1.global_planner

```
GlobalPlanner:                                  # Also see:
http://wiki.ros.org/global_planner
  old_navfn_behavior: false                     # Exactly mirror behavior of
navfn, use defaults for other boolean parameters, default false
  use_quadratic: true                           # Use the quadratic approximation
of the potential. Otherwise, use a simpler calculation, default true
  use_dijkstra: true                            # Use dijkstra's algorithm.
Otherwise, A*, default true
  use_grid_path: false                          # Create a path that follows the
grid boundaries. Otherwise, use a gradient descent method, default false
  allow_unknown: true                           # Allow planner to plan through
unknown space, default true
                                                #Needs to have
track_unknown_space: true in the obstacle / voxel layer (in costmap_commons_param)
to work
  planner_window_x: 0.0                         # default 0.0
  planner_window_y: 0.0                         # default 0.0
  default_tolerance: 0.5                        # If goal in obstacle, plan to the
closest point in radius default_tolerance, default 0.0

  publish_scale: 100                            # Scale by which the published
potential gets multiplied, default 100
  planner_costmap_publish_frequency: 0.0        # default 0.0

  lethal_cost: 253                              # default 253
  neutral_cost: 66                              # default 50
  cost_factor: 0.55                              # Factor to multiply each cost
from costmap by, default 3.0
  publish_potential: true                       # Publish Potential Costmap (this
is not like the navfn pointcloud2 potential), default true
```

move_base 中的`base_global_planner`配置为 `base_global_planner: global_planner/GlobalPlanner`

```
<library path="lib/libglobal_planner">
  <class name="global_planner/GlobalPlanner" type="global_planner::GlobalPlanner" base_class_type="nav_core::BaseGlobalPlanner">
    <description>
      A implementation of a grid based planner using Dijkstras or A*
    </description>
  </class>
</library>
```

先看下`global_planner`的接口定义(前面讲过所有的实际的都是该接口的实现)

```
class BaseGlobalPlanner{
  public:
   /**
    * @brief Given a goal pose in the world, compute a plan
    * @param start The start pose
    * @param goal The goal pose
    * @param plan The plan... filled by the planner
    * @return True if a valid plan was found, false otherwise
    */
   virtual bool makePlan(const geometry_msgs::PoseStamped& start,
      const geometry_msgs::PoseStamped& goal, std::vector<geometry_msgs::PoseStamped>& plan) = 0;

   /**
    * @brief Given a goal pose in the world, compute a plan
    * @param start The start pose
    * @param goal The goal pose
    * @param plan The plan... filled by the planner
    * @param cost The plans calculated cost
    * @return True if a valid plan was found, false otherwise
    */
   virtual bool makePlan(const geometry_msgs::PoseStamped& start,
                   const geometry_msgs::PoseStamped& goal, std::vector<geometry_msgs::PoseStamped>& plan,
                   double& cost)
   {
     cost = 0;
     return makePlan(start, goal, plan);
   }

   /**
    * @brief  Initialization function for the BaseGlobalPlanner
    * @param  name The name of this planner
    * @param  costmap_ros A pointer to the ROS wrapper of the costmap to use for planning
    */
   virtual void initialize(std::string name, costmap_2d::Costmap2DROS* costmap_ros) = 0;

   /**
    * @brief  Virtual destructor for the interface
    */
   virtual ~BaseGlobalPlanner(){}

  protected:
   BaseGlobalPlanner(){}
}? end BaseGlobalPlanner ? ;
```

接口很简单，总共只有三个还有个重载函数，看名字就知道，一个初始化，还有个是规划路径，可以的话你也可以实现这些接口完成你自己的`global_planner`，目前可以使用的有三种

- `navfn/NavfnROS` 使用`Dijkstra’s`算法代价最小的规划
- `global_planner/GlobalPlanner` 提供更多选项支持不同配置
- `carrot_planner/CarrotPlanner`

-`allow unknown(true)`

- `use dijkstra(true)`
- `use quadratic(true)`
- `use grid path(false)`
- `old navfn behavior(false)` 这些设置默认参数即可
- `default_tolerance` 当目标点为障碍时，规划可以有一定的允许误差
- `lethal_cost`
- `neutral_cost`

