

Bluetooth Gravity Mouse

An exploration of socket programming and UI design from QT and QML

Yinhao Qian @ December 2022

Design Overview

High-level Description

The Bluetooth Gravity Mouse is a fully-functional Human Interface Device that allows the users to control their PCs using their smartphone without internet. Users control the cursor on their PC by tilting their smartphones, and simulate mouse clicking of both left button and right button by tapping the screen. The smartphone acts as a bluetooth server while the PC acts as a bluetooth client, and users are able to pause and resume the service without closing and reopening the application. A core feature of this project is the fluid interface design. It not only provides the basic functionalities but also brings splendid user experience.

Original Design Concept

Looking back to our childhood, we are still able to remember the excitement when we first got to play games like Asphalt 5 where we could tilt our devices to control the race car. Are we able to apply the same concept to Human Interface Devices, or more specifically cursor control? Could we control the cursor just like we control the racecar?

Previous Works Expansion

There have been several attempts from developers to make smartphones as HID Mouse, yet all of these apps use a local area network which requires a local router, none of which make use of on-device sensors on smartphones, and more importantly, The interface designs in these projects are sub par. This project, however, did not use a single line of code from the internet, and everything was started from scratch.

Preliminary Design Verification

All client-side testing is conducted on Windows 10 19045 using MSVC2019 from x86-64 architecture, and all server-side testing is conducted on ARM64 architecture.

Client Side

Communication Protocol Testing

Finalizing the software framework selection and communication protocol, a simple program is written to test if the framework is able to correctly identify and gain access to the bluetooth module on the local device. QBluetoothDeviceDiscoveryAgent class is used to provide a simple nearby bluetooth device scanning using the on-device bluetooth module. Turning the logging filter for bluetooth. QBluetoothDeviceDiscoveryAgent is not to be used in the final application, because this is only for testing the bluetooth module. Users are responsible to identify MAC addresses on both devices, as both Windows and Android operating systems do not provide a way to access the host MAC address programmatically.

```
auto deviceDiscoveryAgent = new QBluetoothDeviceDiscoveryAgent(this);
QObject::connect(deviceDiscoveryAgent, &QBluetoothDeviceDiscoveryAgent::deviceDiscovered, this,
[this](const QBluetoothDeviceInfo & info) {
    qInfo() << ("agent device discovered " + info.address().toString() + " " + info.name());
});
QObject::connect(deviceDiscoveryAgent, &QBluetoothDeviceDiscoveryAgent::finished, this, [this]() {
    qInfo() << disp("agent finished " );
});
QObject::connect(deviceDiscoveryAgent,
static_cast<void(QBluetoothDeviceDiscoveryAgent::*)(QBluetoothDeviceDiscoveryAgent::Error)>(&QBluetoothDeviceDiscoveryAgent::error), this, [this](QBluetoothServiceDiscoveryAgent::Error error) {
    qInfo() << disp("agent error " +
QString::fromUtf8(QMetaEnum::fromType<QBluetoothDeviceDiscoveryAgent::Error>().valueToKey(error)));
});
QLoggingCategory::setFilterRules(QStringLiteral("qt.bluetooth* = true"));
deviceDiscoverAgent->start();
```

Running this code yields no error on Android devices, yet some unexpected behaviors occurred while testing on Windows devices which are later resolved. It turned out to be a compiler issue. This issue is discussed in more detail under the Design Testing section.

Cursor Moving Functional Testing

The QCursor class provides functionalities to let users gain direct control of cursor position.

```
QCursor cursor = QCursor();
cursor.setPos(0, 0);
```

The setPos() method successfully sets the cursor to the top left position of the main screen.

Cursor Clicking Functional Testing

The QCursor class mentioned above is solely responsible for moving the cursor, yet it does not provide a way to simulate the mouse clicking event. Because of this, relying on the Windows API is the only viable option.

