键盘控制功能使用与讲解和运动控制原理

1. 功能简介

运功的控制和实现是 ROS 机器人小车最基础、最重要的部分。在我们的小车当中，运动的控制与实现是通过上位机与下位机通信来实现的，即 ROS 与 STM32 之间的通信，通过由 ROS 向 STM32 发布速度、STM32 再根据速度信息控制舵机转向（仅阿克曼小车）与电机转速来使小车运动，同时通过 STM32 向 ROS 返回的数据来获取小车的速度与姿态。

2. 使用方法

使用前需要确保上位机连接小车 WiFi，并已经 SSH 远程登录到小车端。

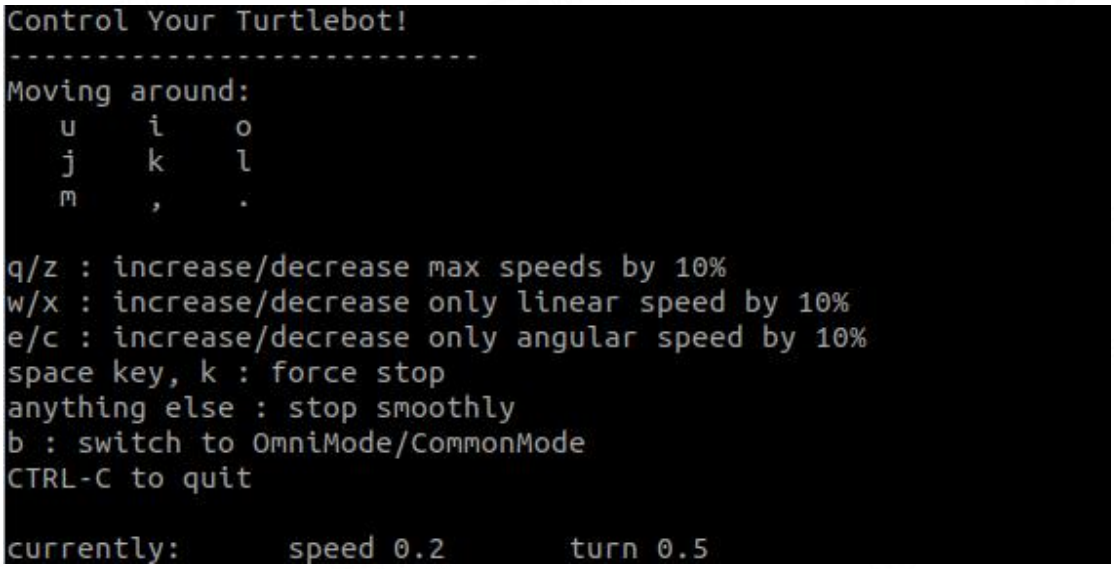
1) 打开终端输入启动底层节点的指令

```
roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch
```

2) 打开终端输入启动键盘控制功能的指令

```
roslaunch wheeltec_robot_rc keyboard_teleop.launch
```

3) 运行后可看到终端提示，如下图所示

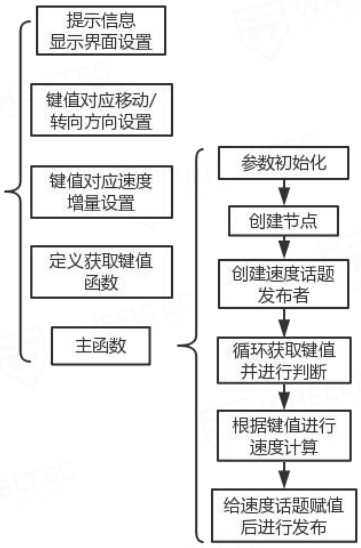


键盘控制终端提示信息

此时就可以通过键盘来控制小车的运动了，根据终端提示来按键可进行小车的移动、转向、速度控制以及模式的切换，全向移动模式仅支持全向轮及麦轮小车使用，阿克曼小车及差速小车因机械结构无法实现自转。

