GUI控制

模板Gui代码

```python
#! /usr/bin/env python
# coding: utf-8

import rospy
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import sys
from std_srvs.srv import SetBool, SetBoolRequest, SetBoolResponse
from sensor_msgs.msg import Imu, MagneticField, BatteryState
from geometry_msgs.msg import Twist
from std_msgs.msg import Float32

import threading
import time


class CtrlWindow(QWidget):

    def __init__(self):
        super(CtrlWindow, self).__init__()

        imu_subscriber = rospy.Subscriber("/zxcar/imu", Imu, self.imu_callback)
        magnetic_subscriber = rospy.Subscriber("/zxcar/mag", MagneticField, self.magnetic_callback)
        vel_subscriber = rospy.Subscriber("/zxcar/get_vel", Twist, self.vel_callback)
        butter_subscriber = rospy.Subscriber("/zxcar/battery", BatteryState, self.battery_callback)
        self.vel_publisher = rospy.Publisher("/zxcar/cmd_vel", Twist, queue_size=20)
```

```
        layout = QHBoxLayout()
        self.setLayout(layout)

        ################## 第一列 #############
        first_layout = QVBoxLayout()
        layout.addLayout(first_layout)

        ########### LED 控制
        group_led = QGroupBox("LED控制")
        first_layout.addWidget(group_led)

        led_layout = QVBoxLayout(group_led)
        btn_led_open = QPushButton("打开LED")
        btn_led_close = QPushButton("关闭LED")
        led_layout.addWidget(btn_led_open)
        led_layout.addWidget(btn_led_close)

        btn_led_open.clicked.connect(self.click_led_open)
        btn_led_close.clicked.connect(self.click_led_close)

        ########### 蜂鸣器 控制
        group_buzzer = QGroupBox("蜂鸣器控制")
        first_layout.addWidget(group_buzzer)

        buzzer_layout = QVBoxLayout(group_buzzer)
        btn_buzzer_open = QPushButton("打开蜂鸣器")
        btn_buzzer_close = QPushButton("关闭蜂鸣器")
        buzzer_layout.addWidget(btn_buzzer_open)
        buzzer_layout.addWidget(btn_buzzer_close)

        btn_buzzer_open.clicked.connect(self.click_buzzer_open)
        btn_buzzer_close.clicked.connect(self.click_buzzer_close)
        ########################################

        ################## 第2列 ##################
        second_layout = QVBoxLayout()
        layout.addLayout(second_layout)

        ################ imu陀螺仪
        group_imu = QGroupBox("imu陀螺仪")
        second_layout.addWidget(group_imu)

        imu_layout = QVBoxLayout(group_imu)
        imu_acc_layout = QHBoxLayout()
        imu_gyro_layout = QHBoxLayout()
        imu_mag_layout = QHBoxLayout()
        imu_layout.addLayout(imu_acc_layout)
        imu_layout.addLayout(imu_gyro_layout)
        imu_layout.addLayout(imu_mag_layout)
        imu_acc_layout.addWidget(QLabel("加速度"))
        imu_gyro_layout.addWidget(QLabel("线速度"))
        imu_mag_layout.addWidget(QLabel("地磁"))

        self.imu_labels = []
        for i in range(9):
            lb = QLabel("0")
            self.imu_labels.append(lb)
```

```
            if i >= 6:
                imu_mag_layout.addWidget(lb)
            elif i >= 3:
                imu_gyro_layout.addWidget(lb)
            else:
                imu_acc_layout.addWidget(lb)

        ################ 舵机
        group_servo = QGroupBox("舵机控制")
        second_layout.addWidget(group_servo)

        servo_layout = QHBoxLayout(group_servo)
        self.le_servo = QLineEdit("0")
        servo_layout.addWidget(self.le_servo)

        btn_servo = QPushButton("旋转")
        servo_layout.addWidget(btn_servo)
        btn_servo.clicked.connect(self.click_servo)

        ################### 第3列 ###################
        third_layout = QVBoxLayout()
        layout.addLayout(third_layout)

        ############### 电池信息显示
        group_battery = QGroupBox("电池信息显示")
        third_layout.addWidget(group_battery)

        battery_layout = QFormLayout(group_battery)
        self.lb_battery_voltage = QLabel("0")
        battery_layout.addRow("当前电压", self.lb_battery_voltage)

        ############### 万向轮结构速度显示
        group_vel = QGroupBox("万向轮结构速度显示")
        third_layout.addWidget(group_vel)

        vel_layout = QFormLayout(group_vel)
        self.lb_linear_vel = QLabel("0")
        vel_layout.addRow("线速度", self.lb_linear_vel)
        self.lb_angle_vel = QLabel("0")
        vel_layout.addRow("角速度", self.lb_angle_vel)

        ############### 万向轮结构速度控制
        group_vel_ctrl = QGroupBox("万向轮结构速度控制")
        third_layout.addWidget(group_vel_ctrl)

        vel_ctrl_layout = QFormLayout(group_vel_ctrl)
        self.le_linear_vel = QLineEdit("0")
        vel_ctrl_layout.addRow("线速度", self.le_linear_vel)
        self.le_angular_vel = QLineEdit("0")
        vel_ctrl_layout.addRow("角速度", self.le_angular_vel)
        self.btn_vel_ctrl = QPushButton("发送")
