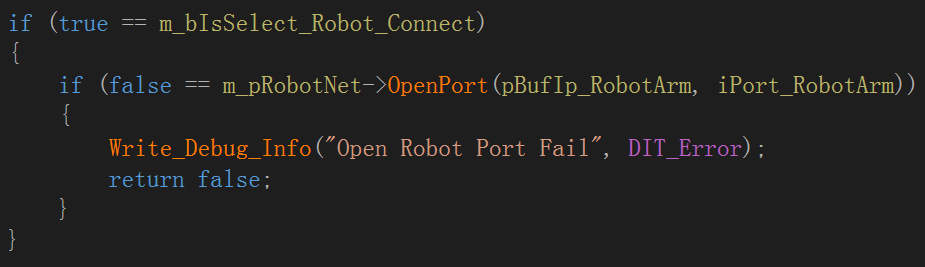
判断配置文件中读取的m\_bIsSelect\_Chasis\_Connect（是否需要连接底盘模块）的值，确定是否需要调用CChasisNet（底盘通信类）的OpenPort函数连接到底盘模块。



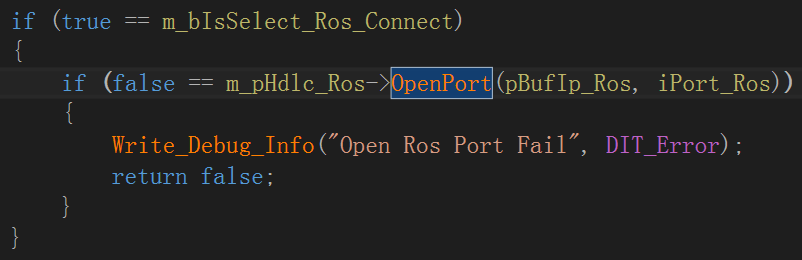
判断配置文件中读取的m\_bIsSelect\_LiftPlat\_Connect（是否需要连接升降平台模块）的值，确定是否需要调用CMotorModbus（升降平台通信类）的OpenPort函数连接到升降平台模块。连接成功后，还需要调用Set\_LiftPlat\_MotorEnable函数，设置升降电机使能。



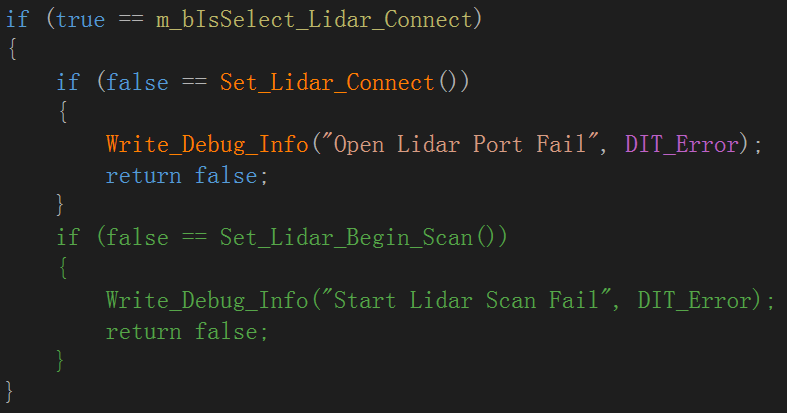
判断配置文件中读取的m\_bIsSelect\_Robot\_Connect（是否需要连接机械臂模块）的值，确定是否需要调用CURRobotArmNet（机械臂通信类）的OpenPort函数连接到机械臂模块。



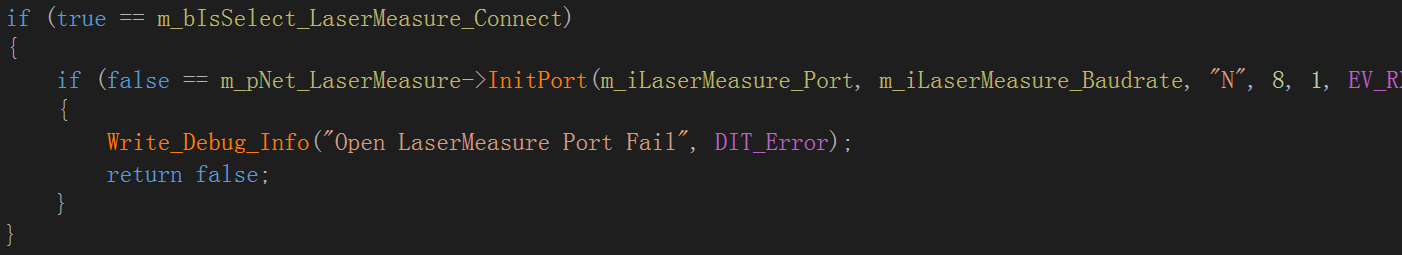
判断配置文件中读取的m\_bIsSelect\_Ros\_Connect（是否需要连接Ros系统模块）的值，确定是否需要调用CRosplatNet（Ros系统通信类）的OpenPort函数连接到Ros系统模块。



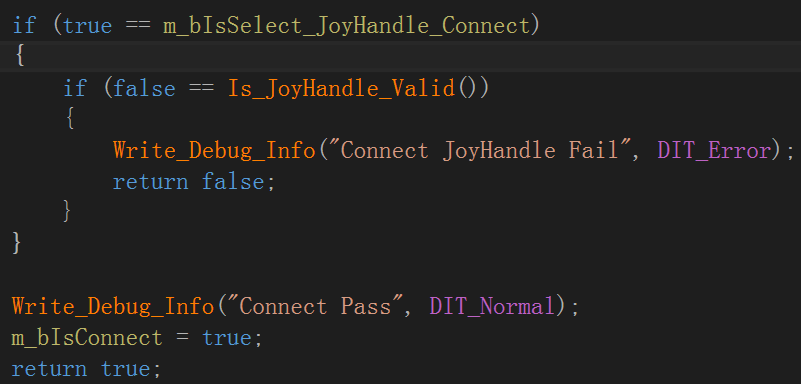
判断配置文件中读取的m\_bIsSelect\_Lidar\_Connect（是否需要连接激光雷达模块）的值，确定是否需要调用全局类中私有函数Set\_Lidar\_Connect()连接到激光雷达模块。



判断配置文件中读取的m\_bIsSelect\_LaserMeasure\_Connect（是否需要连接激光测距模块）的值，确定是否需要调用CSerialPort（串口通信类）中的InitPort函数连接到激光测距模块。

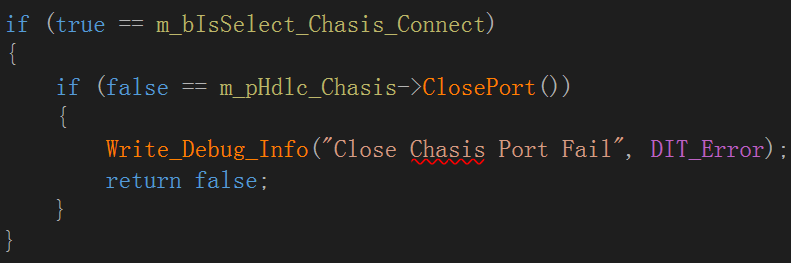


判断配置文件中读取的m\_bIsSelect\_JoyHandle\_Connect（是否需要连接手柄控制模块）的值，确定是否需要调用全局类中的Is\_JoyHandle\_Valid()函数连接到手柄控制模块。

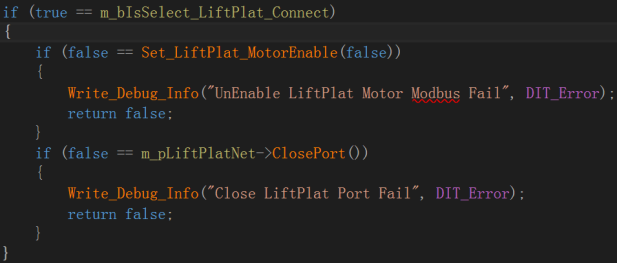


1. 设置系统断开与各模块的连接

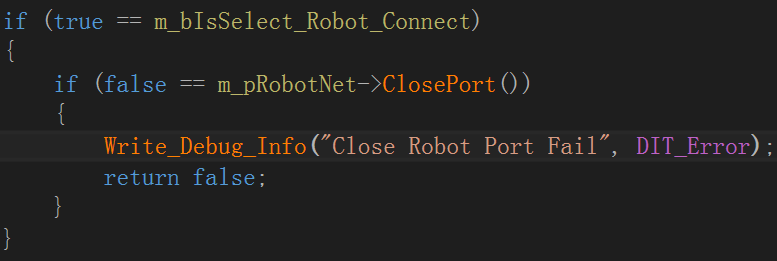
判断配置文件中读取的m\_bIsSelect\_Chasis\_Connect（是否需要连接底盘模块）的值，确定是否需要调用CChasisNet（底盘通信类）的ClosePort函数断开与底盘模块的连接。



