移动射击机器人设计及制作（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. 控制系统框图

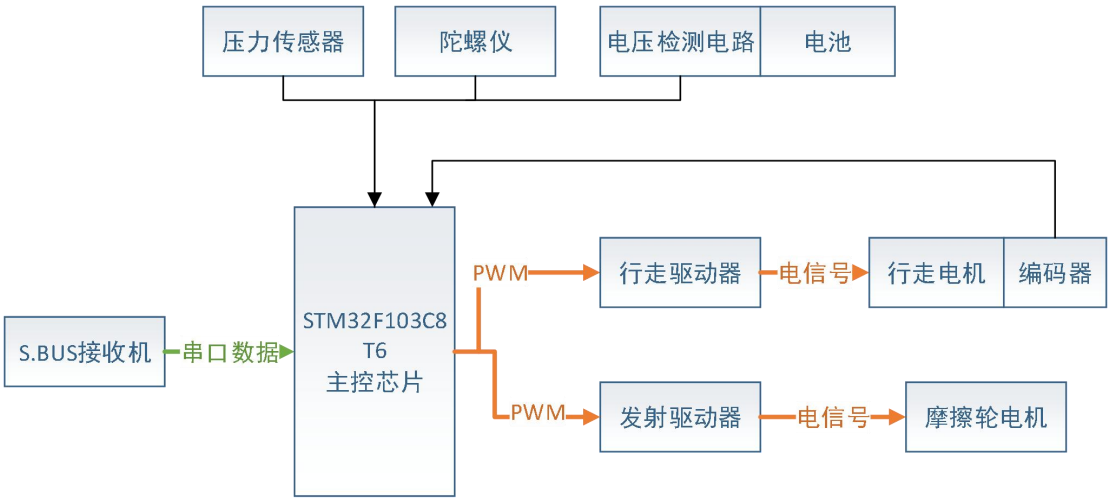

图 1　机器人控制系统

机器人采用中断方式进行串口通信，读取遥控接收端里面的数据，其发射装置使用了两个无刷电机摩擦轮为动力源，以高速旋转发射的方式将炮弹发射并通过差速来调整发射的方向。多个传感器的使用将装填、瞄准再到发射的所有步骤自动完成，有效提高命中和概率。

