1. Struttura del codice

ROS è strutturato in nodi. All'interno di un'applicazione vengono eseguiti più nodi, indipendenti l'uno dall'altro, che comunicano fra loro tramite messages, topics e services(…spiegare…).

Per quanto riguarda la nostra applicazione useremo il package bosch-ros-pkg, che ci offre una buona base per i nostri test e i nostri sviluppi.

L'esplorazione degli ambienti sfrutterà quindi nodi dei seguenti package:

- explore
- stage (stageros)
- gmapping
- move_base
- tf

Nel nostro caso particolare, la struttura del codice è la seguente.

Il nostro package (bosh-ros-pkg/trunk/stacks/exploration/explore_stage) contiene i seguenti file e directory:

- config/
- explore/
- explore.launch
- explore.vcg
- explore.xml
- explore_slam.launch
- explore_slam.xml
- manifest.xml
- move.xml

Il file manifest.xml descrive le dipendenze del nostro package da package esterni ad esso.

Per lanciare l'applicazione richiamiamo il file explore_slam.launch, ce si trova all'interno del path "/trunk/stacks/bosch_demos/explore_stage". Questo file permette di lanciare tutti insieme i diversi nodi che fanno parte della nostra applicazione e di collegarli tra loro, di specificare dei parametri e di includere file xml, che a loro volta possono includere file di configurazione yaml.

Ad esempio, il file explore_slam.launch include i file explore_slam.xml e move.xml.

Il file explore_slam.xml include e dichiara i seguenti file e parametri:

- file con scope limitato al node-package "explore":
    - (file) config/footprint.yaml
    - (file) config/costmap_common.yaml
    - (file) explore/explore_costmap.yaml

- altri parametri, anch'essi con scope limitato al node-package "explore":
    - (parametro) potential_scale

- (parametro) orientation_scale
- (parametro) gain_scale
- (parametro) close_loops

- <remap from="slam_entropy" to="gmapping/entropy"/>

Il file move.xml include i seguenti file con scope limitato al node-package “move_base”:

- config/footprint.yaml
- config/costmap_common.yaml (namespace = “global_costmap”)
- explore/global_costmap.yaml
- explore/navfn_params.yaml
- config/costmap_common.yaml(namespace = “local_costmap”)
- explore/local_costmap.yaml
- explore/trajectory_planner_params.yaml

Parametri

Parametri dichiarati nel file config/ footprint.yaml:

```
footprint: [[-0.2825, -0.395], [0.2825, -0.395], [0.2825, -0.2], [0.4025, -0.2],
[0.4025, 0.2], [0.2825, 0.2], [0.2825, 0.395], [-0.2825, 0.395]]
```

Parametri dichiarati nel file config/costmap_common.yaml:

```
map_type: costmap
transform_tolerance: 0.5
obstacle_range: 2.5
max_obstacle_height: 2.0
raytrace_range: 3.0
inscribed_radius: 0.385
circumscribed_radius: 0.685
inflation_radius: 0.6
cost_scaling_factor: 15.0
lethal_cost_threshold: 100

observation_sources: base_scan

base_scan: {
sensor_frame: base_laser_link,
data_type: LaserScan,
expected_update_rate: 0.2,
observation_persistance: 0.0,
marking: true,
clearing: true
}
```

Parametri dichiarati nel file explore/explore_costmap.yaml:

```
explore_costmap:
  global_frame: /map
  robot_base_frame: base_link
```

```
  update_frequency: 1.0
  publish_frequency: 0.0
  raytrace_range: 5.0
  obstacle range: 5.0
  static_map: false
  rolling_window: false
  width: 60.0
  height: 60.0
  resolution: 0.2
  origin_x: -30.0
  origin_y: -30.0
  track_unknown_space: true
```

Parametri dichiarati nel file explore/global_costmap.yaml:

```
#Independent settings for the planner's costmap
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 1.0
  publish_frequency: 0.5
  raytrace_range: 30.0
  obstacle_range: 5.0
  static_map: false
  rolling_window: false
  width: 60.0
  height: 60.0
  resolution: 0.05
  origin_x: -30.0
  origin_y: -30.0
```

Parametri dichiarati nel file explore/local_costmap.yaml:

```
#Independent settings for the local planner's costmap
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 1.0
  publish_frequency: 0.5
  static_map: false
  rolling_window: true
  width: 3.0
  height: 5.0
  resolution: 0.05
  origin_x: 0.0
  origin_y: 0.0
```

Parametri dichiarati nel file explore/navfn_params.yaml:

```
NavfnROS:
  transform_tolerance: 0.3
```

Parametri dichiarati nel file explore/trajectory_planner_params.yaml:

```
TrajectoryPlannerROS:
  transform_tolerance: 0.3
  world_model: costmap
  sim_time: 1.7
  sim_granularity: 0.025
  dwa: true
  vx_samples: 3
  vtheta_samples: 20
  max vel x: 0.65
  min_vel_x: 0.1
  max_vel_th: 1.0
  min_vel_th: -1.0
  min_in_place_vel_th: 0.4
  xy_goal_tolerance: 0.1
  yaw_goal_tolerance: 0.02
  goal_distance_bias: 0.8
  path_distance_bias: 0.6
  occdist_scale: 0.01
  heading_lookahead: 0.325
  oscillation_reset_dist: 0.05
  escape_reset_dist: 0.15
  escape_reset_theta: 0.30
  acc_lim_th: 1.0
  acc_lim_x: 0.5
  acc_lim_y: 0.0
  heading_scoring: false
  heading_scoring_timestep: 0.8
  holonomic_robot: false
  simple_attractor: false
```

Come utilizzare i parametri all'interno dell'applicazione

```
private_nh.param("planner_frequency", planner_frequency_, 1.0);
```

oppure

```
ros::NodeHandle nh;
std::string global_name, relative_name, default_param;
if (nh.getParam("/global_name", global_name))
{
  ...
}

if (nh.getParam("relative_name", relative_name))
{
...
}

// Default value version
nh.param<std::string>("default_param", default_param, "default_value");
```

Cuore dell'applicazione

Quella che è per noi la parte di maggior interesse dell'applicazione, cioè quella che coinvolge direttamente la strategia di esplorazione, è tutta racchiusa nei seguenti file cpp:

- bosh-ros-pkg/trunk/stacks/exploration/explore/src/explore.cpp
- bosh-ros-pkg/trunk/stacks/exploration/explore/src/explore_frontier.cpp

In particolare, il primo file è importante per discriminare tra percezione/decisione continua/discreta, mentre il secondo è importante per quanto riguarda la scelta dei goal sulla frontiera e la scelta del prossimo goal da raggiungere.

2. Modifiche da apportare al codice

2.1. Diverse strategie di esplorazione

Nuovo parametro per scegliere la strategia di esplorazione:

*exploration_strategy_*: [0, 1, 2, …].

Dove:

- 0 equivale RANDOM
- 1 equivale a WEIGHTED (pesata)
- 2 equivale a MCDM

Aggiungere questo parametro sia nei file di configurazione legati al package explore, sia nel codice sorgente dell'applicazione, più precisamente nel file explore.cpp

Nuove funzioni per le diverse strategie di esplorazione:

Bisognerà aggiungere una nuova funzione per ogni strategia di esplorazione che vogliamo prendere in considerazione e collegarla all'attuale chiamata di funzione per la scelta del prossimo goal (getExplorationGoals() ?), ma anche, ovviamente, collegarla al parametro relativo alla strategia di esplorazione.

Esempi di nomi di funzioni utilizzabili per le sopra citate strategie di esplorazione sono:

- getExplorationGoals_RANDOM()
- getExplorationGoals_WEIGHTED()
- getExplorationGoals_MCDM()

Queste nuove funzioni dovranno essere aggiunte nel file explore_frontier.cpp (valutare se aggiungere i riferimenti anche nel file explore_frontier.h e se aggiungere qualche altro riferimento nei file explore.h ed explore.cpp)

2.2. Diverse combinazioni delle modalità (continua e discreta) di percezione e decisione

Parametro per scegliere le diverse combinazioni delle modalità di esplorazione:

Valutare se aggiungere un nuovo parametro o se sfruttare i seguenti parametri (già esistenti):

- explore_stage/explore/explore_costmap.yaml: *update_frequency*
  frequenza con la quale il robot aggiorna la mappa dell'ambiente
- explore_stage/explore_slam.xml: *planner_frequency* (da aggiungere al file xml)
  frequenza con la quale il robot calcola il nuovo goal da raggiungere

Nel primo caso, è possibile utilizzare un unico nuovo parametro che ci permette di scegliere tra le diverse combinazioni (a, b, d):

decision_perception_discretization: [0, 1, 2].

Dove:

- 0 equivale a percezione continua, decisione continua (caso a.)
- 1 equivale a percezione continua, decisione discreta (caso b.)
- 2 equivale a percezione discreta, decisione discreta (caso d.)

In questo primo caso bisognerà definire il nuovo parametro nel file explore_slam.xml e poi riutilizzarlo nel file explore.cpp.

Bisognerà quindi modificare gli attuali metodi che pianificano l'esplorazione e che costruiscono la mappa, in modo tale che i comportino coerentemente con il parametro appena visto.

Lo stesso vale per il secondo caso, qui sotto, con l'unica differenza che i parametri da tenere in considerazione saranno due, separati fra loro, e che questi parametri sono già definiti.

Nel secondo caso invece sarà necessario assegnare il valore "-1" ai due parametri per ottenere le diverse combinazioni, nei seguenti modi:

- *planner_frequency* != -1 e *update_frequency*!= -1 (caso a.)
- *planner_frequency* != -1 e *update_frequency* = -1 (caso b.)
- *planner_frequency* =-1 e *update_frequency* = -1 (caso d.)

E successivamente fare in modo che l'applicazione si comporti nel modo desiderato, collegando alla scelta di tali parametri le scelte nel codice dell'applicazione.

Controllare che il goal sia stato raggiunto, prima di sceglierne uno nuovo:

- nuova variabile globale: *chosenGoal_pose*

mantiene salvato il goal scelto, per poter calcolare la distanza tra di esso e il robot.

- nuova funzione: *chosenGoal_reached ()*

calcola la distanza tra il robot e il goal scelto; se essa è inferiore a una determinata soglia, allora considera il nodo come raggiunto e consente di sceglierne uno nuovo.

Questi elementi vanno aggiunti nel file explore.cpp e nel file explore.h.

2.3. Nuovo nodo in ascolto

Valutare se creare un nuovo nodo per raccogliere e visualizzare i dati relativi alle performance di esplorazione, o se invece effettuare le modifiche direttamente all'interno del nodo explore.

3. Modifiche apportate al codice

Le modifiche sono segnalate in rosso nei blocchi di codice.