3. 注意事项

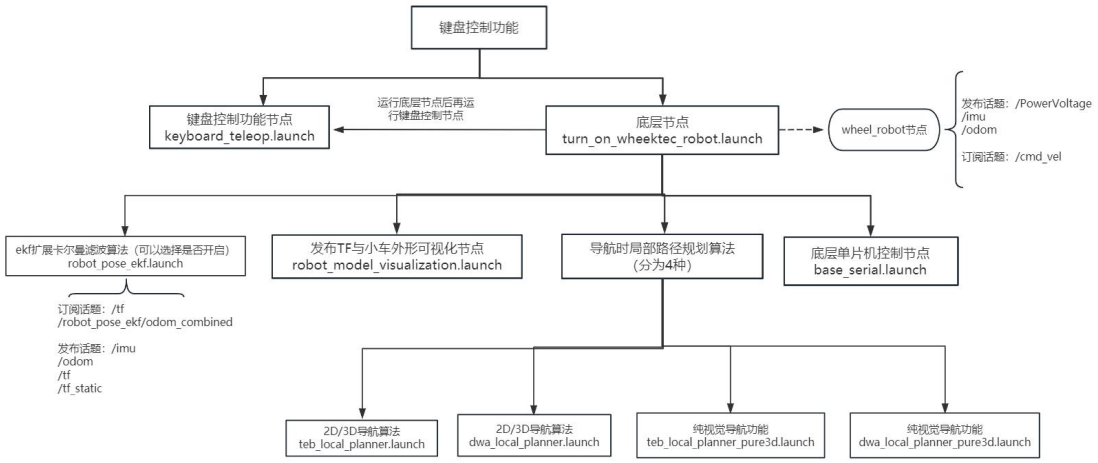
除键盘控制的 launch 文件之外，其他功能的 launch 文件中都嵌套运行了底层节点，无需再运行一次底层节点的 turn_on_wheeltec_robot.launch 文件。键盘控制调用的是 wheeltec_robot_rc 功能包路径下的 keyboard_teleop.launch 文件，launch 文件的内容很简单，调用了 turtlebot_teleop_key.py 这个 python 文件，该 python 文件结构如下：



turtlebot_teleop_key.py 文件结构

4. 功能讲解

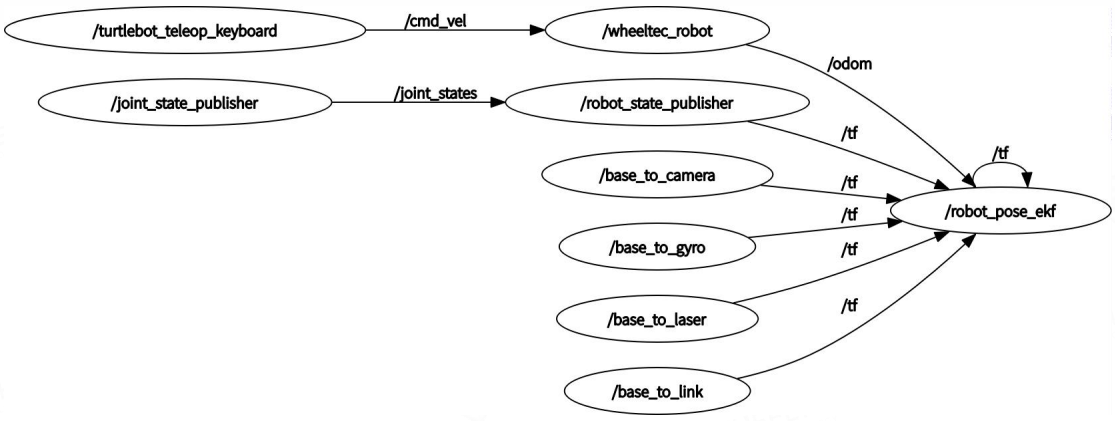
① 启动键盘控制功能



键盘控制功能启动框架

② 键盘控制功能节点关系

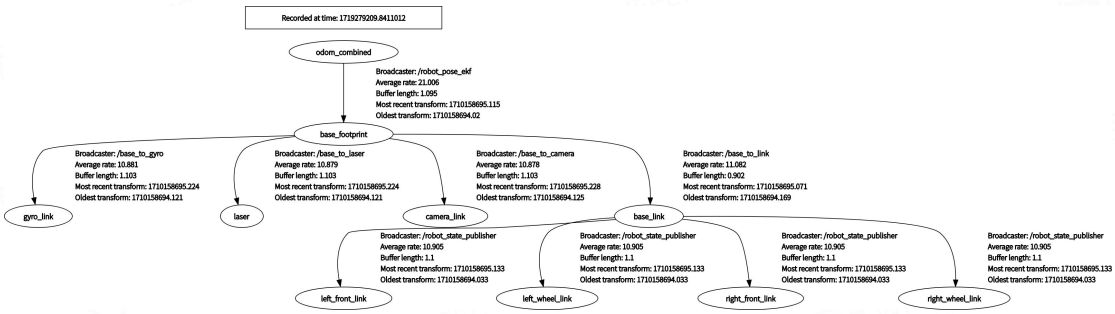
```
rqt_graph
```



键盘控制功能节点关系图

③ 键盘控制功能 TF 树

```
rosrun rqt_tf_tree rqt_tf_tree
```



键盘控制功能 TF 树

④ 小车初始化节点—turn_on_wheeltec_robot.launch

在我们的 ROS 源码当中，通常我们采用 launch 文件来开启一个节点，而小车运动初始化节点的开启是由 turn_on_wheeltec_robot 功能包路径下的 turn_on_wheeltec_robot.launch 文件来实现的，必须启动该 launch 文件才能实现 ROS 与 STM32 的通信，小车才能运动。