- `cost_factor`

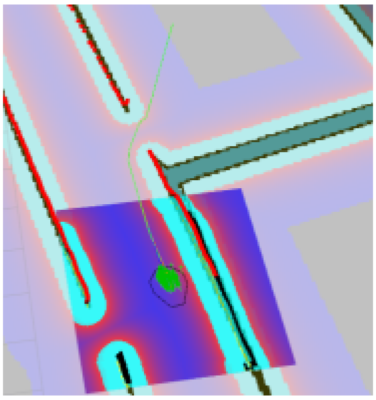

Figure 5: cost_factor = 0.01

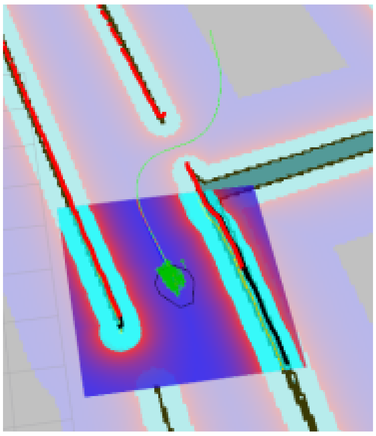

Figure 6: cost_factor = 0.55

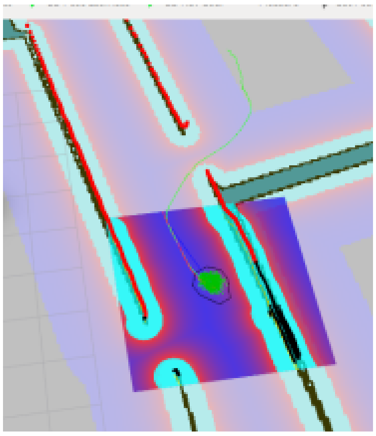

Figure 7: cost_factor = 3.55

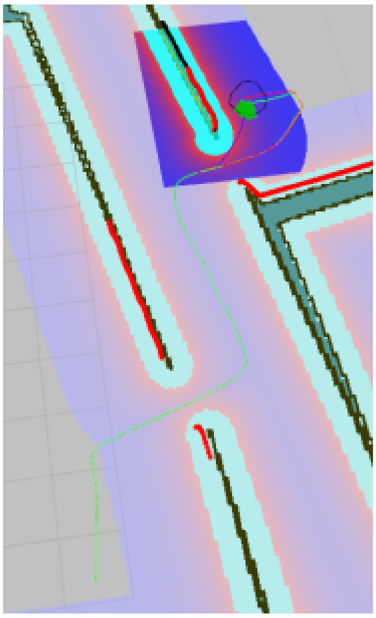

Figure 8: neutral_cost = 1

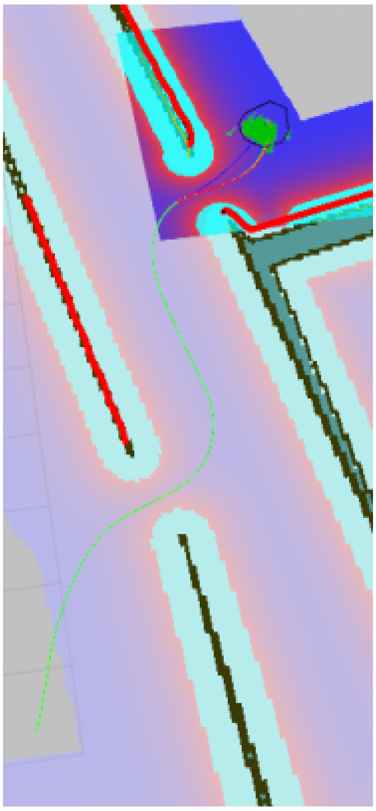

Figure 9: neutral_cost = 66

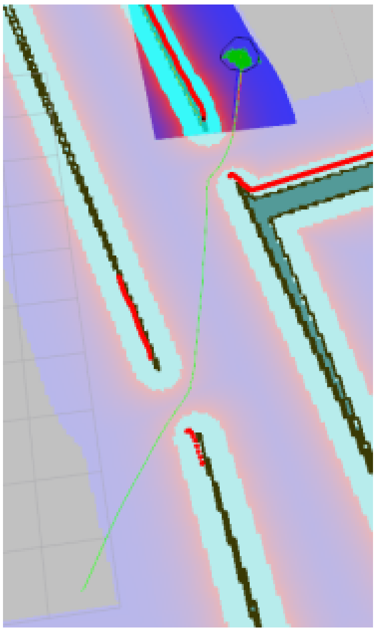

Figure 10: neutral_cost = 233

> 摘自【ROS Navigation Tuning Guide】

2.local_planner

```
DWAPlannerROS:

# Robot Configuration Parameters - Kobuki
  max_vel_x: 0.25
```

```
  min_vel_x: 0.05

  max_vel_y: 0
  min_vel_y: 0

  max_trans_vel: 0.35 # choose slightly less than the base's capability
  min_trans_vel: 0.001  # this is the min trans velocity when there is negligible
rotational velocity
  trans_stopped_vel: 0.05

  # Warning!
  #   do not set min_trans_vel to 0.0 otherwise dwa will always think
translational velocities
  #   are non-negligible and small in place rotational velocities will be created.

  max_rot_vel: 0.6  # choose slightly less than the base's capability
  min_rot_vel: 0.4  # this is the min angular velocity when there is negligible
translational velocity
  rot_stopped_vel: 0.1

  acc_lim_x: 1 # maximum is theoretically 2.0, but we
  acc_lim_theta: 1.5
  acc_lim_y: 0      # diff drive robot

# Goal Tolerance Parameters
  yaw_goal_tolerance: 0.2
  xy_goal_tolerance: 0.15
  latch_xy_goal_tolerance: true

# Forward Simulation Parameters
  sim_time: 2.0       # 1.7
  vx_samples: 10       # 3
  vy_samples: 1
  vtheta_samples: 20  # 20

# Trajectory Scoring Parameters
  path_distance_bias: 32.0      # 32.0   - weighting for how much it should stick
to the global path plan
  goal_distance_bias: 24.0      # 24.0   - wighting for how much it should attempt
to reach its goal
  occdist_scale: 0.4            # 0.01   - weighting for how much the controller
should avoid obstacles
  forward_point_distance: 0.325 # 0.325  - how far along to place an additional
scoring point
  stop_time_buffer: 0.2         # 0.2    - amount of time a robot must stop in
before colliding for a valid traj.
  scaling_speed: 0.25           # 0.25   - absolute velocity at which to start
scaling the robot's footprint
  max_scaling_factor: 0.2       # 0.2    - how much to scale the robot's footprint
when at speed.