The mouse_event() method defined in winuser.h provides a way to simulate both left-button and right-button clicking events by setting corresponding flags from the first argument.

```
#if defined(Q_OS_WINDOWS)
#pragma comment(lib, "User32.lib")
    mouse_event(MOUSEEVENTF_ABSOLUTE | MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP, 1, 1, 0, 0);
    mouse_event(MOUSEEVENTF_ABSOLUTE | MOUSEEVENTF_RIGHTDOWN | MOUSEEVENTF_RIGHTUP, 1, 1, 0, 0);
#endif
```

The function call is guarded by the Q_OS_WINDOWS macro so that this method is not accidentally called on the Android (server) side, and User32.lib import should be explicitly marked to link the library to the executable.

When running on Windows, it successfully simulated mouse left-clicking and right-clicking events at the current cursor position.

Receiving Socket Testing

Before testing for the service-broadcasting server, it is important to construct a socket to receive information that is sent from the server. This can be achieved by using the QBluetoothSocket class.

```
auto socket = new QBluetoothSocket(QBluetoothServiceInfo::RfcommProtocol, this);
QObject::connect(socket, &QBluetoothSocket::disconnected, this, [this]() {
    qInfo("socket disconnected" + "");
});
QObject::connect(socket, &QBluetoothSocket::connected, this, [this]() {
    qInfo("socket connected to" + socket->peerAddress().toString(), false);
});
QObject::connect(socket,
    static_cast
<void(QBluetoothSocket::*)(QBluetoothSocket::SocketError)>(&QBluetoothSocket::error),
    this, [this](QBluetoothSocket::SocketError error) {
        qInfo("socket error occurred" +
            QString::fromUtf8(QMetaEnum::fromType<QBluetoothSocket::SocketError>()
                .valueToKey(error)));
    });
QObject::connect(socket, &QBluetoothSocket::stateChanged, this,
    [this](QBluetoothSocket::SocketState state) {
        qInfo("socket state changed" +
            QString::fromUtf8(QMetaEnum::fromType<QBluetoothSocket::SocketState>()
                .valueToKey(state)));
    });
```

```
QObject::connect(socket, &QBluetoothSocket::readyRead, this, [this]() {
    QString dataRead = QString::fromUtf8(socket->readAll());
    qDebug("socket read" + dataRead);
});
socket->connectToService(QBluetoothAddress("9C:BC:F0:CF:32:AF"), QBluetoothUuid::SerialPort);
```

The socket is now able to connect to the device with a specified address (9C:BC:F0:CF:32:AF in this case) and to read any information sent from that address. This code is not executed until the service-broadcasting Bluetooth server and its corresponding socket from the server side has been initialized. More details can be found under Bluetooth Service Broadcasting subsection under Server Side section.

Server Side

Tilt Sensor Reading Testing

The QT Sensors API provides an easy way to access the current horizontal and vertical values of device tilting angles.

```
auto tileSensor = new QTiltSensor(this);
QObject::connect(tileSensor, &QTiltSensor::readingChanged, this, [this]() {
    qDebug() << "x" << tileSensor->reading()->xRotation();
    qDebug() << "y" << tileSensor->reading()->yRotation();
});
QObject::connect(tileSensor, &QTiltSensor::sensorError, this, [this](int error) {
    qDebug(("tilt sensor error" + "code " + QString::number(error)));
});
tileSensor->start();
```

Although the sensor should not be used on the PC (client) side, it is not necessary to guard the method, as calling this API from client side, unlike calling Windows API methods from Android, will not yield exceptions and will just not trigger readingChanged() signal. After the code above is executed, a real-time monitoring of tilting angles is printed in the debugging console.

Broadcaster Service & Transmitting Socket Testing

The server will broadcast a bluetooth signal by using QBluetoothService, and listens to incoming socket connections of type QBluetoothSocket.

```
auto server = new QBluetoothServer(QBluetoothServiceInfo::Protocol::RfcommProtocol, this);
QBluetoothSocket* socket = nullptr;
QObject::connect(server,
    static_cast<void(QBluetoothServer::*)>(QBluetoothServer::Error)>(&QBluetoothServer::error), this,
    [this](QBluetoothServer::Error error) {
        qDebug("server error occurred" + QString::fromUtf8(QMetaEnum::fromType<QBluetoothServer::Error>().valueToKey(error)));
    });
```

```

});
QObject::connect(server, &QBluetoothServer::newConnection, this, [this]() {
    socket = server->nextPendingConnection();
    qInfo("socket connected to", "code " + socket->peerAddress().toString());
});
auto serviceInfo = server->listen(QBluetoothUuid::SerialPort, "");
if(server && !server.isNull() && server->isListening()) {
    qInfo("service connected to", "port " + QString::number(server->serverPort()));
}

```

Until now, the sockets are ready to be connected. Executing the code above along with the socket and by calling the write() function from the socket on the server's end, a notification can be seen on the client's client.

