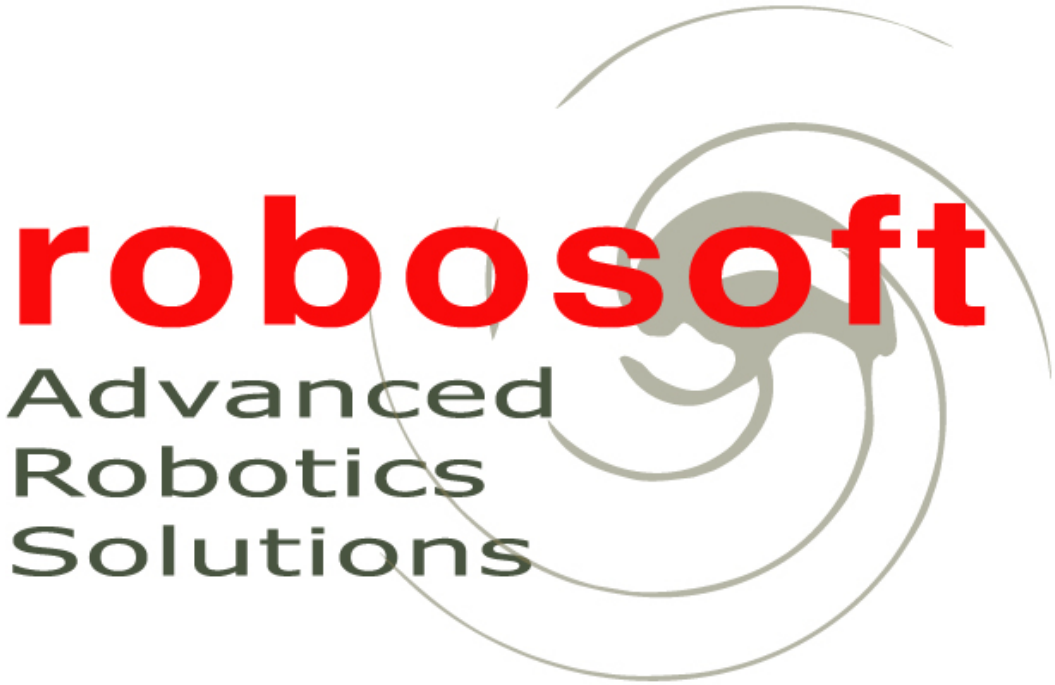


robuBOX Developer s Manual

Release 4.0

Robosoft

March 29, 2012

CONTENTS

CHAPTER

ONE

INTRODUCTION

robuBOX is an **open source** Software Development Kit (SDK) developed by Robosoft, and based on Microsoft Robotics Developer Studio (MSRDS) and developed in C#. Its primary function is to provide support for Robosoft's robots to developers working with MSRDS.

This manual describes the steps needed to get started with **robuBOX**.

Warning: We assume that the user has a basic knowledge of MSRDS and its Decentralized Software Services (DSS) component.
This is not a MSRDS documentation, for MSRDS documentation see:
http://msdn.microsoft.com/en-us/robotics/aa731517.aspx

Note: If you have a robot from robosoft, you need to update PURE before using **robuBOX** (see PURE documentation). The latest PURE version compatible with the **robuBOX** is provided with.

1.1 Who should read it?

This manual is targeted at software developers who want to write their application by interfacing directly with Robosoft middleware.

There are several cases where this might be suitable:

- you don't need real-time behaviour:

robuBOX is used to develop high level application like GUI for monitoring or hight level control management.

- you want to use a simulated environment to develop your own algorithms:

Currently, our high level applications and the software we provide are developed using Microsoft Robotics Developer Studio. It provides a simulation environment in which we add our robuLAB10 robot. Thanks to the design of our interfaces, you can control a simulated robot like a real one.

1.2 Which prerequisites do I need?

This manual supposes that the developer has its own development environment like Visual Studio.

You also need:

- Basic knowledge of C#.
- A basic knowledge of MSRDS. The code is open source and available under the LGPL (Lesser General Public License).
- A Karto licence to use the *Navigation* package.

1.3 What does it contain?

It contains:

- The description of packages containing standard interfaces and services.
- The description of our basic applications.
- Procedures to configure services to communicate with PURE.
- Procedures to build services.

It does not contain information specific to each robot, like the services available, or the kind of hardware installed. These information can be found in the robot manual, available through its technical documentation page.

1.4 How to read this manual?

It is recommended to go through chapter *General Description* and *Packages* a first time to get the basics of the architecture.

Then you can install the **robuBOX** chapter *Installation* then you can do the tutorials to learn the basics of the robuBOX.

Chapter *Compiling sources (optional)* is meant to be used when you want modify services from the **robuBOX**

CHAPTER

TWO

GENERAL DESCRIPTION

RobuBOX is composed of several components.

The most important one is the *Core* component, which contains the definitions of robots actuators and sensors, in the form of DSS abstract contracts.

Note: All the other components interact through these definitions, either by implementing or using them.

The *Algorithms* component contains robotic related functionalities, such as manual control with a gamepad and automatic path following.

The *Drivers* component contains implementations of the *Core* contracts based on hardware.

The *Pure* component contains implementations of the *Core* contracts based on Robosoft low level controller.

The *Simulations* component contains implementations of the *Core* contracts based on MSRDS simulation environment.

The *Lokarria* component gives access to robuBOX functions through HTTP requests.

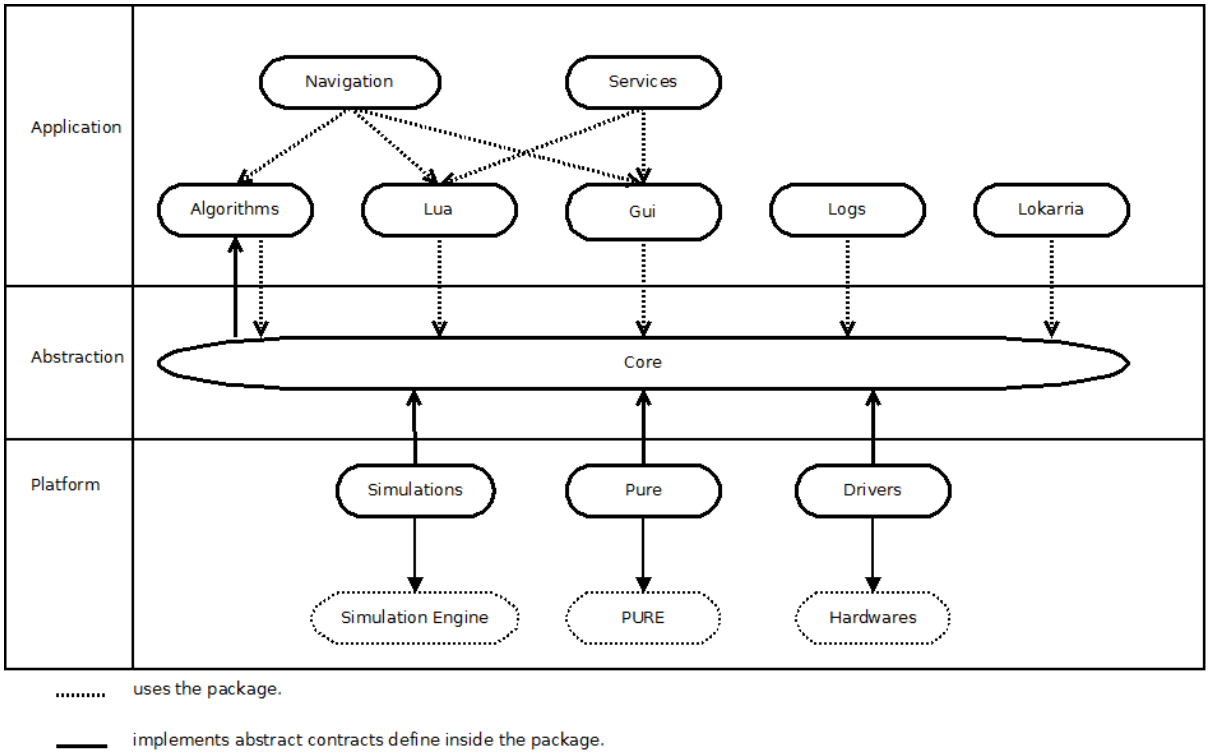
The *Lua* component gives access to robuBOX functions through Lua script language.

The *Logs* component contains loggers for MSRDS contracts.

The *GUI* component contains the graphical user interfaces used to monitor or interact with the robots.

The *Navigation* component contains a set of services that deal with Karto functionnalities (Mapper, Planner and Localizer).

The *Services* component contains services that deal only with not core component but various.



Some specific components may be added if necessary to support new robots functionalities. This can be extended with services developed for the end-user's application needs.

CHAPTER

THREE

INSTALLATION

This chapter describes how to install the robuBOX on your computer.

3.1 Prerequisites

- If you are using a robosoft robot, please update PURE version first. The latest PURE version compatible with robuBOX is provided within the same package. The procedure to update PURE is described in pure-robulab-rXXXX\doc\pure-communication-manual-3.0.pdf, chapter 5.3.1.
- If you wish to write services in C#, you will also need a development environment. It's possible to use Visual Studio C# Express edition for that purpose, which is freely available at www.microsoft.com/express/Windows.
- RobuBOX 4.0 requires MSRDS to be installed on the development computer. This can be found at www.microsoft.com/robotics.

  Note: You should install Visual Studio 2010 **before** you install MSRDS.

- MSRDS requires .NET 4.0
- RobuBOX provides some externals needed for the correct functionning of it.

Important: The *Navigation* package is based on **Karto Robotics SDK 2.1**.

If you buy a robot with a karto licence, dlls from Karto are delivery with the robot. You need to copy the following dlls from the "MSRDS\bin" directory from tablet-PC to your MSRDS\bin directory of your development computer:

- Karto_32.dll
- KartoWrapper35_32.dll
- lobiomp5md.dll
- tbb.dll

For custommers with a previous version of Karto, please contact our support (support@robosoft.com), we will provide you those dlls.

If you don't buy a robot with a Karto licence, go to http://www.kartorobotics.com/. In this case you will need to build the *Navigation* package with the new references (see *Compiling sources (optional)*).

If you don't buy the Karto licence you will not be able to run the *mapperapplication-label* and the *navigationapplication-label*.