这个 launch 文件由五部分组成：

第一部分是各种参数的设置，首先是车型选择的 argument 参数，

```
arg name="car_mode"  default=" mini_mec"
```

参数名称为 car_mode，修改 default 的值可切换不同车型，通常此处设置要与实际车型吻合，因为这里的车型设置影响到了后面将车型相关参数传入其他文件、小车部件的 TF 设置以及对应 urdf 文件的调用。

然后是语音中车型相关的参数设置，若为阿克曼系列车型，则对应参数设置为 yes，这里需要设置是因为在语音功能的声源定位功能中，针对阿克曼系列车型无法自转的问题进行了小车运动控制的优化。

```
  <!--是否为 akm 系列车型 在语音导航功能中需要进行判断-->
  <param if="$(eval car_mode=='mini_akm' or car_mode=='senior_akm' or
car_mode=='top_akm_bs' or car_mode=='top_akm_dl')" name="if_akm_yes_or_no"
value="yes"/>
```

接下来的几个参数与建图导航功能有关，在被调用到时会被重新赋值。

```
    <!--是否开启导航功能 在导航相关文件中开启 此处默认不开启-->
  <arg name="navigation"  default="false"/>
  <arg name="pure3d_nav"  default="false"/>
  <!--是否重复开启底层节点 在语音运行自主建图时开启 此处默认不开启-->
  <arg name="repeat"  default="false"/>
   <!--是否使用 cartographer 建图算法 此处默认不使用-->
  <arg name="is_cartographer"  default="false"/>
  <arg name="odom_frame_id"  default="odom_combined"/>
```

第二部分是开启底层单片机的控制节点，这里嵌套调用了 turn_on_wheeltec_robot 中的 base_serial.launch，也是 turn_on_wheeltec_robot.launch 中最重要的一部分。此处对是否为六自由度机械臂车型有不同的底层节点开启判断。

```
  <group unless="$(eval (car_mode=='mini_mec_moveit_six' or
car_mode=='mini_4wd_moveit_six') and if_voice==true)" >
  <!-- turn on base_serial 开启底层单片机的控制节点  -->
  <include file="$(find
turn_on_wheeltec_robot)/launch/include/base_serial.launch" unless="$(arg
repeat)">
    <arg name="odom_frame_id"  value="$(arg odom_frame_id)"/>
  </include>
  </group>
  <group if="$(eval (car_mode=='mini_mec_moveit_six' or
car_mode=='mini_4wd_moveit_six') and if_voice==true)" >
  <!-- 开启机械臂语音相关节点  -->
  <include file="$(find wheeltec_arm_pick)/launch/base_serial.launch" >
    <arg name="if_voice_control" value="true"/>
    <arg name="moveit_config" value="true"/>
    <arg name="preset" value="true"/>
  </include>
  </group>
```

第三部分是导航时局部路径规划算法的选择，我们的源码中默认选择 teb 算法，如果要使用 dwa 算法可以对开启 teb 算法的语句进行注释，同时将开启 dwa 算法语句的注释去掉，

```
<!--当开启导航功能时 启用导航算法选择-->
  <!--当开启 2D 或 3D 导航功能时-->
```

```
  <!-- 开启 teb_local_planner 导航算法 与 dwa 算法相比效果更佳-->
  <include file="$(find
turn_on_wheeltec_robot)/launch/include/teb_local_planner.launch" if="$(arg
navigation)">
  <!-- 开启 dwa_local_planner 导航算法-->
  <!-- <include file="$(find
turn_on_wheeltec_robot)/launch/include/dwa_local_planner.launch" if="$(arg
navigation)"> -->
    <arg name="car_mode" value="$(arg car_mode)"/>
  </include>

  <!--当开启纯视觉导航功能时-->
  <!-- 开启 teb_local_planner 导航算法 与 dwa 算法相比效果更佳-->
  <include file="$(find
turn_on_wheeltec_robot)/launch/include/teb_local_planner_pure3d.launch"
if="$(arg pure3d_nav)">
  <!-- 开启 dwa_local_planner 导航算法-->
  <!-- <include file="$(find
turn_on_wheeltec_robot)/launch/include/dwa_local_planner_pure3d.launch"
if="$(arg pure3d_nav)"> -->
    <arg name="car_mode" value="$(arg car_mode)"/>
  </include>
```