3. 控制逻辑示意图

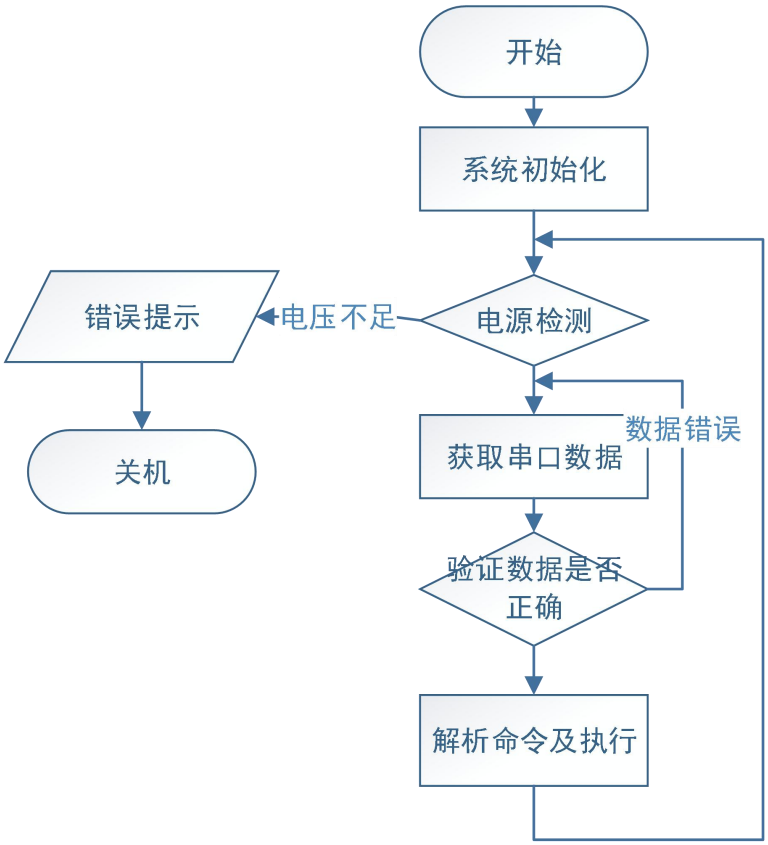

图 2　控制流程图

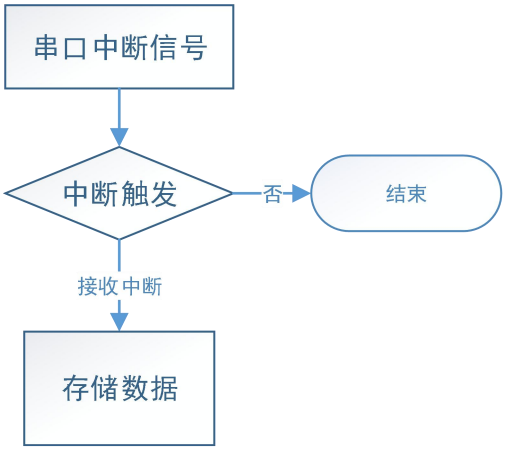

图 3　串口数据接收

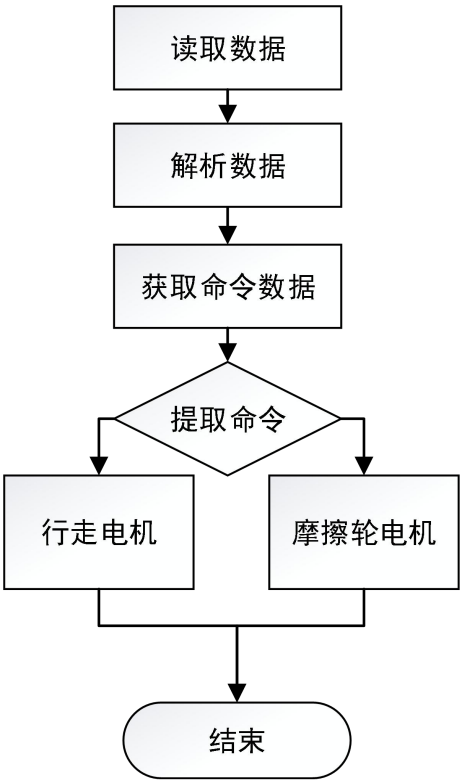

图 4　命令执行流程

4. 使用的算法：

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。

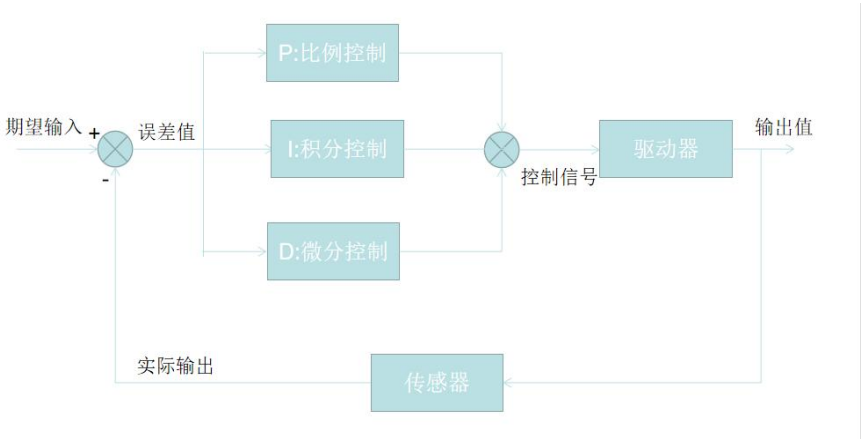

图 5　PID 算法控制框图

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 pwm 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。