# Oscillation Prevention Parameters
  oscillation_reset_dist: 0.05  # 0.05   - how far to travel before resetting
oscillation flags
```

```
# Debugging
  publish_traj_pc : true
  publish_cost_grid_pc: true
  global_frame_id: odom


# Differential-drive robot configuration - necessary?
#  holonomic_robot: false
```

move_base 中的`base_local_planner`配置为 `base_local_planner: "dwa_local_planner/DWAPlannerROS"`

```
<library path="lib/libdwa_local_planner">
  <class name="dwa_local_planner/DWAPlannerROS" type="dwa_local_planner::DWAPlannerROS" base_class_type="nav_core::BaseLocalPlanner">
    <description>
      A implementation of a local planner using either a DWA approach based on configuration parameters.
    </description>
  </class>
</library>
```

同样该类实现了`base_local_planner`的接口，我们看下接口

```
class BaseLocalPlanner{
  public:
    /**
     * @brief  Given the current position, orientation, and velocity of the robot, compute velocity commands to send to the base
     * @param cmd_vel Will be filled with the velocity command to be passed to the robot base
     * @return True if a valid velocity command was found, false otherwise
     */
    virtual bool computeVelocityCommands(geometry_msgs::Twist& cmd_vel) = 0;

    /**
     * @brief  Check if the goal pose has been achieved by the local planner
     * @return True if achieved, false otherwise
     */
    virtual bool isGoalReached() = 0;

    /**
     * @brief  Set the plan that the local planner is following
     * @param plan The plan to pass to the local planner
     * @return True if the plan was updated successfully, false otherwise
     */
    virtual bool setPlan(const std::vector<geometry_msgs::PoseStamped>& plan) = 0;

    /**
     * @brief  Constructs the local planner
     * @param name The name to give this instance of the local planner
     * @param tf A pointer to a transform listener
     * @param costmap_ros The cost map to use for assigning costs to local plans
     */
    virtual void initialize(std::string name, tf::TransformListener* tf, costmap_2d::Costmap2DROS* costmap_ros) = 0;

    /**
     * @brief  Virtual destructor for the interface
     */
    virtual ~BaseLocalPlanner(){}

  protected:
    BaseLocalPlanner(){}
} ? end BaseLocalPlanner ? ;
```

接口也不算复杂，字面理解分别为：

- 计算速度
- 是否到达目标点
- 下发全局路径
- 初始化 参数说明
- `max_vel_x` `min_vel_x` `max_vel_y` `min_vel_y`速度限定值