修改为 dwa 算法举例：

```
     <!-- 开启 teb_local_planner 导航算法-一般用于全向移动阿克曼等车-->
  <!-- <include file="$(find
turn_on_wheeltec_robot)/launch/include/teb_local_planner.launch" if="$(arg
navigation)"> -->
  <!-- 开启 dwa_local_planner 导航算法-一般用于差速车-->
  <include file="$(find
turn_on_wheeltec_robot)/launch/include/dwa_local_planner.launch" if="$(arg
navigation)">
    <arg name="car_mode" value="$(arg car_mode)"/>
  </include>
```

第四部分是发布用于建图、导航的 TF 关系与小车外形可视化，也是嵌套调用了一个 robot_model_visualization.launch 文件来实现，在这个 launch 文件中根据我们的实测数据设定了不同车型小车的 TF 坐标变换关系，同时也根据车型的设定来调用我们根据实际数据所搭建的 urdf 模型，并且将 TF 变换与 urdf 信息进行发布。

第五部分是开启 ekf 扩展卡尔曼滤波算法，当选择建图方式为 cartographer 时不使用该滤波算法，它的主要作用是连接 map 与 footprint，进行小车的定位。

```
<!-- turn on ekf 扩张卡尔曼滤波 发布 map 到 footprint 的 TF,即小车定位 使用
cartographer 算法时不使用该滤波算法-->
  <include file="$(find
turn_on_wheeltec_robot)/launch/include/robot_pose_ekf.launch" unless="$(arg
repeat)">
    <arg name="is_cartographer" value="$(arg is_cartographer)"/>
  </include>
```

⑤ 底层单片机控制节点—base_serial.launch

底层单片机控制节点的开启是通过运行 base_serial.launch 文件来实现的，其中的内容同样可以分为三部分。

第一部分是进行是否开启速度平滑功能的选择，在我们的源码中默认不开启，

```
<arg name="smoother"  default="false"/> <!-- 是否开启速度平滑功能 -->
```

第二部分是开启 wheeltec_robot 节点，wheeltec 节点是小车底层的一个核心节点，将在下一小节中做详细解析。开启节点的同时，对一些参数进行了初始化，串口、波特率、frame_id 等，为数据的传输做准备，

```
 <node pkg="turn_on_wheeltec_robot" type="wheeltec_robot_node"
name="wheeltec_robot" output="screen" respawn="false">
```

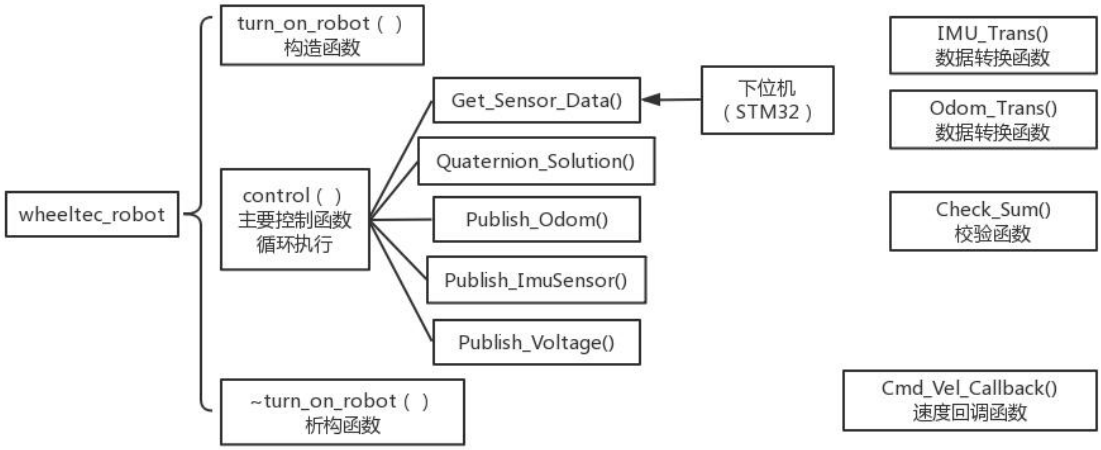
第三部分是运行速度平滑功能包，此处是否运行与前面的参数 smoother 设置相关。

```
<!-- 如果开启了速度平滑功能，则运行速度平滑功能包 -->
  <include if="$(arg smoother)"
    file="$(find
turn_on_wheeltec_robot)/launch/include/velocity_smoother.launch" >
  </include>
```