3.2 Directory layout

The SDK elements are located in the following directories:

msrds:

bin: Contains the binaries (DLL) of robuBOX MSRDS services, and their associated XML documentation files.

store:

applications: Contains manifests and configurations files for basic robot operation.

launchers: Contains .bat files to start application provided in sample.

media: Contains media files used with the MSRDS simulation engine (Environment textures, 3D ground models, 3D robot meshes).

src: Contains the robuBOX C# source files.

doc: Contains the PDF versions of the manuals.

external: Contains some additionnal software needed for using robuBOX.

- AXISMediaControlSDK_redist.exe: necessary to display the video stream from Axis IP camera.
- Visual C++ 2008 Redistributable Package.exe
- Visual C++ 2010 Redistributable Package.exe

3.3 How to install?

- Install MSRDS on you development computer.
- Install externals.
- If a robuBOX is already installed on your computer delete **all** robuBOX dlls from the "...path\to\msrds\bin" folder. You should also delete previous manifests and configurations files installed into "...path\to\msrds\store\applications" directory. You can use the following commands to do so:

```
DEL "path\to\msrds\bin\robubox.*"
DEL /S "path\to\msrds\store\applications"
```

Warning: If you add your own manifests inside "path\to\msrds\store\applications" directory, save them before. But they certainly will no be compatible with new robuBOX.

- The content of the msrds directory can be copied in your MSRDS installation directory, as it matches its structure. You can use the following command to do so:

```
xcopy "path\to\robubox\msrds\*" "path\to\msrds" /E /Y
```

CHAPTER

FOUR

COMPILING SOURCES (OPTIONAL)

Note: This step is not needed unless you want to modify the robuBOX itself. All the binaries that result from the build step are available in the *.\nsrds\bin* directory.

4.1 Principles

The source tree layout is based on the components listed in the general description. Each component can be (re)built separately, except the *Navigation* and *Services* components.

So the principle is to build first the *Core* component, and then the other one in any order (except for the *Navigation* and *Services* components taht must be built at the end).

Each component has a corresponding solution file (extension .sln) with the same name.

4.2 Step 1: Environment setup

The project files (extension .csproj) assume that the MSRDS installation is in the directory **C:\MSRDS**.

If that's the case, you should be able to go to the build step. If not, you can still go to the command line build. You can also use the DssProjectMigration tool as described in the next step.

An alternative solution is to create a link between the original installation directory, with a tool like *Junction* (http://technet.microsoft.com/en-us/sysinternals/bb896768.aspx).

4.3 Step 2: Project files setup

The project files are for **Visual Studio 2010**.

In case you don't have installed your MSRDS under **C:MSRDS** directory, you will need to run the **DssProjectMigration** tool provided with MSRDS.

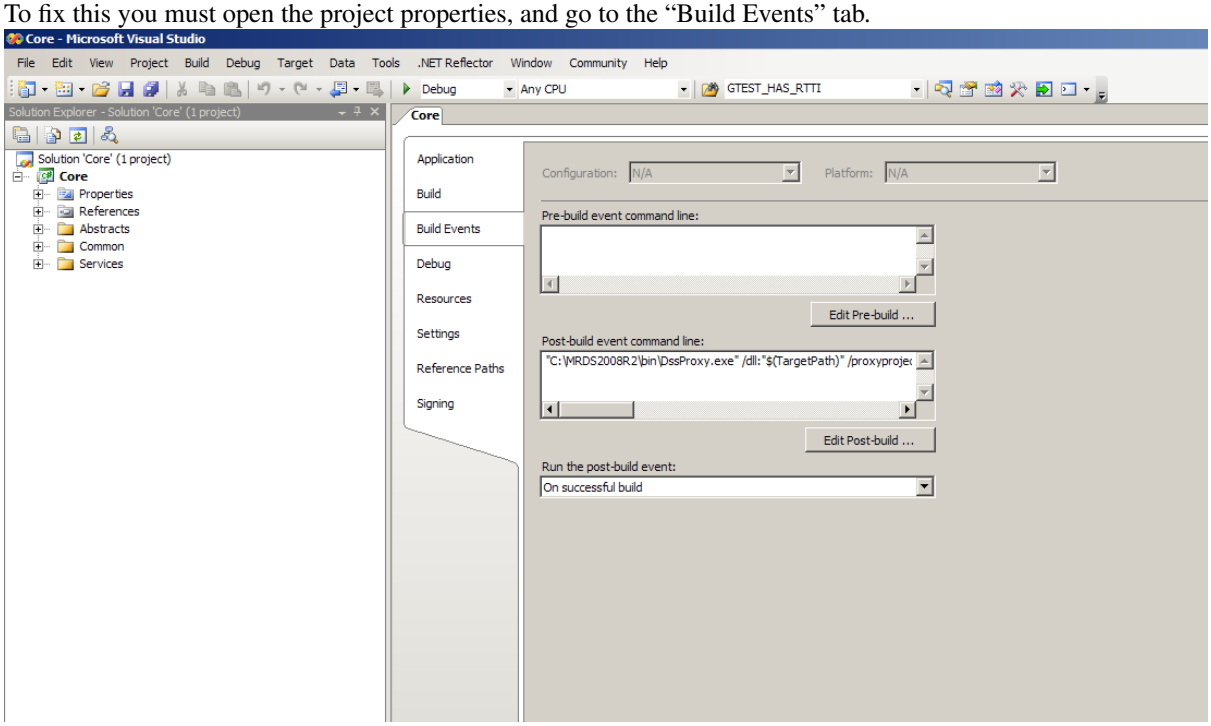
To do so, open a DSS Command Prompt (Start -> Programs -> Microsoft Robotics Developer Studio -> DSS Command prompt).

Type the following command:

```
dssprojectmigration path\to\src
```

> **Warning:** However, it will cause a problem with the **srcCoreCore.csproj** file when compiling the **Core** component and the **srcNavigationMapManagerMapManager.csproj** file when compiling the **Navigation** component. Those projects use a proxy extension file. This requires additional arguments in the Post-Build event, where the proxy is generated.

To fix this you must open the project properties, and go to the "Build Events" tab.



You will need to add the following options at the end of the "Post-Build event command line":

```
/partialclass+ /e:"$(ProjectDir)Extensions"
```

4.4 Step 3A: Command line build

It is possible to build the sources without Visual Studio, using MSBuild. This is the Microsoft build engine, and it is provided with the standard .NET3.5 Framework redistributable. Actually, it is also used by Visual Studio behind the scene.

To do so, open a command prompt, and type the following command:

```
cd %SYSTEMROOT%\Microsoft.NET\Framework\v3.5

msbuild path\to\src\Core\Core.sln
```

This will build the *Core* component.

If your MSRDS installation is somewhere else than C:MSRDS, you can override the options that depends on this, e.g. the reference path and the output path:

```
msbuild path\to\src\Core\Core.sln
/property:OutputPath=path/to/msrds/bin;ReferencePath=path/to/msrds/bin
```

Once you've built the *core-label* component, you can proceed with the others, in the order you like (except for Lua component that must be built before the Gui component):

```
msbuild path\to\src\Drivers\Drivers.sln
msbuild path\to\src\Pure\Pure.sln
msbuild path\to\src\Simulations\Simulations.sln
msbuild path\to\src\Algorithms\Algorithms.sln
msbuild path\to\src\Lua\Lua.sln
msbuild path\to\src\Lokarria\Lokarria.sln
msbuild path\to\src\Logs\Logs.sln
msbuild path\to\src\Gui\Gui.sln
msbuild path\to\src\Navigation\Navigation.sln
msbuild path\to\src\Services\Services.sln
```

4.5 Step 3B: Visual Studio build

As already mentioned, you must build first the Core component, since the other have dependencies on it.

- Double click on path\to\src\Core\Core.sln
- Go to the Build menu.
- Click on Build Solution
- You're done!

You can then proceed with the other solutions, as needed:

- Drivers\Drivers.sln
- Pure\Pure.sln
- Simulations\Simulations.sln
- Algorithms\Algorithms.sln
- Lua\Lua.sln
- Lokarria\Lokarria.sln
- Logs\Logs.sln
- Gui\Gui.sln
- NavigationNavigation.sln
- ServicesServices.sln

CHAPTER

FIVE

PACKAGES

This section describes each packages from robuBOX.

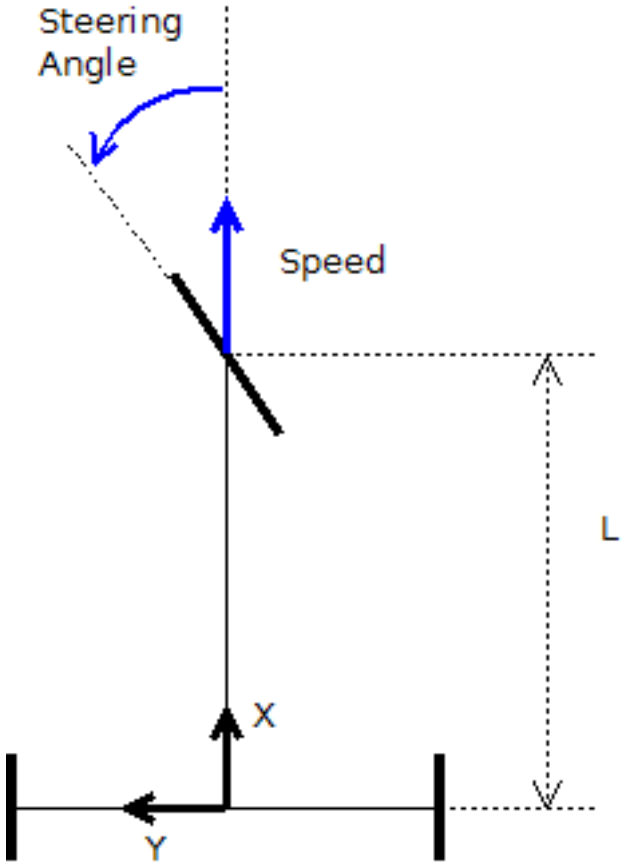
5.1 Core

This component contains the definitions of robots actuators and sensors, in the form of DSS abstract contracts.

Important: All the other components interact through these definitions, either by implementing or using them.