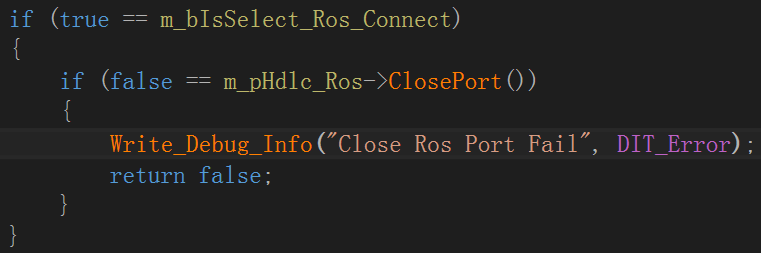
判断配置文件中读取的m\_bIsSelect\_LiftPlat\_Connect（是否需要连接升降平台模块）的值，确定是否需要调用CMotorModbus（升降平台通信类）的ClosePort函数断开与升降平台模块的连接。注意在关闭串口前，还需要先调用Set\_LiftPlat\_MotorEnable函数，设置升降电机不使能。



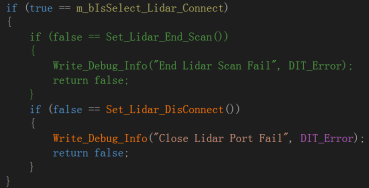
判断配置文件中读取的m\_bIsSelect\_Robot\_Connect（是否需要连接机械臂模块）的值，确定是否需要调用CURRobotArmNet（机械臂通信类）的ClosePort函数断开与机械臂模块的连接。



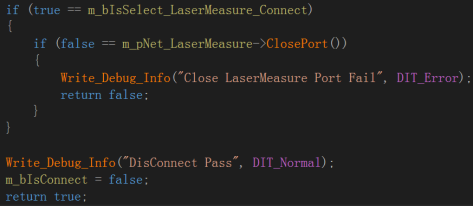
判断配置文件中读取的m\_bIsSelect\_Ros\_Connect（是否需要连接Ros系统模块）的值，确定是否需要调用CRosplatNet（Ros系统通信类）的ClosePort函数断开与Ros系统模块的连接。



判断配置文件中读取的m\_bIsSelect\_Lidar\_Connect（是否需要连接激光雷达模块）的值，确定是否需要调用全局类中私有函数Set\_Lidar\_DisConnect()断开与激光雷达模块的连接。

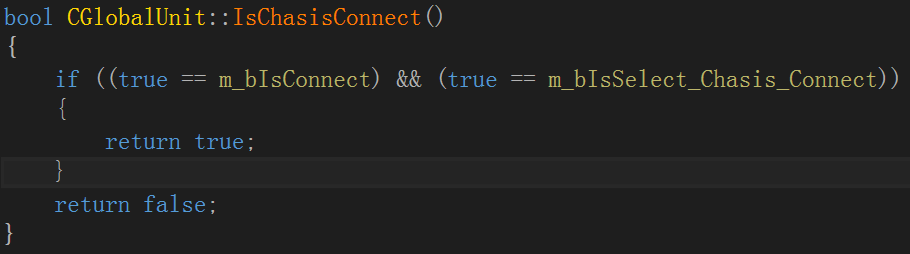


判断配置文件中读取的m\_bIsSelect\_LaserMeasure\_Connect（是否需要连接激光测距模块）的值，确定是否需要调用CSerialPort（串口通信类）中的ClosePort函数断开与激光测距模块的连接。



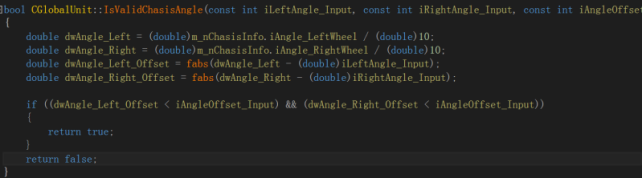
1. 判断底盘模块是否已经连接

判断m\_bIsConnect（系统当前是否已经连接）和m\_bIsSelect\_Chasis\_Connect（当前是否选择了底盘模块）的与运算结果，即为底盘模块是否已经连接。



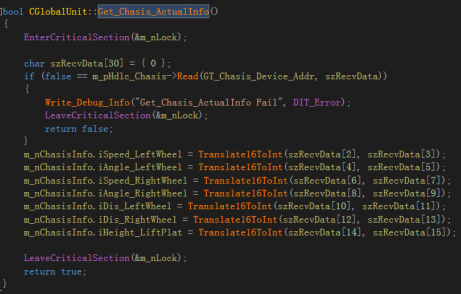
1. 判断底盘舵轮选择的角度是否达到需求的角度

计算输入的目标角度与当前舵轮的实际角度的差值，并与输入的指定的角度偏差比较，确定底盘舵轮是否旋转到指定目标角度。



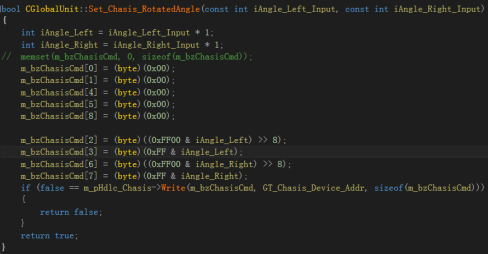
1. 获取底盘的相关数据信息

调用CChasisNet（底盘通信类）的Read函数，读取底盘通信串口缓冲区的数据。调用全局类的Translate16ToInt函数，将16进制数据转换成int类型数据，即为底盘的左轮速度、左轮角度、右轮速度、右轮角度、左轮行程、右轮行程、升降平台高度等信息。



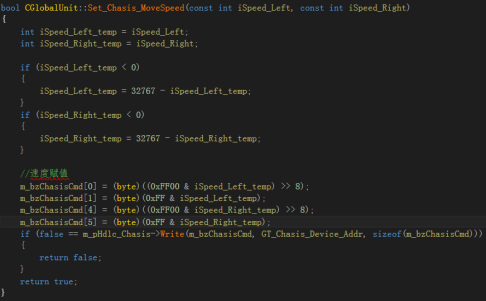
1. 设置底盘舵轮转动指定角度

将指定的左右轮的角度转换成16进制数据，赋值给m\_bzChasisCmd（底盘通信指令），调用CChasisNet（底盘通信类）的Write函数，将配置好的通信指令发送给底盘模块。



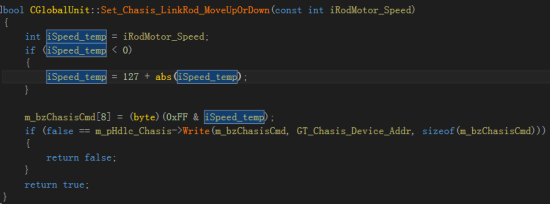
1. 设置底盘舵轮的移动速度

将指定的左右轮的速度转换成16进制数据，赋值给m\_bzChasisCmd（底盘通信指令），调用CChasisNet（底盘通信类）的Write函数，将配置好的通信指令发送给底盘模块。



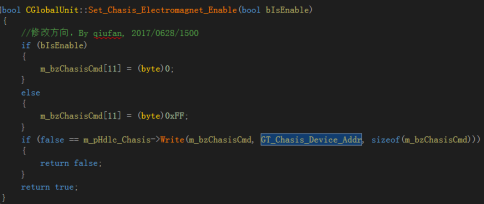
1. 设置底盘的连杆的升降

将输入的连杆升降速度变量iRodMotor\_Speed转换成16进制数据，赋值给m\_bzChasisCmd（底盘通信指令），调用CChasisNet（底盘通信类）的Write函数，将配置好的通信指令发送给底盘模块。



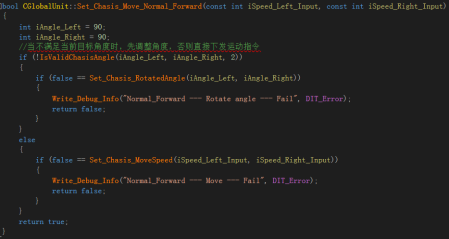
1. 设置底盘模块的电磁阀开关

根据输入参数bIsEnable，确定m\_bzChasisCmd（底盘通信指令）中电磁阀控制的值。调用CChasisNet（底盘通信类）的Write函数，将配置好的通信指令发送给底盘模块。



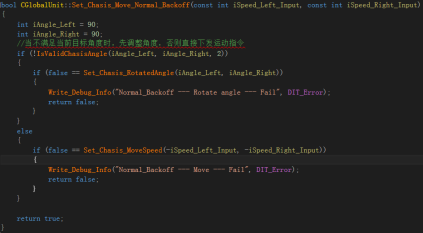
1. 设置底盘以指定的速度正常前进

底盘前进时，舵轮角度需要调整到90度。调用全局类中的IsValidChasisAngle函数判断底盘舵轮是否已经调整到目标角度。



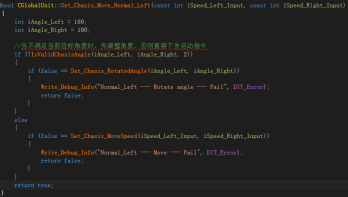
1. 设置底盘以指定的速度正常后退

底盘后退时，舵轮角度需要调整到90度。调用全局类中的IsValidChasisAngle函数判断底盘舵轮是否已经调整到目标角度。若舵轮旋转到90度，则调用全局类的Set\_Chasis\_MoveSpeed函数，且设置输入的速度参数均为负值，使底盘后退。



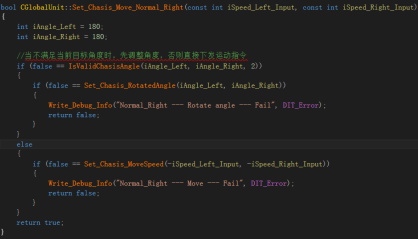
1. 设置底盘以指定的速度正常左移

底盘左移时，舵轮角度需要调整到180度。调用全局类中的IsValidChasisAngle函数判断底盘舵轮是否已经调整到目标角度。若舵轮旋转到180度，则调用全局类的Set\_Chasis\_MoveSpeed函数，且设置输入的速度参数均为正值，使底盘左移。