⑥ wheeltec_robot 节点—wheeltec_robot.cpp

wheeltec_robot 节点是小车的核心节点，它主要实现的是 ROS 与 STM32 的通信，wheeltec_robot 节点的创建与定义在 wheeltec_robot.cpp 文件中。

```
int main(int argc, char** argv)
{
  ros::init(argc, argv, "wheeltec_robot"); //ROS initializes and sets the node name
//ROS 初始化 并设置节点名称
  turn_on_robot Robot_Control; //Instantiate an object //实例化一个对象
  Robot_Control.Control(); //Loop through data collection and publish the topic //
循环执行数据采集和发布话题等操作
  return 0;
}
```



wheeltec_robot.cpp 文件结构

wheeltec_robot.cpp 文件中还由许多子函数组成，下面将从各函数入手来讲解 wheeltec_robot 节点的功能。

第一个要讲的是 Get_Sensor_Data()函数，它的作用是通过串口读取并校验下位机发送过来的数据，然后将数据转换为国际单位。下位机发送过来的一帧数据为 24 位，其中第一位应为帧头，最后一位为帧尾，第 23 位为 BCC 校验位（即倒数第二位），函数首先通过串口读取下位机发送过来的 24 位数据，再检验这一帧数据中的帧头与帧尾位置，并以此对数据进行纠正，纠正后的这一帧数据，第一位定义为帧头，最后一位定义为帧尾。接下来就是对这一帧数据的处理及转化，首先判断帧头是否为 0X7B、帧尾是否为 0X7D，如果同时符合，则进入 BCC 校验，BCC 校验使用的函数为 Check_Sum()，这个函数会将帧头与数据位（即第 1-21 位）按位异或后与校验位（即第 23 位）进行比对，从而判断数据有无错误。BCC 校验通过后，正式进入程序的处理部分。

接收到的第二位数据作为一个预留位，定义为 Receive_Data.Flag_Stop，接下来的 3-8 位分别为运动底盘 X、Y、Z 方向的速度，使用 Odom_Trans()函数将两个八位数据合并为一个十六位数据并做单位转换，转换后单位为 m/s。9-20 位数据为 IMU 三轴加速度与角速度，同样使用 IMU_Trans()函数将两个八位数据合并为一个十六位数据，IMU 所获取的三轴加速度与角速度还需要进行单位转换。所获取的 Odom 与 IMU 数据都是用于计算小车姿态的。21、22 位数据为电池电压数据，同样需要合并两个八位数据后进行单位转换，转换后单位为 V。

第二个要讲的函数为 Publish_Odom()函数，它的主要功能是发布里程计话题，

包含位置、姿态（Z 轴转角）、三轴速度、TF 父子坐标、协方差矩阵。首先将 Z 轴转角转换为四元数进行表达，定义里程计的父坐标为 odom_frame_id、子坐标为 robot_frame_id，并且读取位置与速度信息作为 odom 话题的信息，接下来是扩展卡尔曼滤波相关配置，根据小车为静止还是运动来选择使用的协方差矩阵，完成以上数据处理后就是里程计话题的发布了。

```
void turn_on_robot::Publish_Odom()
{
    //Convert the Z-axis rotation Angle into a quaternion for expression
    //把 Z 轴转角转换为四元数进行表达
    geometry_msgs::Quaternion odom_quat =
tf::createQuaternionMsgFromYaw(Robot_Pos.Z);

    nav_msgs::Odometry odom; //Instance the odometer topic data //实例化里程计话
题数据
    odom.header.stamp = ros::Time::now();
    odom.header.frame_id = odom_frame_id; // Odometer TF parent coordinates //
里程计 TF 父坐标
    odom.pose.pose.position.x = Robot_Pos.X; //Position //位置
    odom.pose.pose.position.y = Robot_Pos.Y;
    odom.pose.pose.position.z = Robot_Pos.Z;
    odom.pose.pose.orientation = odom_quat; //Posture, Quaternion converted by
Z-axis rotation //姿态，通过 Z 轴转角转换的四元数

    odom.child_frame_id = robot_frame_id; // Odometer TF subcoordinates //里程计
TF 子坐标
    odom.twist.twist.linear.x =  Robot_Vel.X; //Speed in the X direction //X 方
向速度
    odom.twist.twist.linear.y =  Robot_Vel.Y; //Speed in the Y direction //Y 方
向速度
    odom.twist.twist.angular.z = Robot_Vel.Z; //Angular velocity around the Z axis
//绕 Z 轴角速度

    //There are two types of this matrix, which are used when the robot is at rest
and when it is moving.Extended Kalman Filtering officially provides 2 matrices for
the robot_pose_ekf feature pack
    //这个矩阵有两种，分别在机器人静止和运动的时候使用。扩展卡尔曼滤波官方提供的
2 个矩阵，用于 robot_pose_ekf 功能包
    if(Robot_Vel.X== 0&&Robot_Vel.Y== 0&&Robot_Vel.Z== 0)
      //If the velocity is zero, it means that the error of the encoder will be
relatively small, and the data of the encoder will be considered more reliable
      //如果 velocity 是零，说明编码器的误差会比较小，认为编码器数据更可靠
      memcpy(&odom.pose.covariance, odom_pose_covariance2,
```

```
sizeof(odom_pose_covariance2)),
      memcpy(&odom.twist.covariance, odom_twist_covariance2,
sizeof(odom_twist_covariance2));
    else
      //If the velocity of the trolley is non-zero, considering the sliding error
that may be brought by the encoder in motion, the data of IMU is considered to be
more reliable
      //如果小车 velocity 非零，考虑到运动中编码器可能带来的滑动误差，认为 imu 的
数据更可靠
      memcpy(&odom.pose.covariance, odom_pose_covariance,
sizeof(odom_pose_covariance)),
      memcpy(&odom.twist.covariance, odom_twist_covariance,
sizeof(odom_twist_covariance));
    odom_publisher.publish(odom); //Pub odometer topic //发布里程计话题
}
```