5.1.1 CarDrive

This abstract contract describes a car-like robot as following:



Car-like robot

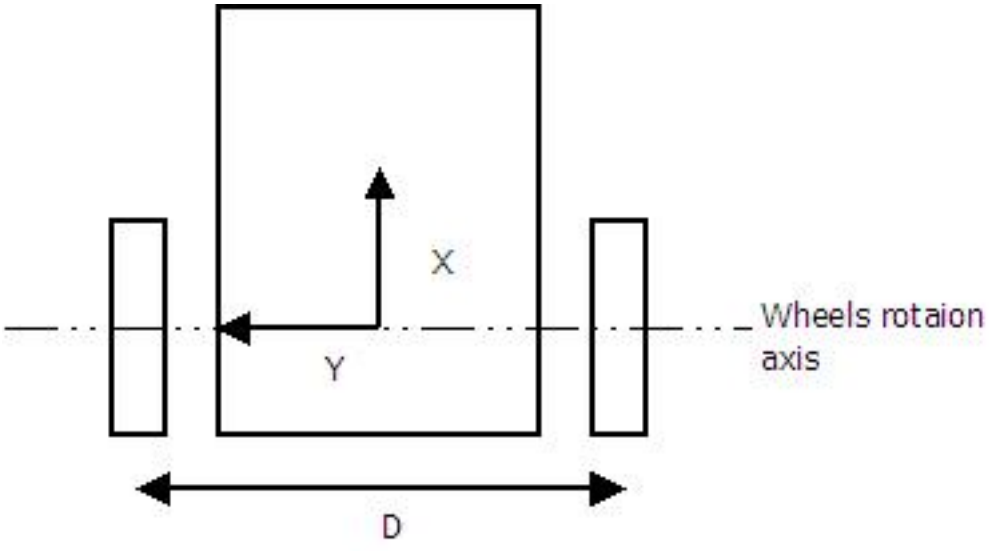
Let V be the vehicle speed, ϕ the vehicle steering, L the length between front and rear axle, T the sampling period and x, y, θ the coordinates of vehicle in global frame.

The robot coordinates in global frame are computed the following way:

$$x_{k+1} = x_k + V * T * \cos(\theta_k)$$
$$y_{k+1} = y_k + V * T * \sin(\theta_k)$$
$$\theta_{k+1} = \theta_k + \frac{V * T}{L} * \tan(\phi)$$

5.1.2 DifferentialDrive

This abstract contract describes a differential-like robot as following:



Differential robot

The robot physically achieves a linear speed by adding the velocities of its two wheels, and an angular speed by subtracting these velocities.

The linear and angular speeds apply to the center of the segment defined by the rotation axis and its intersections with the wheels.

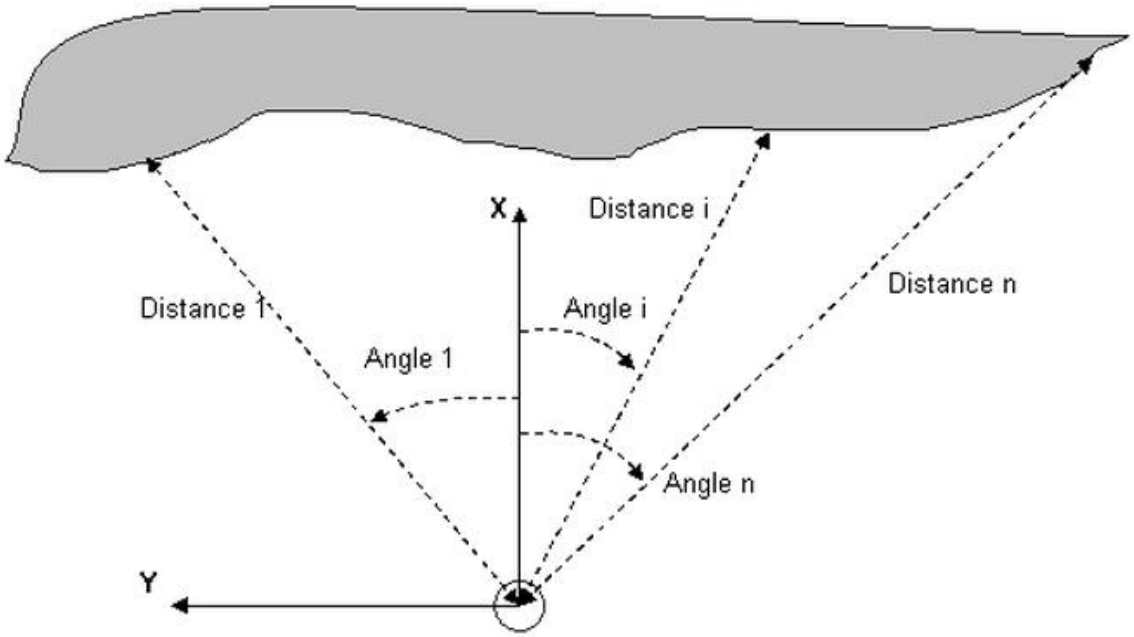
Let V_l be the vehicle linear speed, V_a the vehicle angular speed, D the width between rear wheels, T the sampling period and x, y, θ the coordinates of vehicle in global frame.

The robot coordinates in global frame are computed the following way:

$$x_{k+1} = x_k + V_l * T * \cos(\theta_k)$$
$$y_{k+1} = y_k + V_l * T * \sin(\theta_k)$$
$$\theta_{k+1} = \theta_k + V_a * T$$

5.1.3 Laser

A two-dimensional laser range finder. Each sample is a collection of range measurements in a given plane, but at different angle. This is illustrated on the figure below.



Description of a laser

The measurements are computed from the time of flight of the laser beam. Some of these devices can also determine if there is a reflector, based on a reflectivity measure.

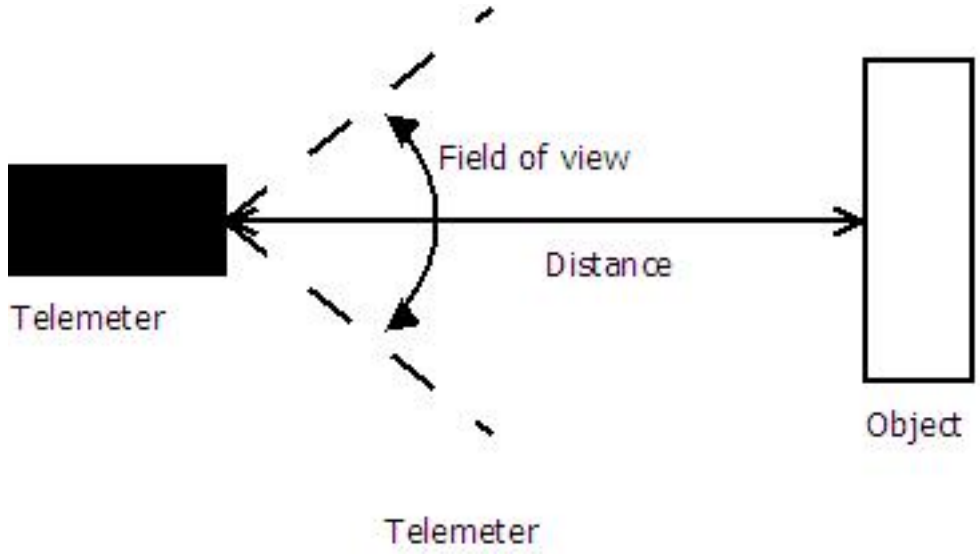
Note: The angles are measured counter clock wise, and the zero is the X axis.

5.1.4 Logger

Generic interface common to all loggers.

5.1.5 Telemeter

A telemeter is a hardware that gives a one dimensional measure of distance (ultrasound sensor, infrared sensor, laser sensor...).



5.1.6 Battery

Robosoft's battery representation.

5.1.7 Drive

Motor drive interface.

5.1.8 GPS

GPS interface.

5.1.9 Step

Control interface to manage robot's motions by steps (ex: request to move X meters forward or to rotate Y radians).

5.1.10 Inclinometer

An inclinometer is a hardware for measuring angles of an object with respect to gravity. This is a 2D inclinometer.

5.1.11 IOCard

Hardware that group analogs and digitals inputs/outputs.

5.1.12 Localization

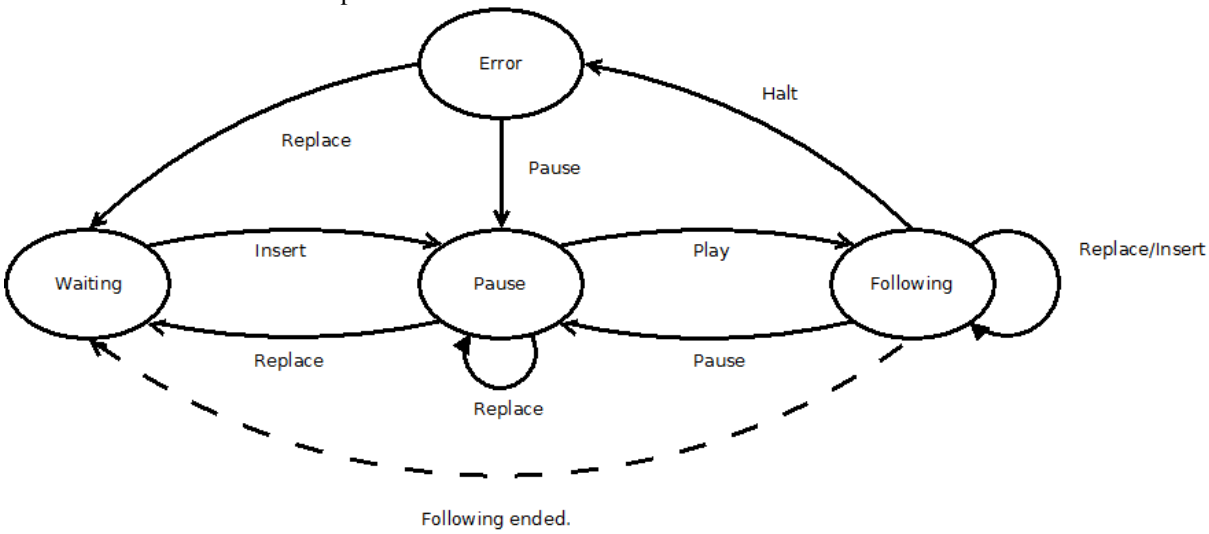
3D localization of robot. Orientation is represented by a Quaternion.

