Preser

A Matlab Framework For Professional Education in Service Robotics.

Release 1.0.0

2016

Contents

1 About Preser

The goal of the project is to develop and test the new area of Professional Education in Service Robotics. The main motivation comes from the fact that there is a significant gap between research education and professiontal education, i.e. addressing the needs of inegrators of service robots.

PRESER will provides lectures for addressing the practical aspects of using a service robot in a domestic environment and a simulation framework based on MATLAB and the simulator V-REP for testing your knowledge during the lessons.

1.1 Download Preser

Our framework is available for OSX, Windows and Debian .You can download the binary included in Preser for free following the link below, or you can get the source code via our public repositories.

- BINARIES
- SOURCE CODE

1.2 Dependencies

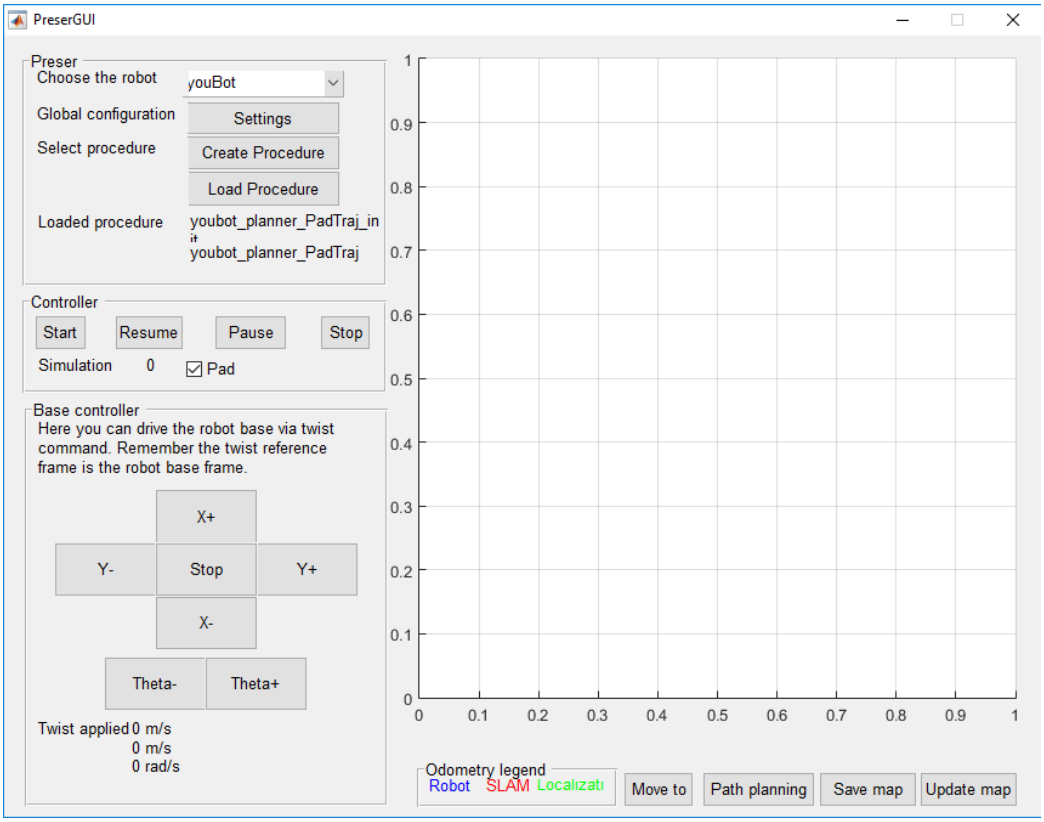
1.2.1 Java

Check your java version in Matlab, with following command :
>> version -java
Preser Runner Thread is builded with Java 1.7.0 .

2 PreserGUI

In this section we introduce the main interface.
Inside Matlab, move to the **Preser Workspace**, and write in the command window :
>> *start_me_up*

With this command we open the *GUI* and we read the ini configuration file *Preser.ini*. It contains 3 different sections, described in the tables.