File bosch-ros-pkg/trunk/stacks/exploration/explore_stage/explore.xml

```
File (riporto qui le stesse modifiche fatte per explore_slam.xml)
```

File bosch-ros-pkg/trunk/stacks/exploration/explore_stage/explore_slam.xml

```
file
```

File: bosh-ros-pkg/trunk/stacks/exploration/explore/include/explore/explore.h

```
file
```

File: bosh-ros-pkg/trunk/stacks/exploration/explore/src/explore.cpp

```
file
```

File: -ros-pkg/trunk/stacks/exploration/explore/include/explore/explore_frontier.h

```
file
```

File: bosh-ros-pkg/trunk/stacks/exploration/explore/src/explore_frontier.cpp

```
file
```

File: bosh-ros-pkg/trunk/stacks/exploration/explore_stage/explore.launch

```
<node pkg="stage" type="stageros" name="stage" args="-g $(find bosch_worlds)/maze-noisy.world"
respawn="false" output="screen"/>
```

File: bosh-ros-pkg/trunk/stacks/exploration/explore_stage/explore_slam.launch

```
 <node pkg="stage" type="stageros" name="stage" args="-g $(find bosch_worlds)/maze-noisy.world"
respawn="false" output="screen"/>
```

File: explore_test/config/explore.launch

```
file
```

File: explore_test/config/explore_slam.launch

```
file
```

3.1. Aggiunto il parametro “planner_frequency”

Aggiunto il parametro “planner_frequency” nel file explore_slam.xml per permettere a chi lo volesse di modificare a piacimento la frequenza di pianificazione del nuovo goal verso il quale dirigersi.

Aggiunta la riga seguente nel file: explore_slam.xml

```
<param name="planner_frequency" value="1.0"/>
```

3.2. Aggiunto il parametro “decision_perception_discretization”

Aggiunto il parametro “decision_perception_discretization” nel file explore_slam.xml per evitare “effetti collaterali” in altre parti dell’applicazione in seguito alla eventuale modifica dei parametri “planner_frequency” e “update_frequency”, e per mantenere un unico parametro dedicato alla scelta della modalità di esplorazione e solo ad essa, sicuramente la soluzione più pulita.

Al parametro in questione è stato assegnato il valore “0”, che prevede decisione e percezione continue.

Aggiunta la riga seguente nel file: explore_slam.xml

```
<param name=”decision_perception_discretization” value=”0”/>
```

Per poter utilizzare questo parametro è necessario aggiungerlo anche nel codice sorgente, in particolare nei file explore.h ed explore.cpp.

Aggiunta la riga seguente nel file: explore.h

```
int decision_perception_discretization_;
```

Aggiunta la riga seguente nel file: explore.cpp

```
private_nh.param("decision_perception_discretization", decision_perception_discretization_, 0);
```

3.3. Aggiunta la variabile “chosenGoal_pose”

Manteniamo memorizzato in questa variabile il goal che stiamo cercando di raggiungere.

La variabile va aggiunta sia nel file explore.cpp che nel file explore.h

Aggiunta la riga seguente nel file: explore.h

```
geometry_msgs::PoseStamped chosenGoal_pose_;
```

Aggiunta la parte in rosso, all'interno del metodo "makePlan()", nel file: explore.cpp

```
......
......

bool valid plan = false;
std::vector<geometry msgs::PoseStamped> plan;
PoseStamped goal_pose, robot_pose_msg;
tf::poseStampedTFToMsg(robot_pose, robot_pose_msg);

goal pose.header.frame id = explore costmap ros ->getGlobalFrameID();
goal pose.header.stamp = ros::Time::now();
planner ->computePotential(robot pose msg.pose.position); // just to be safe, though this should
already have been done in explorer_->getExplorationGoals

//:::dt:::
ROS_INFO(":::dt::: goals.size()= %d", goals.size());

int blacklist_count = 0;
for (unsigned int i=0; i<goals.size(); i++) {
  goal_pose.pose = goals[i];

  if (goalOnBlacklist(goal pose)) {
    blacklist count++;
    continue;
  }

  valid plan = ((planner ->getPlanFromPotential(goal pose, plan)) && (!plan.empty()));
  if (valid plan) {

    //:::dt:::
    chosenGoal_pose_.header.frame_id = goal_pose.header.frame_id;
    chosenGoal_pose_.header.stamp = goal_pose.header.stamp;
    chosenGoal_pose_.pose = goal_pose.pose;
    ROS_INFO("chosenGoal_pose_ : x=%f -
y=%f",(float)chosenGoal_pose_.pose.position.x,(float)chosenGoal_pose_.pose.position.y);

    break;
  }
}

//:::dt:::
ROS_INFO(":::dt::: blacklist_count= %d", blacklist_count);

......
......
```

3.4. Aggiunto il parametro "chosenGoalReached_tolerance"

Aggiunto il parametro "*chosenGoalReached_tolerance*" nel file explore_slam.xml . Esso indica il limite di distanza entro il quale possiamo considerare un goal come raggiunto, e quindi possiamo procedere ad effettuare una nuova percezione (nel caso di prcezione discreta).

Al parametro in questione è stato assegnato il valore "0.5".

Aggiunta la riga seguente nel file: explore_slam.xml

```
<param name="chosenGoalReached_tolerance" value="0.5"/>
```

Per poter utilizzare questo parametro è necessario aggiungerlo anche nel codice sorgente, in particolare nei file explore.h ed explore.cpp.

Aggiunta la riga seguente nel file: explore.h

```
double chosenGoalReached_tolerance_;
```

Aggiunta la riga seguente nel file: explore.cpp

```
private_nh.param("chosenGoalReached_tolerance ", chosenGoalReached_tolerance _, 1.0);
```

3.5. Aggiunto il metodo "chosenGoal_reached( )"

In questo metodo si controlla se il goal scelto in precedenza sia stato raggiunto, o se comunque siamo ad una distanza minima da esso: si calcola la distanza tra il robot e il goal scelto, se essa è inferiore a una determinata soglia, allora si considera il goal come raggiunto e si consente di sceglierne uno nuovo.

L'importante è scegliere accuratamente la soglia della distanza entro la quale consideriamo raggiunto il goal. (tale soglia è identificata dal parametro "chosenGoalReached_tolerance")

Aggiunto il metodo "chosenGoal_reached()" nel file explore.cpp

```
//:::dt::: checks if the chosen goal has been reached
bool Explore::chosenGoal_reached(){

  tf::Stamped<tf::Pose> robot_pose_local;
  explore_costmap_ros_->getRobotPose(robot_pose_local);
  PoseStamped robot_pose_local_msg;
  tf::poseStampedTFToMsg(robot_pose_local, robot_pose_local_msg);
  float map_resolution = explore_costmap_ros_->getResolution();

  ROS_INFO("robot_pose_local_msg : x=%f -
y=%f",(float)robot_pose_local_msg.pose.position.x,(float)robot_pose_local_msg.pose.position.y);
  ROS_INFO("chosenGoal_pose : x=%f -
y=%f",(float)chosenGoal_pose_.pose.position.x,(float)chosenGoal_pose_.pose.position.y);
  ROS_INFO("map_resolution : %f",(float)map_resolution);
  ROS_INFO("chosenGoalReached_tolerance_ : %f",(float)chosenGoalReached_tolerance_);

  float distance_x = (float)robot_pose_local_msg.pose.position.x -
(float)chosenGoal_pose_.pose.position.x;
  float distance_y = (float)robot_pose_local_msg.pose.position.y -
(float)chosenGoal_pose_.pose.position.y;
  float distance_from_goal = sqrt((distance_x * distance_x) + (distance_y * distance_y));
  ROS_INFO("distance_from_goal : %f",distance_from_goal);

  if(distance_from_goal<chosenGoalReached_tolerance_){
    return true;
  }

  return false;
}
```

Aggiunto il metodo "chosenGoal_reached()", nel file explore.h

```
private:

  /**
   * @brief  :::dt::: decides whether to choose a new goal or not
   */
```

```
  bool performDecision_startStopPerception();

  /**
   * @brief  :::dt::: checks if the chosen goal has been reached
   */
  bool chosenGoal_reached();
```

3.6. Aggiunto il metodo "printStrategyAndPerception ()

Aggiunto il nuovo metodo

Mostriamo a video quali modalità di esplorazione siano state selezionate, in base al parametro "decision_perception_discretization" e al parametro "exploration_strategy".

Aggiunto il metodo "printStrategyAndPerception()", nel file explore.h

```
private:

  /**
   * @brief  :::dt::: puts in output strategy and perception discretization
   */
  void printStrategyAndPerception();

  /**
   * @brief  :::dt::: decides whether to choose a new goal or not
   */
  bool performDecision_startStopPerception();

… … … … … …
… … … … … …
```

Aggiunto il metodo "printStrategyAndPerception()", nel file explore.cpp

```
//:::dt:::  puts in output strategy and perception discretization
void Explore:: printStrategyAndPerception () {
  if(exploration strategy ==0){ //RANDOM
        ROS_INFO(":::dt::: exploration_strategy_ = %d (RANDOM)", exploration_strategy_);
  }
  else if(exploration_strategy_==1){ //WEIGHTED
        ROS_INFO(":::dt::: exploration_strategy_ = %d (WEIGHTED)", exploration_strategy_);
  }
  else if(exploration strategy ==2){ //MCDM
        ROS_INFO(":::dt::: exploration_strategy_ = %d (MCDM)", exploration_strategy_);
  }

  if(decision perception discretization ==0){ //continuous perception - continuous decision
        ROS INFO(":::dt::: decision perception discretization  = %d (continuous perception -
continuous decision)", decision_perception_discretization_);
  }
  else if(decision perception discretization ==1){ //continuous perception - discrete decision
        ROS INFO(":::dt::: decision perception discretization  = %d (continuous perception -
discrete decision)", decision perception discretization );
  }
  else if(decision_perception_discretization_==2){ //discrete perception - discrete decision
        ROS_INFO(":::dt::: decision_perception_discretization_ = %d (discrete perception - discrete
decision)", decision perception discretization );
  }
}
```

Modificato il metodo execute

Questo metodo è quello che gestisce la frequenza di pianificazione e ri-pianificazione del prossimo goal da raggiungere.

Interveniamo qui solo per mostrare quanto è previsto dal metodo printStrategyAndPerception().

Aggiunta la parte in rosso nel metodo "execute()", nel file explore.cpp

```
void Explore::execute() {
  while (! move_base_client_.waitForServer(ros::Duration(5,0)))
    ROS_WARN("Waiting to connect to move_base server");

  ROS_INFO("Connected to move base server");

  // This call sends the first goal, and sets up for future callbacks.
  makePlan();

  ros::Rate r(planner_frequency_);

  //:::dt:::
  printStrategyAndPerception();

  while (node_.ok() && (!done_exploring_)) {

    if (close_loops_) {
      tf::Stamped<tf::Pose> robot_pose;
      explore_costmap_ros_->getRobotPose(robot_pose);
      loop_closure_->updateGraph(robot_pose);
    }

    makePlan();
    r.sleep();
  }

  move_base_client_.cancelAllGoals();
}
```

3.7. Aggiunto il metodo "performDecision_startStopPerception ( )"

Questo metodo consente di decidere se effettuare una nuova percezione e se ri-calcolare il goal da raggiungere.

Dobbiamo quindi intervenire qui: se il parametro "decision_perception_discretization" è stato impostato ad un valore diverso da 0, dobbiamo gestire la discretizzazione della decisione e/o della percezione, bloccandole finché il goal non sia stato raggiunto.