Design Implementation

Overall System

Instead of using the legacy QWidgets, this project uses Qt Quick 2.0 as the front-end, powered by QML Engine. The back-end business logic is implemented using C++. The project will automatically detect the server and client side and adapt different interfaces without users specifying it.

Communication Protocol

To establish the communication between the smartphone and PC, a unified communication protocol needs to be defined. In this project, classic bluetooth is used. The smartphone (server) broadcasts the service with a specific UUID while the PC (client) listens to the nearby services by filtering out unrelated ones that have different UUID.

Visual Components

The final interface of the project presented to the users consists of multiple reusable visual components plus a non-visual singleton styling component.

The project went over two phases of interface design. The original implementation uses square components inspired by Windows 10 UI accompanied by linear motion transitioning. However, a redesigning process was initiated at the midpoint of the software development cycle, since the original design gives too much of an abstruse feel.

The second interface design is inspired by the new Android 12, circular components with spring-like transitions, and the way how the interface communicates with the users replicates a basic calculator – information flows out of the device from the top through a digital display pane, and subsequently flows into the device through huge buttons.

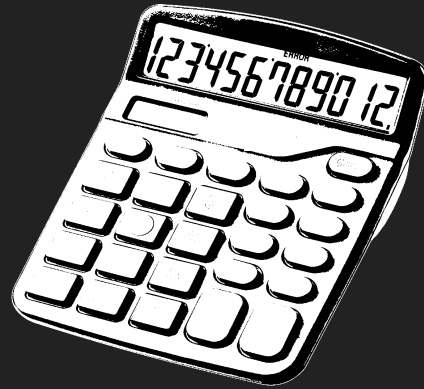


Figure. Grid-like Layout in Calculator

The Styling Singleton (MyStyle.qml)

To provide a clean and unified look, a singleton styling sheet is declared and loaded as singleton. It serves two functions - scaling and coloring.

Because the software is runned on a wide variety of devices with distinct screen resolutions and DPI settings, and having only a pair of fixed line width and font size would make the software look okay on one device but disproportional on others. The font size, radius, and line width are all directly controlled by a base value, which is overridden when the main application is rendered.

The coloring of the entire application is classified as highlight(900), foreground(600), buffer(300), background(white). All components from the application are composed of different combinations of these four colors.

```
pragma Singleton
import QtQuick 2.15
Item {
    property int baseValue: 10
    readonly property color color_neutral900: "#1A202C"
    readonly property color color_neutral600: "#717D96"
    readonly property color color_neutral300: "#E2E7F0"
    readonly property color color_neutralWhite: "#FFFFFF"
    readonly property double param_fontSize: baseValue
    readonly property double param_largeFontSize: baseValue * 1.5
    readonly property double param_radius: baseValue
    readonly property double param_lineWidth: baseValue * 0.1
}
```

The Single Line 14 Segment Display (MyLineDisplayer.qml)

The digital display is a multi-row display which contains multiple arrays of single-line display:



Figure. Single Line Digital Display

```
Rectangle {
    id: rootRectangle_myLineDisplay
    property string dispText: "!!!NOT!IN!SERVICE!!!"
    color: "transparent"
    implicitHeight: text_backgroundText.paintedHeight
    implicitWidth: 200
    Text {
        id: text_backgroundText
        anchors.fill: parent
        font.family: "DSEG14 Classic"
        font.pixelSize: MyStyle.param_largeFontSize * 4
        horizontalAlignment: Text.AlignHCenter
        verticalAlignment: Text.AlignVCenter
        styleColor: "#ffffff"
        font.letterSpacing: MyStyle.param_lineWidth
        fontSizeMode: Text.Fit
        text: "~~~~~"
        opacity: 0.15
        color: "#ffffff"
    }
    Text {
        id: text_foregroundText
        anchors.fill: parent
        font.family: "DSEG14 Classic"
        font.pixelSize: MyStyle.param_largeFontSize * 4
        horizontalAlignment: Text.AlignHCenter
        verticalAlignment: Text.AlignVCenter
        font.letterSpacing: MyStyle.param_lineWidth
        fontSizeMode: Text.Fit
        text: {
            let toDisplay = parent.dispText
            if (toDisplay.length < 20)
                toDisplay = toDisplay.padEnd(20, '!')
            else if (toDisplay.length > 20)
                toDisplay = toDisplay.substring(0, 20)
            return toDisplay.replace(/\\W/g, '!').replace(' ', '!')
        }
        color: MyStyle.color_neutralWhite
    }
}
```