Section : Hilas

parameter	value	content
start˙thread	true/false	Start the simulation loop
thread˙sleep	1-n	Sleep time between each simulation step
init˙procedure	string	Initialization procedure
procedure	string	Procedure executed every simulation step
matlab˙warning	true/false	Enable warning in the Matlab console
java˙log	true/false	Enable log for simulation runner
matlabcontrol˙log	true/false	Enable debug console for jmi.jar

Section : Simulator

parameter	value	content
socketaddr	ip address	Ip address where the simulator resides
socketport	number	Port number for the socket connection
scene	string	Name of the scene to be loaded at startup
startScene	true/false	Scene is loaded via Preser

Section : Robot

parameter	value	content
name	string	Name of the robot used in the simulation

We manage the configuration file via Interface.

Users can manually choose all the available robots in the popup menu. Then there are some "Push Buttons", that help users initial configuration :

- Settings : Global Configuration.
- Create Procedure : Users can create their custom procedures.
- Load Procedure : Users can load default procedures, that exist within software.

We will see these buttons in detail in the following sections.

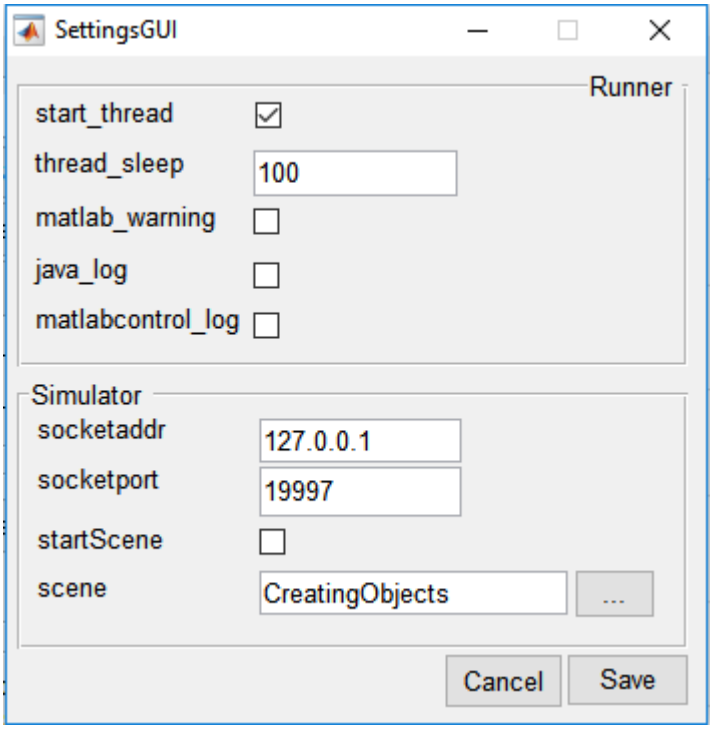
The Controller Panel is the main connection between Matlab and the Simulator, it communicates directly with the runner thread created on *"Start"* push button, which build a variable called **Hilas** .
With the *"Pause"* and *"Resume"* , we respectively suspend and resume the runner thread. The *Stop* push button, stop the execution of the runner thread, stop the simulator and exit from the loop. This will not clean the Matlab Workspace!
Inside Controller Panel there is also a checkbox *Pad* , it's helpfull to switch quickly between Joypad and calculated trajectory .

In the bottom left of the PreserGUI we have a Base Controller, here users can drive the robot via twist command in robot base frame. If in the procedure GUIJoypad is enable. Users can also use also a different joypad to

move the robot.

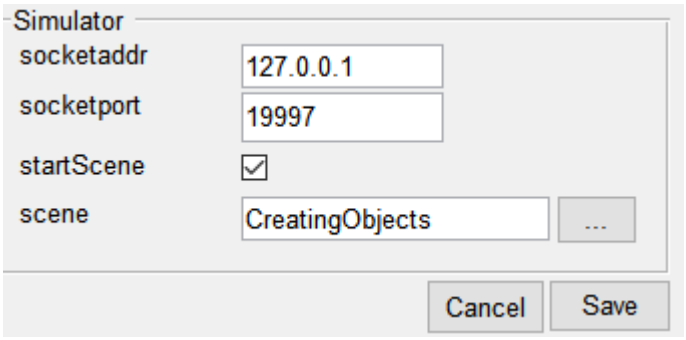
The figure inside the PreserGUI shows the Map of the Simulator scene or if you want a create a new scene ,the image corresponding to the planimetry. Users are able to Save, Update or generate a trajectory inside the Map with the push buttons below.