Aggiunto il metodo "performDecision_startStopPerception()", nel file explore.cpp

```
//:::dt::: decides whether to choose a new goal or not
bool Explore::performDecision_startStopPerception(){
    bool performDecision = false;
    if(decision_perception_discretization_ == 0){
      //continuous perception \u2013 continuous decision
      performDecision = true;
    }
    else if(decision_perception_discretization_ == 1 || decision_perception_discretization_ == 2){
      //continuous perception \u2013 discrete decision

      if(chosenGoal_reached()){
        if(decision_perception_discretization_ == 2){
          //discrete perception \u2013 discrete decision
            ROS_INFO(":::dt::: goal reached - starting perception");
          explore_costmap_ros_->start();
          //far girare il robot di 360 gradi????
        }
          performDecision = true;
      }
      else{
        if(decision_perception_discretization_ == 2){
          //discrete perception \u2013 discrete decision
            ROS_INFO(":::dt::: goal not reached - stopping perception");
          explore_costmap_ros_->pause();
```

```
          //explore_costmap_ros_->stop(); ?????
        }
          performDecision = false;
      }
    }
    return performDecision;
}
```

Aggiunto il metodo "performDecision_startStopPerception()", nel file explore.h

```
private:

  /**
   * @brief  :::dt::: decides whether to choose a new goal or not
   */
  bool performDecision_startStopPerception();
```

In questo nuovo metodo abbiamo fatto ricorso ai seguenti metodi del nodo costmap_2d_ros:

```
/**
 * @brief  Subscribes to sensor topics if necessary and starts costmap
 * updates, can be called to restart the costmap after calls to either
 * stop() or pause()
 */
void start();

/**
 * @brief  Stops costmap updates and unsubscribes from sensor topics
 */
void stop();


/**
 * @brief  Stops the costmap from updating, but sensor data still comes in over the wire
 */
void pause();
```

3.8. Aggiunta la variabile "savedGoals_"

Manteniamo memorizzato in questa variabile l'elenco ordinato dei goal che stiamo cercando di raggiungere. Lo utilizzeremo poi nel metodo makeplan.

La variabile va aggiunta nel file explore.h

Aggiunta la riga seguente nel file: explore.h

```
std::vector<geometry_msgs::Pose> savedGoals_;
```

3.9. Modificato il metodo "makeplan( )"

Aggiunta la gestione di percezione e decisione attraverso il metodo performDecision_startStopPerception() e sfruttando la variabile savedGoals, che ci permette di ottenere i goal salvati nella precedente chiamata a getExplorationGoals nel caso in cui non sia consentito effettuare una nuova scelta dei goal.

Aggiunta la parte in rosso nel metodo "makeplan()", nel file explore.cpp

```
void Explore::makePlan() {
  //since this gets called on handle activate
```

```
  if(explore_costmap_ros_ == NULL)
    return;

  tf::Stamped<tf::Pose> robot pose;
  explore_costmap_ros_->getRobotPose(robot_pose);

  std::vector<geometry msgs::Pose> goals;
  explore costmap ros ->clearRobotFootprint();

  //:::dt:::
  bool performDecision = performDecision_startStopPerception();
  if(performDecision==true){
    explorer ->getExplorationGoals(*explore costmap ros , robot pose, planner , goals,
potential scale , orientation scale , gain scale , exploration strategy );
    savedGoals  = goals;
  }
  else{
    goals = savedGoals_;
  }

  if (goals.size() == 0)
    done_exploring_ = true;

  bool valid_plan = false;
  std::vector<geometry msgs::PoseStamped> plan;
  PoseStamped goal pose, robot pose msg;
  tf::poseStampedTFToMsg(robot pose, robot pose msg);

 ......
 ......
```

3.10. Aggiunto il parametro “exploration_strategy”

Aggiunto il parametro “exploration_strategy” nel file explore_slam.xml per poter scegliere una tra le strategie di esplorazione disponibili.

Al parametro in questione è stato assegnato il valore “1”, che prevede la scelta della strategia WEIGHTED.

Aggiunta la riga seguente nel file: explore_slam.xml

```
<param name=”exploration_strategy” value=”1”/>
```

Per poter utilizzare questo parametro è necessario aggiungerlo anche nel codice sorgente, in particolare nei file explore.h ed explore.cpp.

Aggiunta la riga seguente nel file: explore.h

```
int exploration_strategy_;
```

Aggiunta la riga seguente nel file: explore.cpp

```
private_nh.param("exploration_strategy", exploration_strategy_, 0);
```

3.11. Aggiunto un nuovo metodo per ogni nuova strategia di esplorazione

Sono stati aggiunti i seguenti metodi all’interno del file explore_frontier.cpp:

- getRandomCost()
- getWeightedCost()
- getMcdmCost()

Il metodo getWeightedCost(), oltre a calcolare il costo "pesato" per ogni frontiera, viene temporaneamente utilizzato anche per loggare i costi rilevati (nella parte compresa tra i commenti *"dt - only for logging utility/cost values*" e "*dt – end*").

Una volta rilevati, sono stati utilizzati per stabilire i pesi da dare ai diversi costi, sia per questo metodo, sia per il metodo che prevede l'utilizzo di MCDM.

Aggiunti i seguenti metodi all'interno del file explore_frontier.h

```
private:

  /**
   * @brief  :::dt::: returns the random cost for the current frontier
   */
  virtual int getRandomCost();

  /**
   * @brief  :::dt::: returns the weighted cost for the current frontier
   */
  virtual float getWeightedCost(WeightedFrontier& weightedFrontier, tf::Stamped<tf::Pose>
robot pose, double potential scale, double orientation scale, double gain scale);

  /**
   * @brief Calculates MCDM frontier cost
   */
  virtual float getMcdmCost(WeightedFrontier weightedFrontier, tf::Stamped<tf::Pose> robot pose,
double potential scale, double orientation scale, double gain scale , double
potentialOrientation_scale, double potentialGain_scale, double orientationGain_scale, double
potentialOrientationGain_scale);
```

Metodo getRandomCost( )

Questo metodo consente di scegliere in modo casuale il prossimo goal da raggiungere.

Aggiunto il seguente metodo all'interno del file explore_frontier.cpp

```
int ExploreFrontier::getRandomCost(WeightedFrontier weightedFrontier)
{
  // srand(static cast<int>(time(NULL)));
  // int magic = rand() % 100 + 1;
  int cost = rand();

  //dt - only for logging utility/cost values
  if(logDebugInfo  == 1){
    FILE *stream;
    int  numwritten;
    char s cost[30];

    float pos_x = weightedFrontier.frontier.pose.position.x;
    float pos y = weightedFrontier.frontier.pose.position.y;
    char s pos x[30];
    char s pos y[30];
    sprintf(s_pos_x, "%f", pos_x);
    sprintf(s_pos_y, "%f", pos_y);

    std::string goal pose string = "";
    goal pose string = std::string("(").append(s pos x).append(std::string(" ,
")).append(s pos y).append(std::string(")"));

    sprintf(s_cost, "%d", cost);
```

```
    if( (stream = fopen(debugInfo_filename_.c_str(), "a" )) != NULL )
    {
      numwritten = fputs( goal pose string.c str(), stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s_cost, stream );
      numwritten = fputs( "\n", stream );

      fclose( stream );
    }
    else
      printf( "Problem opening the file\n" );
  }

  return cost;
```

Metodo getWeightedCost( )

Questo metodo implementa la strategia di esplorazione “pesata”.

Aggiunto il seguente metodo all’interno del file explore_frontier.cpp

```
float ExploreFrontier::getWeightedCost(WeightedFrontier weightedFrontier, tf::Stamped<tf::Pose>
robot pose, double potential scale, double orientation scale, double gain scale){

  float frontierCost = getFrontierCost(weightedFrontier.frontier);
  double orientationChange = fabs(getOrientationChange(weightedFrontier.frontier, robot_pose));
  float frontierGain = getFrontierGain(weightedFrontier.frontier, costmapResolution_);

  float cost = potential scale * frontierCost + orientation scale * orientationChange - gain scale *
frontierGain;
//      weightedFrontier.cost = getFrontierCost(weightedFrontier.frontier) -
getFrontierGain(weightedFrontier.frontier, costmapResolution );
//      ROS DEBUG("cost: %f (%f * %f + %f * %f - %f * %f)",
//          weightedFrontier.cost,
//          potential scale,
//          getFrontierCost(weightedFrontier.frontier),
//          orientation_scale,
//          getOrientationChange(weightedFrontier.frontier, robot pose),
//          gain scale,
//          getFrontierGain(weightedFrontier.frontier, costmapResolution ) );


  //dt - only for logging utility/cost values
  if(logDebugInfo  == 1){
    FILE *stream;
    int  numwritten;
    char s frontierCost[30];
    char s_orientationChange[30];
    char s_frontierGain[30];
    char s cost[30];

    float pos x = weightedFrontier.frontier.pose.position.x;
    float pos_y = weightedFrontier.frontier.pose.position.y;
    char s_pos_x[30];
    char s pos y[30];
    sprintf(s pos x, "%f", pos x);
    sprintf(s pos y, "%f", pos y);

    std::string goal_pose_string = "";
    goal_pose_string = std::string("(").append(s_pos_x).append(std::string(" ,
")).append(s pos y).append(std::string(")"));

    sprintf(s frontierCost, "%f", frontierCost);
    sprintf(s_orientationChange, "%f", orientationChange);
    sprintf(s_frontierGain, "%f", frontierGain);
    sprintf(s_cost, "%f", cost);

    if( (stream = fopen(debugInfo filename .c str(), "a" )) != NULL )
    {
      numwritten = fputs( goal_pose_string.c_str(), stream );
      numwritten = fputs( " ", stream );
```

```
      numwritten = fputs( s_frontierCost, stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s orientationChange, stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s_frontierGain, stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s cost, stream );
      numwritten = fputs( "\n", stream );

      fclose( stream );
    }
    else
      printf( "Problem opening the file\n" );
  }

  return cost;
}
```

Metodo getMcdmCost( )

Questo metodo implementa la strategia di esplorazione “MCDM”.

Per poter utilizzare il metodo getMcdmCost(), dobbiamo modificare anche il metodo getExplorationGoals (come vedremo in seguito) e aggiungere alcune variabili globali, per poter sfruttare i pesi associati alle diverse possibili combinazioni di criteri.