中间 PID 的控制部分的连续型公式如下：

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

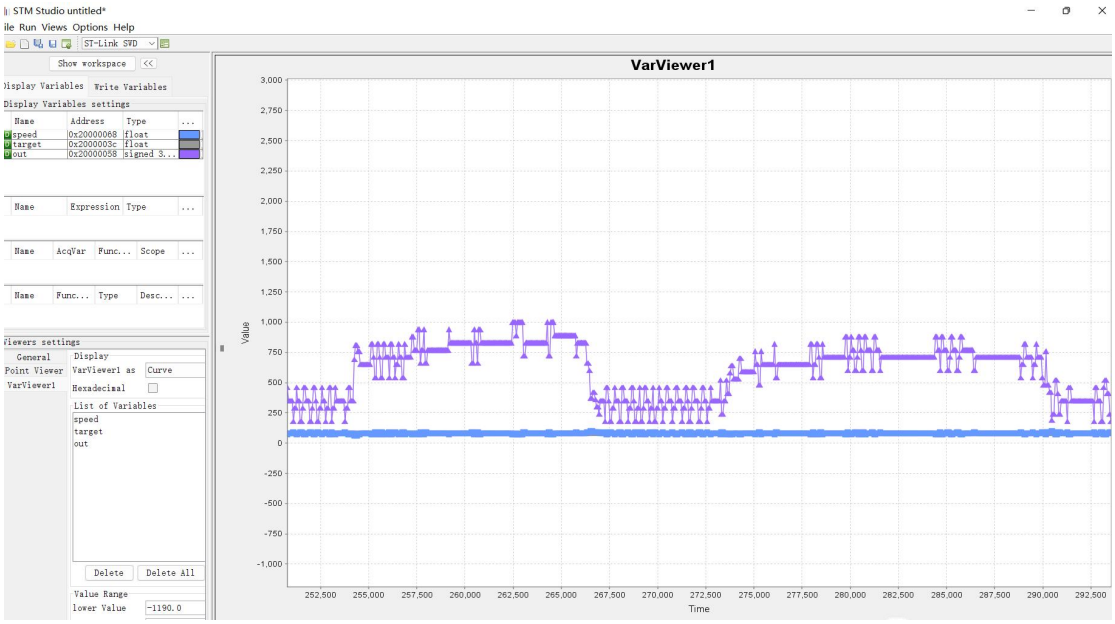

图 6　PID 控制部分连续型图像

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

隔 T 为 1，离散型 PID 公式如下：

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

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

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

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。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。