        vel_ctrl_layout.addRow(self.btn_vel_ctrl)

        self.le_loop_count = QLineEdit("500")
        vel_ctrl_layout.addRow("循环次数", self.le_loop_count)

        self.le_loop_delay = QLineEdit("0.1")
        vel_ctrl_layout.addRow("循环间隔(s)", self.le_loop_delay)
```

```
143
144         self.btn_vel_loop_ctrl = QPushButton("循环发送")
145         vel_ctrl_layout.addRow(self.btn_vel_loop_ctrl)
146 
147         self.btn_vel_ctrl.clicked.connect(self.click_vel_ctrl)
148         self.btn_vel_loop_ctrl.clicked.connect(self.click_vel_loop_ctrl)
149 
150     def imu_callback(self, msg):
151         if not isinstance(msg, Imu):
152             return
153         self.imu_labels[0].setText(str(msg.linear_acceleration.x))
154         self.imu_labels[1].setText(str(msg.linear_acceleration.y))
155         self.imu_labels[2].setText(str(msg.linear_acceleration.z))
156         self.imu_labels[3].setText(str(msg.angular_velocity.x))
157         self.imu_labels[4].setText(str(msg.angular_velocity.y))
158         self.imu_labels[5].setText(str(msg.angular_velocity.z))
159 
160     def magnetic_callback(self, msg):
161         if not isinstance(msg, MagneticField):
162             return
163         self.imu_labels[6].setText(str(msg.magnetic_field.x))
164         self.imu_labels[7].setText(str(msg.magnetic_field.x))
165         self.imu_labels[8].setText(str(msg.magnetic_field.x))
166 
167     def vel_callback(self, msg):
168         if not isinstance(msg, Twist):
169             return
170         self.lb_linear_vel.setText(str(msg.linear.x))
171         self.lb_angle_vel.setText(str(msg.angular.x))
172 
173     def battery_callback(self, msg):
174         if not isinstance(msg, BatteryState):
175             return
176         self.lb_battery_voltage.setText(str(msg.voltage))
177 
178     def click_led_open(self):
179         client = rospy.ServiceProxy("/zxcar/led", SetBool)
180         client.wait_for_service()
181 
182         request = SetBoolRequest()
183         request.data = True
184         client.call(request)
185 
186         client.close()
187 
188     def click_led_close(self):
189         client = rospy.ServiceProxy("/zxcar/led", SetBool)
190         client.wait_for_service()
191 
192         request = SetBoolRequest()
193         request.data = False
194         client.call(request)
195 
196         client.close()
197 
198     def click_buzzer_open(self):
199         client = rospy.ServiceProxy("/zxcar/buzzer", SetBool)
200         client.wait_for_service()
```

```
201
202         request = SetBoolRequest()
203         request.data = True
204         client.call(request)
205
206         client.close()
207
208     def click_buzzer_close(self):
209         client = rospy.ServiceProxy("/zxcar/buzzer", SetBool)
210         client.wait_for_service()
211
212         request = SetBoolRequest()
213         request.data = False
214         client.call(request)
215
216         client.close()
217
218     def click_servo(self):
219         angle = float(self.le_servo.text())
220         msg = Float32()
221         msg.data = angle
222         self.servo_publisher.publish(msg)
223
224     def click_vel_ctrl(self):
225         linear = float(self.le_linear_vel.text())
226         angular = float(self.le_angular_vel.text())
227         twist = Twist()
228         twist.linear.x = linear
229         twist.angular.z = angular
230         self.vel_publisher.publish(twist)
231
232     def __do_loop_ctrl(self, linear, angular, loop_count, loop_delay):
233         self.btn_vel_ctrl.setEnabled(False)
234         self.btn_vel_loop_ctrl.setEnabled(False)
235
236         count = 0
237         while count < loop_count:
238             # twist = Twist()
239             # twist.linear.x = linear
240             # twist.angular.z = angular
241             # self.vel_publisher.publish(twist)
242
243             count += 1
244             time.sleep(loop_delay)
245
246         self.btn_vel_ctrl.setEnabled(True)
247         self.btn_vel_loop_ctrl.setEnabled(True)
248
249     def click_vel_loop_ctrl(self):
250         linear = float(self.le_linear_vel.text())
251         angular = float(self.le_angular_vel.text())
252
253         loop_count = float(self.le_loop_count.text())
254         loop_delay = float(self.le_loop_delay.text())
255
256         threading.Thread(target=self.__do_loop_ctrl, args=(linear, angular,
    loop_count, loop_delay)).start()
257
```

```
258
259 if __name__ == '__main__':
260     # 创建node
261     node_name = "zxcar_ctrl_node"
262     rospy.init_node(node_name)
263
264     app = QApplication(sys.argv)
265
266     window = CtrlWindow()
267     window.show()
268
269     app.exec_()
```