- max_trans_vel min_trans_vel 平移速度限定值
- trans_stopped_vel未使用
- max_rot_vel min_rot_vel 旋转的速度限定值
- rot_stopped_vel未使用
- acc_lim_x acc_lim_theta acc_lim_y 加速度限定值
- yaw_goal_tolerance xy_goal_tolerance 到达目标点的允许误差
- latch_xy_goal_tolerance 如果为true 当机器人到达目标点后通过旋转调整姿态（方向）后，偏离了目标点，也认为完成。这个实际应用中还是比较酷的
- sim_time 模拟机器人以采样速度行走的时间，太小(<2)会导致行走不流畅，特别在遇到障碍或狭窄的空间，因为没有足够多时间获取路径；太大(>5)会导致以僵硬的轨迹行走使得机器人不太灵活

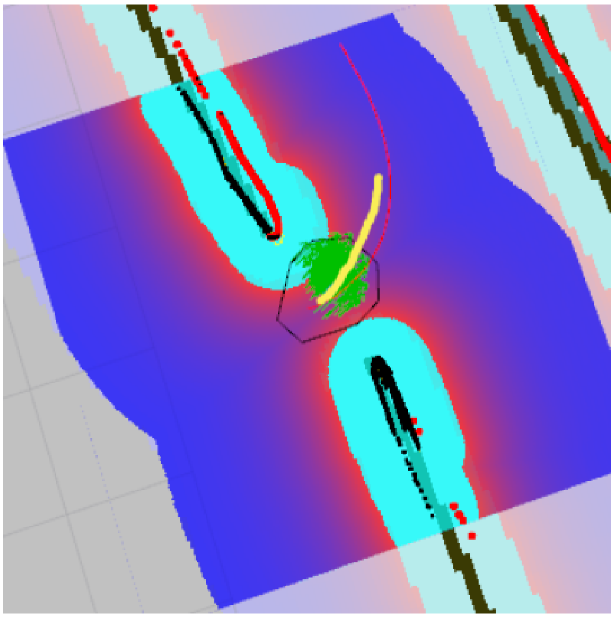

Figure 11: sim_time = 1.5

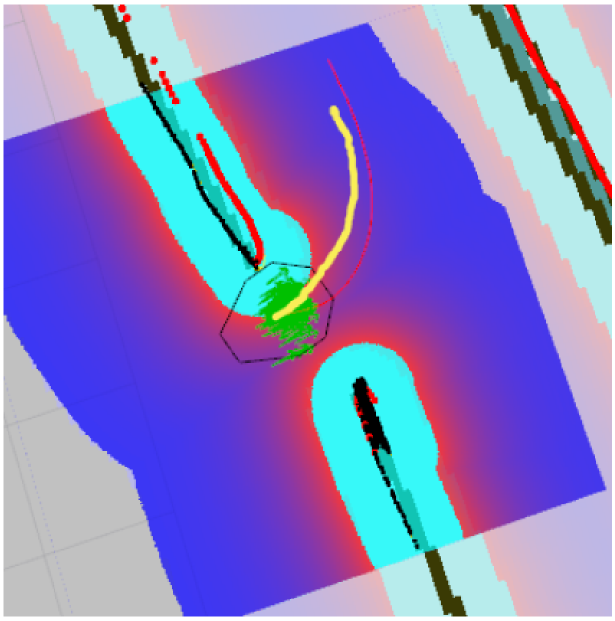

Figure 12: sim_time = 4.0

- vx_samples vy_samples vtheta_samples采样速度个数, 一般vtheta_samples大于vx_samples vy_samples怎么不是0?查看源码即可得到答案， 最小为1,即使设置<=0也会重新置1

```
if (vx_samp <= 0) {
  ROS_WARN("You've specified that you don't want any samples in the x dimension.
  vx_samp = 1;
  config.vx_samples = vx_samp;
}

if (vy_samp <= 0) {
  ROS_WARN("You've specified that you don't want any samples in the y dimension.
  vy_samp = 1;
  config.vy_samples = vy_samp;
}

if (vth_samp <= 0) {
  ROS_WARN("You've specified that you don't want any samples in the th dimension.
  vth_samp = 1;
  config.vth_samples = vth_samp;
}
```

- `path_distance_bias` `goal_distance_bias` `occdist_scale` 轨迹代价计算
  ```
  cost = pdist_scale_ * path_dist + goal_dist * gdist_scale_ + occdist_scale_ * occ_cost;
  ```
  - `path_dist` 规划最后一个点距离全局路径的距离，即决定`local_plan`多接近`global_plan`
  - `goal_distance` 规格最后一个点距离local目标距离，决定机器人接近目标
  - `occdist_scale` 路径中避障代价

另外还有

- `sim_granularity` 轨迹上的点的密集程度