Aggiunte le seguenti variabili globali all’interno del file explore.h

```
… … …
… … …

  //:::dt:::
  int decision_perception_discretization_;
  int exploration_strategy_;
  geometry_msgs::PoseStamped chosenGoal_pose_;
  std::vector<geometry msgs::Pose> savedGoals ;
  double chosenGoalReached tolerance ;
  double potentialOrientation scale ;
  double potentialGain_scale_;
  double orientationGain_scale_;
  double potentialOrientationGain scale ;

… … …
… … …
```

Aggiunte le seguenti variabili globali all’interno del file explore.cpp

```
… … …
… … …

  private nh.param("potential scale", potential scale , 0.3);
  private nh.param("orientation scale", orientation scale , 0.1);
  private nh.param("gain scale", gain scale , 0.5);

  //:::dt:::
  private nh.param("decision perception discretization", decision perception discretization , 0);
  private nh.param("exploration strategy", exploration strategy , 0);
  private nh.param("chosenGoalReached tolerance", chosenGoalReached tolerance , 0.5);
  private nh.param("potentialOrientation scale", potentialOrientation scale , 0.4);
  private_nh.param("potentialGain_scale", potentialGain_scale_, 0.9);
  private_nh.param("orientationGain_scale", orientationGain_scale_, 0.7);
  private nh.param("potentialOrientationGain scale", potentialOrientationGain scale , 1.0);
  exploring status publisher  = private nh.advertise<std msgs::Bool>("exploring status", 10);
… … …
… … …
```

Aggiunte le seguenti variabili (in rosso) globali all'interno del file explore_slam.xml (e anche al file explore.xml)

```
<launch>
  <node pkg="explore" type="explore" respawn="false" name="explore" output="screen" >
    <rosparam file="$(find explore stage)/config/footprint.yaml" command="load" />

    <rosparam file="$(find explore stage)/config/costmap common.yaml" command="load"
ns="explore_costmap" />
    <rosparam file="$(find explore_stage)/explore/explore_costmap.yaml" command="load" />

    <param name="potential scale" value="0.3"/>
    <param name="orientation scale" value="0.1"/>
    <param name="gain_scale" value="0.5"/>
    <param name="close_loops" value="true"/>
    <remap from="slam_entropy" to="gmapping/entropy"/>

    <param name="planner frequency" value="1.0"/>
    <param name="decision perception discretization" value="0"/>
    <param name="exploration_strategy" value="1"/>
    <param name="chosenGoalReached_tolerance" value="1.0"/>

    <param name="potentialOrientation_scale" value="0.4"/>
    <param name="potentialGain_scale" value="0.9"/>
    <param name="orientationGain_scale" value="0.7"/>
    <param name="potentialOrientationGain_scale" value="1.0"/>
  </node>
</launch>
```

Per poter implementare correttamente il metodo getMcdmCost(), è necessario definire le variabili globali:

- utilities_ (array di elementi di tipo Utility)
- weights_ (array di elementi di tipo Weight)

Definiamo quindi i tipi Utility e Weight nel file explore_frontier.h:

```
enum MinimizeOrMaximize { minimize, maximize };
enum Criterion { potential, orientation, gain };

struct Utility {
  float normalizedValue;
  float minValue;
  float maxValue;
  Criterion relatedCriterion;
  MinimizeOrMaximize minimizeOrMaximize;
  // "operator <" is used when we call the std::sort or std::stable_sort(myvector.begin(),
myvector.end()) methods.
  bool operator<(const Utility& o) const { return normalizedValue < o.normalizedValue; }
};

struct Weight {
  double weight;
  std::vector<Criterion> relatedCriteria;
};
```

Definiamo le variabili globali all'interno del file explore_frontier.h

```
public:

… … …
… … …

  //:::dt:::
```

```
  std::vector<Utility> utilities_;
  std::vector<Weight> weights_;
};

}

#endif /* EXPLORE_FRONTIER_H_ */
```

Aggiungiamo i seguenti 2 metodi:

- setWeights() per salvare nella variabile globale weights i pesi dei diversi criteri
- setMinMaxValues() per ottenere i valori minimi e massimi di utilità/costo per ogni criterio.

La funzione setWeights() dovrà essere eseguita una sola volta, mentre la funzione setMinMaxWeights() dovrà essere richiamata ogni volta che otteniamo un nuovo elenco di frontiere (ad ogni percezione).

Aggiunti i seguenti metodi al file explore_frontier.h

```
… … …
… … …

protected:
  std::vector<Frontier> frontiers_;

  … … …
  … … …

  /**
   * @brief Calculates MCDM frontier cost
   */
  virtual float getMcdmCost(WeightedFrontier weightedFrontier, tf::Stamped<tf::Pose> robot pose,
double potential scale, double orientation scale, double gain scale, double
potentialOrientation scale, double potentialGain scale, double orientationGain scale, double
potentialOrientationGain_scale);

  /**
   * @brief Sets the weights vector with the values obtained from the configuration file
   */
  virtual bool setWeights(double potential scale, double orientation scale, double gain scale,
double potentialOrientation_scale, double potentialGain_scale, double orientationGain_scale, double
potentialOrientationGain_scale);

  /**
   * @brief Calculates min and max utility/cost values for every criteria, for every perception
   */
  virtual bool setMinMaxValues(costmap_2d::Costmap2DROS& costmap, WeightedFrontier weightedFrontier,
tf::Stamped<tf::Pose> robot_pose, double potential_scale, double orientation_scale, double
gain scale);

… … …
… … …
```

Aggiunto il metodo setWeights() all'interno del file explore_frontier.cpp

```
bool ExploreFrontier::setWeights(double potential_scale, double orientation_scale, double
gain_scale, double potentialOrientation_scale, double potentialGain_scale, double
orientationGain scale, double potentialOrientationGain scale){
  if (weights .size() > 0)
    return false;

  Weight potential_weight;
  potential_weight.weight = potential_scale;
  potential weight.relatedCriteria.push back(Criterion(potential));

  Weight orientation_weight;
  orientation_weight.weight = orientation_scale;
```

```
  orientation_weight.relatedCriteria.push_back(Criterion(orientation));

  Weight gain weight;
  gain weight.weight = gain scale;
  gain_weight.relatedCriteria.push_back(Criterion(gain));

  Weight potentialOrientation weight;
  potentialOrientation weight.weight = potentialOrientation scale;
  potentialOrientation weight.relatedCriteria.push back(Criterion(potential));
  potentialOrientation weight.relatedCriteria.push back(Criterion(orientation));

  Weight potentialGain_weight;
  potentialGain weight.weight = potentialGain scale;
  potentialGain weight.relatedCriteria.push back(Criterion(potential));
  potentialGain weight.relatedCriteria.push back(Criterion(gain));

  Weight orientationGain_weight;
  orientationGain_weight.weight = orientationGain_scale;
  orientationGain weight.relatedCriteria.push back(Criterion(orientation));
  orientationGain weight.relatedCriteria.push back(Criterion(gain));

  Weight potentialOrientationGain_weight;
  potentialOrientationGain_weight.weight = potentialOrientationGain_scale;
  potentialOrientationGain_weight.relatedCriteria.push_back(Criterion(potential));
  potentialOrientationGain weight.relatedCriteria.push back(Criterion(orientation));
  potentialOrientationGain weight.relatedCriteria.push back(Criterion(gain));

  weights_.push_back(potential_weight);
  weights_.push_back(orientation_weight);
  weights .push back(gain weight);
  weights .push back(potentialOrientation weight);
  weights .push back(potentialGain weight);
  weights_.push_back(orientationGain_weight);
  weights_.push_back(potentialOrientationGain_weight);

  return true;
}
```

Aggiunto il metodo setMinMaxValues () all'interno del file explore_frontier.cpp

```
bool ExploreFrontier::setMinMaxValues(Costmap2DROS& costmap, WeightedFrontier weightedFrontier,
tf::Stamped<tf::Pose> robot pose, double potential scale, double orientation scale, double
gain_scale){
  if (utilities_.size() > 0)
    return false;

  Utility potential utility;
  potential utility.minValue = -1;
  potential_utility.maxValue = -1;
  potential_utility.relatedCriterion = Criterion(potential);
  potential utility.minimizeOrMaximize = MinimizeOrMaximize(minimize);

  Utility orientation utility;
  orientation_utility.minValue = -1;
  orientation_utility.maxValue = -1;
  orientation utility.relatedCriterion = Criterion(orientation);
  orientation utility.minimizeOrMaximize = MinimizeOrMaximize(minimize);

  Utility gain_utility;
  gain_utility.minValue = -1;
  gain_utility.maxValue = -1;
  gain utility.relatedCriterion = Criterion(gain);
  gain utility.minimizeOrMaximize = MinimizeOrMaximize(maximize);

  costmapResolution_ = costmap.getResolution();

  //we'll make sure that we set goals for the frontier at least the circumscribed
  //radius away from unknown space
  float step = -1.0 * costmapResolution ;
  int c = ceil(costmap.getCircumscribedRadius() / costmapResolution );
  WeightedFrontier goal;
  std::vector<WeightedFrontier> weightedFrontiers;
  weightedFrontiers.reserve(frontiers_.size() * c);
```

```
  for (uint i=0; i < frontiers_.size(); i++) {
    Frontier& frontier = frontiers [i];
    WeightedFrontier weightedFrontier;
    weightedFrontier.frontier = frontier;

    tf::Point p(frontier.pose.position.x, frontier.pose.position.y, frontier.pose.position.z);
    tf::Quaternion bt;
    tf::quaternionMsgToTF(frontier.pose.orientation, bt);
    tf::Vector3 v(cos(bt.getAngle()), sin(bt.getAngle()), 0.0);

    for (int j=0; j <= c; j++) {
      tf::Vector3 check_point = p + (v * (step * j));
      weightedFrontier.frontier.pose.position.x = check point.x();
      weightedFrontier.frontier.pose.position.y = check point.y();
      weightedFrontier.frontier.pose.position.z = check point.z();

      //:::dt::: here we calculate the cost/utility values for each criteria
      // and then we save the minimum and the maximum values for each criteria
      float frontierCost = getFrontierCost(weightedFrontier.frontier);
      double orientationChange = fabs(getOrientationChange(weightedFrontier.frontier, robot pose));
      float frontierGain = getFrontierGain(weightedFrontier.frontier, costmapResolution );

      if(potential_utility.minValue == -1 || frontierCost < potential_utility.minValue){
        potential_utility.minValue = frontierCost;
      }
      if(potential utility.maxValue == -1 || frontierCost > potential utility.maxValue){
        potential utility.maxValue = frontierCost;
      }

      if(orientation utility.minValue == -1 || orientationChange < orientation utility.minValue){
        orientation utility.minValue = orientationChange;
      }
      if(orientation_utility.maxValue == -1 || orientationChange > orientation_utility.maxValue){
        orientation_utility.maxValue = orientationChange;
      }

      if(gain utility.minValue == -1 || frontierGain < gain utility.minValue){
        gain utility.minValue = frontierGain;
      }
      if(gain_utility.maxValue == -1 || frontierGain > gain_utility.maxValue){
        gain_utility.maxValue = frontierGain;
      }
    }
  }

  utilities_.push_back(potential_utility);
  utilities_.push_back(orientation_utility);
  utilities .push back(gain utility);

  return true;
}
```