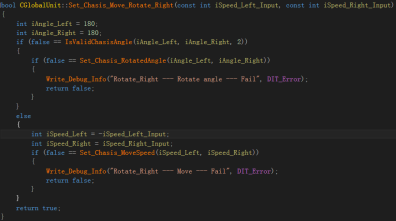
1. 设置底盘以指定的速度正常右移

底盘右移时，舵轮角度需要调整到180度。调用全局类中的IsValidChasisAngle函数判断底盘舵轮是否已经调整到目标角度。若舵轮旋转到180度，则调用全局类的Set\_Chasis\_MoveSpeed函数，且设置输入的速度参数均为负值，使底盘右移。



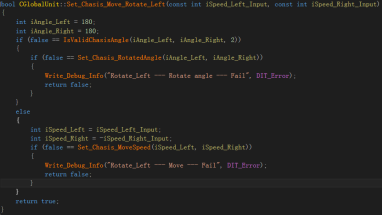
1. 设置底盘以指定的速度顺时针旋转

底盘旋转时，舵轮角度需要调整到180度。调用全局类中的IsValidChasisAngle函数判断底盘舵轮是否已经调整到目标角度。若舵轮旋转到180度，则调用全局类的Set\_Chasis\_MoveSpeed函数，且设置输入的左轮速度为负值，右轮速度为正值，使底盘顺时针旋转。



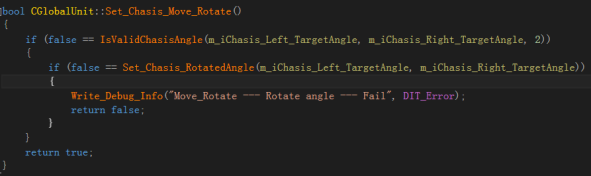
1. 设置底盘以指定的速度逆时针旋转

底盘旋转时，舵轮角度需要调整到180度。调用全局类中的IsValidChasisAngle函数判断底盘舵轮是否已经调整到目标角度。若舵轮旋转到180度，则调用全局类的Set\_Chasis\_MoveSpeed函数，且设置输入的左轮速度为正值，右轮速度为负值，使底盘逆时针旋转。



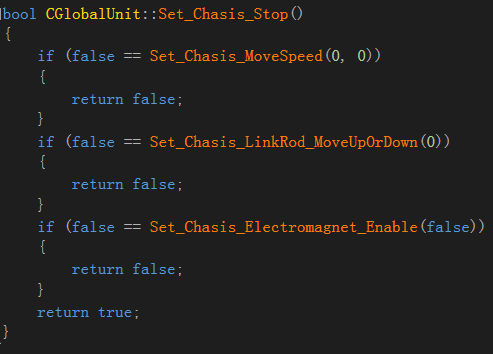
1. 设置底盘的舵轮旋转

调用全局类中的IsValidChasisAngle函数判断底盘舵轮是否已经调整到指定的目标角度。若为到达指定角度，则会调用全局类中Set\_Chasis\_RotatedAngle函数，使舵轮旋转至指定的目标角度。



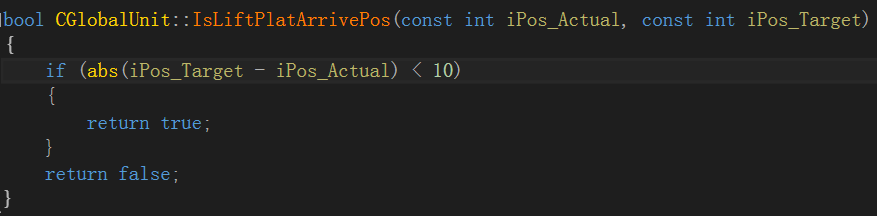
1. 设置底盘停止运动

调用全局类中的Set\_Chasis\_MoveSpeed，将输入的舵轮速度均设置为0，使舵轮停止移动。同时调用Set\_Chasis\_LinkRod\_MoveUpOrDown(0)函数，使连杆停止升降。并调用Set\_Chasis\_Electromagnet\_Enable(false)函数，使电磁阀关闭。



1. 判断升降平台是否到达指定位置

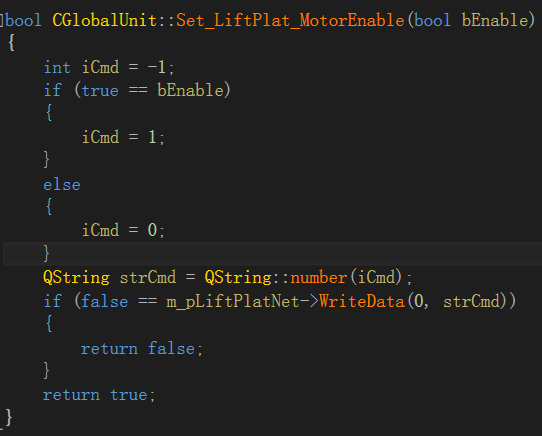
通过判断升降平台当前高度与指定的高度的绝对差值是否小于10mm，判断出升降平台当前是否有到达指定高度位置。



1. 获取升降平台电机的数据信息

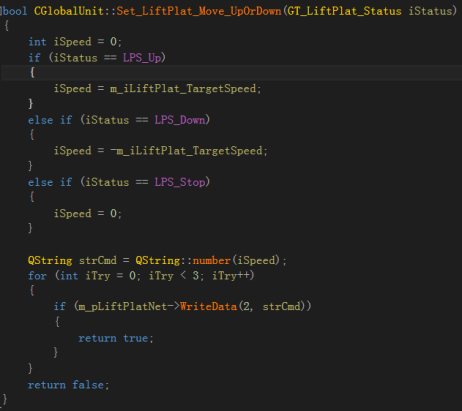
调用CMotorModbus（升降平台串口通信类）的ReadData函数，读取串口缓冲区的数据。根据升降平台电机驱动器的协议格式，解析出获取的电机的电压、电流、速度、脉冲数、当前转速等相关信息。

1. 设置升降平台电机使能



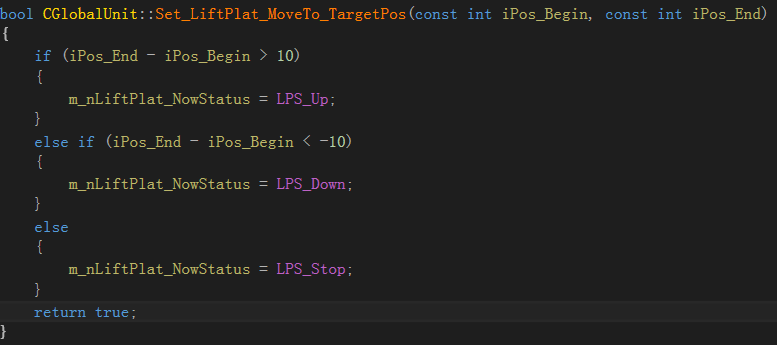
1. 设置升降平台的升降

根据输入参数iStatus的状态，确定当前需要设定的升降平台的转动速度值。将目标转动速度转换成字符串格式数据后，调用CMotorModbus（升降平台串口通信类）的WriteData函数，依据升降平台的电机驱动协议格式，将指令发送至升降平台电机驱动，使升降平台执行指定的目标动作。



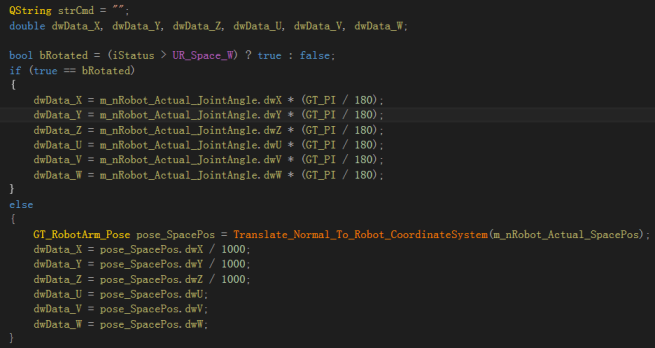
1. 设置升降平台从指定的初始位置运动至指定的终点位置

根据输入参数iPos\_End（指定的终点位置）和输入参数iPos\_Begin（指定的初始位置）的差值，判断升降平台当前需要执行的目标状态动作。

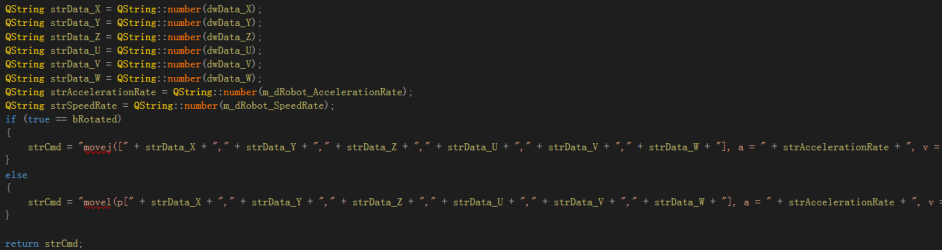


1. 获取机械臂运动指令

根据输入参数iStatus（机械臂关节轴状态）和输入参数iDirection（运动方向）的值，判断当前是位置伺服还是角度伺服，用于确定X、Y、Z、U、V、W的值。