5.1.13 Diagnostic

Represents specific robot hardware status or errors (like drive, laser or iocard status etc...).

5.1.14 TrajectoryFollower

Abstarct contract to make a robot to follow a trajectory. A trajectory is a list of segments referenced inside world frame associated to maximum speed allowed on them.



A trajectory is a buffer of points (representing a 2D point with the maximum speed allowed to robot at this point) that robot must reach.

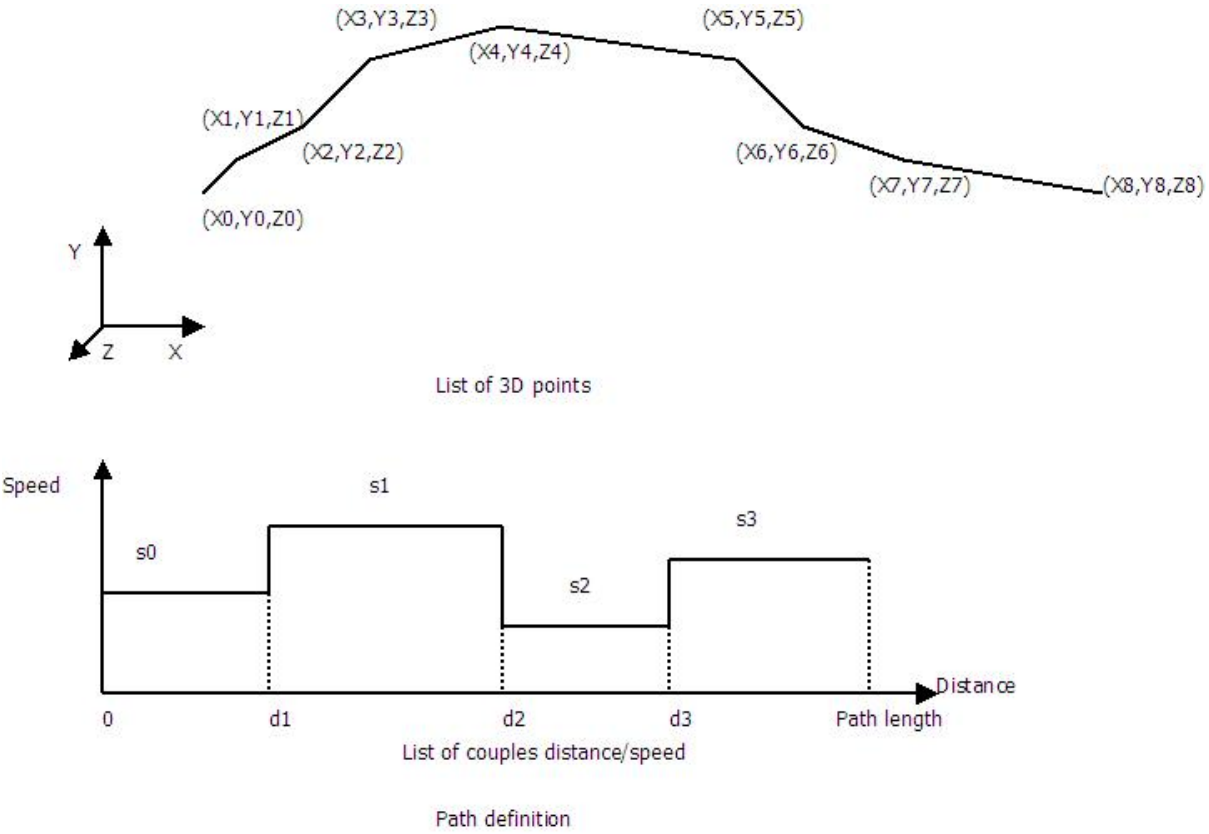
When the service starts, the status is *Waiting*. To fill the buffer use the **Insert** message. If you send various **Insert** messages, new points will be added at the end of the buffer.

Use the **Replace** message to replace current buffer by a new one. You can clear the buffer using the **Replace** message without specifying new buffer.

5.1.15 PathManager

This service is used to store paths into an .xml file (configuration file of the service) and share them with other services.

A path is defined by a list of points referenced inside global frame. To this list of points is added a list of SpeedSteps. A SpeedStep is made of a **distance** and a **speed**. The **distance** corresponds to the distance from the beginning of the trajectory after which the **speed** will be applied to motor drives.



5.1.16 TextToSpeech

Use it to make your application speek.

5.1.17 SpeechRecognition

Speech recognition service.

5.2 Algorithms

This component contains some services that use sensor data for control or computation.

5.2.1 JoystickCarControl

Use a joystick and send *CarDrive* commands. The joystick must be plugged on the machine before launching the service, and must be detected by windows. A button on the joystick (the number 0) is used as a dead man switch, i.e it must be pressed to validate commands while driving robot.

5.2.2 JoystickDifferentialDrive

Use a joystick and send *DifferentialDrive* commands. The joystick must be plugged on the machine before launching the services. A button on the joystick (the number 0) is used as a dead man switch, i.e it must be pressed to validate commands while driving robot.

5.2.3 CarDriveTrajectoryFollower

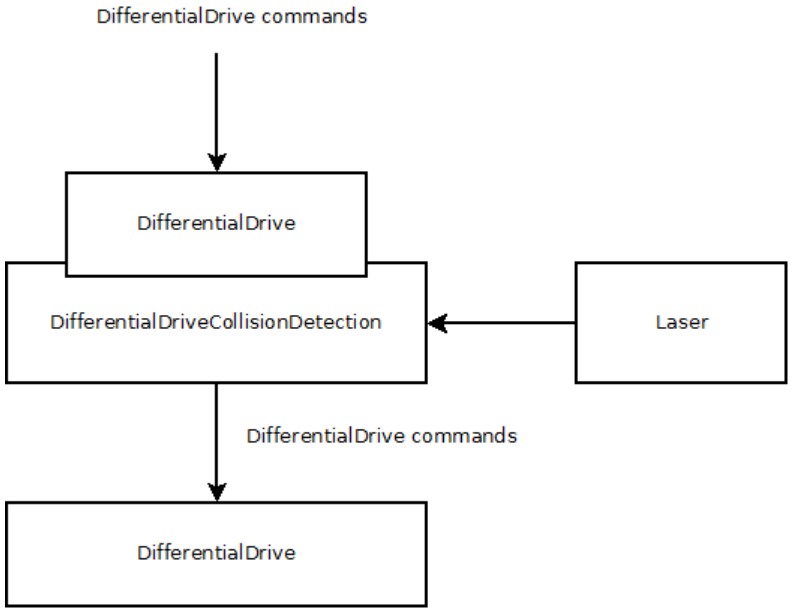
This service implements the *TrajectoryFollower* absctract contract. It make a car like robot following a trajectory using morin-samson law [1] [2].

5.2.4 DifferentialDriveTrajectoryFollower

This service implements the *TrajectoryFollower* absctract contract. It make a differential like robot following a trajectory using morin-samson law [1] [2].

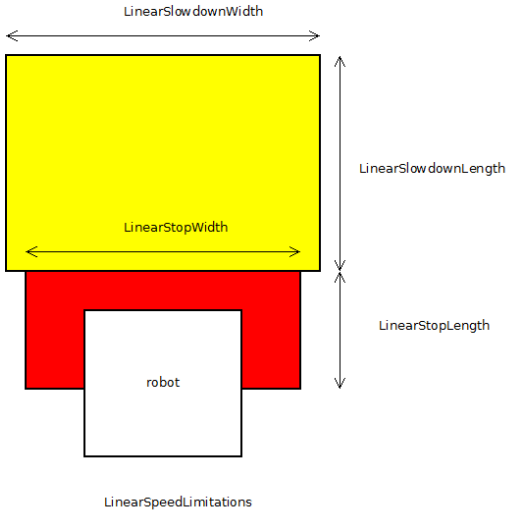
5.2.5 DifferentialRobotCollisionDetection

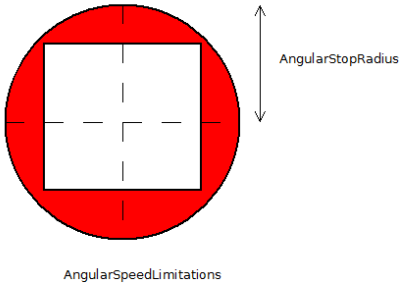
This service is a filter on *DifferentialDrive* commands. It uses the laser data to detect objects inside warning or stop areas of robot. Those areas can be configured using the configuration file of the service.



[1] Motion control of wheeled mobile robots, Pascal Morin and Claude Samson, INRIA, 2004, Route des Lucioles 06902 Sophia-Antipolis Cedex, France, Firstname.Lastname@inria.fr, August 24, 2007

[2] eq: 34.15





5.2.6 DockingStation

This service manages the docking of the robot.

Drive the robot about 0.5 meter in front of the docking station to request a dock action.

5.2.7 StepDifferentialControl

This service implements the *Step* absctract contract and send commands to a *DifferentialDrive*.

5.3 Drivers

This component contains various hardware drivers.

5.3.1 IAMCameraController

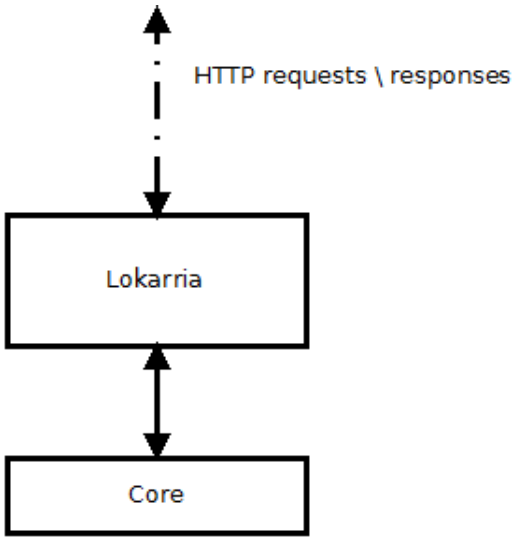
This service manages the pan/tilt control of the "QuickCam Orbit/Sphere AF" usb camera, and more generally pan/tilt camera implementing IAMCameraControl interface from directshow library.

It expose the **Core** abstract contract **Drive**. First axle (0) is the pan one. Second axle (1) is tilt.

Only position control is accepted.

