文档说明 for MASProject aka libbitctrl

1. 参数初始化

函数表

|  |  |  |
| --- | --- | --- |
| 类别 | 归属 | 函数名 |
| 配置函数 | B | void setConfigDir(const std::string& dir) |
| 初始化 | B | Controller(int robotID); |
|  | D | Combined(int robotID); |
| 读取函数 | B | int loadconfig(sscfg::ConfigFile& file) virtual; |
|  | B | int loadconfig(const std::string& file\_name); |
|  | B | int loadmapdata(bool overwrite); |
|  | D | int loadconfig(sscfg::ConfigFile& file) virtual; |
|  | D | int loadconfig\_debug(sscfg::ConfigFile& file); |
|  | D | int loadconfig\_shape(sscfg::ConfigFile& file); |
|  |  | size\_t MapInfo::load(const std::string& name, bool append); |
| 调用函数 |  | int ControllerInit(int expID, int robotID); |
|  | D | int setFunction(int funcID, bool is\_on); |

（B表示基类Controller，D表示子类Combined）

（1）初始化过程

1. ControllerInit中new Combined(robotID)
   1. Contoller(robotID)中将基本参数加入ConfigSet对象config
   2. Combined(robotID)中将各算法参数等信息加入ConfigSet对象config
2. ControllerInit中设置配置文件目录  
   pderived->setConfigDir(config\_dir);
3. ControllerInit中读取配置pbase->loadconfig(conf)，因为pbase指向的是子类的对象，所以此处会调用子类的函数，在子类的loadconfig函数中：
   1. 调用基类的loadconfig，基类的loadconfig中：
      1. 调用loadmapdata(true);后者调用mapinfo.load(obfile, false);  
         最终从obstacles.txt中载入并覆盖静态障碍物信息于mapinfo中
      2. 调用config.load(cfgfile);  
         更新**所有**由config进行管理的参数（包括基类和子类的参数）  
         Note: 此load函数中如果发现找不到要读取的参数会直接退出
   2. 调用loadconfig\_debug(file)读取用于配置Debug信息的参数
   3. 调用loadconfig\_shape(file)读取队形配置参数dxy, dxy\_base  
      调用onFormationChange()更改相应参数
   4. 读取不进行故障检测的节点，更新dont\_detect
   5. 读取要进行延迟测试的节点，更新latency\_test
   6. 依据是否自动故障，更新auto\_faulty
   7. 依据是否虚拟人的位置，更新tracer\_default\_shift
   8. 依据是否采用NSAvoid算法，更新nsavoid，包括  
      利用nsavoid.set\_unicycle\_point设置线性化原点位置  
      利用mapinfo中的障碍物信息更新nsavoid.ob\_static() (重置已有)  
      调用nsavoid.ob\_static\_changed();
   9. 将configset中的参数输出到参数文件示例config\_example.txt中

一、利用proto和SimuPanel监视车的运动状态

（1）