The Multi Line 14 Segment Display (MyDisplayer.qml)

The final display pane is populated using the single-line display using model-view-delegate pattern:

```
Rectangle {
    id: rootRectangle_myDisplay
    implicitHeight: 400
    implicitWidth: 500
    color: MyStyle.color_neutral900
    radius: MyStyle.param_radius
    property int numberOfLines: 5
    property string dispText: "t\\ne\\ngeeg"
    onDispTextChanged: passDispTextToLines()
    Column {
        id: column_linesOfDisplay
        anchors.fill: parent
        anchors.margins: MyStyle.param_lineWidth * 10
        spacing: MyStyle.param_lineWidth * 10
        clip: true
        Repeater {
            id: repeater_linesOfDisplay
            model: numberOfLines
            delegate: MyLineDisplay {
                anchors.left: column_linesOfDisplay.left
                anchors.right: column_linesOfDisplay.right
                height: (column_linesOfDisplay.height
                    - (column_linesOfDisplay.spacing
                        * (rootRectangle_myDisplay.numberOfLines -
1)))
                    / rootRectangle_myDisplay.numberOfLines
            }
            Component.onCompleted: passDispTextToLines()
        }
    }

    function passDispTextToLines() {
        let splittedDispText = rootRectangle_myDisplay.dispText.split('\n')
        let smallerIndex = Math.min(rootRectangle_myDisplay.numberOfLines,
            splittedDispText.length)

        for (var i = 0; i < smallerIndex; ++i) {
            if (repeater_linesOfDisplay.itemAt(i) !== null)
                repeater_linesOfDisplay.itemAt(i).dispText = i
                < splittedDispText.length ? splittedDispText[i] :
"!!!!-!!!!-!!!!-!!!"
        }
    }
}
```

Although it is specified in this very specific project to use 2 lines, it is possible to have more than 2 lines, and the height of each line is automatically calculated.



Figure 1. Multi-line Segment Display

The Readings Observer (MyReadingsObserver.qml)

Another cool component is the reading observer. The positions of the two levers will slide horizontally, and they are the readings of tilt sensors reading data received from the smartphone.

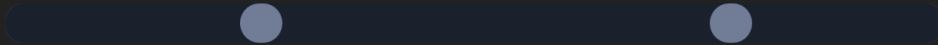
```
Rectangle {
    id: rootRectangle_myReadingsObserver
    implicitHeight: 100
    implicitWidth: 400
    color: MyStyle.color_neutral900
    radius: MyStyle.param_radius
    property double xVal: -0
    property double yVal: -0
    property color readingsColor: MyStyle.color_neutral600
    Rectangle {
        id: rightGauge
        x: {
            let xMinimum = rootRectangle_myReadingsObserver.x
                        + rootRectangle_myReadingsObserver.width / 2
            let xMaximum = rootRectangle_myReadingsObserver.x
                        + rootRectangle_myReadingsObserver.width
                        - rootRectangle_myReadingsObserver.radius * 2
            let xMiddle = (xMaximum + xMinimum) / 2
            return xMiddle + (xMaximum - xMinimum) / 2 * xVal
        }
        Behavior on x {
            SpringAnimation {
                spring: 1
                damping: 0.2
            }
        }
        color: readingsColor
        anchors.top: rootRectangle_myReadingsObserver.top
        anchors.bottom: rootRectangle_myReadingsObserver.bottom
        radius: MyStyle.param_radius
        width: MyStyle.param_radius * 2
    }
    Rectangle {
        id: rectangle_leftGauge
        x: {
            let xMinimum = rootRectangle_myReadingsObserver.x
            let xMaximum = rootRectangle_myReadingsObserver.x
                        + rootRectangle_myReadingsObserver.width / 2
                        - rootRectangle_myReadingsObserver.radius * 2
            let xMiddle = (xMaximum + xMinimum) / 2
            return xMiddle + (xMaximum - xMinimum) / 2 * yVal
        }
    }
}
```

```

    }
    Behavior on x {
        SpringAnimation {
            spring: 1
            damping: 0.2
        }
    }
    color: readingsColor
    anchors.top: rootRectangle_myReadingsObserver.top
    anchors.bottom: rootRectangle_myReadingsObserver.bottom
    radius: MyStyle.param_radius
    width: MyStyle.param_radius * 2
}
}