5.4 Lokarria

This package gives access to robuBOX functions through HTTP requests. Each service from this package interfaces one **Core** contract.



Note: URL given are specified into default manifest used to launch this package.

5.4.1 Battery

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/battery
method	GET
content type	application/json

Response

```
{
    "Properties":
    {
        "Critical":10,
        "Power":20,
        "Voltage":24
    },
    "Remaining":66,
    "Status":2,
    "Timestamp":14636000
}
```

5.4.2 CarDrive

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/cardrive
method	GET
content type	application/json

Response


```
{
    "Command":
    {
        "TargetSpeed":0,
        "TargetSteer":0,
        "Enabled":true
    },
    "Feedback":
    {
        "CurrentSpeed":0,
        "CurrentSteer":0,
        "Enabled":true
    },
    "Properties":
    {
        "MaxSpeed":3,
        "MinSpeed":-2,
        "MaxSteer":0.5,
        "MinSteer":-0.5,
        "MaxAcceleration":2,
        "MaxDeceleration":-2,
        "Length":2
    },
    "Timestamp":14703680
}
```


5.4.3 Console

This service stores console data into a ".txt" file. From Lokarria you can download files saved by this service, once downloaded the file is deleted.

Request

Send it to retrive the name of files that can be downloaded.

URL	/lokarria/console/files
method	GET
content type	application/json

Response


```
[
    "Console_2011-9-7_11h45min15s.txt",
    "Console_2011-9-7_11h53min18s.txt",
    "Console_2011-9-7_11h59min19s.txt"
]
```


Request

Send it to download the specified file.

URL	/lokarria/console/filename
method	GET
content type	application/txt

Response


```
{
    "Category":"Console:http:\/\/schemas.microsoft.com\/xw\/2005\/12\/dss.html",
    "CodeSite":"Void Start()() at line:0, file",
    "Detail":null,
    "Level":"Info:http:\/\/schemas.microsoft.com\/xw\/2004\/10\/consoleoutput.html",
    "Source":"http:\/\/pc-arnaud:50000\/constructor",
    "Subject":"Service started",
    "Time":"\/Date(1315388697934+0200)\/"
}

{
    "Category":"Activation:http:\/\/schemas.microsoft.com\/xw\/2005\/12\/dss.html",
    "CodeSite":"Boolean MoveNext()() at line:0, file",
    "Detail":null,
    "Level":"Info:http:\/\/schemas.microsoft.com\/xw\/2004\/10\/consoleoutput.html",
    "Source":"http:\/\/pc-arnaud:50000\/manifestloaderclient",
    "Subject":"Starting manifest load: file:\/\/\/C:\/MRDS\/samples\/config\/apartment.simulation
    "Time":"\/Date(1315388698043+0200)\/"
}

...

{
    "Category":"StdOut:http:\/\/www.robubox.com\/simulations\/models\/laser.html",
    "CodeSite":"Boolean MoveNext()() at line:155, filef:\\svn\\robubox\\RobuBOX\\trunk\\Simulatio
    "Detail":null,
    "Level":"Info:http:\/\/schemas.microsoft.com\/xw\/2004\/10\/consoleoutput.html",
    "Source":"http:\/\/pc-arnaud:50000\/simulation\/laser",
    "Subject":"Parent entity not found in partner list. Trying SimulatedRobot partner.",
    "Time":"\/Date(1315388701819+0200)\/"
}
```


5.4.4 Diagnostic

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/diagnostic
method	GET
content type	application/json

Response

5.4.5 DifferentialDrive

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/differentialdrive
method	GET
content type	application/json

Response


```
{
    "Command":
    {
        "TargetAngularSpeed":0,
        "TargetLinearSpeed":0
    },
    "Feedback":
    {
        "CurrentAngularSpeed":2.3046566639095545E-05,
        "CurrentLinearSpeed":4.1944749682443216E-06,
        "Enabled":true
    },
    "Properties":
    {
        "MaxAngularAcceleration":6,
        "MaxAngularDeceleration":-6,
        "MaxAngularSpeed":1.5,
        "MaxLinearAcceleration":2,
        "MaxLinearDeceleration":-2,
        "MaxLinearSpeed":1,
        "MinAngularSpeed":-1.5,
        "MinLinearSpeed":-1,
        "Width":0.36399999260902405
    },
    "Timestamp":14703680
}
```


Request

Send a **Drive** request.

URL	/lokarria/differentialdrive
method	POST
content type	application/json

Content


```
{
    "TargetAngularSpeed":0,
    "TargetLinearSpeed":0
}
```


Response

5.4.6 Drive

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/drive
method	GET
content type	application/json

Response


```
{
    "Commands":
    [
        {
            "Enable":true,
            "Mode":0,
            "Target":0
        },
        {
            "Enable":true,
            "Mode":0,
            "Target":0
        }
    ],
    "Feedbacks":
    [
        {
            "Enabled":true,
            "Mode":0,
            "Position":0,
            "Speed":0,
            "Target":0,
            "Torque":0
        },
        {
            "Enabled":true,
            "Mode":0,
            "Position":0,
            "Speed":0,
            "Target":0,
            "Torque":0
        }
    ],
    "Properties":
    [
        {
            "MaxAcceleration":1,
            "MaxPosition":1.2217304763960306,
            "MaxSpeed":1,
            "MaxTorque":1,
            "MinPosition":-1.2217304763960306,
            "MinSpeed":-1,
            "MinTorque":-1,
            "Type":0
        },
        {
            "MaxAcceleration":1,
            "MaxPosition":0.52359877559829882,
            "MaxSpeed":1,
            "MaxTorque":1,
            "MinPosition":-0.52359877559829882,
            "MinSpeed":-1,
            "MinTorque":-1,
            "Type":0
        }
    ],
    "Timestamp":0
}
```

Request

Send a **Drive** request.

URL	/lokarria/drive
method	POST
content type	application/json

Content

```
[
    "Command":
    {
        "Enable":true,
        "Mode":0,
        "Target":0
    },
    "Command":
    {
        "Enable":true,
        "Mode":0,
        "Target":0
    }
]
```

Response

5.4.7 Inclinometer

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/inclinometer
method	GET
content type	application/json

Response

```
{
    "PitchAngle":0.1,
    "RollAngle":-0.4,
    "Timestamp":153876
}
```

5.4.8 IOCard

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/iocard
method	GET
content type	application/json

Response

5.4.9 Laser

Request

Send it to retrive the properties of laser.

URL	/lokarria/laser/properties
method	GET
content type	application/json

Response


```
{
     "AngleIncrement": 0.017388889226511769,
     "EndAngle":2.3561944901923448,
     "Pose":
     {
          "Orientation":
          {
               "W":1,
               "X":0,
               "Y":0,
               "Z":0
          },
          "Position":
          {
               "X":0.15,
               "Y":0,
               "Z":0.2
          }
     },
     "StartAngle":-2.3561944901923448
 }
```


Request

Send it to retrive the properties of laser.

URL	/lokarria/laser/echoes
method	GET
content type	application/json

Response


```
{
     "Echoes":
     [
          0.884,0.9,0.916,0.932,0.95,0.969,0.989,1.01,1.032,1.056,1.064,1.06,1.047,
          1.035,1.024,1.013,1.002,0.992,0.983,0.974,0.965,0.957,0.95,0.942,0.936,
          0.929,0.923,0.917,0.912,0.907,0.902,0.898,0.893,0.89,0.886,0.883,0.88,
          0.877,0.875,0.873,0.871,0.87,0.869,0.868,0.867,0.866,0.866,0.866,0.867,
          0.867,0.868,0.869,0.871,0.872,0.874,0.876,0.879,0.882,0.885,0.888,0.892,
          0.896,0.9,0.905,0.91,0.915,0.921,0.927,0.933,0.94,0.947,0.954,0.962,0.97,
          0.979,0.988,0.998,1.008,1.019,1.03,1.042,1.055,1.068,1.082,1.096,1.111,
          1.127,1.144,1.162,1.18,1.2,1.221,1.242,1.265,1.289,1.315,1.342,1.37,1.401,
          1.433,1.466,1.503,1.541,1.582,1.625,1.672,1.722,1.775,1.833,1.895,1.962,
```

```
        2.034,2.113,2.198,2.28,2.279,2.264,2.25,2.236,2.224,2.212,2.201,2.191,2.182,
        2.174,2.166,2.159,2.152,2.146,2.141,2.137,2.133,2.13,2.128,2.126,2.16,2.159,
        2.159,4.016,4.018,4.022,4.028,4.034,4.042,4.051,4.061,4.073,4.086,4.1,4.116,
        4.133,4.151,4.171,4.192,4.215,4.24,4.266,4.278,2.358,2.342,2.36,2.346,2.36,
        2.38,2.402,2.424,2.448,2.473,2.5,2.527,2.556,2.587,2.619,2.653,2.688,2.726,
        2.765,2.806,2.85,2.895,2.943,2.994,3.047,3.103,3.162,3.225,3.291,3.361,3.435,
        3.513,3.597,3.685,3.779,3.879,3.986,4.101,4.156,4.115,4.076,4.039,4.004,3.97,
        3.938,3.908,3.88,3.853,3.828,3.804,3.781,3.76,3.741,3.722,3.705,3.69,3.675,
        3.662,3.65,3.639,3.629,3.621,3.613,3.607,3.602,3.598,3.595,3.593,3.592,3.592,
        3.594,3.596,3.6,3.605,3.611,3.618,3.626,3.635,3.645,3.408,3.122,2.882,2.677,
        2.499,2.345,2.209,2.089,1.981,1.885,1.798,1.719,1.647,1.582,1.522,1.467,1.416,
        1.369,1.325,1.285,1.247,1.211,1.178,1.148,1.119,1.091,1.066,1.042,1.019,0.997,
        0.977,0.958,0.94,0.922
    ],
    "Overflow":
    [
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ],
    "Reflector":
    [
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ],
    "Timestamp":147070
}
```


5.4.10 Localization

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/localization
method	GET
content type	application/json

Response


```
{
    "Pose":
    {
        "Orientation":
        {
            "W":-0.70752808315921567,
            "X":0,
            "Y":0,
            "Z":0.70668522804785294
        },
        "Position":
```

```
        {
            "X":0.062024351309485075,
            "Y":-4.8787359764368405,
            "Z":0
        }
    },
    "Status":0,
    "Timestamp":75949220
}
```

5.4.11 Step

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/step
method	GET
content type	application/json

Response

```
{
    "Command":
    {
        "Distance":1,
        "MaxSpeed":1,
        "MotionType":"Translation"
    },
    "Feedback":
    {
        "DistanceTravelled":1.23,
        "Status":"Moving"
    },
    "Properties":
    {
        "KpGain" = 1,
        "RotationErrorThreshold" = 0.01,
        "TranslationErrorThreshold" = 0.02
    }
}
```

Request

Request a translation (Distance in m and MaxSpeed in m/s).