第三个要讲的函数为 Publish_ImuSensor()，设置 IMU 对应 TF 坐标为 gyro_frame_id，接着读取转换获得的四元数来作为 IMU 三轴姿态信息，再将三轴加速度与角速度的数据作为 IMU 相关话题中的内容，并定义了三轴姿态协方差矩阵和三轴角速度协方差矩阵，最后将 IMU 话题的信息发布出去。

```
void turn_on_robot::Publish_ImuSensor()
{
  sensor_msgs::Imu Imu_Data_Pub; //Instantiate IMU topic data //实例化 IMU 话题数
据
  Imu_Data_Pub.header.stamp = ros::Time::now();
  Imu_Data_Pub.header.frame_id = gyro_frame_id; //IMU corresponds to TF
coordinates, which is required to use the robot_pose_ekf feature pack
                                                //IMU 对应 TF 坐标，使用
robot_pose_ekf 功能包需要设置此项
  Imu_Data_Pub.orientation.x = Mpu6050.orientation.x; //A quaternion represents a
three-axis attitude //四元数表达三轴姿态
  Imu_Data_Pub.orientation.y = Mpu6050.orientation.y;
  Imu_Data_Pub.orientation.z = Mpu6050.orientation.z;
  Imu_Data_Pub.orientation.w = Mpu6050.orientation.w;
  Imu_Data_Pub.orientation_covariance[0] = 1e6; //Three-axis attitude covariance
matrix //三轴姿态协方差矩阵
  Imu_Data_Pub.orientation_covariance[4] = 1e6;
  Imu_Data_Pub.orientation_covariance[8] = 1e-6;
  Imu_Data_Pub.angular_velocity.x = Mpu6050.angular_velocity.x; //Triaxial
angular velocity //三轴角速度
  Imu_Data_Pub.angular_velocity.y = Mpu6050.angular_velocity.y;
  Imu_Data_Pub.angular_velocity.z = Mpu6050.angular_velocity.z;
  Imu_Data_Pub.angular_velocity_covariance[0] = 1e6; //Triaxial angular velocity
```

```
covariance matrix //三轴角速度协方差矩阵
  Imu_Data_Pub.angular_velocity_covariance[4] = 1e6;
  Imu_Data_Pub.angular_velocity_covariance[8] = 1e-6;
  Imu_Data_Pub.linear_acceleration.x = Mpu6050.linear_acceleration.x; //Triaxial
acceleration //三轴线性加速度
  Imu_Data_Pub.linear_acceleration.y = Mpu6050.linear_acceleration.y;
  Imu_Data_Pub.linear_acceleration.z = Mpu6050.linear_acceleration.z;
  imu_publisher.publish(Imu_Data_Pub); //Pub IMU topic //发布 IMU 话题
}
```

第四个要讲的函数为 Publish_Voltage()，它的作用是发布电压相关信息，这里面做了一个降低发布频率的处理，使电压发布的频率比其它数据发布频率降低 10 倍。

```
void turn_on_robot::Publish_Voltage()
{
    std_msgs::Float32 voltage_msgs; //Define the data type of the power supply
voltage publishing topic //定义电源电压发布话题的数据类型
    static float Count_Voltage_Pub=0;
    if(Count_Voltage_Pub++>10)
      {
        Count_Voltage_Pub=0;
        voltage_msgs.data = Power_voltage; //The power supply voltage is obtained
//电源供电的电压获取
        voltage_publisher.publish(voltage_msgs); //Post the power supply voltage
topic unit: V, volt //发布电源电压话题单位：V、伏特
      }
}
```