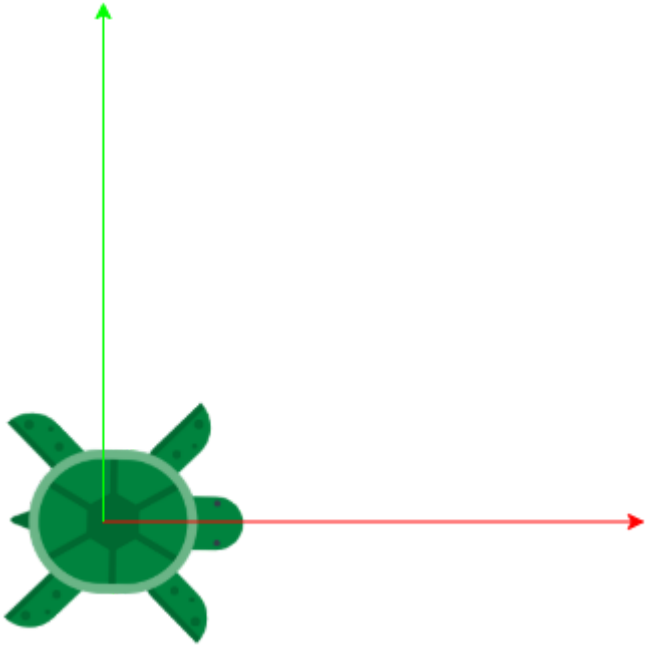
自定义信号

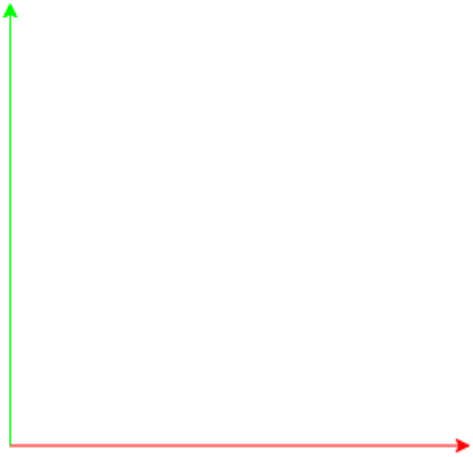
```
1  class CtrlWindow(QWidget):
2      _imu_signal = pyqtSignal(list)
3      _magnetic_signal = pyqtSignal(list)
4      _vel_signal = pyqtSignal(float, float)
5      _battery_signal = pyqtSignal(float)
6      _vel_loop_ctrl_signal = pyqtSignal(bool)
7
8      def __init__(self):
9          ...
10         self._imu_signal.connect(self.imu_update)
11         self._magnetic_signal.connect(self.magnetic_update)
12         self._vel_signal.connect(self.vel_update)
13         self._battery_signal.connect(self.battery_update)
14         self._vel_loop_ctrl_signal.connect(self._loop_ctrl_update)
15         ...
16
17     def imu_update(self, arr):
18         self.imu_labels[0].setText(str(arr[0]))
19         self.imu_labels[1].setText(str(arr[1]))
20         self.imu_labels[2].setText(str(arr[2]))
21         self.imu_labels[3].setText(str(arr[3]))
22         self.imu_labels[4].setText(str(arr[4]))
23         self.imu_labels[5].setText(str(arr[5]))
24
25     def imu_callback(self, msg):
26         if not isinstance(msg, Imu):
27             return
28         self._imu_signal.emit(
29             [msg.linear_acceleration.x, msg.linear_acceleration.y,
msg.linear_acceleration.z, msg.angular_velocity.x,
30              msg.angular_velocity.y, msg.angular_velocity.z])
31
32     def magnetic_update(self, arr):
33         self.imu_labels[6].setText(str(arr[0]))
34         self.imu_labels[7].setText(str(arr[1]))
35         self.imu_labels[8].setText(str(arr[2]))
36
37     def magnetic_callback(self, msg):
38         if not isinstance(msg, MagneticField):
39             return