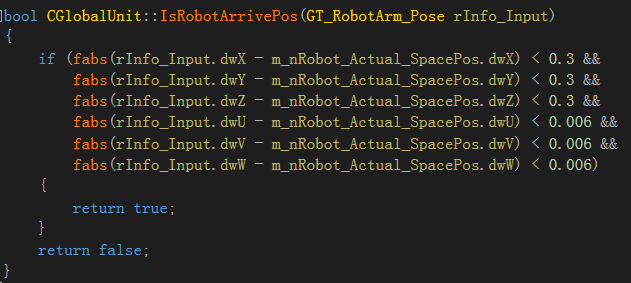


将确定好的机械臂的6个关节轴的坐标（或是角度）值，以及界面上选择的机械臂的加速度和速率值转换成指定的字符串格式指令。



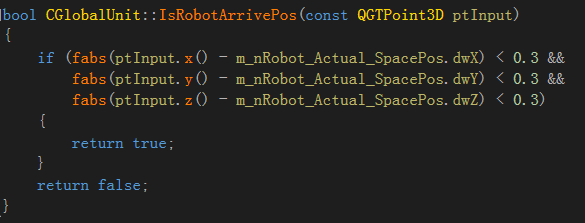
1. 判断机械臂是否达到指定的坐标位置

通过比较输入参数rInfo\_Input的坐标姿态值与机械臂当前的坐标姿态，判断机械臂是否到达指定的坐标姿态。坐标差值在0.3mm以内，且姿态差值在0.06以内，则认为机械臂已经到达指定的坐标位置。



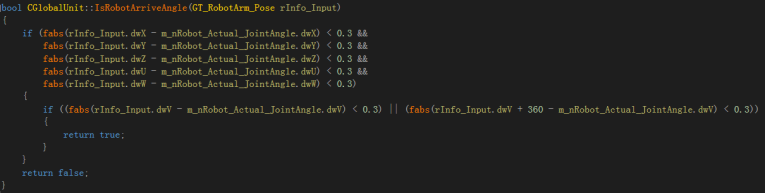
1. 判断机械臂是否达到指定的坐标位置

通过比较输入的三维坐标点参数ptInput的坐标值与机械臂当前的坐标值，判断机械臂是否到达指定的坐标姿态。若坐标差值在0.3mm以内，则认为机械臂已经到达指定的坐标位置。



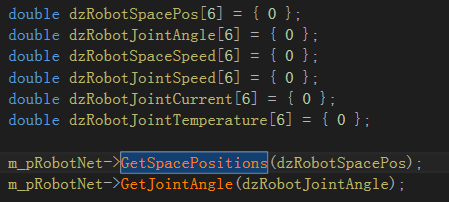
1. 判断机械臂是否到达指定的角度位置

通过比较输入参数rInfo\_Input的关节角度值与机械臂当前的各关节角度值，判断机械臂是否到达指定的角度位置。各关节角度差值在0.3mm以内，则认为机械臂已经到达指定的角度位置。

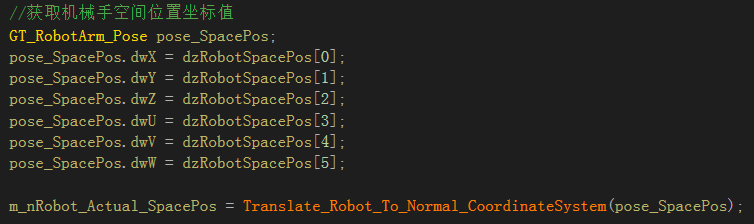


1. 获取机械臂的当前状态信息

调用CURRobotArmNet（机械臂通信类）的GetSpacePositions函数，获取机械臂当前的坐标位姿。调用CURRobotArmNet（机械臂通信类）的GetJointAngle函数，获取机械臂当前的关节角度。

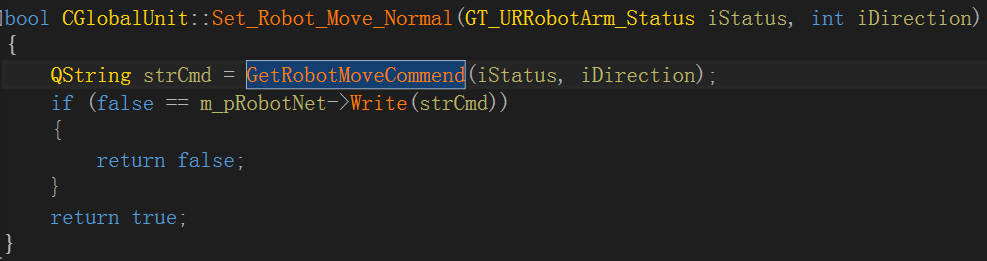


调用全局类的Translate\_Robot\_To\_Normal\_CoordinateSystem函数，将机械臂当前坐标值从基坐标系转换到标准坐标系下。



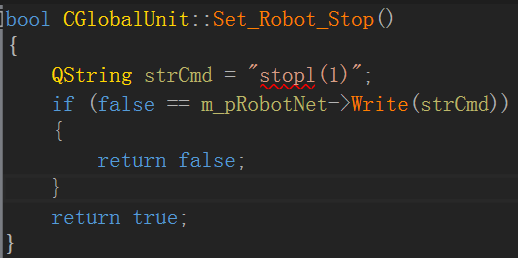
1. 设置机械臂正常运动

调用全局类的GetRobotMoveCommend函数，获取当前机械臂运动指令。调用CURRobotArmNet（机械臂通信类）的Write函数，发送运动指令给机械臂控制器，使机械臂按输入的指定关节轴和指定方向正常运动。



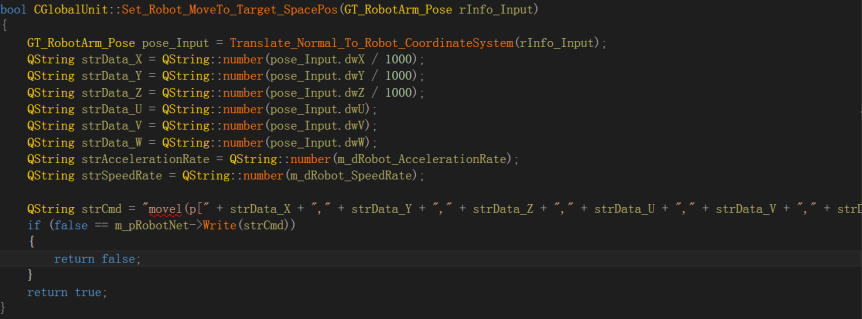
1. 设置机械臂停止运动

将机械臂运动指令赋值为"stopl(1)"，调用CURRobotArmNet（机械臂通信类）的Write函数，发送运动指令给机械臂控制器，使机械臂停止运动。



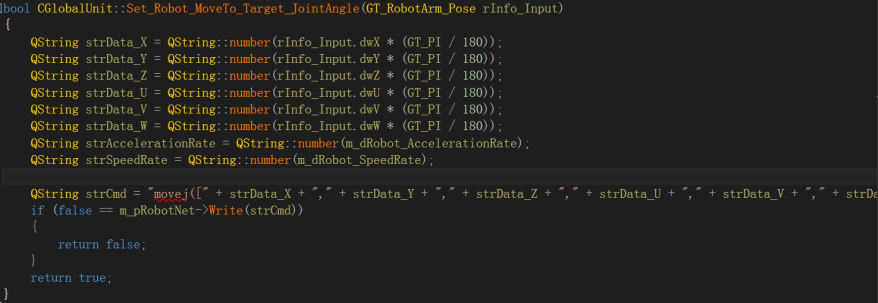
1. 设置机械臂运动到指定的坐标位置

调用全局类的Translate\_Normal\_To\_Robot\_CoordinateSystem函数，将输入的坐标位置，从标准坐标系下转换成机械臂基坐标系下。并将坐标值转换成指定格式的字符串指令，调用CURRobotArmNet（机械臂通信类）的Write函数，发送运动指令给机械臂控制器，使机械臂运动至指定的坐标位置。



1. 设置机械臂运动到指定的关节角度位置

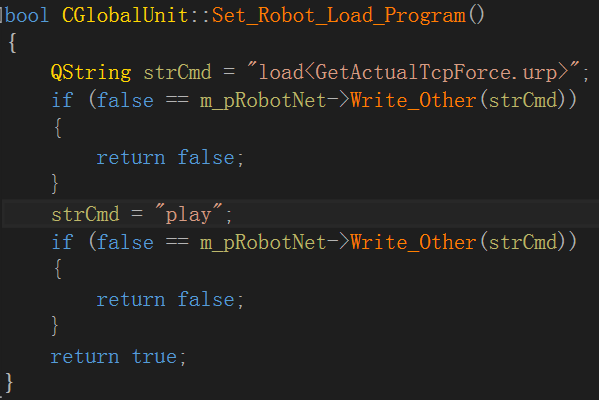
将输入的关节角度参数转换成指定格式的字符串指令，调用CURRobotArmNet（机械臂通信类）的Write函数，发送运动指令给机械臂控制器，使机械臂运动至指定的关节角度位置。