第五个要讲的函数是 Control()函数，这个函数是 whheltec_robot 节点的最主要的功能函数，功能在于循环获取下位机数据与发布话题。首先获取两次进入该函数的时间间隔，用于后面进行速度积分来获得里程数据，接着通过 Get_Sensor_Data()函数来进行串口读取并校验下位机发送过来的数据，利用串口所读取到的数据进行 X 方向与 Y 方向位移的计算与绕 Z 轴的角位移的计算，使用 Quaternion_Solution()进行四元数解算 Quaternion_Solution()函数的定义在同路径下的 Quaternion_Solution.cpp 文件中，通过 IMU 绕三轴角速度与三轴加速度计算三轴姿态，最后调用 Publish_Odom()、Publish_ImuSensor()、Publish_Voltage()函数进行里程计、IMU 以及电量话题的发布。

```
void turn_on_robot::Control()
{
  while(ros::ok())
```

```
  {
    if (true == Get_Sensor_Data_New()) //The serial port reads and verifies the data sent by the lower computer, and then the data is converted to international units
                                       //通过串口读取并校验下位机发送过来的数据,然后数据转换为国际单位
    {
      _Now = ros::Time::now();
      if(_Last_Time.toSec()==0) _Last_Time=_Now; //Perform this operation when entering for the first time to avoid excessive integration time
                                                 //首次进入时进行此操作，避免积分时间过大
      Sampling_Time = (_Now - _Last_Time).toSec(); //Retrieves time interval, which is used to integrate velocity to obtain displacement (mileage)
                                                   //获取时间间隔，用于积分速度获得位移(里程)

      //Odometer correction parameters
      //里程计误差修正
      Robot_Vel.X=Robot_Vel.X*odom_x_scale;
      Robot_Vel.Y=Robot_Vel.Y*odom_y_scale;
      if(Robot_Vel.Z>=0)
        Robot_Vel.Z=Robot_Vel.Z*odom_z_scale_positive;
      else
        Robot_Vel.Z=Robot_Vel.Z*odom_z_scale_negative;

      //Speed * Time = displacement (odometer)
      //速度*时间=位移（里程计）
      Robot_Pos.X+=(Robot_Vel.X * cos(Robot_Pos.Z) - Robot_Vel.Y * sin(Robot_Pos.Z)) * Sampling_Time; //Calculate the displacement in the X direction, unit: m //计算 X 方向的位移，单位：m
      Robot_Pos.Y+=(Robot_Vel.X * sin(Robot_Pos.Z) + Robot_Vel.Y * cos(Robot_Pos.Z)) * Sampling_Time; //Calculate the displacement in the Y direction, unit: m //计算 Y 方向的位移，单位：m
      Robot_Pos.Z+=Robot_Vel.Z * Sampling_Time; //The angular displacement about the Z axis, in rad //绕 Z 轴的角位移，单位：rad

      //Calculate the three-axis attitude from the IMU with the angular velocity around the three-axis and the three-axis acceleration
      //通过 IMU 绕三轴角速度与三轴加速度计算三轴姿态
      Quaternion_Solution(Mpu6050.angular_velocity.x, Mpu6050.angular_velocity.y, Mpu6050.angular_velocity.z,\
                  Mpu6050.linear_acceleration.x, Mpu6050.linear_acceleration.y, Mpu6050.linear_acceleration.z);
```

```
      Publish_Odom();      //Pub the speedometer topic //发布里程计话题
      Publish_ImuSensor(); //Pub the IMU topic //发布 IMU 话题
      Publish_Voltage();   //Pub the topic of power supply voltage //发布电源电
压话题

      _Last_Time = _Now; //Record the time and use it to calculate the time interval
//记录时间，用于计算时间间隔
    }

    ros::spinOnce();   //The loop waits for the callback function //循环等待回调
函数
    }
}
```

