障碍清理机器人设计及制作（七）

控制程序设计

完成机器人硬件系统搭建后即可开始进行控制程序设计，类似于赋予机器人灵魂，让机器人按工作要求实现运动与完成任务动作。

1. 控制程序实现

（1）工具

本机器人基于 c 语言的开发，使用了 STM studio、串口助手等辅助软件。

（2）平台

所有的控制程序都基于 stm32f103c8t6。

（3）算法

机器人使用到了多个算法。最典型的是 PID 控制算法。本算法用于对电机的精确控制和对任务动作的动态调节。

2. 系统简述

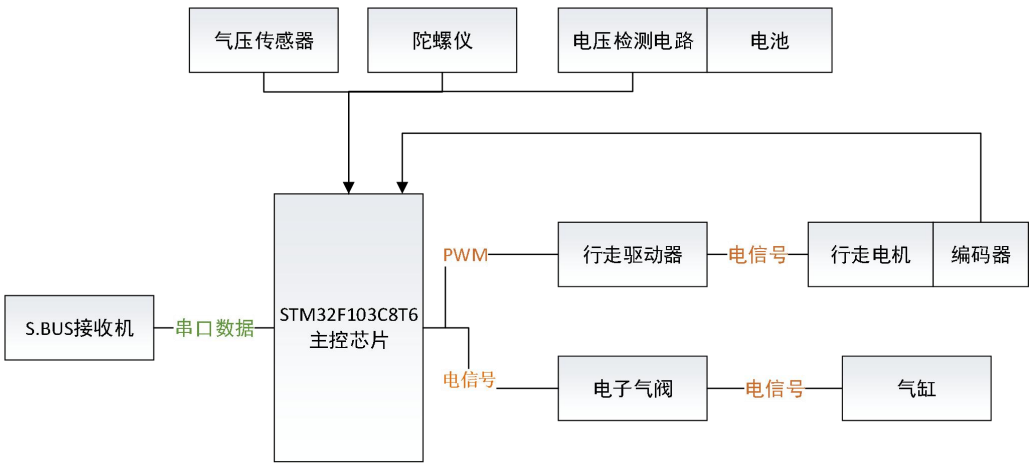

图 1 机器人控制结构

如上图所示，机器人采用电气混合驱动的方式实现目标功能，其中，机器人的移动使用了电机驱动万向轮，在行走灵活的同时，也能提供良好的动力。对障碍杆的清除，动力部分采用了气动弹射的方式，经过精密计算的抛射轨道，能快速而准确地完成对障碍物的清理。