1. 设置机械臂控制箱加载脚本并运行

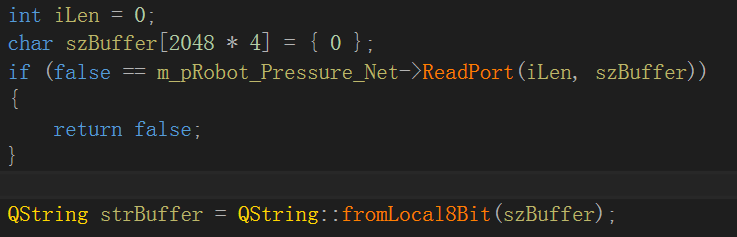
在机械臂示教盘上编写获取机械臂末端的力大小的例程，指定机械臂控制指令为"load<GetActualTcpForce.urp>"，调用CURRobotArmNet（机械臂通信类）的Write函数，发送运动指令给机械臂控制器，使机械臂控制系统加载之前编写的获取力大小的脚本例程。

再次更新机械臂控制指令为"play"，发送至机械臂控制器，使机械臂控制系统运行指定的脚本例程。

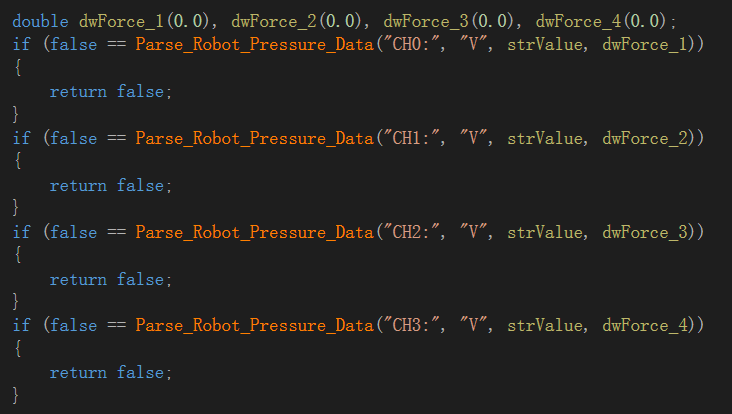


1. 通过外部压力传感器获取机械臂末端力大小

当前使用4个小型压力传感器，并粘贴在机械臂tcp末端处。调用CSerialPort（串口通信类）的ReadPort函数，读取串口缓冲区内的数据strBuffer。



调用全局类的Parse\_Robot\_Pressure\_Data函数，对串口数据strBuffer解析，得到4个压力传感器的压力值。

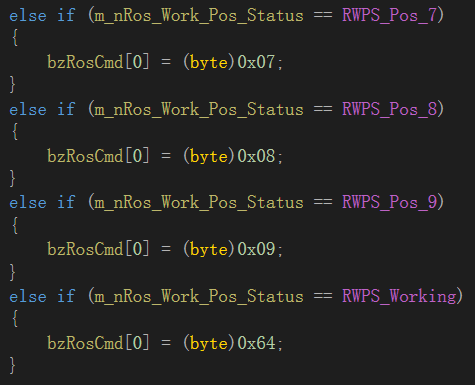
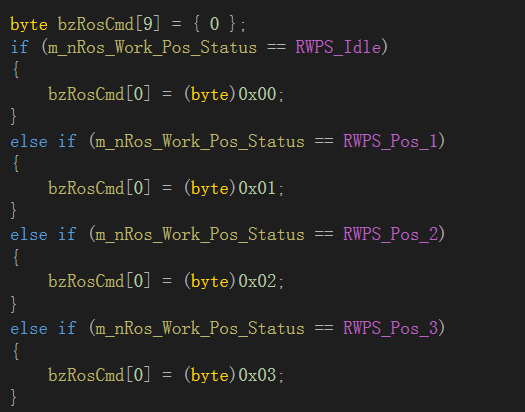


1. 设置Ros系统运行

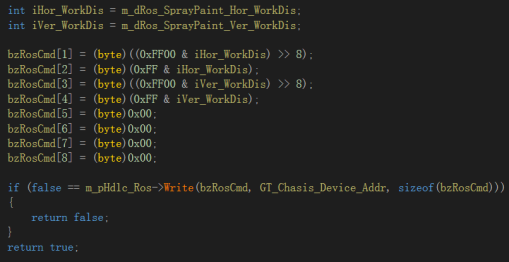
Ros系统是当时为演示喷漆实验，使机械臂运行更流畅，故机将械臂控制部分从windows上位机，转移至linux系统下，由Ros框架控制。上位机主控电脑通过网络通信与linun主机连接，并通过指定的协议格式，告知liunx系统的Ros框执行相应的操作。

演示过程中，约定了左边阴角的上、中、下三个位置状态，正常墙面喷漆的上、中、下三个位置状态，右边阴角的上、中、下三个位置状态。

根据全局变量m\_nRos\_Work\_Pos\_Status（Ros框架下机械臂当前状态）的值，确定当前需要发送至Ros框架的控制指令。

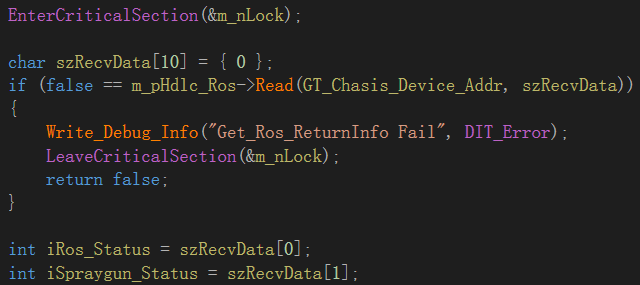


调用CRosplatNet（Ros系统通信类）的Write函数，将指令发送至linux系统下的Ros框架，使机械臂运行。

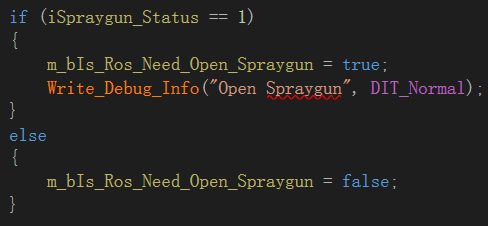
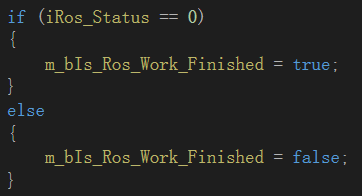


1. 获取Ros框架的返回信息

调用CRosplatNet（Ros系统通信类）的Read函数，读取Ros通信串口缓冲区内的数据szRecvData，得到iRos\_Status（机械臂的状态）和iSpraygun\_Status（喷枪的状态）。

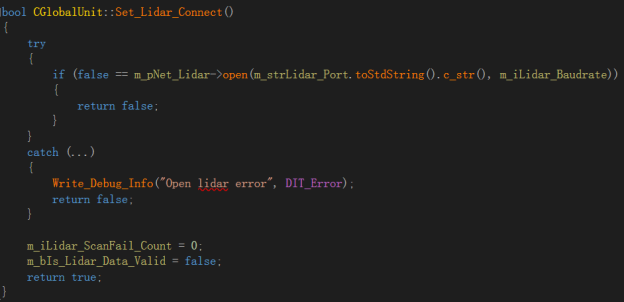


根据iRos\_Status确定m\_bIs\_Ros\_Work\_Finished（Ros平台下单次工作是否完成）的值。根据iSpraygun\_Status确定m\_bIs\_Ros\_Need\_Open\_Spraygun（是否需要打开喷枪）的值。