```

To provide a better visual experience for the users, the movement of two levers follows the spring-like movement, following a pattern of simple harmonic motion.



The Dialer (MyDialer.qml)

The dialer component is the keypad that allows the user to control the software:

```

Rectangle {
    id: rootRectangle_myDialer
    implicitHeight: 500
    implicitWidth: 600
    color: MyStyle.color_neutral900
    radius: MyStyle.param_radius
    property bool isClient: true
    property color buttonsColor: MyStyle.color_neutral600
    property color buttonColor2: MyStyle.color_neutral800
    property color buttonLabelColor: MyStyle.color_neutral300
    signal buttonClicked(string buttonText)
    Grid {
        id: grid_buttons
        rows: isClient ? 4 : 1
        columns: isClient ? 4 : 1
        anchors.top: parent.top
        anchors.left: parent.left
        anchors.right: parent.right
        height: parent.height * 0.7
        anchors.margins: MyStyle.param_lineWidth * 10
        spacing: MyStyle.param_lineWidth * 10
        Repeater {
            id: repeater_buttons
            model: isClient ? ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B",
"C", "D", "E", "F"] : ["CL"]
            delegate: Rectangle {
                width: (grid_buttons.width - (grid_buttons.columns - 1)
                    * grid_buttons.spacing) / grid_buttons.columns
            }
        }
    }
}

```

```

        height: (grid_buttons.height - (grid_buttons.rows - 1)
            * grid_buttons.spacing) / grid_buttons.rows
        radius: mouseArea_button.pressed ? MyStyle.param_radius * 4 :
MyStyle.param_radius

        Behavior on radius {
            SpringAnimation {
                spring: 5
                damping: 500
            }
        }

        color: mouseArea_button.pressed ? buttonsColor : buttonColor2
        Behavior on color {
            ColorAnimation {
                duration: 200
            }
        }

        Text {
            anchors.fill: parent
            anchors.margins: mouseArea_button.pressed ?
MyStyle.param_lineWidth

            * 10 : MyStyle.param_lineWidth * 5
            Behavior on anchors.margins {
                SpringAnimation {
                    spring: 5
                    damping: 500
                }
            }
            font.family: "STHeiti TC"
            font.pixelSize: MyStyle.param_largeFontSize * 10
            horizontalAlignment: Text.AlignHCenter
            verticalAlignment: Text.AlignVCenter
            fontSizeMode: Text.Fit
            id: text_buttonLabel
            visible: isClient
            color: buttonLabelColor
            text: modelData
        }
        MouseArea {
            id: mouseArea_button
            anchors.fill: parent
            onClicked: {
                buttonClicked(modelData)
            }

            onPressAndHold: {
                if (modelData === "CL")
                    buttonClicked("HO")
            }
        }
    }
}
Grid {
    rows: 1
    columns: isClient ? 3 : 2
    id: grid_bottomButtons