Aggiunto il metodo getMcdmCost() all'interno del file explore_frontier.cpp

```
float ExploreFrontier::getMcdmCost(Costmap2DROS& costmap, WeightedFrontier weightedFrontier,
tf::Stamped<tf::Pose> robot pose, double potential scale, double orientation scale, double
gain scale, double potentialOrientation scale, double potentialGain scale, double
orientationGain scale, double potentialOrientationGain scale){
  float goalCost = 0;
  float goalUtility = 0;
  float prevGoalUtility = 0;

  //if not already done, saves the weights into the global variable "weights "
  setWeights(potential scale, orientation scale, gain scale, potentialOrientation scale,
potentialGain_scale, orientationGain_scale, potentialOrientationGain_scale);

  //if not already done, saves the max and min utility values for the current set of goals into the
global variable "utilities "
  setMinMaxValues(costmap, weightedFrontier, robot pose, potential scale, orientation scale,
gain scale);

  //:::dt::: here we calculate the cost/utility values for each criteria, for the current
goal/frontier
  float frontierCost = getFrontierCost(weightedFrontier.frontier);
```

```
  double orientationChange = fabs(getOrientationChange(weightedFrontier.frontier, robot_pose));
  float frontierGain = getFrontierGain(weightedFrontier.frontier, costmapResolution );

  //computes the normalized utilities for the current goal, for each criteria
  for(uint k=0; k<utilities_.size(); k++){
    float gap = utilities_[k].maxValue - utilities_[k].minValue;
    if(gap == 0){
      gap = 0.000001;
    }

    float minValue = utilities_[k].minValue;
    float value = 0;
    if(utilities [k].relatedCriterion == Criterion(potential))
      value = frontierCost;
    else if(utilities [k].relatedCriterion == Criterion(orientation))
      value = orientationChange;
    else if(utilities_[k].relatedCriterion == Criterion(gain))
      value = frontierGain;

    float normalizedValue = (value - minValue)/gap;

    if(utilities_[k].minimizeOrMaximize == MinimizeOrMaximize(minimize)){
      utilities_[k].normalizedValue = 1 - normalizedValue;
    }
    else{
       utilities [k].normalizedValue = normalizedValue;
    }
  }

  //utilities are ordered by normalizedValue
  std::stable sort(utilities .begin(), utilities .end());

  //computes the global utility for the current goal/frontier
  std::vector<Criterion> availableCriteria;
  availableCriteria.push_back(Criterion(potential));
  availableCriteria.push back(Criterion(orientation));
  availableCriteria.push back(Criterion(gain));
  uint numberOfUtilities = utilities .size(); //must be equal to availableCriteria.size();
  for(uint k=0; k<numberOfUtilities; k++){
    float higherValue = 0;
    float lowerValue = 0;
    double currentCriteriaWeight = 0;

    if(k>0){
      lowerValue = utilities_[k-1].normalizedValue;
    }
    higherValue = utilities_[k].normalizedValue;

    //searches for the weight of the current criteria set
    for(uint z=0; z<weights_.size(); z++){
      //ROS_INFO(":::dt:::: weights_[%d]",z);
      bool allCriteriaFound = false;
      uint availableCriteriaNumber = availableCriteria.size();
      uint weightRelatedCriteriaNumber = weights [z].relatedCriteria.size();
      //ROS_INFO(":::dt:::: availableCriteriaNumber = %d - weightRelatedCriteriaNumber =
%d",availableCriteriaNumber,weightRelatedCriteriaNumber);
      if(availableCriteriaNumber == weightRelatedCriteriaNumber){
        //ROS INFO(":::dt:::: availableCriteriaNumber = weightRelatedCriteriaNumber =
%d",availableCriteriaNumber);
        for(uint availableCriteria index=0; availableCriteria index<availableCriteriaNumber;
availableCriteria_index++){
          bool currentCriteriaFound = false;
          for(uint weightCriteria index=0; weightCriteria index<weightRelatedCriteriaNumber &&
currentCriteriaFound==false; weightCriteria index++){
            if(availableCriteria[availableCriteria index] ==
weights [z].relatedCriteria[weightCriteria index]){
              currentCriteriaFound = true;
              //ROS_INFO(":::dt::: currentCriteriaFound :::dt:::");
            }
          }
          if(availableCriteria index==0){
            allCriteriaFound = currentCriteriaFound;
          }
          else{
            allCriteriaFound = allCriteriaFound && currentCriteriaFound;
          }
        }
        if(allCriteriaFound == true){
```

```
          currentCriteriaWeight = weights_[z].weight;
          //ROS_INFO(":::dt:::: allCriteriaFound: z = %d : %f :::dt:::",z,weights_[z].weight);
          break;
        }
      }
    }

    //removes the criteria of the utility[k] from the available criteria set
    bool erased = false;
    for(uint z=0; z<availableCriteria.size() && erased==false; z++){
      if(availableCriteria[z] == utilities_[k].relatedCriterion){
        availableCriteria.erase(availableCriteria.begin()+z);
        erased = true;
        //ROS_INFO(":::dt:::: availableCriteria[%d] erased :::dt:::", z);
      }
    }

    //the global utility for thecurrent goal/frontier
    prevGoalUtility = goalUtility;
    goalUtility = prevGoalUtility + ((higherValue - lowerValue) * currentCriteriaWeight);
    //ROS_INFO(":::dt:::: partial goal utility computed: goalUtility = %f + ( %f - %f ) * %f = %f",
prevGoalUtility, higherValue, lowerValue, currentCriteriaWeight, goalUtility);
  }

  // returns the global cost for the current goal/frontier
  goalCost = (-1) * goalUtility;
  //ROS_INFO(":::dt:::: overall goal cost computed :::dt:::");

  //dt - only for logging utility/cost values
  if(logDebugInfo  == 1){
    FILE *stream;
    int  numwritten;
    char s_frontierCost[30];
    char s_orientationChange[30];
    char s_frontierGain[30];
    char s_cost[30];


    float normalized_frontierCost;
    float normalized_orientationChange;
    float normalized_frontierGain;
    char s_normalized_frontierCost[30];
    char s_normalized_orientationChange[30];
    char s_normalized_frontierGain[30];

    float pos_x = weightedFrontier.frontier.pose.position.x;
    float pos_y = weightedFrontier.frontier.pose.position.y;
    char s_pos_x[30];
    char s_pos_y[30];
    sprintf(s_pos_x, "%f", pos_x);
    sprintf(s_pos_y, "%f", pos_y);

    std::string goal_pose_string = "";
    goal_pose_string = std::string("(").append(s_pos_x).append(std::string(" ,
")).append(s_pos_y).append(std::string(")"));

    sprintf(s_frontierCost, "%f", frontierCost);
    sprintf(s_orientationChange, "%f", orientationChange);
    sprintf(s_frontierGain, "%f", frontierGain);
    sprintf(s_cost, "%f", goalCost);

    for(uint k=0; k<utilities_.size(); k++){
      if(utilities_[k].relatedCriterion == Criterion(potential))
        normalized_frontierCost = utilities_[k].normalizedValue;
      else if(utilities_[k].relatedCriterion == Criterion(orientation))
        normalized_orientationChange = utilities_[k].normalizedValue;
      else if(utilities_[k].relatedCriterion == Criterion(gain))
        normalized_frontierGain = utilities_[k].normalizedValue;
    }

    sprintf(s_normalized_frontierCost, "%f", normalized_frontierCost);
    sprintf(s_normalized_orientationChange, "%f", normalized_orientationChange);
    sprintf(s_normalized_frontierGain, "%f", normalized_frontierGain);

    if( (stream = fopen(debugInfo_filename_.c_str(), "a" )) != NULL )
    {
      numwritten = fputs( goal_pose_string.c_str(), stream );
      numwritten = fputs( " ", stream );
```

```
      numwritten = fputs( s_frontierCost, stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s orientationChange, stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s_frontierGain, stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s cost, stream );
      numwritten = fputs( "\n", stream );

      numwritten = fputs( "NORMALIZED VALUES:      ", stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s_normalized_frontierCost, stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s normalized orientationChange, stream );
      numwritten = fputs( " ", stream );
      numwritten = fputs( s normalized frontierGain, stream );
      numwritten = fputs( "\n\n", stream );

      fclose( stream );
    }
    else
      printf( "Problem opening the file\n" );
  }

  return goalCost;
}
```

3.12. Modificato il metodo getExplorationGoals( )

Modificato, nel file explore_frontier.cpp, il metodo getExplorationGoals(), che restituiva i goal da esplorare in ordine di costo decrescente. Ora il metodo controlla quale strategia è stata scelta e richiama il corrispondente metodo per scegliere e ordinare i goal da raggiungere.

Modificata l'intestazione del metodo getExplorationGoals() nel file explore_frontier.h

```
  /**
   * @brief Returns a list of frontiers, sorted by the planners estimated cost to visit each
frontier
   * @param costmap The costmap to search for frontiers
   * @param start The current position of the robot
   * @param goals Will be filled with sorted list of current goals
   * @param planner A planner to evaluate the cost of going to any frontier
   * @param potential scale A scaling for the potential to a frontier goal point for the frontier's
cost
   * @param orientation scale A scaling for the change in orientation required to get to a goal
point for the frontier's cost
   * @param gain_scale A scaling for the expected information gain to get to a goal point for the
frontier's cost
   * @param exploration_strategy The chosen exploration strategy
   * @param potentialOrientation_scale The weight for the combined criteria potential and
orientation to get to a goal point for the frontier's cost
   * @param potentialGain_scale The weight for the combined criteria potential and gain to get to a
goal point for the frontier's cost
   * @param orientationGain_scale The weight for the combined criteria orientation and gain to get
to a goal point for the frontier's cost
   * @param potentialOrientationGain_scale The weight for the combined criteria potential,
orientation and gain to get to a goal point for the frontier's cost
   *
   * @return True if at least one frontier was found
   *
   * The frontiers are weighted by a simple cost function, which prefers
   * frontiers which are large and close:
   *   frontier cost = travel cost / frontier size
   *
   * Several different positions are evaluated for each frontier. This
   * improves the robustness of goals which may lie near other obstacles
   * which would prevent planning.
   */
```

```
  virtual bool getExplorationGoals(costmap_2d::Costmap2DROS& costmap, tf::Stamped<tf::Pose>
robot pose, navfn::NavfnROS* planner, std::vector<geometry msgs::Pose>& goals, double cost scale,
double orientation_scale, double gain_scale, int exploration_strategy , double
potentialOrientation_scale, double potentialGain_scale, double orientationGain_scale, double
potentialOrientationGain_scale);
```