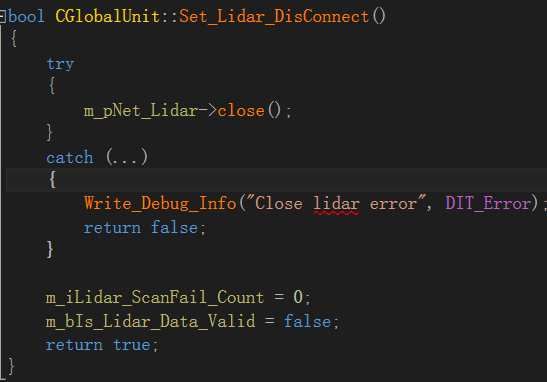
1. 连接激光雷达

调用激光雷达SDK的open函数，打开激光雷达串口。



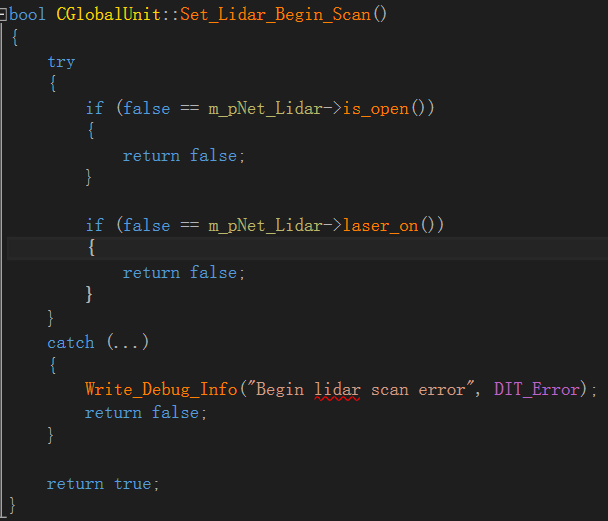
1. 断开与激光雷达的连接

调用激光雷达SDK的close()函数，关闭激光雷达串口。



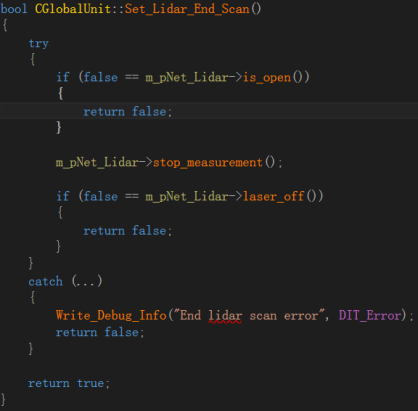
1. 设置激光雷达开始扫描

调用激光雷达SDK的is\_open()函数，判断激光雷达是否已经打开。若已经打开，则调用laser\_on()函数，使激光雷达工作。



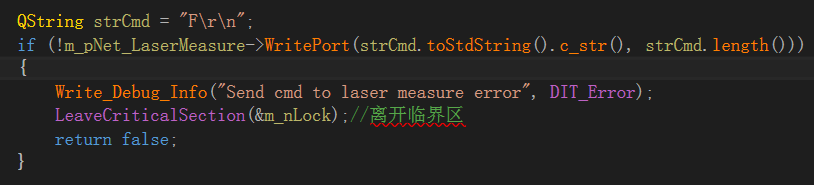
1. 设置激光雷达停止扫描

调用激光雷达SDK的is\_open()函数，判断激光雷达是否已经打开。若已经打开，则先调用stop\_measurement()函数停止测量，再调用laser\_off()函数，使激光雷达停止工作。

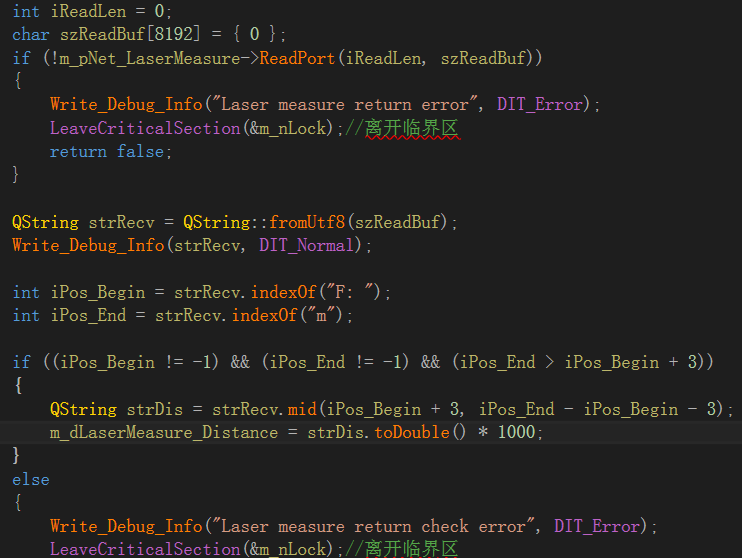


1. 设置激光测距模块开始测量

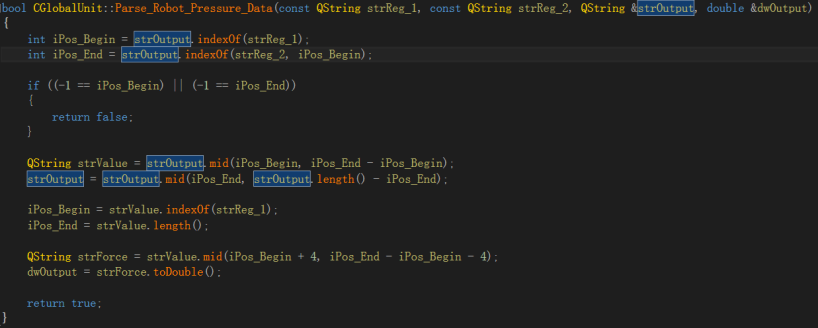
配置通信指令字符串strCmd为"F\r\n"，调用CSerialPort（串口通信类）的WritePort函数，将指令发送给激光测距模块。



调用CSerialPort（串口通信类）的ReadPort函数，读取激光测距模块串口缓冲区内的数据。解析读取的数据，得到当次测量的距离值。



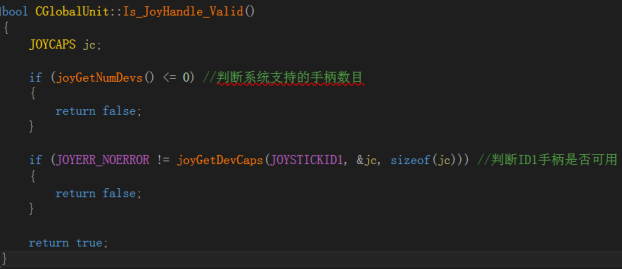
1. 解析机械臂tcp末端的压力传感器数值



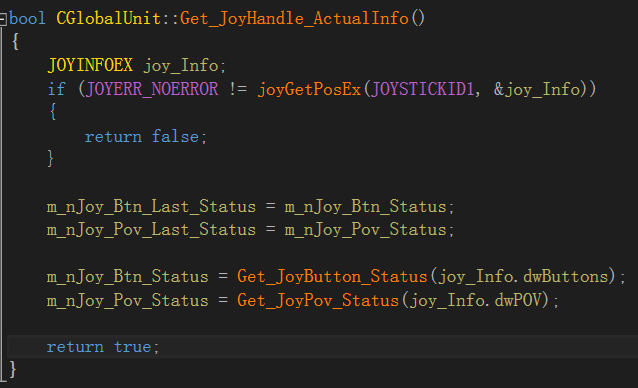
1. 判断手柄是否有效

调用windows的API函数joyGetNumDevs()，获取系统当前支持的手柄数目，判断当前主机系统是否支持手柄控制。

调用windows的API函数joyGetDevCaps，判断ID1手柄是否可用。

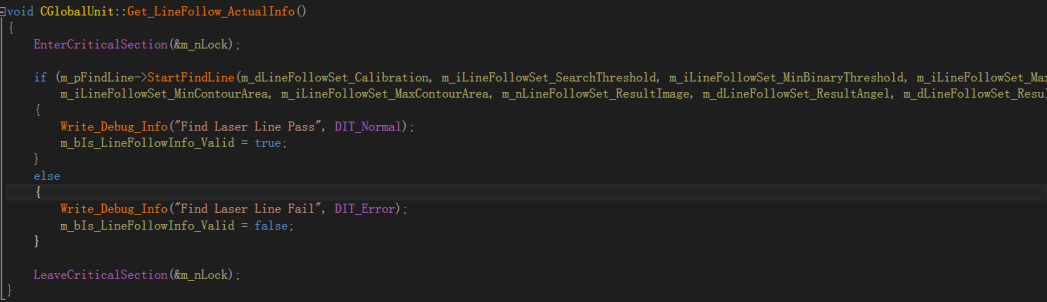


1. 获取手柄的相关状态信息



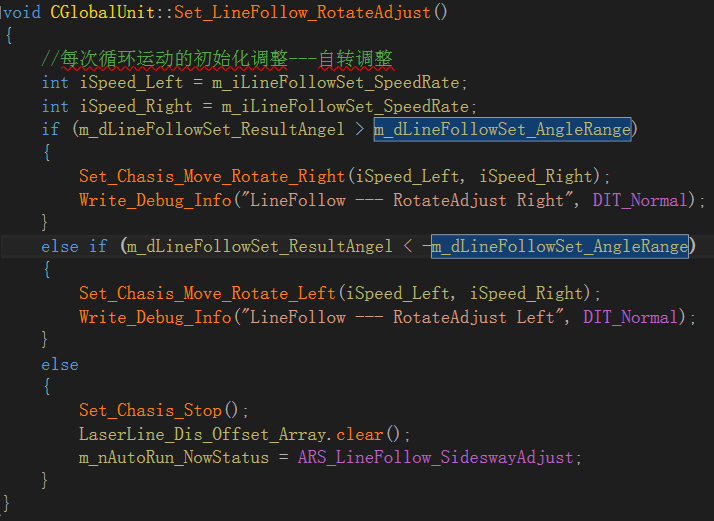
1. 获取循线检测的相关信息（Get\_LineFollow\_ActualInfo）

调用CFindLine（直线检测类）的StartFindLine函数，使相机采集单帧图像并检测图像上的激光线中心线与默认坐标系的偏移角度和偏移距离。



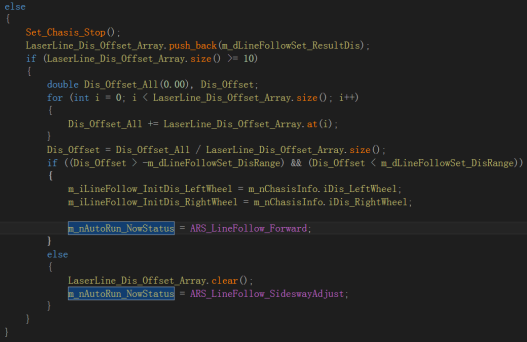
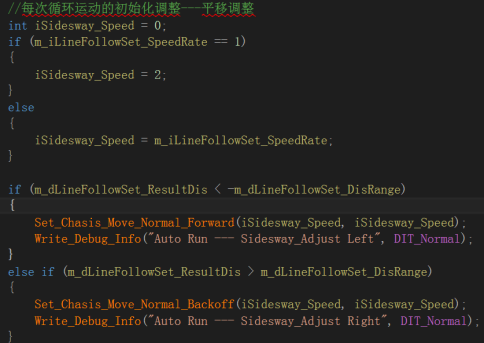
1. 设置循线过程中底盘自转调整