速度话题的回调函数是 Cmd_Vel_Callback()，在这个回调函数中，向下位机传送的数据一帧为 11 位，第 1 位为帧头 0x7B，第 2、3 位为预留位，第 4-9 位分别为/机器人 x、y 轴的目标线速度和 z 轴目标角速度，其中，只有麦轮、全向轮小车才可以直接进行 y 轴移动，而差速与阿克曼系列车型无 y 轴速度，在 ROS 中虽然有做 y 轴速度处理，但在 STM32 源码中不做处理，第 10 位为校验位，第 11 位为帧尾 0X7D，完成 11 位数据的处理之后，将回调函数的数据通过串口向下位机写入。

```
void turn_on_robot::Cmd_Vel_Callback(const geometry_msgs::Twist &twist_aux)
{
  short  transition;  //intermediate variable //中间变量

  Send_Data.tx[0]=FRAME_HEADER; //frame head 0x7B //帧头 0X7B
  Send_Data.tx[1] = 0; //set aside //预留位
  Send_Data.tx[2] = 0; //set aside //预留位

  //The target velocity of the X-axis of the robot
  //机器人 x 轴的目标线速度
  transition=0;
  transition = twist_aux.linear.x*1000; //将浮点数放大一千倍，简化传输
  Send_Data.tx[4] = transition;      //取数据的低 8 位
  Send_Data.tx[3] = transition>>8;  //取数据的高 8 位

  //The target velocity of the Y-axis of the robot
  //机器人 y 轴的目标线速度
  transition=0;
  transition = twist_aux.linear.y*1000;
  Send_Data.tx[6] = transition;
  Send_Data.tx[5] = transition>>8;
```

```
  //The target angular velocity of the robot's Z axis
  //机器人 z 轴的目标角速度
  transition=0;
  transition = twist_aux.angular.z*1000;
  Send_Data.tx[8] = transition;
  Send_Data.tx[7] = transition>>8;

  Send_Data.tx[9]=Check_Sum(9,SEND_DATA_CHECK); //For the BCC check bits, see the
Check_Sum function //BCC 校验位，规则参见 Check_Sum 函数
  Send_Data.tx[10]=FRAME_TAIL; //frame tail 0x7D //帧尾 0X7D
  try
  {
    Stm32_Serial.write(Send_Data.tx,sizeof (Send_Data.tx)); //Sends data to the
downloader via serial port //通过串口向下位机发送数据
  }
  catch (serial::IOException& e)
  {
    ROS_ERROR_STREAM("Unable to send data through serial port"); //If sending data
fails, an error message is printed //如果发送数据失败，打印错误信息
  }
}
```

wheeltec_robot.cpp 文件中还有两个特殊的函数：构造函数与析构函数，它们都只执行一次，其中构造函数 turn_on_robot()在开始时执行一次，主要用于初始化，在这个函数中创建了一个节点句柄，用于参数服务器的调用，对程序中使用的参数进行初始化，

```
ros::NodeHandle private_nh("~"); //Create a node handle //创建节点句柄
```

接下来创建了三个发布者，发布的话题分别为电量话题 PowerVoltage、里程计话题 odom 和 IMU 话题 mobile_base/sensors/imu_data，

```
 voltage_publisher = n.advertise<std_msgs::Float32>("PowerVoltage", 10); //Create
a battery-voltage topic publisher //创建电池电压话题发布者
  odom_publisher    = n.advertise<nav_msgs::Odometry>("odom", 50); //Create the
odometer topic publisher //创建里程计话题发布者
  imu_publisher     = n.advertise<sensor_msgs::Imu>("imu", 20); //Create an IMU
topic publisher //创建 IMU 话题发布者
```

同时创建一个订阅者，订阅各车型的速度话题，并定义相应的回调函数，

```
 Cmd_Vel_Sub= n.subscribe(cmd_vel,100, &turn_on_robot::Cmd_Vel_Callback, this);
```

最后就是开启串口准备与下位机进行通信的部分了，同时打印相应的提示消息

```
try
```

```
  {
    //Attempts to initialize and open the serial port //尝试初始化与开启串口
    Stm32_Serial.setPort(usart_port_name); //Select the serial port number to
enable //选择要开启的串口号
    Stm32_Serial.setBaudrate(serial_baud_rate); //Set the baud rate //设置波特率
    serial::Timeout _time = serial::Timeout::simpleTimeout(2000); //Timeout //
超时等待
    Stm32_Serial.setTimeout(_time);
    Stm32_Serial.open(); //Open the serial port //开启串口
  }
  catch (serial::IOException& e)
  {
    ROS_ERROR_STREAM("wheeltec_robot can not open serial port,Please check the
serial port cable! "); //If opening the serial port fails, an error message is printed
//如果开启串口失败，打印错误信息
  }
  if(Stm32_Serial.isOpen())
  {
    ROS_INFO_STREAM("wheeltec_robot serial port opened"); //Serial port opened
successfully //串口开启成功提示
  }
}
```

析构函数~turn_on_robot()则是在节点关闭时执行一次，它通过向下位机发送停止运动命令，除帧头帧尾保留外其他数据都清零，使小车停止运动，至此，节点运行就结束了，终端会打印对应的提示信息。