Aggiunta la parte in rosso all'interno del file explore_frontier.cpp

```
bool ExploreFrontier::getExplorationGoals(Costmap2DROS& costmap, tf::Stamped<tf::Pose> robot_pose,
navfn::NavfnROS* planner, std::vector<geometry_msgs::Pose>& goals, double potential_scale, double
orientation_scale, double gain_scale, int exploration_strategy , double potentialOrientation_scale,
double potentialGain_scale, double orientationGain_scale, double potentialOrientationGain_scale)
{
  findFrontiers(costmap);
  if (frontiers_.size() == 0)
    return false;

  geometry msgs::Point start;
  start.x = robot pose.getOrigin().x();
  start.y = robot pose.getOrigin().y();
  start.z = robot_pose.getOrigin().z();

  planner->computePotential(start);

  planner  = planner;
  costmapResolution  = costmap.getResolution();

  //we'll make sure that we set goals for the frontier at least the circumscribed
  //radius away from unknown space
  float step = -1.0 * costmapResolution ;
  int c = ceil(costmap.getCircumscribedRadius() / costmapResolution );
  WeightedFrontier goal;
  std::vector<WeightedFrontier> weightedFrontiers;
  weightedFrontiers.reserve(frontiers_.size() * c);

  //:::dt:::
  srand(static_cast<int>(time(NULL)));
  utilities_.clear();

  for (uint i=0; i < frontiers_.size(); i++) {
    Frontier& frontier = frontiers [i];
    WeightedFrontier weightedFrontier;
    weightedFrontier.frontier = frontier;

    tf::Point p(frontier.pose.position.x, frontier.pose.position.y, frontier.pose.position.z);
    tf::Quaternion bt;
    tf::quaternionMsgToTF(frontier.pose.orientation, bt);
    tf::Vector3 v(cos(bt.getAngle()), sin(bt.getAngle()), 0.0);

    for (int j=0; j <= c; j++) {
      tf::Vector3 check point = p + (v * (step * j));
      weightedFrontier.frontier.pose.position.x = check point.x();
      weightedFrontier.frontier.pose.position.y = check point.y();
      weightedFrontier.frontier.pose.position.z = check_point.z();

// intervengo sul calcolo del costo in base al parametro exploration_strategy:
// i goal verranno poi ordinati in base al costo, in ordine crescente

      //ROS_INFO(":::dt::: exploration_strategy = %d", exploration_strategy);

      if(exploration_strategy==0){ //RANDOM
        //ROS_INFO(":::dt::: exploration_strategy = RANDOM");
        weightedFrontier.cost = getRandomCost(weightedFrontier);
      }
      else if(exploration_strategy==1){ //WEIGHTED
        //ROS_INFO(":::dt::: exploration_strategy = WEIGHTED");
        weightedFrontier.cost = getWeightedCost(weightedFrontier, robot_pose, potential_scale,
orientation scale, gain scale);
      }
      else if(exploration_strategy==2){ //MCDM
        //ROS_INFO(":::dt::: exploration_strategy = MCDM");
        weightedFrontier.cost = getMcdmCost(weightedFrontier, robot_pose, potential_scale,
orientation_scale, gain_scale, potentialOrientation_scale, potentialGain_scale,
orientationGain_scale, potentialOrientationGain_scale);
```

```
      }

      weightedFrontiers.push_back(weightedFrontier);

    }
  }

  goals.clear();
  goals.reserve(weightedFrontiers.size());
  std::sort(weightedFrontiers.begin(), weightedFrontiers.end());

  for (uint i = 0; i < weightedFrontiers.size(); i++) {
    //:::dt:::
    //ROS_INFO("goals[%d] : x=%f - y=%f -
cost=%f",(int)i,(float)weightedFrontiers[i].frontier.pose.position.x,(float)weightedFrontiers[i].fro
ntier.pose.position.y,(float)weightedFrontiers[i].cost);

    goals.push_back(weightedFrontiers[i].frontier.pose);
  }
  return (goals.size() > 0);
}
```

Modificata la seguente chiamata all'interno del metodo "makePlan()" nel file explore.cpp

```
explorer ->getExplorationGoals(*explore costmap ros , robot pose, planner , goals, potential scale ,
orientation_scale_, gain_scale_, exploration_strategy_ , potentialOrientation_scale_,
potentialGain_scale_, orientationGain_scale_, potentialOrientationGain_scale_);
```

3.13. Nuovo file di configurazione per RVIZ

Creata nuova cartella "rviz" all'interno della cartella explore_stage. Copiato all'interno di tale cartella il file stage/rviz/stage.vcg e rinominato: rviz_config.vcg.

NB: il topic per la visualizzazione della mappa è: **/explore/map**

D'ora in avanti useremo il seguente comando per lanciare RVIZ:

```
rosrun rviz rviz -d $(rospack find explore_stage)/rviz/rviz_config.vcg
```

Modifico il file in modo tale da avere subito a disposizione tutti i topic necessari per la visualizzazione dell'esplorazione:

Modificato il file explore_stage/rviz/stage.vcg

```
Background\ ColorR=0
Background\ ColorG=0
Background\ ColorB=0
Fixed\ Frame=/map
Target\ Frame=<Fixed Frame>
Grid.Alpha=0,5
Grid.Cell\ Size=1
Grid.ColorR=0,5
Grid.ColorG=0,5
Grid.ColorB=0,5
Grid.Enabled=1
Grid.Line\ Style=0
Grid.Line\ Width=0,03
Grid.Normal\ Cell\ Count=0
Grid.OffsetX=0
Grid.OffsetY=0
Grid.OffsetZ=0
Grid.Plane=0
Grid.Plane\ Cell\ Count=50
Grid.Reference\ Frame=<Fixed Frame>
Laser\ Scan.Alpha=1
```

```
Laser\ Scan.Autocompute\ Intensity\ Bounds=0
Laser\ Scan.Billboard\ Size=0,01
Laser\ Scan.Channel=0
Laser\ Scan.Decay\ Time=60
Laser\ Scan.Enabled=1
Laser\ Scan.Max\ ColorR=1
Laser\ Scan.Max\ ColorG=1
Laser\ Scan.Max\ ColorB=1
Laser\ Scan.Max\ Intensity=0
Laser\ Scan.Min\ ColorR=0
Laser\ Scan.Min\ ColorG=0
Laser\ Scan.Min\ ColorB=0
Laser\ Scan.Min\ Intensity=0
Laser\ Scan.Selectable=1
Laser\ Scan.Style=1
Laser\ Scan.Topic=/base scan
Map.Alpha=0,7
Map.Draw\ Behind=0
Map.Enabled=1
Map.Topic=/explore/map
Markers.Enabled=1
Markers.Marker\ Topic=visualization_marker
Markers.explore_goal=1
Markers.frontiers=1
Markers.loop closure=1
Path.Alpha=1
Path.ColorR=0,1
Path.ColorG=1
Path.ColorB=0
Path.Enabled=1
Path.Topic=/move base/NavfnROS/plan
Transforms.All\ Enabled=1
Transforms.Enabled=1
Transforms.Show\ Arrows=0
Transforms.Show\ Axes=1
Transforms.Show\ Names=1
Transforms.Update\ Rate=1
Tool\ 2D\ Nav\ GoalTopic=goal
Tool\ 2D\ Pose\ EstimateTopic=initialpose
Camera\ Type=rviz::FixedOrientationOrthoViewController
Camera\ Config=14.0493 -1.31134e-06 30 -5.68434e-14 -0.707107 -0 -0 0.707107
Property\ Grid\ State=selection=Map.Enabled.Map.Topic;expanded=.Global
Options,Transforms./base footprint/base footprint,Transforms./base laser link/base laser link,Transf
orms./base_link/base_link,Transforms./map/map,Transforms./odom/odom,Transforms.Enabled.Transforms.Tr
ee,Transforms./mapTree/map,Transforms./odomTree/odom,Transforms./base_footprintTree/base_footprint,T
ransforms./base_linkTree/base_link,Transforms./base_laser_linkTree/base_laser_link,Map.Enabled,Marke
rs.Enabled.Markers.StatusTopStatus,Markers.Enabled.Markers.Namespaces;scrollpos=0,0;splitterpos=150,
286;ispageselected=1
[Display0]
Name=Grid
Package=rviz
ClassName=rviz::GridDisplay
[Display1]
Name=Transforms
Package=rviz
ClassName=rviz::TFDisplay
[Display2]
Name=Laser Scan
Package=rviz
ClassName=rviz::LaserScanDisplay
[Display3]
Name=Map
Package=rviz
ClassName=rviz::MapDisplay
[Display4]
Name=Markers
Package=rviz
ClassName=rviz::MarkerDisplay
[Display5]
Name=Path
Package=rviz
ClassName=rviz::PathDisplay
[Transforms.]
base_footprintEnabled=1
base laser linkEnabled=1
base_linkEnabled=1
mapEnabled=1
```

```
odomEnabled=1
```

3.14. Integrazione del package explore_test realizzato da Simone Pignatelli

- ***Creazione di un nuovo package nella nostra cartella principale: tesi_ros***

```
cd tesi ros
roscreate-pkg explore test nav msgs move base msgs  roscpp
rospack profile
rospack find explore_test
roscd explore test
rosdep explore test
rosmake explore_test
```

Copiare dal package di Simone Pignatelli al nostro nuovo package, tutti i files e le cartelle, ad eccezione dei seguenti file:

- CMakeLists.txt
- mainpage.dox
- Makefile
- manifest.xml

Aggiungere in fondo al file CMakeLists.txt la seguente riga:

```
rosbuild_add_executable(explore_test src/test.cpp)
```

- ***Integrazione con il nostro package di esplorazione (exploration/explore)***

Modifica dei file explore.cpp ed explore.h per l'invio di un messaggio al package explore_test nel momento in cui l'esplorazione sia stata completata.

Aggiunta la parte in rosso nel file explore.h

```
#ifndef NAV_EXPLORE_H_
#define NAV_EXPLORE_H_

#include <ros/ros.h>
#include <move base msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>
#include <geometry_msgs/PoseStamped.h>
#include <nav msgs/GetMap.h>
#include <costmap 2d/costmap 2d ros.h>
#include <costmap 2d/costmap 2d.h>
#include <navfn/navfn_ros.h>
#include <explore/explore_frontier.h>
#include <explore/loop closure.h>
#include <visualization msgs/MarkerArray.h>
#include <vector>
#include <string>
#include <boost/thread/mutex.hpp>
#include <std_msgs/Bool.h>

namespace explore {

/**
 * @class Explore
 * @brief A class adhering to the robot_actions::Action interface that moves the robot base to
explore its environment.
 */
```

```
class Explore {
public:
  /**
   * @brief  Constructor
   * @return
   */
  Explore();

  /**
   * @brief  Destructor - Cleans up
   */
  virtual ~Explore();

  /**
   * @brief  Runs whenever a new goal is sent to the move base
   * @param goal The goal to pursue
   * @param feedback Feedback that the action gives to a higher-level monitor, in this case, the
position of the robot
   * @return The result of the execution, ie: Success, Preempted, Aborted, etc.
   */
//  virtual robot actions::ResultStatus execute(const ExploreGoal& goal, ExploreFeedback& feedback);
  void execute();

  void spin();

private:

  /**
   * @brief  :::dt::: decides whether to choose a new goal or not
   */
  bool performDecision startStopPerception();

  /**
   * @brief  :::dt::: checks if the chosen goal has been reached
   */
  bool chosenGoal reached();

  /**
   * @brief  Make a global plan
   */
  void makePlan();

  /**
   * @brief  Publish a goal to the visualizer
   * @param  goal The goal to visualize
   */
  void publishGoal(const geometry_msgs::Pose& goal);

  /**
   * @brief publish map
   */
  void publishMap();

  void reachedGoal(const actionlib::SimpleClientGoalState& status, const
move_base_msgs::MoveBaseResultConstPtr& result, geometry_msgs::PoseStamped frontier_goal);

  /**
   * @brief  Resets the costmaps to the static map outside a given window
   */
//  void resetCostmaps(double size x, double size y);

  bool mapCallback(nav_msgs::GetMap::Request  &req, nav_msgs::GetMap::Response &res);

  bool goalOnBlacklist(const geometry msgs::PoseStamped& goal);

  ros::NodeHandle node ;
  tf::TransformListener tf_;
  costmap_2d::Costmap2DROS* explore_costmap_ros_;

  actionlib::SimpleActionClient<move base msgs::MoveBaseAction> move base client ;

  navfn::NavfnROS* planner_;
  std::string robot_base_frame_;
  bool done_exploring_;

  ros::Publisher marker publisher ;
  ros::Publisher marker_array_publisher_;
  ros::Publisher map_publisher_;
```

```
  ros::Publisher exploring_status_publisher_;
  ros::ServiceServer map server ;

  ExploreFrontier* explorer ;

  tf::Stamped<tf::Pose> global_pose_;
  double planner frequency ;
  int    visualize ;
  LoopClosure* loop closure ;
  std::vector<geometry msgs::PoseStamped> frontier blacklist ;
  geometry_msgs::PoseStamped prev_goal_;
  unsigned int prev_plan_size_;
  double time since progress , progress timeout ;
  double potential scale , orientation scale , gain scale ;
  boost::mutex client mutex ;
  bool   close loops ;

  //:::dt:::
  int decision perception discretization ;
  int exploration strategy ;
  geometry msgs::PoseStamped chosenGoal pose ;
  std::vector<geometry_msgs::Pose> savedGoals_;
  double chosenGoalReached_tolerance_;

};

}

#endif
```