3. 控制逻辑示意图

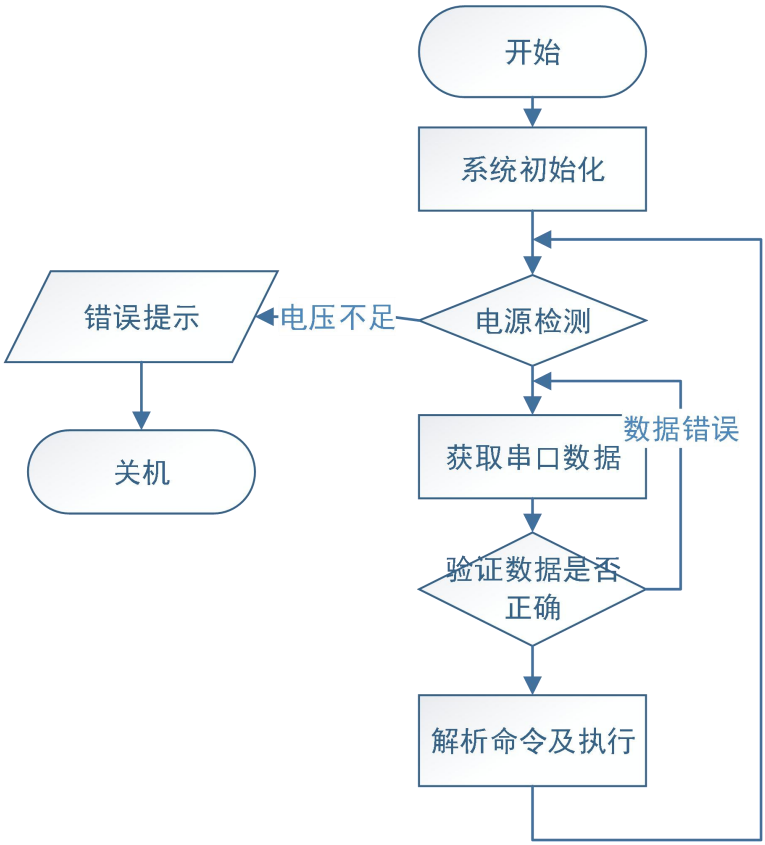

图 2 控制流程图

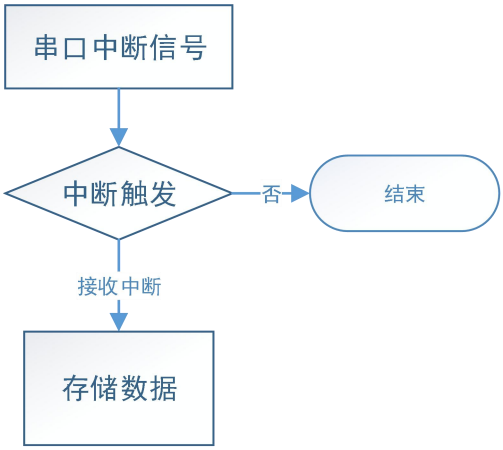

图 3 串口数据接收

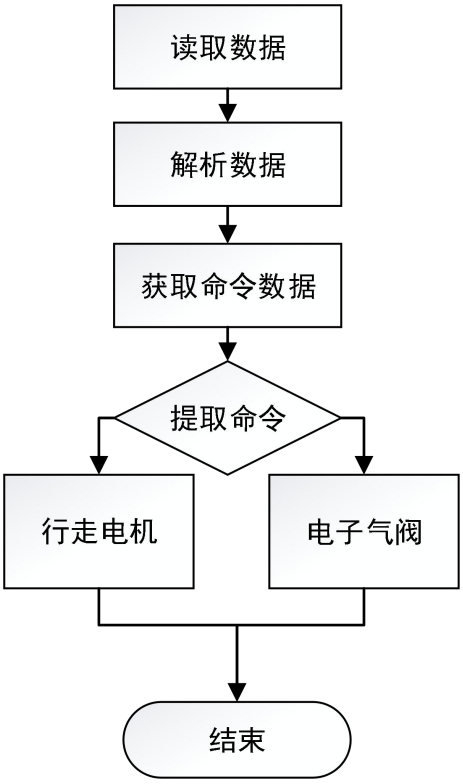

图 4 命令执行流程

机器人的所有动作均采用了自动控制原理，在控制器发出动作指令的同时，自动地对电机、气缸等动作系统进行动态的调节，保证所有的动作都能完成预期目标。

4. 算法部分

1. PID 算法

PID 算法是上个世纪 30 年代左右提出的控制算法，大至航空航天、小至家庭温度调控都可以使用 PID 算法，虽然 PID 算法从提出到现在已经历经了快一个世纪，其后也出现了很多现代的智能算法，比如蒙特卡洛、智能控制等等，但现在 PID 仍然经久不衰，可以说目前 80%以上的控制仍然使用 PID 算法。

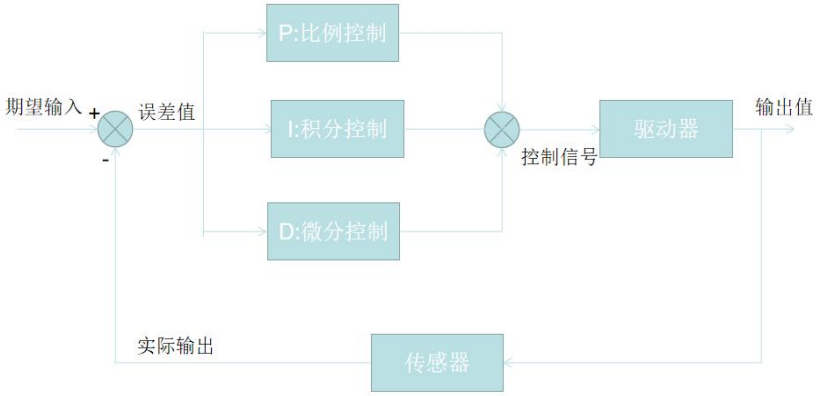

图 5 PID 算法控制框图

上图为 PID 算法的控制框图，在我们控制电机速度时，期望输入就是电机的期望速度值，期望输入与由编码器测得的实际速度作差，求出的误差值传给 PID 的控制部分，算出需要输出的控制信号，将该控制信号传给控制器，也就是输出给电机驱动板 L298N，这样形成一个循环，就实现了对电机速度的精准控制。