2.1 SettingsGUI

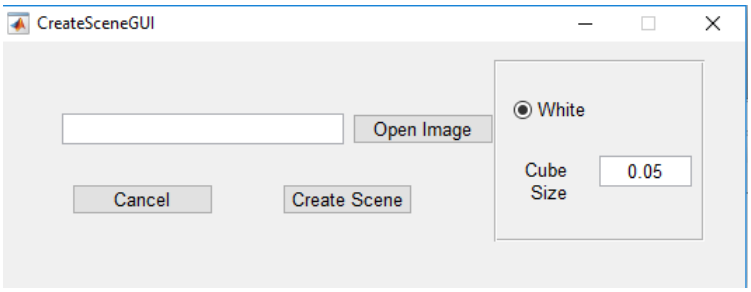


In SettingsGUI we manipulate the global settings of the ini configuration file, that we presented in the previous section. It's important to note the *startScene* checkbox, if enable and vrep is running , we will open the Scene called in the edit text below.

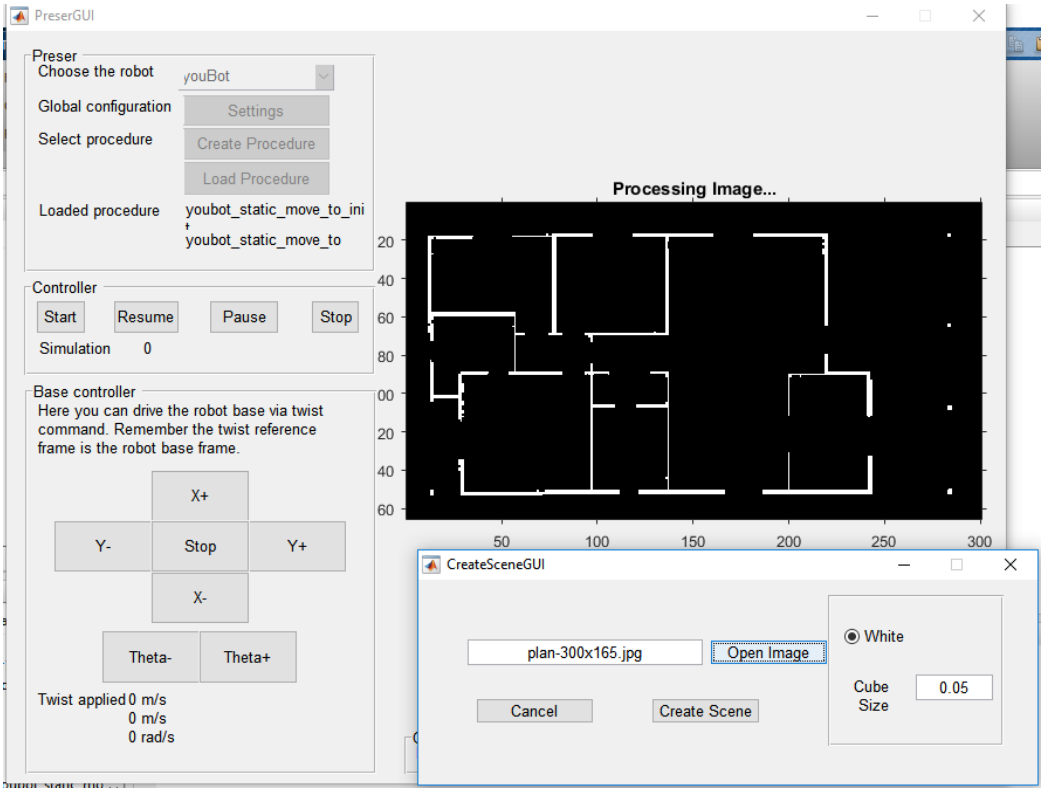
If you want to create a new V-REP Scene , given the image of the planimetry , users have to enable the *startScene* checkbox.