URL	/lokarria/step/translate
method	POST
content type	application/json

Content

```
{
    "Distance": 1,
    "MaxSpeed": 1,
}
```

Response

HTTP status 204.

Request

Request a relative rotation (Distance in rad and MaximumSpeed in rad/s).

URL	/lokarria/step/relative
method	POST
content type	application/json

Content

```
{
    "Distance": 1,
    "MaxSpeed": 1,
}
```

Response

HTTP status 204.

Request

Request an absolute rotation (Distance in rad and MaximumSpeed in rad/s).

URL	/lokarria/step/absolute
method	POST
content type	application/json

Content

```
{
    "Distance": 1,
    "MaxSpeed": 1,
}
```

Response

HTTP status 204.

Request

Request robot stops its motion.

URL	/lokarria/step/stop
method	POST
content type	application/json

Content

No content.

Response

HTTP status 204.

5.4.12 Telemeter

Request

Send it to retrive the properties of each telemeter.

URL	/lokarria/telemeter/properties
method	GET
content type	application/json

Response

```
[
    "Properties":
    {
        "FieldOfView": 0.3,
        "MaxDistance": 2,
        "MinDistance":0.2,
        "Pose":
        {
            "Orientation":
            {
                "W":-0.7,
                "X":0,
                "Y":0,
                "Z":0.2
            },
            "Position":
            {
                "X":0.2,
                "Y":-0.4,
                "Z":0
            }
        }
    }
    ...
    "Properties":
    {
        "FieldOfView": 0.3,
        "MaxDistance": 2,
        "MinDistance":0.2,
        "Pose":
        {
            "Orientation":
            {
                "W":-0.7,
                "X":0,
                "Y":0,
                "Z":0.2
            },
            "Position":
            {
                "X":0.2,
                "Y":0.4,
                "Z":0
            }
```

```
        }
    }
]
```

Request

Send it to retrive the distance measured by telemeter

URL	/lokarria/telemeter/echoes
method	GET
content type	application/json

Response

```
{
     "Echoes":
     [
         0.8,
         1.5467,
         ...
         0.564
     ]
     "Timestamp":75949220
 }
```

5.4.13 TrajectoryFollower

Request

Send it to retrive the state of the **Core** contract interfaced.

URL	/lokarria/trajectoryfollower
method	GET
content type	application/json

Response

```
{
    "DistanceCovered":0,
    "HeadingError":
    {
        "W":1,
        "X":0,
        "Y":0,
        "Z":0
    },
    "LateralError"
    {
        "X":0,
        "Y":0.1,
        "Z":0
    }
    "Status":Waiting,
    "Timestamp":14585300
}
```

Request

Send an **Insert** message.

URL	/lokarria/trajectoryfollower/insert
method	POST
content type	application/json

Content

```
[
    {
        "MaxSpeed":0.5,
        "Point":
        {
            "X":-0.018430978059768677,
            "Y":0.014290516264736652,
            "Z":0
        }
    },
    {
        "MaxSpeed":0.5,
        "Point":
        {
            "X":0.030706623569130898,
            "Y":0.023533949628472328,
            "Z":0
        }
    },
    ....
]
```

Response

HTTP status 204.

Request

Send an **Replace** message.

URL	/lokarria/trajectoryfollower/replace
method	POST
content type	application/json

Content

If you want to reset current path, no content is needed.

If you want to replace current path followed, fill the content with a trajectory as following.

```
[
    {
        "MaxSpeed":0.5,
        "Point":
        {
            "X":-0.018430978059768677,
            "Y":0.014290516264736652,
            "Z":0
        }
```

```
    },
    {
        "MaxSpeed":0.5,
        "Point":
        {
            "X":0.030706623569130898,
            "Y":0.023533949628472328,
            "Z":0
        }
    },
    ....
]
```

Response

HTTP status 204.

Request

Send an **Play** message.

URL	/lokarria/trajectoryfollower/play
method	POST
content type	application/json

Content

No content.

Response

HTTP status 204.

Request

Send an **Pause** message.

URL	/lokarria/trajectoryfollower/pause
method	POST
content type	application/json

Content

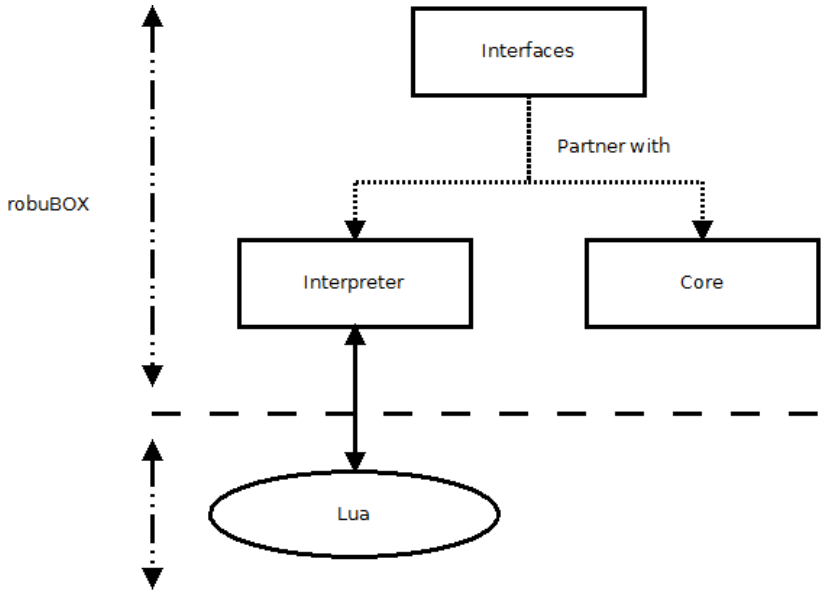
No content.

Response

HTTP status 204.

5.5 Lua

This package gives access to robuBOX functions through Lua script language. It is made of an **Interpreter** for Lua scripts, and **Interfaces** that register functions into **Interpreter** and links those functions with MRDS.



5.5.1 Interpreter

This services is an interpreter of LUA language, the clients register their functions into it. It is used to run and stop scrips.

This service introduces 4 default functions that can be called inside a script:

- **Wait("time")**: This function wait specified time before going to the next line, the time is in milliseconds. This function only delay the time to the script to execute the next line.
- **Log("string")**: Log into the MRDS console the specified string.
- **ScriptEnded()**: This function should be called to notify that the script ended normally to MRDS. This is generally the last function to call.
- **ScriptStopped()**: If it exists, this function is called when the script is stopped manually using graphic interface. This function must implement the special shutdown logic.

The interpreter can handled HTTP GET and POST requests, responses are under JSOn format.

Request

Send it to retrive the state of the service.

URL	/lua/interpreter
method	GET
content type	application/json

Response

```
{
    "AutoStart":false,
    "Directory":"C:\\MRDS\\store\\scripts",
    "Functions":
        [
```

```
            {
                "Key":"ScriptEnded",
                "Value":"dssp.tcp:\/\/pc-arnaud:50001\/kompai\/domeo\/lua\/interpreter"
            },
            {
                "Key":"ScriptStopped",
                "Value":"dssp.tcp:\/\/pc-arnaud:50001\/kompai\/domeo\/lua\/interpreter"
            },
            {
                "Key":"Wait",
                "Value":"dssp.tcp:\/\/pc-arnaud:50001\/kompai\/domeo\/lua\/interpreter"
            },
            {
                "Key":"Log",
                "Value":"dssp.tcp:\/\/pc-arnaud:50001\/kompai\/domeo\/lua\/interpreter"
            },
            {
                "Key":"tts.Say",
                "Value":"dssp.tcp:\/\/pc-arnaud:50001\/kompai\/domeo\/lua\/tts"
            }
        ],
    "Path":"",
    "Running":false
}
```


Request

Returns the scripts name located into directory specified in state.

URL	/lua/interpreter/scripts
method	GET
content type	application/json

Response


```
[
    "Script1.Lua",
    "Script2.Lua",
    "Script3.Lua"
]
```


Request

Returns if a script is currently running or not.

URL	/lua/interpreter/running
method	GET
content type	application/json

Response


```
{
    "Path":"",
    "Running":false
}
```

Request

Run a script specifying the name.

URL	/lua/interpreter/run
method	POST
content type	application/json

Content

```
"script.lua"
```

Response

HTTP status 204.

Request

Stop the current script.

URL	/lua/interpreter/stop
method	POST
content type	application/json

Content

No content.

Response

HTTP status 204.

5.5.2 Interfaces

Interfaces are services that maps Lua with robuBOX functionalities. They register Lua functions into the **Interpreter**. Each service interfaces one MRDS contract.

Because it is possible to start various instance of a same functionality inside the same **MRDS** node, each interfaces must have a specific name into Lua. To retrive the name given to the specific instance you should proceed as following:

- retrieve the url of the **MRDS** partner service
- split this url using '/' as separator
- keep the last but one word

Example:

we have two instances of IOCard service implementation on your **MRDS** node which urls are:

- http://localhost:50000/iocard1/iocard
- http://localhost:50000/iocard2/iocard

Two Lua interfaces services are needed, one per instance of IOCard. When a function relative to iocard1 must be called, you need to target the service that interface iocard1.