Aggiunta la parte in rosso nel metodo "Explore()", nel file explore.cpp

```
Explore::Explore() :
  node_(),
  tf (ros::Duration(10.0)),
  explore costmap ros (NULL),
  move_base_client_("move_base"),
  planner_(NULL),
  done_exploring_(false),
  explorer_(NULL),
  prev plan size (0)
{
  ros::NodeHandle private_nh("~");

  … … … … … …
  … … … … … …

  //:::dt:::
  private_nh.param("decision_perception_discretization", decision_perception_discretization_, 0);
  private_nh.param("exploration_strategy", exploration_strategy_, 0);
  private nh.param("chosenGoalReached tolerance", chosenGoalReached tolerance , 0.5);
  exploring status publisher  = private nh.advertise<std msgs::Bool>("exploring status", 10);

  … … … … … …
  … … … … … …
```

Aggiunta la parte in rosso nel metodo "execute()", nel file explore.cpp

```
void Explore::execute() {
… … … … … …
… … … … … …

  while (node_.ok() && (!done_exploring_)) {

    if (close_loops_) {
      tf::Stamped<tf::Pose> robot pose;
      explore costmap ros ->getRobotPose(robot pose);
      loop_closure_->updateGraph(robot_pose);
    }
```

```
    makePlan();
    r.sleep();
  }

//:::dt::: block added in order to make this node able to communicate with the explore_test node
  std msgs::Bool exploration done;
  exploration done.data = true;
  exploring status publisher .publish(exploration done);

  move_base_client_.cancelAllGoals();
}
```

- ***Modifica del file test.cpp***

  Modificata la parte in rosso del file /explore_test/src/test.cpp

```
… … …
… … …

// scrive la prima linea del file
void scriviIntestazione()
{
          if(intestazioneScritta == false){
                    fprintf(log_file, "time --- map --- distance\n");
                    intestazioneScritta = true;
          }
}

//scrive i dati sul file di log
void scriviDati(const ros::TimerEvent& e)
{
          ROS_INFO("mappa coperta: %d. stop a %d", covered map, stop map);
          ROS_INFO("distance: %f", distance);
          ROS_INFO("time: %f", ros::Time::now().toSec());
          scriviIntestazione();
          fprintf(log_file, "\n%f      %d       %f", ros::Time::now().toSec(), covered_map,
distance);
}

… … …
… … …

//interrompi la simulazione uccidendo tutti i nodi di ros (brutale ma efficace)
void stopSimulation()
{
          fprintf(log_file, "\n%f      %d       %f", ros::Time::now().toSec(), covered_map,
distance);
          fclose(log file);
          system("rosnode kill -a");
}

… … …
… … …

//rileva se l'esplorazione è terminata ascoltando il messaggio sul topic apposito
//se è terminata chiude tutto

void statoEsplorazione(const std_msgs::Bool::ConstPtr& exploration_done)
{
          if(exploration_done)
          {
                    ROS_INFO("FINITA ESPLORAZIONE");
                    stopSimulation();
          }
}
```

- ***Sostituire tutti i file di configurazione con quelli della nostra versione di explore***

Da eseguire al fine di mantenere le modifiche fatte da noi al package explore ed evitare che eventuali differenze di configurazione possano compromettere il lavoro svolto finora.

- Rimuovere tutte le cartelle e i file di configurazione contenuti nella cartella tesi_ros/explore_test/config, tranne le seguenti cartelle e i seguenti file:
    - bosch_maps/
    - bosch_worlds/
    - explore_slam.launch
    - explore.launch
    - launch.php
    - test_parameters.xml
- Sostituire tutti i file contenuti in “bosch_worlds/” con quelli contenuti in “*bosh-ros-pkg/trunk/stacks/bosch_demos/bosch_worlds/”*
- Sostituire il file di configurazione “explore_test/config/explore_slam.launch” con il file copiato da “exploration/explore_stage/explore_slam.launch”.
- Modificare il file “explore_test/config /explore_slam.launch”:

```
<launch>
  <master auto="start"/>
  <param name="/use sim time" value="true"/>
  <node pkg="stage" type="stageros" name="stage" args="-g $ (find
explore_test)/config/bosh_worlds/maze-noisy.world" respawn="false" output="screen"/>
  <node pkg="gmapping" type="slam_gmapping" name="gmapping" output="screen">
    <remap from="scan" to="base scan" />
    <param name="inverted laser" value="false" />
    <param name="maxUrange" value="30.0" />
    <param name="particles" value="30" />
    <param name="delta" value="0.10" />
    <param name="xmin" value="-15.0" />
    <param name="xmax" value="15.0" />
    <param name="ymin" value="-15.0" />
    <param name="ymax" value="15.0" />
    <param name="angularUpdate" value="0.5" />
    <param name="linearUpdate" value="1.0" />
    <param name="map_update_interval" value="1.0" />
    <param name="resampleThreshold" value="0.3" />
    <param name="llsamplerange" value ="0.05" />
    <param name="llsamplestep" value ="0.05" />
    <param name="lasamplerange" value ="0.05" />
    <param name="lasamplestep" value ="0.05" />
  </node>
  <include file="$(find explore stage)/move.xml" />
  <include file="$(find explore_stage)/explore_slam.xml" />
  <include file="$(find explore_test)/config/test_parameters.xml" />
</launch>
```

Modificare anche il file “explore_test/config /explore.launch” nel caso in cui si volesse provare l’esplorazione con percezione perfetta:

```
<launch>
  <master auto="start"/>
  <param name="/use sim time" value="true"/>
  <node pkg="stage" type="stageros" name="stage" args="-g $(find
explore_test)/config/bosh_worlds/maze.world" respawn="false" output="screen"/>
  <node pkg="tf" type="static_transform_publisher" name="fake_localize" args="0 0 0 0 0 0 map odom
10"/>
  <include file="$(find explore stage)/move.xml" />
  <include file="$(find explore stage)/explore.xml" />
  <include file="$(find explore_test)/config/test_parameters.xml" />
</launch>
```

- ***Correggere i file che utilizzeremo per le mappe (maze.world, maze-noisy.world)***

Affinchè la struttura del codice sia coerente con la parte restante dei file utilizzati, e affinchè si possa intervenire con il file launch.php per effettuare il parsing di questi file e modificare in modo automatico la

posizione di partenza, dobbiamo modificare il codice come illustrato, portando a capo la parentesi dopo "segway":

```
segway
(
  pose [0 0 0]
  name "segway"
  color "gray"
)
```

- ***Modificare/correggere il file launch.php***

Aggiungere la riga **$logFileName = $logFileName.$err_count**.

Ridurre la soglia di mappa esplorata per considerare valido un test (ridotta da 9900 a **9350**).

Salvare i log nella cartella "**explore_test/logs/**".

```
<?php
	//lancia la batteria di simulazioni con i punti di partenza indicati nel vettore
sottostante
	//-- IMPORTANTE -- occorre prima far partire da shell il core di ROS col comando roscore

	function setLogFile($logFileName)
	{
		$xml = simplexml_load_file("test_parameters.xml");
		foreach($xml->node as $node)
			if($node['pkg'] == "explore_test")
				foreach($node->param as $param)
					if($param['name'] == "log filename")
						$param['value'] = "$(find
explore_test)/logs/".$logFileName;
		file_put_contents("test_parameters.xml",$xml->asXML());
	}

	function setStartPose($x, $y, $z, $t)
	{
		$pose = $x. " ". $y. " ". $z. " ". $t;
		$stringa = file get contents("./bosh worlds/maze-noisy.world");
		$segway pos = strpos($stringa, "segway\n");
		$lpar pos = strpos($stringa, "[", $segway pos);
		$rpar pos = strpos($stringa, "]", $lpar pos);
		$stringa_left = substr($stringa,0,$lpar_pos+1);
		$stringa_right = substr($stringa,$rpar_pos);
		$stringa = $stringa left . $pose. $stringa right;
		file put contents("./bosh worlds/maze-noisy.world",$stringa);
		echo $stringa;
	}

	$pose[0] = array(0, 0, 0, 0);
	$pose[1] = array(-9, 8.5, 0, 0);
	$pose[2] = array(8.5, 8.5, 0, 225);
	$pose[3] = array(-7, 5, 0, 180);
	$pose[4] = array(2, 5, 0, 0);
	$pose[5] = array(-7, -4, 0, 270);
	$pose[6] = array(-2, -6, 0, 0);
	$pose[7] = array(4, -9, 0, 90);
	$pose[8] = array(8, -8, 0, 90);
	$pose[9] = array(9, 0, 0, 180);

	$min_speed = 0.4;
	$min_area = 9350;
	$err count = 0;

	for($i=0;$i<count($pose);$i++)
	{
		$logFileName = "explore".$i.".log";
		setLogFile($logFileName);
		setStartPose($pose[$i][0],$pose[$i][1],$pose[$i][2],$pose[$i][3]);
		system('roslaunch explore_test explore_slam.launch');
		$logFileName = "../logs/".$logFileName;
```

```
                    //calcola velocitÃ  e rifai il test se Ã¨ inferiore alla soglia
                    //oppure se non ha scoperto abbastanza area
                    $rows = file($logFileName);
                    end($rows);
                    $gap = 0.0;
                    while($gap < 0.005)
                    {
                              $last = current($rows);
                              prev($rows);
                              $secondLast = current($rows);
                              $data = explode("\t", $last);
                              $distance = trim($data[2]);
                              $prevData = explode("\t", $secondLast);
                              $prevDistance = trim($prevData[2]);
                              $gap = ($distance - $prevDistance) / $distance;
                    }
                    $time = trim($data[0]);
                    $area = trim($data[1]);
                    $speed = $distance/$time;
                    if($speed<$min speed || $area < $min area)
                    {
                              rename($logFileName,$logFileName.$err_count);
                              $logFileName = $logFileName.$err_count;
                              $i--;
                              $err count++;
                    }
                    else
                              $err_count=0;

                    //replace dots whit commas in logs
                    $s1 = file get contents($logFileName);
                    $s2 = str_replace(".",",",$s1);
                    file_put_contents($logFileName,$s2);
          }

?>
```

- ***Ri-compilare i package exploration ed explore_test***

```
rosmake exploration
rosmake explore_stage
```

- ***Lanciare i test***

Posizionarsi in explore_test/config ed eseguire il seguente comando:

```
php launch.php
```

3.15. Sistema di debug – loggare i valori calcolati e le scelte fatte

In aggiunta al package explore_test, è importante poter valutare con precisione se le modifiche apportate all'applicazione ci abbiano portato ad ottenere il comportamento desiderato.