2.2 CreateSceneGUI



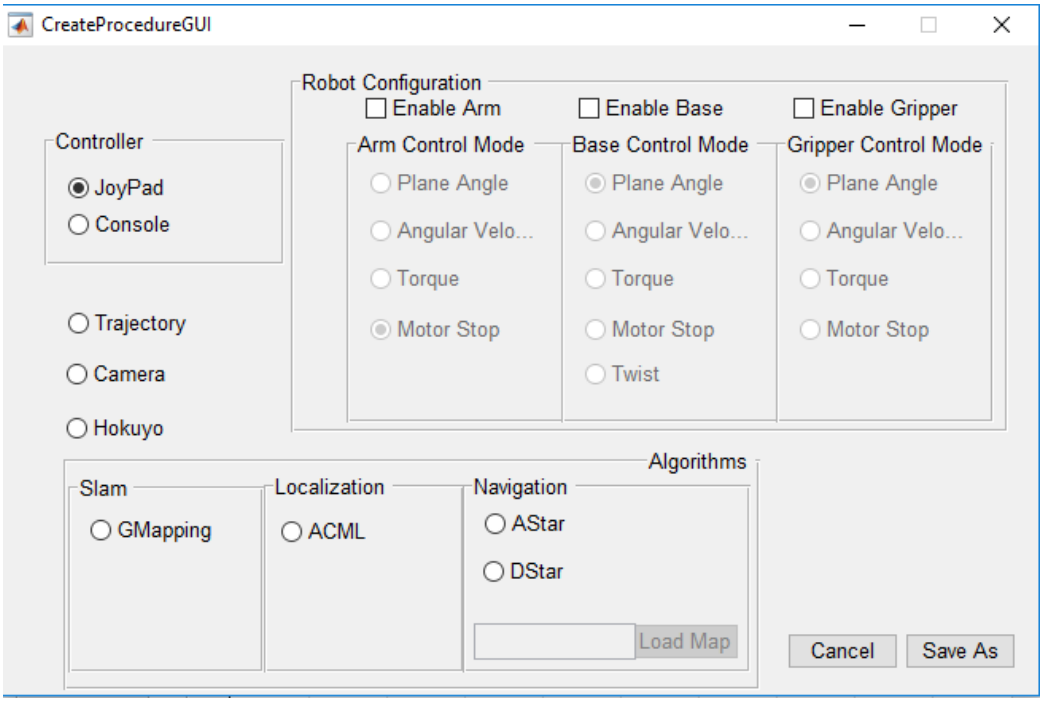
CreateSceneGUI will popup if users enable the *startScene* checkbox in SettingsGUI and choose *CreatingObjects* as scene. Users can navigate to the image path location and choose a *png or jpg* image. It loads image and it shows in the PreserGUI figure,as in the example



Additional settings:

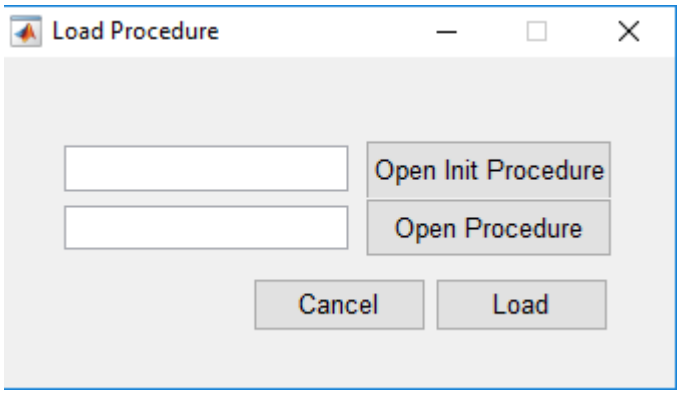
- *White radio button* : if toggle it means you are processing white pixel inside the Simulator.
- *Cube size* : it's the size of the cube created inside the Simulator.

2.3 CreateProcedureGUI



CreateProcedureGUI allows users to create a new custom procedures , by toggle / untoggle radio buttons.
In top left of the GUI you can choose a controller to move the robot, if *JoyPad* is enabled , the *Base Controller* in the PreserGUI will disable.
In Robot Configuration Panel , you can choose to enable individual parts of the robot, to set the *Control Mode* for every single parts.
The control modes parameter set the type of loop controller. I.e Setting the control mode to PLANE˙ANGLE the control loop will only apply the joint position target. The MOTOR˙STOP mode is different for the other ones. Specifying it a joint position controller is activated which keeps fixed the instantaneous joints position .
The control modes can be specified via enumeration or integer values via **Robot API**. In Algorithms Panel users can select the implemented algorithms, and also can choose to load a *Map* saved previously in the PreserGUI.
Save As push button will save one *init˙procedure*, and one *procedure* that will execute in each simulation step. These procedures will be save in the ini configuration file.