Example:

iocard1.MyFunction();

Battery

- **name.GetStatus()**: returns the current battery status.

```
batteryStatus = battery1.GetStatus();
```

- **name.GetLevel()**: returns the current battery level in percent.

```
batteryLevel = battery1.GetLevel();
```

IOCard

- **name.WriteDigitalOutput(id, "value")**: set the value of the specified digital output.
  - **id**: identifier of the digital output.
  - **value**: new value off the digital output(true or false).

```
iocard1.WriteDigitalOutput(0, true);
```

- **name.WriteAnalogOutput(id, value)**: set the value of the specified analog output.
  - **id**: identifier of the analog output.
  - **value**: value in volts to apply (minimum and maximum value depends on hardware).

```
iocard1.WriteAnalogOutput(3, 5);
```

- **name.ReadDigitalInput(id)**: returns the value of the specific digital input (true or false).
  - **id**: identifier of the digital input.

```
value = iocard1.ReadDigitalInput(2);
```

- **name.ReadAnalogInput(id)**: Return the value in volts (between minimum and maximum depends on hardware) of the specific analog input.
  - **id**: identifier of the analog input.

```
value = iocard1.ReadAnalogInput(3);
```

- **name.SubscribeDigitalInput(id, "function")**: subscribe to the modification of the value of a specific digital input and call the specified function.
  - **id**: identifier of the digital input.
  - **"function"**: LUA functio to call when the value of the input changes.

```
iocard1.SubscribeDigitalInput(1, "myLuaFunction");
```

- **name.UnSubscribeDigitalInput(id)**: unsubscribe to the digital input. Stop notification reception.
  - **id**: identifier of the digital input.

```
iocard1.UnSubscribeDigitalInput(1);
```

TrajectoryFollower

This service interface the **TrajectoryFollower** and the **PathManager** contracts.

- **name.GetStatus()**: returns the current trajectory follower status (Waiting, Paused, Following, Error).

```
status = follower.GetStatus();
```

- **name.GetDistanceCovered()**: returns the distance covered since begining or since last Reset-DistanceCovered request.

```
status = follower.GetDistanceCovered();
```

- **name.Insert(pathName)**: add the path specified into the buffer of points to be followed.
  - **pathName**: Name of the path to insert. This path must be stored into the PathManager.

```
follower.Insert(path1);
```

- **name.Play()**: Once a path is inserted, launch the following.

```
follower.Play();
```

- **name.Pause()**: pause the following.

```
follower.Pause();
```

- **name.Reset()**: reset buffer of points.

```
follower.Reset();
```

The following script show how to load a path (path1), launch the following and wait its completion.

```
follower.Insert(path1);
status = follower.GetStatus();
while not(status == "Paused")
do
    Wait(500);
    status = follower.GetStatus();
end
follower.Play();
while not(status == "Waiting")
do
    Wait(500);
    status = follower.GetStatus();
end
```

5.6 Logs

This package contains services that logs data from MRDS contract into a ".txt" file. Each service from this package logs one **Core** contract and implement **Log** abstract contract.

Each service (except diagnostic) creates two files

- a header (name_header.txt) that contains the meaning of data and specific properties,
- a log file (name.txt) that contains data logged.

5.6.1 LogManager

This service launch all logs at the same moment and put them into the same directory.

5.6.2 Battery

header:

```
Timestamp(ms) Critical Power Voltage Remaining Status
```

5.6.3 CarDrive

header:

```
Timestamp(ms) TargetSpeed TargetSteer CurrentSpeed CurrentSteer DriveEnabled
```

5.6.4 DifferentialDrive

header:

```
Timestamp(ms) TargetLinearSpeed TargetAngularSpeed CurrentLinearSpeed CurrentAngularSteer DriveEna
```

5.6.5 Diagnostic

5.6.6 Drive

header:

```
Timestamp(ms)  Mode Target Position Speed Torque  Mode Target Position Speed Torque ...
```

Note: Lenght depends on the number of drives targeted by partner service.

5.6.7 GPS

header:

```
Timestamp(ms) Latitude(rad) Logitude(rad) Altitude(m) Mode SateliteNumber DilutionOfPrecision utc
```

5.6.8 Laser

header:

```
PoseX(m) PoseY(m) Theta(rad) MinAngle(rad) MaxAngle(rad) Resolution(rad)
0.150 0.000 0.00000 -2.33001 2.31256 0.0173879241228997
Timestamp(ms) Reading...
```

Note: Lenght of readings depends on laser.

5.6.9 Localization

header:

```
Timestamp(ms) locX LocY Theta Status
```

5.6.10 Telemeters

header:

```
PoseX(m) PoseY(m) Theta(rad) MinDistance(rad) MaxDistance(rad) FieldOfView(rad)
0.150 0.000 0.00000 0.2 2 0.2
Timestamp(ms) Reading(m)...
```

5.6.11 TrajectoryFollower

header:

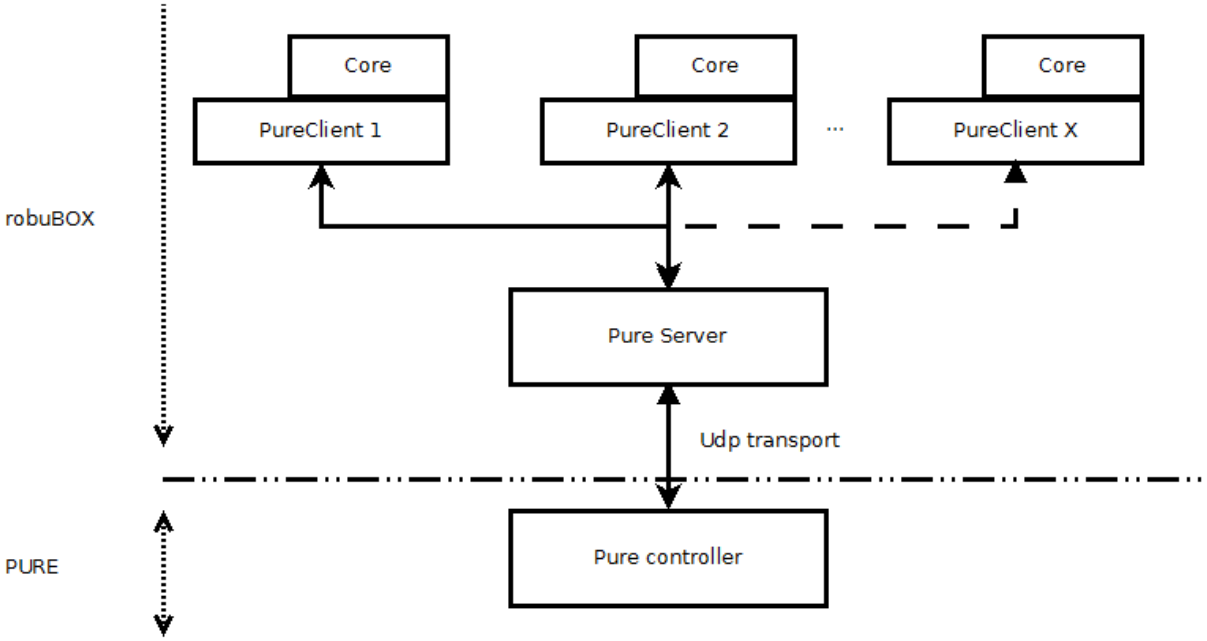
```
Timestamp(ms) Status(W=0,P=1,F=2,E=-1) LateralError(m) HeadingError(rad) DistanceCovered(m)
```

5.7 Pure

This component implements the communication protocol and services offered by the Robosoft low level controller, **PURE**. It is made of a **Server** that communicate dirrectly with PURE and **Clients** that expose **Core** contracts.

If needed, complete and detailed information on this protocol can be found in the PURE User Manual. However, if you use robuBOX, you should be able to proceed only with the information given in this section.

The schema below describes the partnerships between the services of the **Pure** component.



5.7.1 Server

This service implements the communication mechanisms needed to communicate with the low level controller. One instance per PURE controller is required.

Theses mechanisms are request/response kind, mainly for initialization and capabilities discovery, and notifications, either PURE toward client to receive sensors data, or client toward PURE, to send actuators commands.

Configuration

The server configuration file is composed by two parameters (the default file generated **"store\server.config.xml"**, matches the default PURE configuration)

- **Port**: UDP port used (default 60000) to connect to Pure.
- **IpAdress**: IP address of the PURE controller (default 192.168.1.2, with subnet mask 255.255.255.0) .

To get it working, you will have to assign a static IP address on the same subnet to your network interface.

ex: if the IP adress of the PURE controller is 192.168.1.2 with a subnet mask 255.255.255.0, the IP adress of your computer should be 192.168.1.x (x different from 0 or 2) with 255.255.255.0 as subnet mask.

5.7.2 Clients

The data exchanges using the PURE protocol are abstracted from robuBOX applications by Client services. On one side, a client service partners with the Server service to exchange data with PURE. On the other side, it presents this data through a Core abstract contract.

The PURE functionalities have been modularized with the same philosophy used for robuBOX abstract contracts, so each client maps directly to each PURE functionality.

Configuration

The clients will autoconfigure the part of their state relative to the **Core** contracts they exposed, based on the data retreived from the PURE controller.

The only thing they need are:

- **"Instance"** number they should use when exchanging data with the PURE controller.

These numbers can be found by starting the **Server** alone, and examining its state (from a web browser, for example); it will list the functionalities available on the PURE controller.

- **"NotificationPeriod"** period of notifications sent by PURE. (0 = on event, 1 = every 10ms, 2 = every 20ms, ..., 255 every 2550ms).

Client list

- PureBatteryClient
- PureCarDriveClient
- PureDifferentialDriveClient
- PureDriveClient
- PureGPSClient
- PureIOCardClient
- PureLaserClient
- PureLocalizationClient
- PureDiagnosticClient
- PureTelemeterClient
- PureTrajectoryFollowerClient

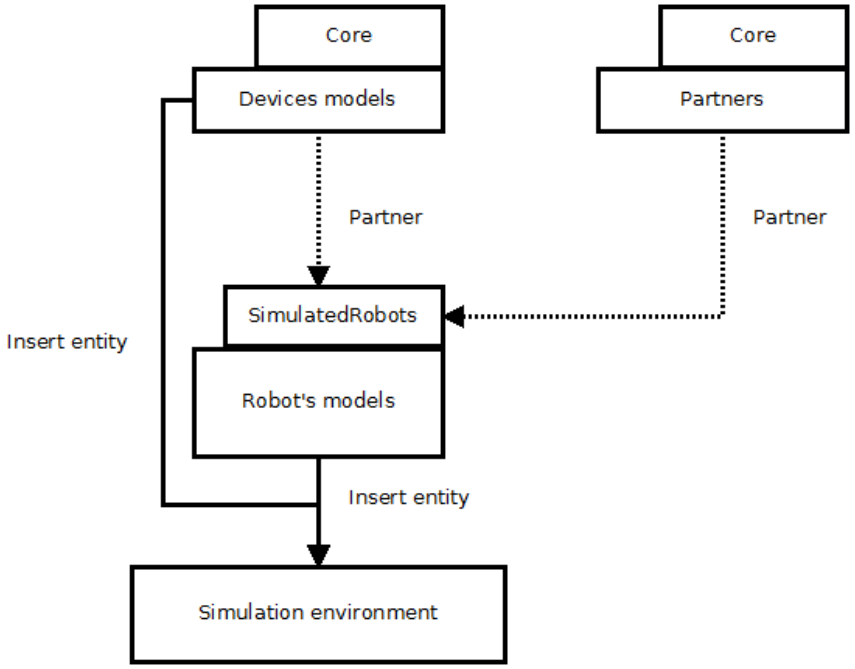
5.8 Simulations

The robuBOX simulation services are organized in two layers:

- The **Model** services: which create and insert robots and/or sensors entities in the simulation engine. Some models simulate a device without inserting any entity inside the simulation environment (ex: Battery, IO-Card).