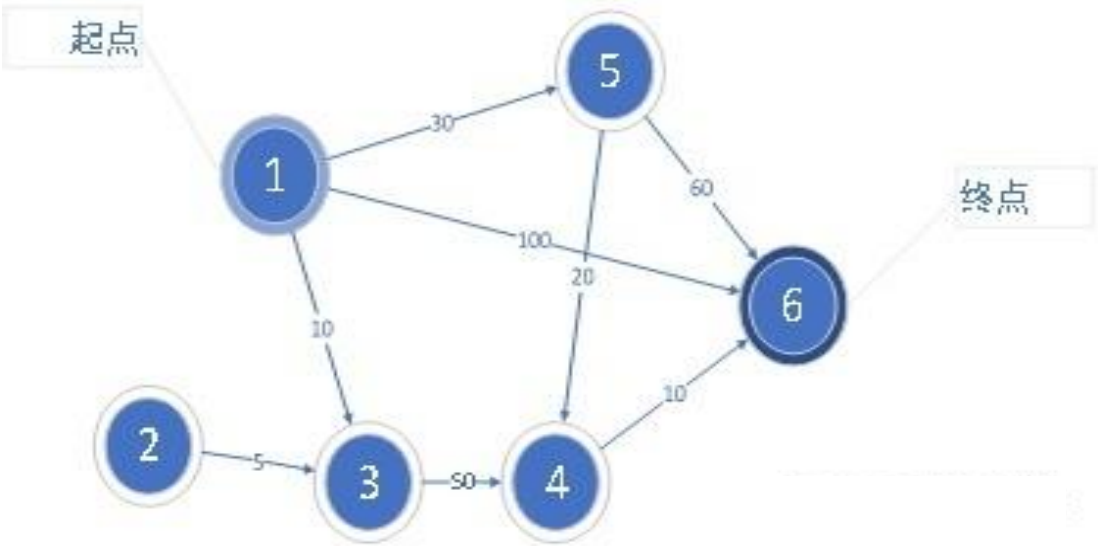

图 7 节点有向图

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。

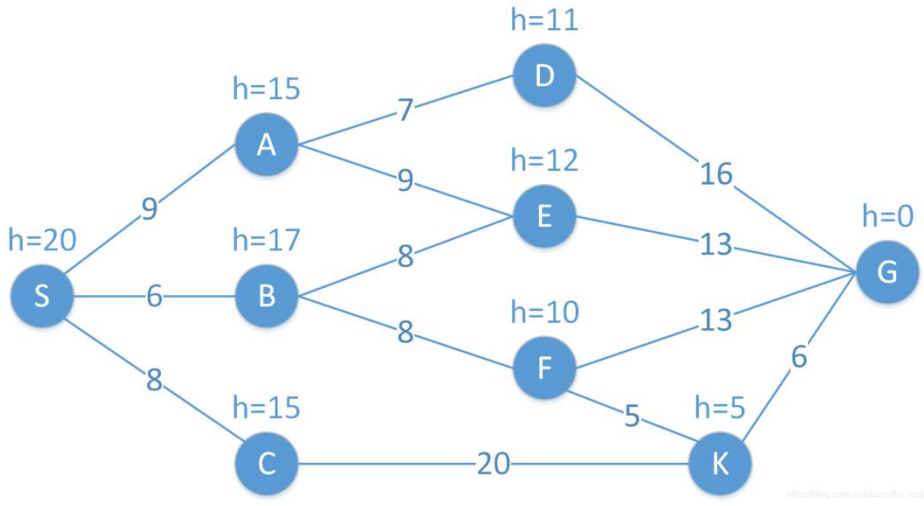

图 8　Dijkstra 算法节点有向图

3 A*算法

A* 算法是启发式搜索算法，是根据 Dijkstra 算法改进而来，启发式搜索即在搜索过程中建立启发式搜索规则，以此来衡量实时搜索位置和目标位置的距离关系，使搜索方向优先朝向目标点所处位置的方向，最终达到提高搜索效率的效果，此处用于目标方向的确定。

A*算法的基本思想如下：引入当前节点 x 的估计函数 f(x),当前节点 x 的估计函数定义为：

f (x)= g(x)+h(x)

其中 g(x)是从起点到当前节点 x 的实际距离量度（代码中用两点之间距离代替）；h(x)是从节点 x 到终点的最小距离估计，h(x)的形式从欧几里得距离（）或者曼哈顿距离中选取。

算法基本实现过程为：从起始点开始计算其每一个子节点的 f 值，从中选择 f 值最小的子节点作为搜索的下一点，往复迭代，直到下一子节点为目标点。

如下图所示，S 为起始（start）节点，G 为目标（goal）节点。

节点之间连线是两点的路径长度，如 A 到 E 的路径长度 c(A,E) = 9。DF

节点旁的 h 值时当前节点到达目标节点（G）的预估值，如 h(A)=15,

表示从当前点 A 到达目标点 G 的估计路径长度为 15，此处 h(x)即为启发函数，启发函数的设计有很多方法，具体可参考链接，此处不再扩展。

从起点 S 到达当前节点 x 的路径长度表示为 g(x) 。

从起点 S 到达目标 G 并经过点 x 的估计距离长度表示为 f(x) = g(x) + h(x)，该公式是 A*算法的核心公式。