$$U(t) = K_p \cdot Err + K_i \cdot \int Err(t)dt + K_d \cdot \frac{dErr}{dt}$$

中间 PID 的控制部分的连续型公式如下：

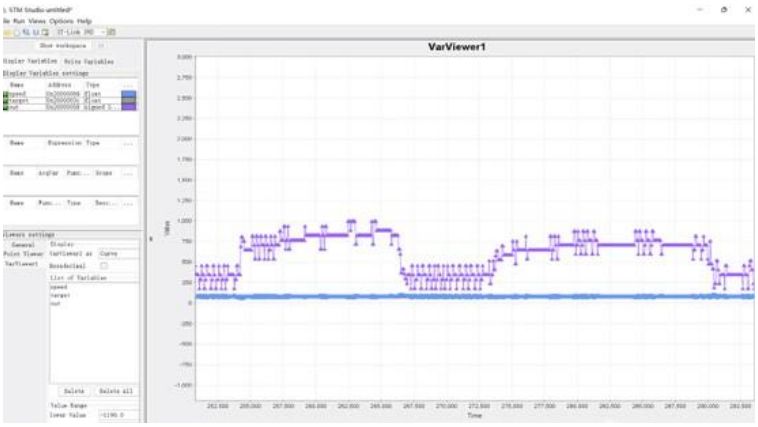

图 6 PID 控制部分连续型图像

但是在计算机中计算机很难实现连续型变量的积分或者微分操作，因此在计算机中，我们使用离散型的积分和微分，就是取时间间

$$U(n) = K_p \cdot Err + K_i \cdot \sum_{n}^{i} Err(i) + K_d \cdot (Err(n) - Err(n-1))$$

隔 T 为 1，离散型 PID 公式如下：

各个项的主要作业及效果如下：

P：增加快速性，过大会引起震荡和超调，P 单独作用会一直有静态误差

I：减少静态误差，过大会引起震荡

D：减小超调，过大会使响应速度变慢

```
//计时器计时
void TIM3_IRQHandler (void)
{
  if(TIM_GetITStatus(TIM3,TIM_IT_Update)==SET)
  {
    circle=Read_circle_count();//圈数
    scale= Read_scale();//走过的刻度值
    speed=circle;

      //增量式
      cnt=speed;
      speed_now=cnt;//当前实际速度
      err_now=target-speed_now;//误差速度=期望值-实际值
      jisuan=KP*(err_now-err_last)+KI*err_now+KD*(err_now+err_last-2*err_last);
      out+=jisuan;
      if(out>19000)
      out=19000;
      PWM_SetCompare1(out);
      err_last_last=err_last;
      err_last=err_now;

      printf("%d",speed_now);
  }
    TIM_ClearITPendingBit(TIM3,TIM_IT_Update);
}
```

图 7 算法实现代码事宜

通过上述操作，即可以实现电机的精准速度控制

在电机的速度 PID 控制算法中，因为使用 PI 算法就够了，所以我们使用了增量式 PID 算法，这样可以让我们的公式和代码更加简洁。

$$U(n) = U(n-1) + K_p \cdot (Err(n) - Err(n-1)) + K_i \cdot Err(n)$$

积分限幅

因为积分的效果是累加，随着时间的推移，积分项的值会升到很高，积分本来的作用是用来减小静态误差，但积分项过大会引起过大的震荡，所以我们可以加一个判断函数 if，当积分项的值达到一定值后，就让积分项保持这个值，避免引起更大的震荡。

积分分离

如果刚开始的误差比较大，那么积分项则会在刚开始就累计到了一个很大的数值，那么当第一次实际输出达到期望值时，不会立刻停止，而是会产生一个很大的过冲。这时就需要用到积分分离，就是当误差值过大时，我们就不使用积分项，只让 PD 项单独作用，当误差值较小后，在加入积分项，以减小静态误差。

2. **Dijkstra 算法**

针对道路狭窄，障碍物较多等复杂环境下无法快速探索行进路径曲折等问题，机器人使用了本算法。Dijkstra 算法又叫迪杰斯特拉算法。该算法采用了一种贪心模式，其解决的是有向图中单个节点到另一节点的最短路径问题，其主要特点是每次迭代时选择的下一个节点是当前节点最近的子节点，也就是说每一次迭代行进的路程是最短的。而为了保证最终搜寻到的路径最短，在每一次迭代过程中，都要对起始节点到所有遍历到的点之间的 最短路径进行更新，具体看一下过程。