判断m\_dLineFollowSet\_ResultAngel（相机反馈的偏移角度）是否在限定的偏移范围内，若在限定范围内，则调用Set\_Chasis\_Stop()函数停止底盘。若偏移角度大于限定范围的上限值，则调用Set\_Chasis\_Move\_Rotate\_Right函数，使底盘顺时针旋转；若偏移角度小于限定范围的下限值，则调用Set\_Chasis\_Move\_Rotate\_Left函数，使底盘逆时针旋转。



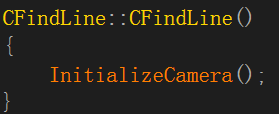
1. 设置循线过程中底盘平移调整

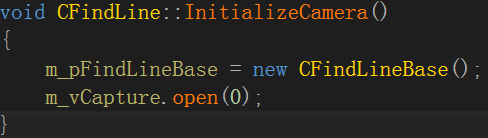
判断m\_dLineFollowSet\_ResultDis（相机反馈的偏移距离）是否在限定的偏移范围内，若偏移距离大于限定范围的上限值，则调用Set\_Chasis\_Move\_Normal\_Forward函数，使底盘前进；若偏移距离小于限定范围的下限值，则调用Set\_Chasis\_Move\_Normal\_Backoff函数，使底盘后退；否则，调用Set\_Chasis\_Stop()函数，使底盘停止运动。



1. 其它
2. 直线检测类
3. 构造函数

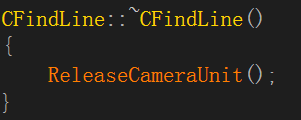
调用当前类中的InitializeCamera()函数，初始化m\_pFindLineBase（直线检测基础类的指针对象），并调用OpenCV的VideoCapture类的open函数，打开当前可用的相机。

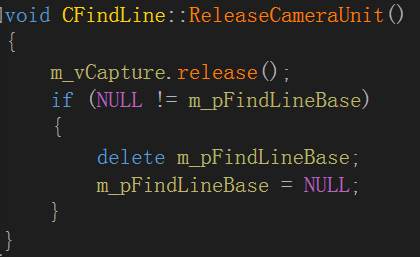




1. 析构函数

调用当前类的ReleaseCameraUnit()函数，释放资源。即调用OpenCV的VideoCapture类的release()函数，释放相机资源。并释放之前构造的m\_pFindLineBase（直线检测基础类的指针对象）。



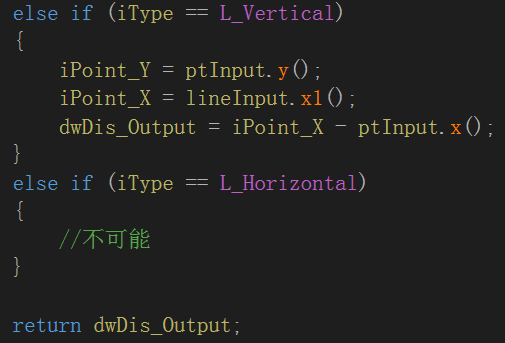


1. 计算点到直线的距离

调用CFindLineBase（直线检测基础类）的QueryLineType函数，查询当前直线的类型。

![C:\Users\Administrator\AppData\Roaming\Tencent\Users\819470192\QQ\WinTemp\RichOle\_TD${LZ[W8W]9](IPQX9(`A.png](data:image/png;base64,)

若当前直线为平行于Y轴的垂线，则点到直线的距离即为直线上的点的横坐标与输入点的横坐标的差值。

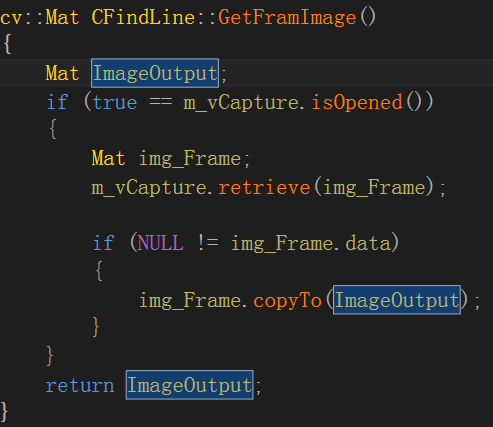


若当前直线为普通类型的直线，则根据直线端点参数计算出当前直线的斜率。再计算输入点到直线的垂线交点，以及当前直线与过输入点的水平直线的交点。通过计算的两个交点与输入点，计算得到输入点到当前直线的距离。



1. 获取单帧图像

使用OpenCV的VideoCapture类中的isOpened()函数，判断相机当前是否已经打开。若已经打开，则调用VideoCapture类中的retrieve函数，从缓冲区内取出一帧图像。

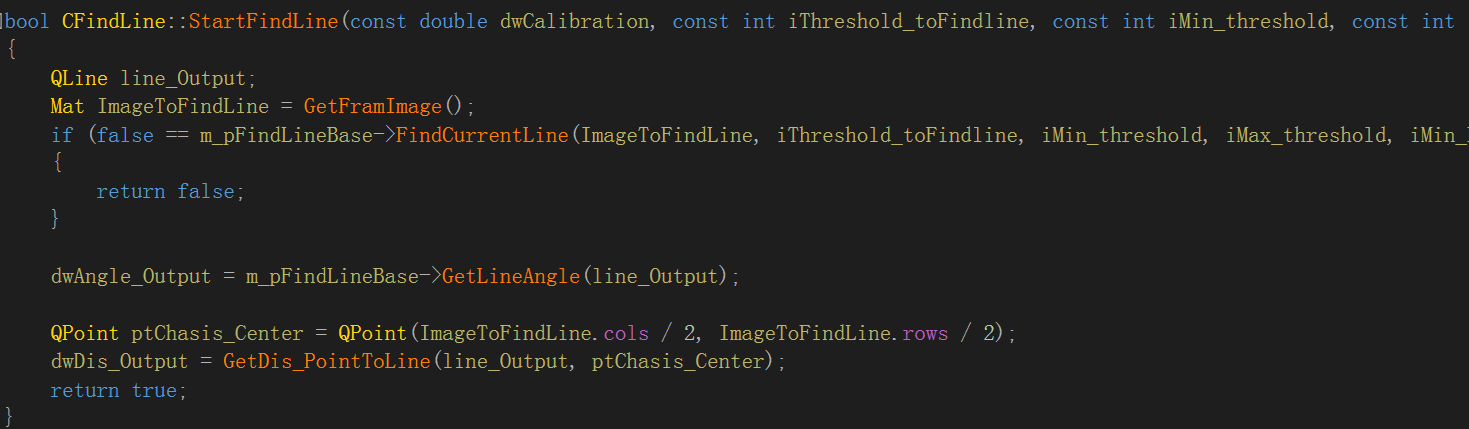


1. 检测直线

调用当前类中的GetFramImage()函数，获取一帧图像。

调用CFindLineBase（直线检测基础类）中的FindCurrentLine函数，检测出当前直线。

再调用CFindLineBase（直线检测基础类）中的GetLineAngle函数，获取当前直线与默认坐标系的Y轴的偏移角度，使用当前类中的GetDis\_PointToLine函数计算得到默认坐标系原点到当前直线的距离。



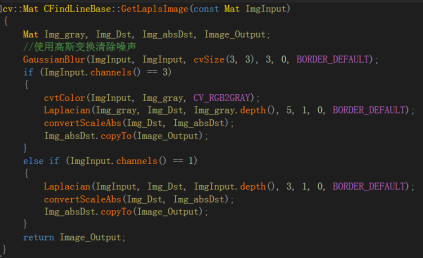
1. 直线检测基础类（CFindLineBase）
2. 边缘检测

使用blur函数对输入图像进行方框滤波，再使用Canny函数，检测方框滤波后的图像的边缘，阈值大小设置为120。



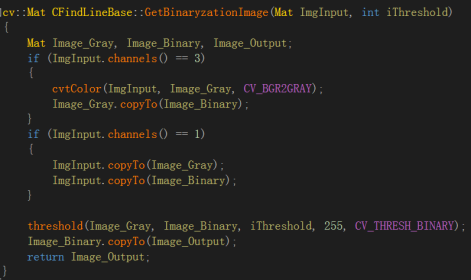
1. 拉普拉斯滤波

先使用GaussianBlur函数对输入图像做高斯滤波处理，清除图像噪点。判断高斯滤波后的图像的通道类型，若为RGB三通道图像，则需要使用cvtColor函数将图像转换为单通道的灰度图。再调用Laplacian函数，对图像做拉普拉斯变换。