A tal fine è stato inserito un nuovo parametro come variabile globale, che decida se effettuare il debug della nostra applicazione, salvando i nostri dati all'interno di file di log.

Ovviamente, sfruttando questà modalità, i risultati e le performance potrebbero risultare falsati, di conseguenza è importante che questa modalità di debug sia disabilitata durante i test veri e propri.

Aggiunto il parametro “logDebugInfo” nel file explore_slam.xml e nel file explore.xml per permettere a chi lo volesse di attivare il salvataggio delle informazioni di debug all’interno di un file. (se impostato a 0, disattiva il log, altrimenti se impostato a 1 lo attiva).

Aggiunta la riga seguente nei file explore_slam.xml ed explore.xml

```
<param name="logDebugInfo" value="0"/>
```

Dichiarata la seguente variabile globale nel file explore.h

```
int logDebugInfo_;
```

Dichiarata le seguenti variabili globali nel file explore_frontier.h

```
 int logDebugInfo_;
 string debugInfo_filename_;
```

Importato in valore della variabile dal file di configurazione, nel file explore.cpp

```
private_nh.param("logDebugInfo", logDebugInfo_, 0);
```

Modificata la chiamata al metodo getExploratinoGoals, aggiungendo la nuova variabile, nel file explore.cpp

```
explorer ->getExplorationGoals(*explore costmap ros , robot pose, planner , goals, potential scale ,
orientation scale , gain scale , exploration strategy , potentialOrientation scale ,
potentialGain_scale_, orientationGain_scale_, potentialOrientationGain_scale_, logDebugInfo_);
```

modificata la dichiarazione del metodo, nel file explore_frontier.h

```
/**
   * @brief Returns a list of frontiers, sorted by the planners estimated cost to visit each
frontier
   * @param costmap The costmap to search for frontiers
   * @param start The current position of the robot
   * @param goals Will be filled with sorted list of current goals
   * @param planner A planner to evaluate the cost of going to any frontier
   * @param potential scale A scaling for the potential to a frontier goal point for the frontier's
cost
   * @param orientation_scale A scaling for the change in orientation required to get to a goal
point for the frontier's cost
   * @param gain scale A scaling for the expected information gain to get to a goal point for the
frontier's cost
   * @param exploration_strategy The chosen exploration strategy
   * @param potentialOrientation_scale The weight for the combined criteria potential and
orientation to get to a goal point for the frontier's cost
   * @param potentialGain scale The weight for the combined criteria potential and gain to get to a
goal point for the frontier's cost
   * @param orientationGain_scale The weight for the combined criteria orientation and gain to get
to a goal point for the frontier's cost
   * @param potentialOrientationGain_scale The weight for the combined criteria potential,
orientation and gain to get to a goal point for the frontier's cost
   * @param logDebugInfo if set to 1, logging debug info is enabled
   *
   * @return True if at least one frontier was found
   *
   * The frontiers are weighted by a simple cost function, which prefers
   * frontiers which are large and close:
   *   frontier cost = travel cost / frontier size
   *
   * Several different positions are evaluated for each frontier. This
```

```
   * improves the robustness of goals which may lie near other obstacles
   * which would prevent planning.
   */
  virtual bool getExplorationGoals(costmap 2d::Costmap2DROS& costmap, tf::Stamped<tf::Pose>
robot_pose, navfn::NavfnROS* planner, std::vector<geometry_msgs::Pose>& goals, double cost_scale,
double orientation_scale, double gain_scale, int exploration_strategy, double
potentialOrientation scale, double potentialGain scale, double orientationGain scale, double
potentialOrientationGain_scale, int logDebugInfo_);
```

aggiunto il metodo setLogFilename(exploration_strategy) nel file explore_frontier.h

```
/**
 * @brief sets the global variable debugInfo_filename_ with the filename of the log file for the
debug info
 */
virtual bool setLogFilename(int exploration_strategy);
```

aggiunto il metodo setLogFilename(exploration_strategy) nel file explore_frontier.cpp

```
bool ExploreFrontier::setLogFilename(int exploration_strategy, double potential_scale, double
orientation_scale, double gain_scale, double potentialOrientation_scale, double potentialGain_scale,
double orientationGain_scale, double potentialOrientationGain_scale){
  if(debugInfo_filename_.compare("") != 0)
    return false;

  ROS_INFO(":::dt::: setting logFilename");

  std::string path = "/home/tortorella/tesi_ros/bosch-ros-pkg/trunk/stacks/exploration/explore/";
  std::string filename = "";

  std::string strategy = "";
  std::string parameters = "";
  std::string header = "";
  std::string backn = "\n";

  char s_potential_scale[30];
  char s_orientation_scale[30];
  char s_gain_scale[30];
  char s_potentialOrientation_scale[30];
  char s_potentialGain_scale[30];
  char s_orientationGain_scale[30];
  char s_potentialOrientationGain_scale[30];

  sprintf(s_potential_scale, "%f", potential_scale);
  sprintf(s_orientation_scale, "%f", orientation_scale);
  sprintf(s_gain_scale, "%f", gain_scale);
  sprintf(s_potentialOrientation_scale, "%f", potentialOrientation_scale);
  sprintf(s_potentialGain_scale, "%f", potentialGain_scale);
  sprintf(s_orientationGain_scale, "%f", orientationGain_scale);
  sprintf(s_potentialOrientationGain_scale, "%f", potentialOrientationGain_scale);


  if(exploration_strategy==0){ //RANDOM
    filename = "randomUtilities_log.txt";

    strategy = "RANDOM";

    header = "TABLE: goal pose - random cost";
  }
  else if(exploration_strategy==1){ //WEIGHTED
    filename = "weightedUtilities_log.txt";

    strategy = "WEIGHTED";

    parameters = std::string("potential_scale").append(s_potential_scale).append(backn);
    parameters.append(std::string("orientation_scale").append(s_orientation_scale).append(backn));
    parameters.append(std::string("gain_scale").append(s_gain_scale).append(backn));

    header = "TABLE: goal pose - frontier cost - orientation cost - frontier gain - global cost";
  }
  else if(exploration_strategy==2){ //MCDM
    filename = "mcdmUtilities_log.txt";
```

```
    strategy = "MCDM";

    parameters = std::string("potential_scale").append(s_potential_scale).append(backn);
    parameters.append(std::string("orientation_scale").append(s_orientation_scale).append(backn));
    parameters.append(std::string("gain_scale").append(s_gain_scale).append(backn));

parameters.append(std::string("potentialOrientation_scale").append(s_potentialOrientation_scale).app
end(backn));

parameters.append(std::string("potentialGain_scale").append(s_potentialGain_scale).append(backn));

parameters.append(std::string("orientationGain_scale").append(s_orientationGain_scale).append(backn)
);

parameters.append(std::string("potentialOrientationGain_scale").append(s_potentialOrientationGain_sc
ale).append(backn));

    header = "TABLE: goal pose - frontier cost - orientation cost - frontier gain - global cost";
  }
  else{
    return false;
  }

  debugInfo_filename_ = path.append(filename);

  //dt - initialize the log file with some info
  if(logDebugInfo_ == 1){
    FILE *stream;
    int  numwritten;

    stream = fopen(debugInfo_filename_.c_str(), "w" );
    fclose( stream );

    if( (stream = fopen(debugInfo_filename_.c_str(), "a" )) != NULL )
    {
      numwritten = fputs( "STRATEGY: ", stream );
      numwritten = fputs( strategy.c_str(), stream );
      numwritten = fputs( "\n\n", stream );

      numwritten = fputs( "PARAMETERS:\n", stream );
      numwritten = fputs( parameters.c_str(), stream );
      numwritten = fputs( "\n\n", stream );

      numwritten = fputs( header.c_str(), stream );
      numwritten = fputs( "\n\n", stream );

      fclose( stream );
    }
    else
      printf( "Problem opening the file\n" );
  }

  return true;
}
```

modificato il metodo getExplorationGoals nel file explore_frontier.cpp

```
bool ExploreFrontier::getExplorationGoals(Costmap2DROS& costmap, tf::Stamped<tf::Pose> robot pose,
navfn::NavfnROS* planner, std::vector<geometry_msgs::Pose>& goals, double potential_scale, double
orientation_scale, double gain_scale, int exploration_strategy, double potentialOrientation_scale,
double potentialGain scale, double orientationGain scale, double potentialOrientationGain scale, int
logDebugInfo)
{
  logDebugInfo_ = logDebugInfo;
  setLogFilename(exploration_strategy, potential_scale, orientation_scale, gain_scale,
potentialOrientation_scale, potentialGain_scale, orientationGain_scale,
potentialOrientationGain_scale);

  findFrontiers(costmap);
  if (frontiers_.size() == 0)
    return false;
… … …
… … …
goals.clear();
```

```
  goals.reserve(weightedFrontiers.size());
  std::sort(weightedFrontiers.begin(), weightedFrontiers.end());
  for (uint i = 0; i < weightedFrontiers.size(); i++) {
    //:::dt:::
    //ROS_INFO("goals[%d] : x=%f - y=%f -
cost=%f",(int)i,(float)weightedFrontiers[i].frontier.pose.position.x,(float)weightedFrontiers[i].fro
ntier.pose.position.y,(float)weightedFrontiers[i].cost);

    goals.push back(weightedFrontiers[i].frontier.pose);
  }

  //dt - saves the first goal pose
  if(logDebugInfo_ == 1){
    FILE *stream;
    int  numwritten;

    float goal_pos_x = weightedFrontiers[0].frontier.pose.position.x;
    float goal_pos_y = weightedFrontiers[0].frontier.pose.position.y;
    char s_goal_pos_x[30];
    char s_goal_pos_y[30];
    sprintf(s_goal_pos_x, "%f", goal_pos_x);
    sprintf(s_goal_pos_y, "%f", goal_pos_y);

    std::string goal_pose_string = "";
    goal_pose_string = std::string("chosen goal pose: (").append(s_goal_pos_x).append(std::string("
, ")).append(s_goal_pos_y).append(std::string(")"));

    float robot_pos_x = robot_pose.getOrigin().x();
    float robot_pos_y = robot_pose.getOrigin().y();
    char s_robot_pos_x[30];
    char s_robot_pos_y[30];
    sprintf(s_robot_pos_x, "%f", robot_pos_x);
    sprintf(s_robot_pos_y, "%f", robot_pos_y);

    std::string robot_pose_string = "";
    robot_pose_string = std::string("robot pose: (").append(s_robot_pos_x).append(std::string(" ,
")).append(s_robot_pos_y).append(std::string(")"));

    if( (stream = fopen(debugInfo_filename_.c_str(), "a" )) != NULL )
    {
      //numwritten = fputs( "\n", stream );
      numwritten = fputs( goal_pose_string.c_str(), stream );
      numwritten = fputs( " --- ", stream );
      numwritten = fputs( robot_pose_string.c_str(), stream );
      numwritten = fputs( "\n\n", stream );

      fclose( stream );
    }
    else
      printf( "Problem opening the file\n" );
  }

  return (goals.size() > 0);
}
```