2.4 LoadProcedureGUI



LoadProcedureGUI allows users to load default procedures or procedures created previously in the CreateProcedureGUI. Loaded Procedures are stored in the ini configuration .

3 Preser API

3.1 General API

The General API have a couple of functions usefull for plotting data.

3.1.1 plotRobotKine()

Plot robot chain kinematic in default figure.

3.1.2 plotRobotKineWorkspace(cube_size)

Plot robot chain kinematic 'workspace' in a figure with *cube_size* axes.
Where workspace is the " Size of robot 3D ".

3.1.3 plotLaserPoints(laser)

Plot the 2D laser points in the laser reference frame.
laser is sensor struct .

3.1.4 plotStaticMap(static_map,amcl,robot)

Plot the occupancy grid from a static map. Amcl and robot pose are displayed in overlay with circles of different colour (green = amcl correction, blue = robot odometry).

3.1.5 overlayRobotData(static_map,robot)

Rescale robot position inside Map ,and plot robot .

3.1.6 overlayLocalizationData(static˙map,amcl,robot)

Rescale robot position inside Map ,and plot over the map.

3.1.7 overlaySlamData(slam,robot,amcl)

With 2 input argument :
Plot Slam Data , and robot position .
With 3 input argument :

TODO..

3.1.8 plotMap(slam,robot,amcl)

Plot the occupancy grid from slam as an image. Slam, amcl and robot pose are displayed in overlay with circles of different colour (red = slam correction, green = amcl correction, blue = robot odometry)..
It recalls overlaySlamData

3.1.9 saveMap(slam,filename)

Save current map in "saved/*filename*" directory, if doesn't exist a directory named "saved" will be created.

3.1.10 loadMap(filename)

Load *filename* Map, located in "saved/" directory.

3.1.11 logic = to˙logic˙string(value)

Convert *value* to logic.

3.1.12 saveConfigToFile()

Save configurations in *Preser Ini* file.

3.1.13 readConfigFromFile()

Read configuations from *Preser Ini* file.

3.2 Robot API

With the Robot API is possible to modify the robot control loop parameters and joint target.

Control Mode	enumeration
PLANE_ANGLE	1
ANGULAR_VELOCITY	2
TORQUE	3
MOTOR_STOP	4
TWIST	5

i.e armSetCtrlModes(1,MOTOR˙STOP) is equal as armSetCtrlModes(1,5).

3.2.1 armSetCtrlModes(index, k)

Set the specified control mode for the arm service.

3.2.2 armSetPos(index,joint˙pos)

Set the control mode to PLANE˙ANGLE and set the target positions.

3.2.3 armSetVel(index,joint˙vel)

Set the control mode to ANGULAR˙VELOCITY and set the target velocities.

3.2.4 armSetTor(index,joint˙tor)

Set the control mode to TORQUE and set the target torques.

3.2.5 baseSetCtrlModes(index, k)

Set the control mode for the base service.

3.2.6 baseSetPos(index,joint˙pos)

Set the control mode to PLANE˙ANGLE and set the target positions.

3.2.7 baseSetVel(index,joint˙vel)

Set the control mode to ANGULAR˙VELOCITY and set the target velocities.

3.2.8 baseSetTor(index,joint˙tor)

Set the control mode to TORQUE and set the target torques.

3.2.9 baseSetTwist(index,twist)

Set the control mode to TWIST and set the specified twist command.

3.2.10 gripperSetCtrlModes(index, k)

Set the specified control mode for the gripper service.

3.2.11 gripperSetStat(index,isToBeOpened)

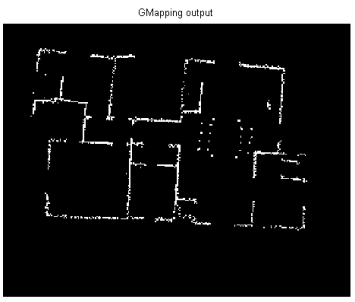
Set the specified target state for the gripper.