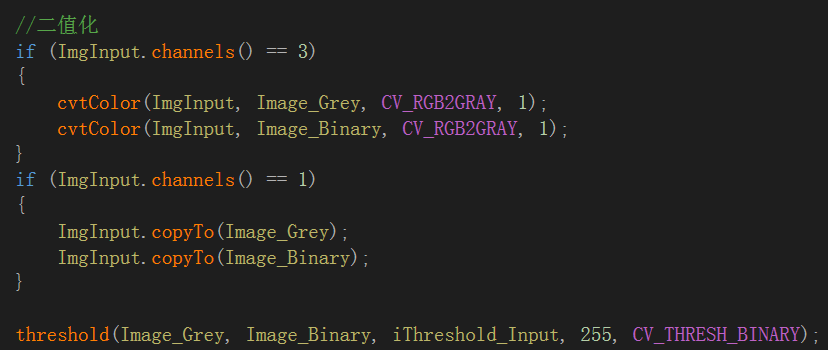
1. 图像二值化

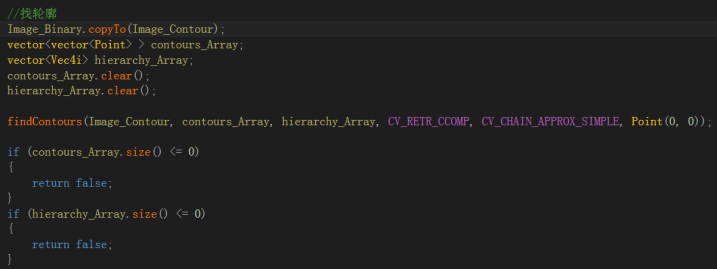
使用cvtColor函数将输入图像转换为单通道的灰度图，再调用threshold函数，根据输入的阈值大小，对当前图像做二值化处理，得到黑白灰度图。

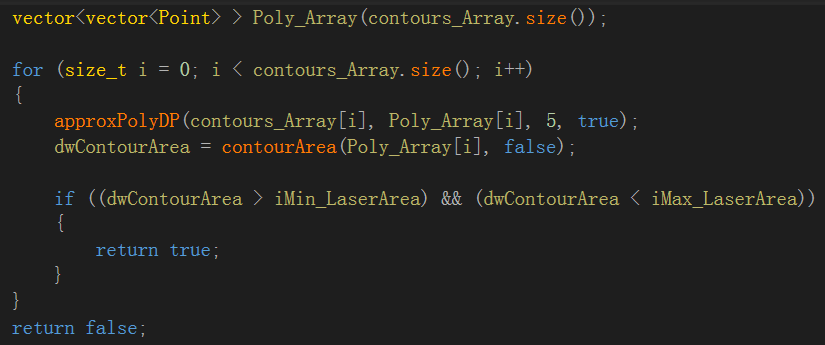


1. 搜索图像上的轮廓

使用cvtColor函数将输入图像转换为单通道灰度图后，使用threshold函数对图像做二值化处理。再调用findContours函数，搜索当前图像上的所有轮廓，并计算所有轮廓的面积大小，取出在面积阈值范围内的轮廓。

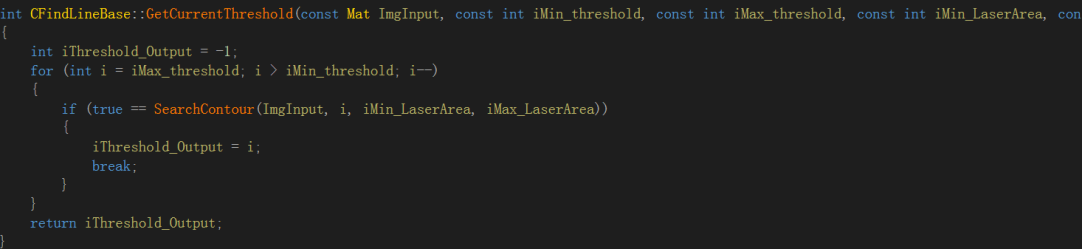






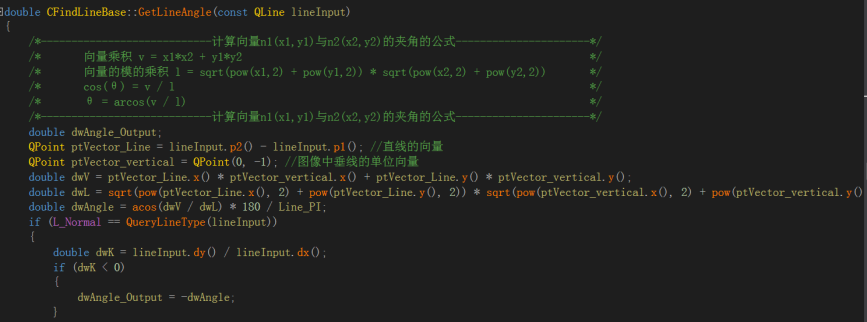
1. 获取适合当前图像二值化的阈值

在for循环中，将阈值从阈值范围的上限不断减小，调用当前类中的SearchContour函数，判断当前阈值下，是否存在有效的轮廓。若存在，则输出当前阈值。

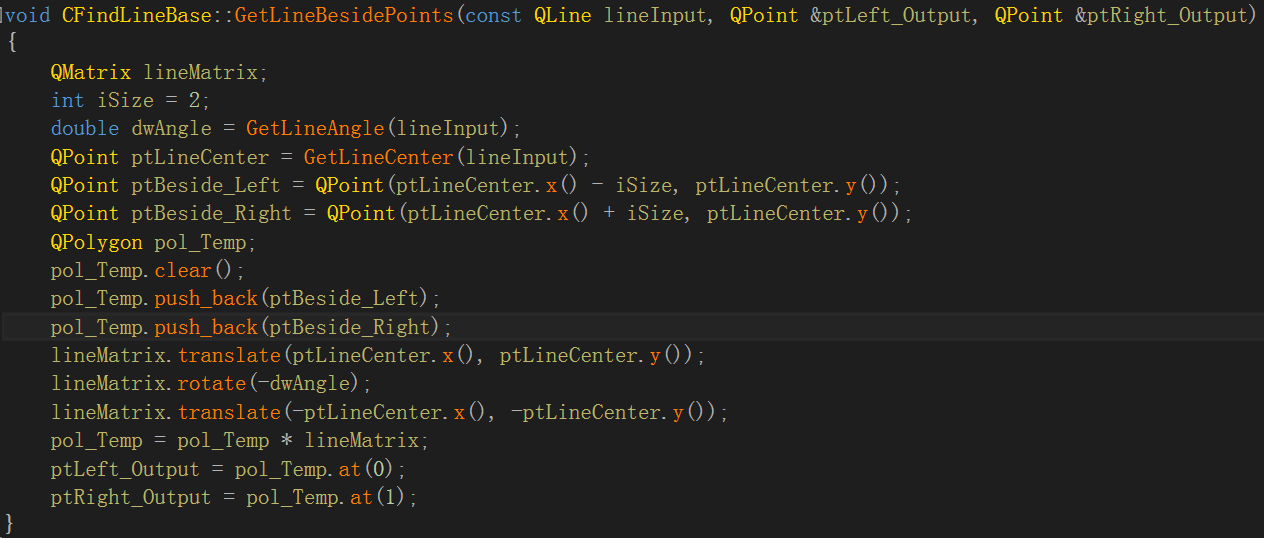


1. 计算输入直线与默认坐标系的Y轴的角度

根据两个向量的夹角公式，计算出输入直线与Y轴的夹角。

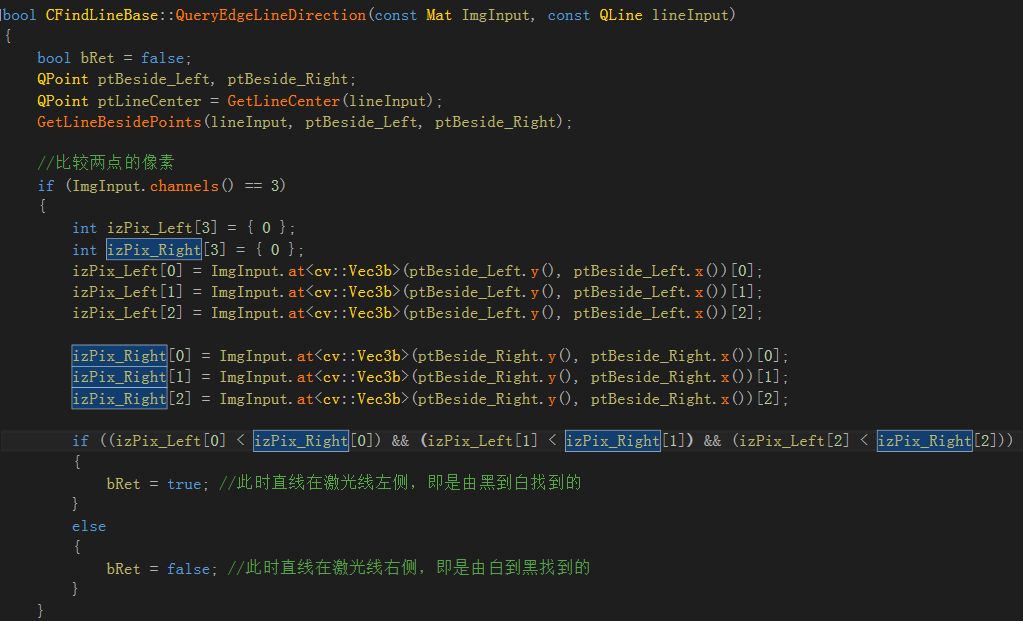


1. 获取当前直线的中心点两边的坐标点



1. 查询当前检测到的直线的方向（是在激光线的左侧还是右侧）

调用GetLineBesidePoints函数，获取当前检测到的直线的中心点两边的坐标点。取出两边的坐标点的灰度值，并比较，确定当前直线是在激光线的左侧还是右侧。



1. 其它
2. 其它
3. 其它
4. 其它