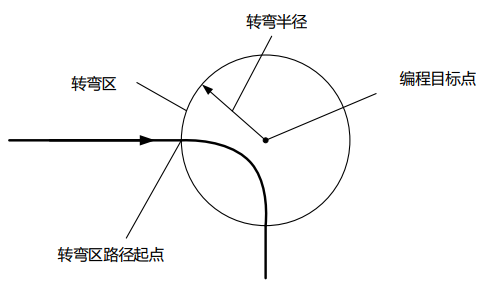
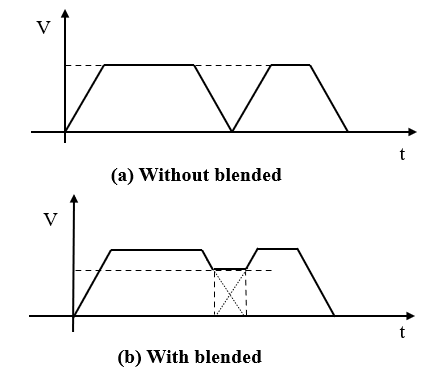
# 轨迹融合

对于同一个机器人指令目标点，在运动指令中有两种处理方式：  
1. 当做停止点处理，机器人将运动到目标点且到达目标点时的速度为 0，之后才会继续执  
行下一条指令；  
2. 当做过渡点处理，机器人不会运动到目标点，而是从距离该目标点若干毫米的地方开始  
转向往下一个目标点运动，转弯路径会偏离编程路径。两条轨迹之间的过渡区域我们称  
之为转弯区，见下图：



第二种策略有利于减少机器人的启停过程，提高工作节拍。实现第二种策略的方法有两种。

* 空间上融合。如上图所示，可在机器人空间相邻两条轨迹直接设计一条过渡曲线，该曲线需考虑与两边曲线过渡中的位置、速度、加速度、加加速度等的连续性；对于相邻直线间设计这类曲线较容易，可考虑Bezier曲线，圆弧过渡时曲率变化不连续，而Bezier曲线可保证曲率连续。Codeit系统中MoveLL指令间是通过该策略实现的。
* 时间上融合。如下图所示，将相邻两条轨迹的速度进行交错融合，也可实现该目的。Coceit中的MoveJ、MoveC、MoveL三类指令间两两间是通过该策略实现的



Codeit中的类型变量zone {dis, per}是用来定义这种融合的程度，其中MoveLL融合使用的是zone.dis来定义融合的长度，zone.dis小于相邻两条直线长度的一般（当大于时，底层会将其变为较小长度的一半）；zone.per用来定义MoveJ、MoveL、MoveC指令间的融合程度，为相邻两个轨迹的时间百分比。假设前后两条轨迹的完成时间分别为t1和t2，当t1\*zone.per>t2时，zone.per会设定为t2/t1。

一条轨迹可以前后两条轨迹同时融合。当此轨迹在与前一条融合时，该轨迹内部的

轨迹融合时关于数据保存说明：codeit对于每个指令，会自动生成一个rt\_log—time—id.txt文档，保存这个指令中通过”lout<<”保存的一些数据。而对于需要进行前瞻和轨迹融合的指令，比如指令A与指令B，若在指令A的executeRT()函数调用过一次指令B的executeRT()函数，则A、B指令通过”lout<<”保存的数据都会保存在指令A对应的txt中；若此后的B指令也会调用到C指令的executeRT()函数，则指令C的内容依然会保存在A对应的txt中。

造成这种现象的本质原因是指令对应的txt的生成是在server.cpp中的tg()函数判断++imp\_->count==1来实现的，正常count值的累加只在该处实现；而若通过指令A调用过了指令B的executeRT()函数，会在该函数体内实现imp\_->count++,从而导致tg()中指令B的txt文档生成失败，指令B的数据依然保存在A对应的txt文档。