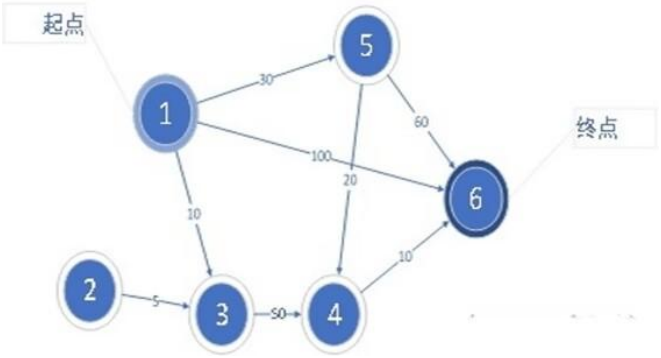

图 8 节点有向图

初始化：

建立 distance[](起点到其他所有点的距离信息)、Top_node[]（最短路径信息）两个列表存放信息。其中，distance[]的维度为节点的个数，每一个数值为到达对应索引节点的最短路径距离，比如 distance[2]的值代表当前迭代时刻到达 3 号节点的最短距离。初始状态 distance[0 inf 10 inf 30 100]，其中 0 代表自身，inf 代表无法到达；Top_node[num1]，其中 num1 代表一号节点并以此类推。

搜索最小点：

找到当前节点到下一点的最小值，即从 num1 开始搜索到 1->5/1->3/1->6 三条路，并找到距离最小的路 1->3。则此时到达 num3 点的最短路径确定为 10，将 num3 存入 Top_node[]。

松弛：

确定 num3 找到最短路径，然后 num3 开始搜寻其弧尾，找到 3->4 路径，此时 1->3->4 路径距离为 10+50=60，小于 inf，故将列表更新为 distance[0 inf 10 60 30 100]。注意这里通过 3->4 这条路径缩短 1->4 这条路径的过程叫做“松弛”，该算法证实通过这样的方法进行路径寻优。

重复迭代：

除去 num1 和 num3，从剩余点搜寻距离最小，找到 num5，故将 num5 加入 Top_node[]。找到弧尾路径 5->4/5->6，进行松弛，其中 1->5->4 距离为 30+20=50<60，1->5->6 距离为 30+60=90<100，所以列表更新为 distance[0 inf 10 50 30 90]。

重复迭代：

除去 num1、num3 和 num5，其余点寻最小，找到 num4，将其加入 Top_node[]。然后找到弧尾 4->6，进行松弛，1->5->4->6 距离 30+20+10=60<90，1->3->4->6 距离 10+50+10=70>60，进行列表更新 distance[0 inf 10 50 30 60]。

重复迭代：

除去 num1、num3、num4 和 num5，其余点寻最小，找到 num6，将其加入 Top_node[]，然后没有找到弧尾，则此时到达 num6 的最优路径找到。

以上就是 Dijkstra 算法的简单实现，其主要是通过贪心原则逐个遍历最小子节点，然后利用松弛方法去优化路径选择，最终将最优路径存放到可读列表当中，以此来解决最优路径规划问题。

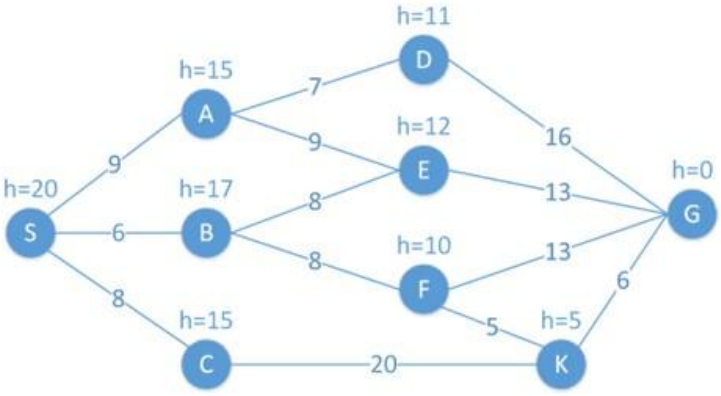

图 9 Dijkstra 算法节点有向图