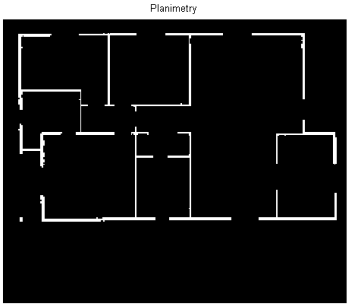
4 Algorithms

4.1 SLAM

4.1.1 GMapping

GMapping is a highly efficient Rao-Blackwellized particle filer to learn grid maps from laser range data.
Recently Rao-Blackwellized particle filters have been introduced as effective means to solve the simultaneous localization and mapping (SLAM) problem. This approach uses a particle filter in which each particle carries an individual map of the environment. Accordingly, a key question is how to reduce the number of particles. We present adaptive techniques to reduce the number of particles in a Rao- Blackwellized particle filter for learning grid maps. We propose an approach to compute an accurate proposal distribution taking into account not only the movement of the robot but also the most recent observation. This drastically decrease the uncertainty about the robot's pose in the prediction step of the filter. Furthermore, we apply an approach to selectively carry out re-sampling operations which seriously reduces the problem of particle depletion.





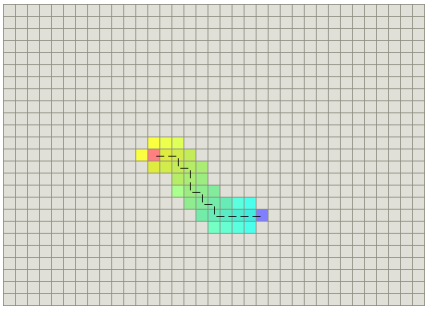
Authors: *Giorgio Grisetti; Cyrill Stachniss; Wolfram Burgard;*
link : openslam

4.2 Navigation

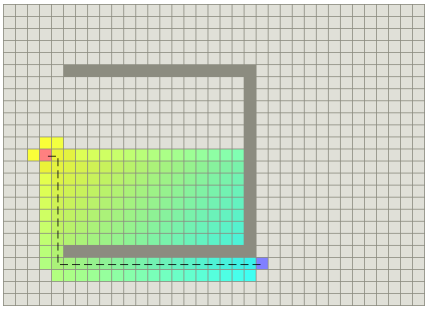
4.2.1 AStar

A* is the most popular choice for pathfinding, because its fairly flexible and can be used in a wide range of contexts.

A* is like Dijkstra's algorithm in that it can be used to find a shortest path. A* is like Greedy Best-First-Search in that it can use a heuristic to guide itself. In the simple case, it is as fast as Greedy Best-First-Search:



In the example with a concave obstacle, A* finds a path as good as what Dijkstras algorithm found:



The secret to its success is that it combines the pieces of information that Dijkstras algorithm uses (favoring vertices that are close to the starting point) and information that Greedy Best-First-Search uses (favoring vertices that are close to the goal). In the standard terminology used when talking about A*, g(n) represents the exact cost of the path from the starting point to any vertex n, and h(n) represents the heuristic estimated cost from vertex n to the goal. In the above diagrams, the yellow (h) represents vertices far from the goal and teal (g) represents vertices far from the starting point. A* balances the two as it moves from the starting point to the goal. Each time through the main loop, it examines the vertex n that has the lowest f(n) = g(n) + h(n).

4.2.2 DStar

D* is any one of the following three related incremental search algorithms:

The original D* by Anthony Stentz, is an informed incremental search algorithm.

- Focused D* is an informed incremental heuristic search algorithm by Anthony Stentz that combines ideas of A* and the original D*.
- Focused D* resulted from a further development of the original D*

All three search algorithms solve the same assumption-based path planning problems, where a robot has to navigate to given goal coordinates in unknown terrain. It makes assumptions about the unknown part of the terrain (for example: that it contains no obstacles) and finds a shortest path from its current coordinates to the goal coordinates under these assumptions. The robot then follows the path.

D* and its variants have been widely used for mobile robot and autonomous vehicle navigation. Current systems are typically based on D* Lite rather than the original D* or Focused D*. In fact, even Stentz's lab uses D* Lite rather than D* in some implementations. Such navigation systems include a prototype system tested on the Mars rovers Opportunity and Spirit and the navigation system of the winning entry in the DARPA Urban Challenge, both developed at Carnegie Mellon University.

The original D* was introduced by Anthony Stentz in 1994. The name D* comes from the term "Dynamic A*", because the algorithm behaves like A* except that the arc costs can change as the algorithm runs.

4.3 Localization

4.3.1 AMCL

AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map