```

```

anchors.top: grid_buttons.bottom
anchors.left: parent.left
anchors.right: parent.right
anchors.bottom: parent.bottom
anchors.margins: MyStyle.param_lineWidth * 10
spacing: MyStyle.param_lineWidth * 10
Repeater {

    id: repeater_bottomButtons
    model: isClient ? ["确认/OK", "地址/AD", "开关/PO"] : ["校正/CA", "开关/PO"]
    delegate: Rectangle {
        width: (grid_bottomButtons.width - (grid_bottomButtons.columns - 1)
                * grid_bottomButtons.spacing) /
grid_bottomButtons.columns
        height: (grid_bottomButtons.height - (grid_bottomButtons.rows - 1)
                * grid_bottomButtons.spacing) /
grid_bottomButtons.rows
        radius: mouseArea_bottomButton.pressed ? MyStyle.param_radius
* 4 : MyStyle.param_radius
        Behavior on radius {
            SpringAnimation {
                spring: 5
                damping: 500
            }
        }

        color: mouseArea_bottomButton.pressed ? buttonsColor : buttonColor2
        Behavior on color {
            ColorAnimation {
                duration: 200
            }
        }

        Text {
            anchors.fill: parent
            anchors.margins: mouseArea_bottomButton.pressed ?
MyStyle.param_lineWidth * 30 : MyStyle.param_lineWidth * 10
            Behavior on anchors.margins {
                SpringAnimation {
                    spring: 5
                    damping: 500
                }
            }
            font.family: "STHeiti TC"
            font.pixelSize: MyStyle.param_largeFontSize * 10
            horizontalAlignment: Text.AlignHCenter
            verticalAlignment: Text.AlignVCenter
            fontSizeMode: Text.Fit
            id: text_bottomButtonLabel
            color: buttonLabelColor
            text: modelData
        }
        MouseArea {
            id: mouseArea_bottomButton
            anchors.fill: parent
            onClicked: buttonClicked(modelData.split('/')[1])
        }
    }
}

```

```
}  
}  
}
```

The cool thing about this component is it will automatically detect if the device software is running on is a client or a server, and it will dynamically change its layout to properly describe the kinds of input.