40         self._magnetic_signal.emit([msg.magnetic_field.x,
msg.magnetic_field.y, msg.magnetic_field.z])
```

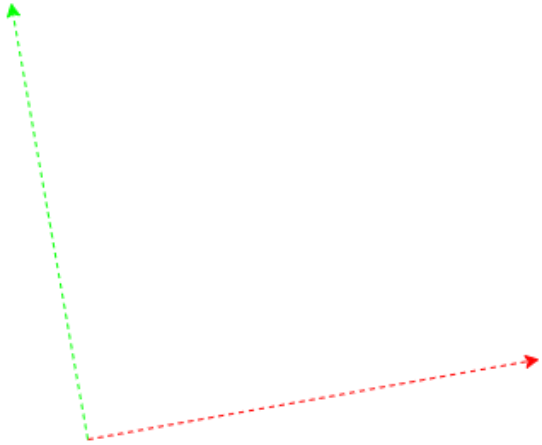
```
41
42      def vel_update(self, linear, angular):
43          self.lb_linear_vel.setText(str(linear))
44          self.lb_angle_vel.setText(str(angular))
45
46      def vel_callback(self, msg):
47          if not isinstance(msg, Twist):
48              return
49          self._vel_signal.emit(msg.linear.x, msg.angular.z)
50
51      def battery_update(self, voltage):
52          self.lb_battery_voltage.setText(str(voltage))
53
54      def battery_callback(self, msg):
55          if not isinstance(msg, BatteryState):
56              return
57          self._battery_signal.emit(msg.voltage)
58
59      def click_vel_ctrl(self):
60          linear = float(self.le_linear_vel.text())
61          angular = float(self.le_angular_vel.text())
62          twist = Twist()
63          twist.linear.x = linear
64          twist.angular.z = angular
65          self.vel_publisher.publish(twist)
66
67      def _loop_ctrl_update(self, enable):
68          self.btn_vel_ctrl.setEnabled(enable)
69          self.btn_vel_loop_ctrl.setEnabled(enable)
70
71      def __do_loop_ctrl(self, linear, angular, loop_count, loop_delay):
72          self._vel_loop_ctrl_signal.emit(False)
73
74          count = 0
75          while count < loop_count:
76              twist = Twist()
77              twist.linear.x = linear
78              twist.angular.z = angular
79              self.vel_publisher.publish(twist)
80
81              count += 1
82              time.sleep(loop_delay)
83
84          self._vel_loop_ctrl_signal.emit(True)
```

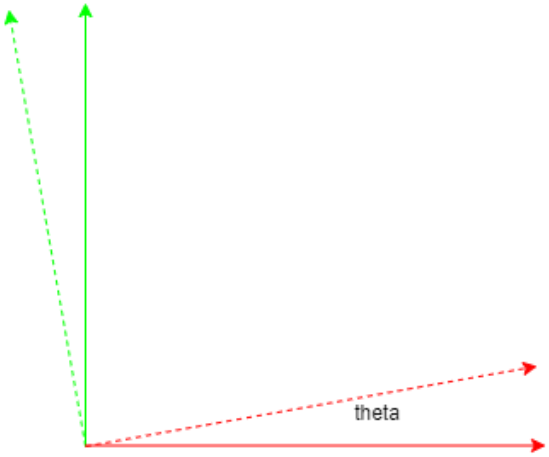
里程计

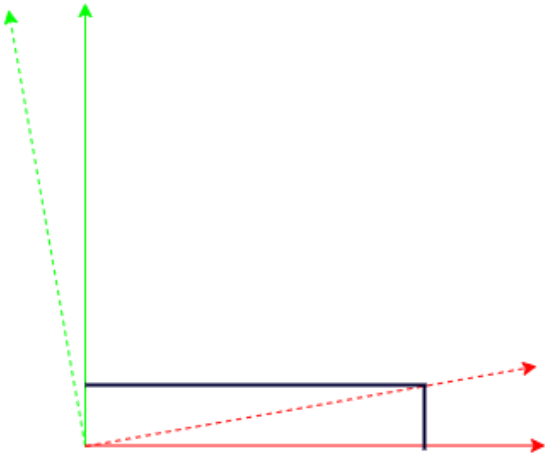
分析











距离 = 速度 * 时间

计算

```
# vel
v_x = msg.linear.x
v_y = msg.linear.y
v_theta = msg.angular.z

# time
current_time = rospy.Time().now()

dt = current_time.to_sec() - last_time.to_sec()
delta_x = (v_x * cos(theta) - v_y * sin(theta)) * dt
delta_y = (v_x * sin(theta) + v_y * cos(theta)) * dt
delta_theta = v_theta * dt

x += delta_x
y += delta_y
theta += delta_theta
```

!!!note
线速度和角速度都是瞬时的，我们按照时间差，可以计算出在这个瞬时时间内移动的距离和转动的角度，然后将所有的数据进行累加，就可以得到小车移动的实际距离。

```
当前小车线速度方向只存在于x轴方向，但是考虑到以后要兼容，麦克纳姆轮结构，速度可能存在于y轴，所以计算的时候加上了y轴方向的速度。
```

坐标关系发布

```
translation = (x, y, 0.0)
rotation = quaternion_from_euler(0, 0, theta)
broadcaster.sendTransform(translation, rotation, current_time, 'base_link',
'odom')
```