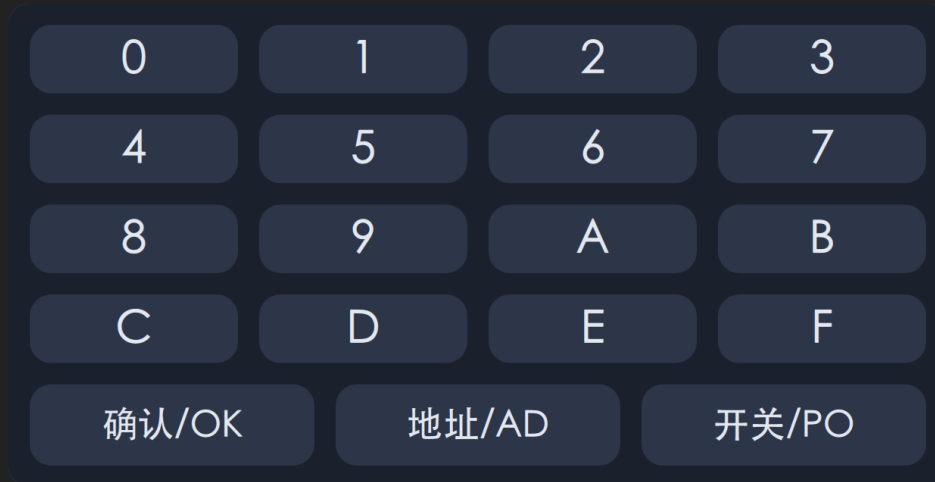


Figure. Dialer Pane on Client Side

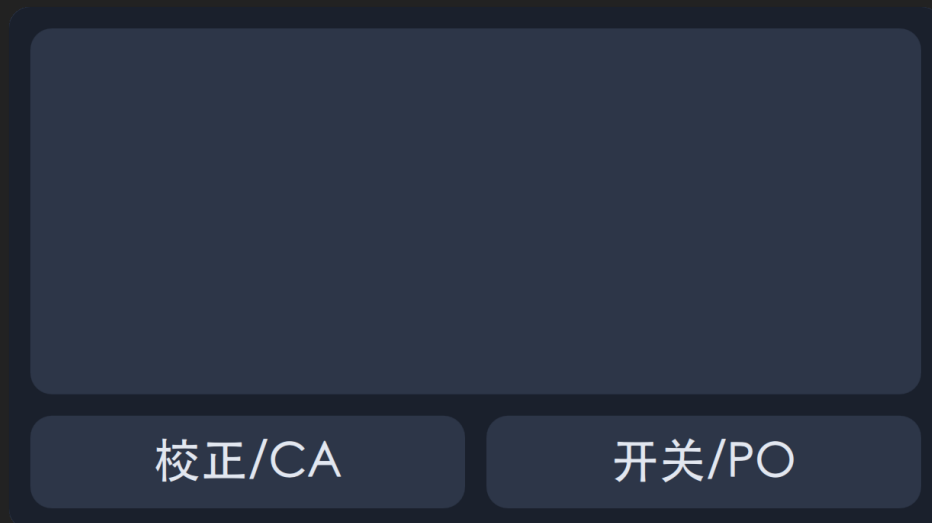


Figure. Dialer Pane on Server Side

The Final Assembly! (MyInterface.qml)

After each individual components have been defined, the only remaining job is to glue them together:

```
Rectangle {
    implicitHeight: 500
    implicitWidth: 640

    objectName: "rootRectangle_myInterface"
    id: rootRectangle_myInterface
    property bool isClient: MyServer ? MyServer.role_q === 1 : true
    property bool isServer: !isClient
    property double x_m: MyServer ? MyServer.xRatio_m_q : 0
    property double y_m: MyServer ? MyServer.yRatio_m_q : 0
    property bool debuggerShow: mouseArea_debuggerShow.pressed
    property string dispText: MyServer ? MyServer.currentMessage_q : "NOT IN SERVICE\nNOT IN SERVICE"

    Rectangle {
        id: rectangle_blurFrame
        anchors.fill: parent
        color: MyStyle.color_neutral800
        MyDisplay {
            id: myDisplay_myDisplay
            anchors.left: parent.left
            anchors.right: parent.right
            anchors.top: parent.top
            height: parent.height * 0.2
            anchors.margins: MyStyle.param_lineWidth * 10
            numberOfLines: 2
            dispText: rootRectangle_myInterface.dispText
        }

        MyReadingsObserver {
            id: myReadingsObserver_myReadingsObserver
            visible: isClient
            anchors.top: myDisplay_myDisplay.bottom

            height: parent.height * 0.05
            anchors.left: parent.left
            anchors.right: parent.right
            anchors.margins: MyStyle.param_lineWidth * 10
            xVal: x_m
            yVal: y_m
        }

        MyDialer {
            id: myDialer_myDialer
            anchors.top: isClient ? myReadingsObserver_myReadingsObserver.bottom :
myDisplay_myDisplay.bottom
            anchors.bottom: parent.bottom
            anchors.left: parent.left
            anchors.right: parent.right
            anchors.margins: MyStyle.param_lineWidth * 10
            isClient: !isServer
            onClicked: buttonText => {
                console.log("BUTTON_" + buttonText)
                if (MyServer) MyServer.execute("BUTTON_" + buttonText)
            }
        }
    }
}
```

```

    }
  }
}

FastBlur {
  opacity: debuggerShow ? 1.0 : 0.0
  Behavior on opacity {
    SpringAnimation {
      spring: 1
      damping: 0.2
    }
  }
  source: rectangle_blurFrame
  anchors.fill: parent
  radius: 40
}

Rectangle {
  id: rectangle_debugger
  x: myDisplay_myDisplay.x
  y: myDisplay_myDisplay.y
  anchors.left: parent.left
  anchors.right: parent.right
  anchors.top: parent.top
  //      color: "transparent"
  height: debuggerShow ? myDialer_myDialer.height : myDisplay_myDisplay.height
  Behavior on height {
    NumberAnimation {
      easing.type: Easing.InOutQuad
    }
  }
  color: MyStyle.color_neutral300
  radius: MyStyle.param_lineWidth * 10
  anchors.margins: MyStyle.param_lineWidth * 10
  visible: true
  opacity: debuggerShow ? 0.5 : 0.0
  Behavior on opacity {
    SpringAnimation {
      spring: 1
      damping: 0.2
    }
  }
  MyDebugger {
    anchors.fill: parent
    anchors.margins: MyStyle.param_lineWidth * 10
  }
  MouseArea {
    id: mouseArea_debuggerShow
    anchors.fill: parent
  }
}
}

```

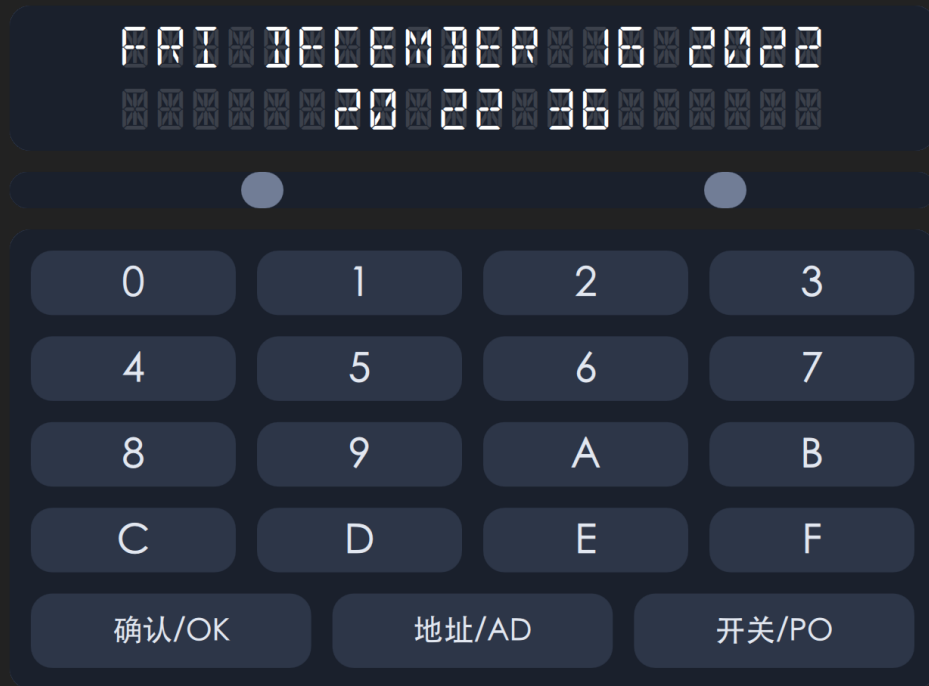


Figure Client-Side Full Interface Design



Figure. Server-Side Full Interface Design

Design Testing

The final project is tested on two Android Devices, and one PC Device. No obvious bugs have been detected so far. However, as this is a pure software project with very little hardware design element in it, all functional testings are completed during the prototyping stage, and after the final integration it would be the users' job to report bugs. Unsuccessful Attempts.

Debugging Console

To make users aware of the running status, a minimal debugging console is embedded in the software. To enable it, users can press the digital display panel, and all logging information will be available. This console is useful whenever there are unexpected disconnection or socket errors.

```
Rectangle {
    implicitHeight: 480
    implicitWidth: 400
    id: rootRectangle_myDebugger
    objectName: "rootRectangle_myDebugger"
    color: "transparent"
    border.color: "#ffffff"
    clip: true
    Text {
        anchors.fill: parent
        text: MyServer ? MyServer.debuggerMessages_q : "-> Debugger Initializing..."
        color: MyStyle.color_neutral900
        font.family: "YaHei Fira Icon Hybrid"
        font.pixelSize: MyStyle.param_fontSize
        wrapMode: Text.Wrap
        fontSizeMode: Text.VerticalFit
    }
}
```

Summary

Future Work

One imperfection of this project is the signal handling algorithms. As mentioned above, the current signal stabilizing techniques are acceptable but have rooms for improvement. In addition to readings from the tilt sensor, it is advisable to track the readings from the gyroscope. The gyroscope is able to detect the angular velocity, with which we can apply a Kalman Filter to smooth out the curve by making predictions of the next reading, comparing that with the actual reading from both sensors, and calculating optimized outputs.

Apart from that, it is also possible to explore other communication standards and integrate them into the software. Common ones would include ZigBee, 5G, NFC. Some other brutal protocols such as hearable sound, or even constantly-moving two-dimensional barcode using some advanced image processing algorithms.

Final Presentation