When a model inserts a robot entity it must expose the **SimulatedRobot** abstract contract.

- The **Partner** services: which interact with entities created by models, and expose them through **Core** abstract contracts.



5.8.1 SimulatedRobot

Abstract contract used to expose an entity name, so a partner can find it in the simulation environment.

This has a use for:

- **Partner services**: they can find the entity they must expose through a Core contract.
- **Model services**: that must insert their entity as a child of another, or synchronize the timestamp with the clock of the simulation environment.

For example, the LaserModel service needs to insert an entity on a platform.

5.8.2 Partners

Partners expose a **Core** abstract contract extracting data from entity.

DifferentialDrive

Use it with a model that inherited from the IDifferentialDriveEntity interface.

CarDrive

Use it with a model that inherited from the ICarDriveEntity interface.

5.8.3 Models

Battery

This service simulates the battery contract. It doesn't need any partners, it doesn't insert any entity.

CarDrive

Inserts a car drive entity inside the simulation environment, inherited from ICarDriveEntity interface.

Camera

Insert a camera entity inside simulated environment. It can be attached to an other entity for tracking.

DifferentialDrive

Inserts a differential drive entity inside the simulation environment, inherited from IDifferentialDriveEntity interface.

Inclinometer

Returns the slope of the partner entity.

IOCard

Simulates an IOCard. It uses a SimulatedRobot as partner only to get a timestamp with respect of the clock of the simulation environment.

Laser

Inserts a raycast entity inside the simulation environment. This entity can be attached to another (specify which one using SimulatedRobot partner).

Localization

Returns the localization of the partner entity.

RobuRide

Inserts the robuRide entity inside the simulation environment. A robuRide have a from axle which rotation center is on it's middle. The robuRide entity inherits from IDifferentialDriveEntity and ICarDriveEntity interfaces, so CarDrive or DifferentialDrive partner can be used with it.

5.9 GUI

This component provides graphical user interfaces for **Core** contracts. There is basically one viewer for each contract. For example, the LaserViewer service displays a Laser service state, etc.

- There is one central service, the Dashboard, whose role is only to host Control (in the Winforms meaning) created by the other services.

These controls can be for example Oscilloscope, available in the Viewers library, or it can be any custom control developed to fit a particular need.

- There is special service, the MapViewer, which allows gathering sensor information in one coherent localization frame.

This control can also be used to display data in the robot frame.

- The path related viewer services (e.g. Record, Follow, Manage and Display) partner with the PathManager.

This allows a modification from the PathManagerViewer service to be immediately available to the PathFollowerViewer, as well as an addition from the Recorder.

5.9.1 Dashboard

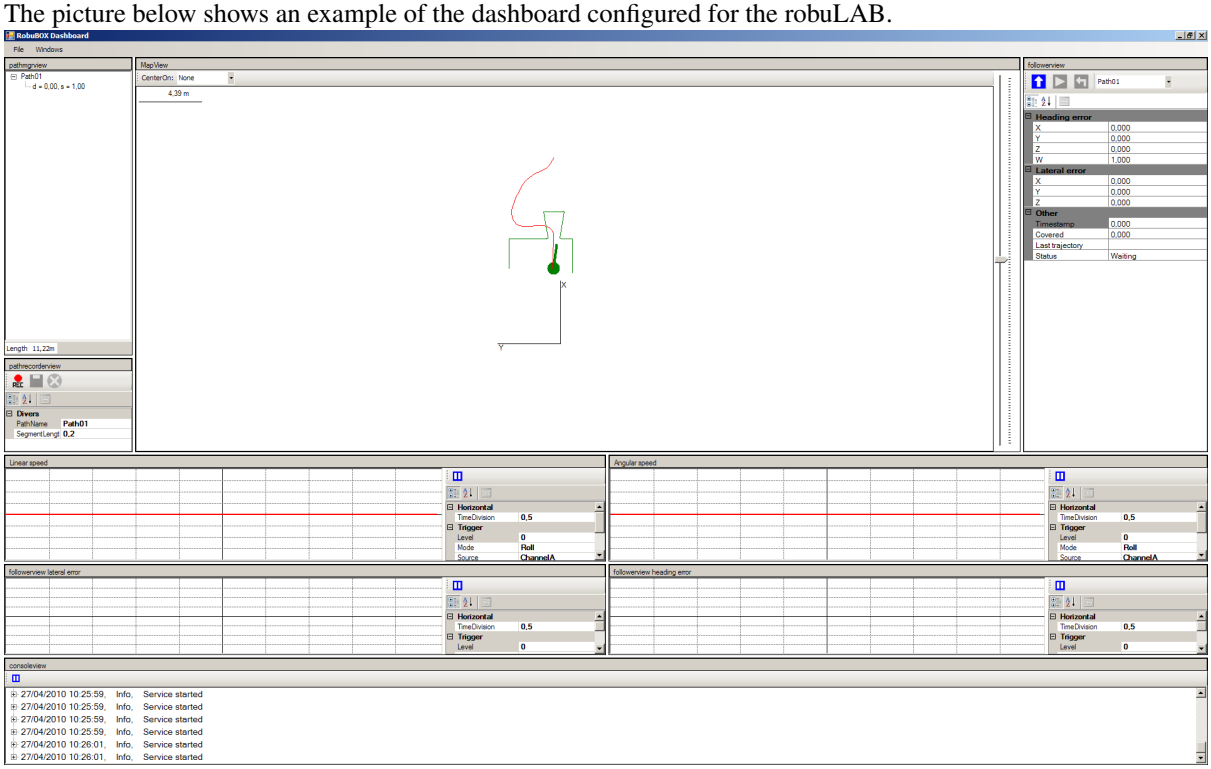
This service creates a window that can be used by other services to host a control. This service will handle display layout and save/load it to/from its configuration file.

The picture below show as an example the dashboard configured for the robuLAB.

The configuration file path can be overridden in a manifest by specifying the StatePartner.

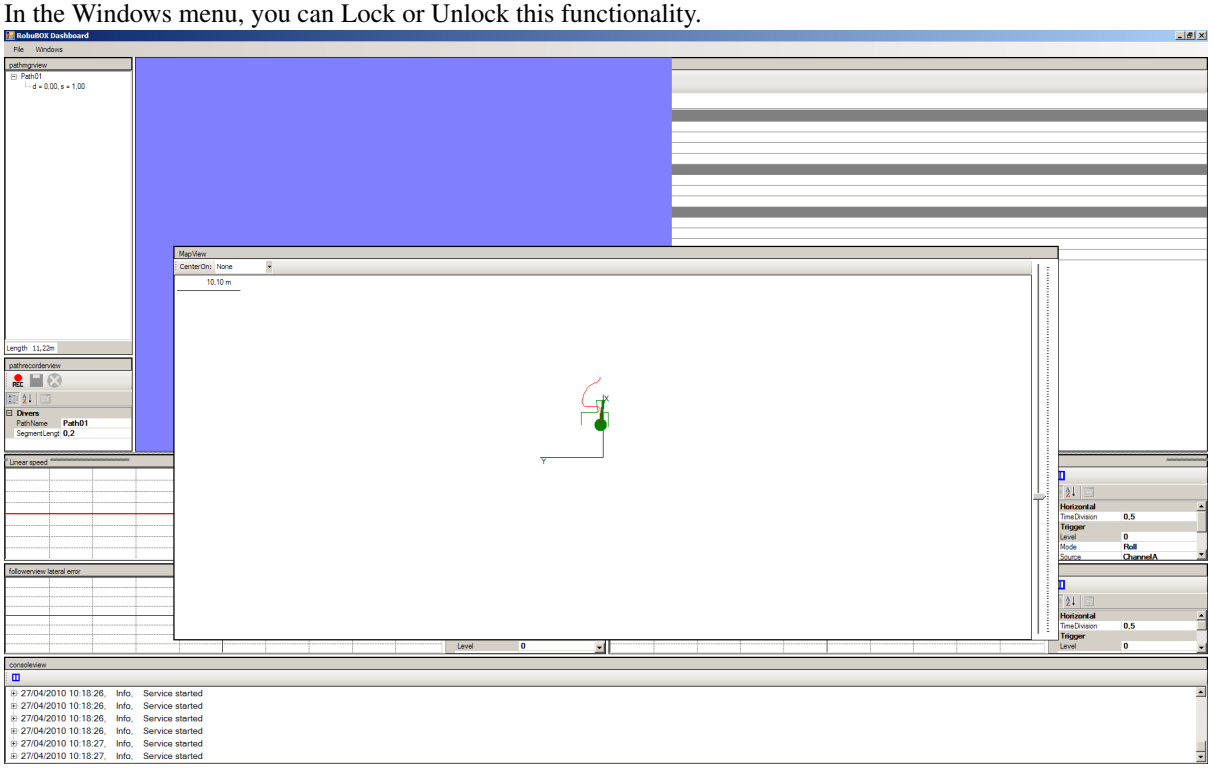
Once the service is loaded, the layout can be saved by clicking on File->Save.

The picture below shows an example of the dashboard configured for the robuLAB.



It is possible to change the organization of the windows by dragging them around, as shown on the next picture. The blue area indicates where the window will be docked when the mouse is released. The principle is similar to Visual Studio.

In the Windows menu, you can Lock or Unlock this functionality.



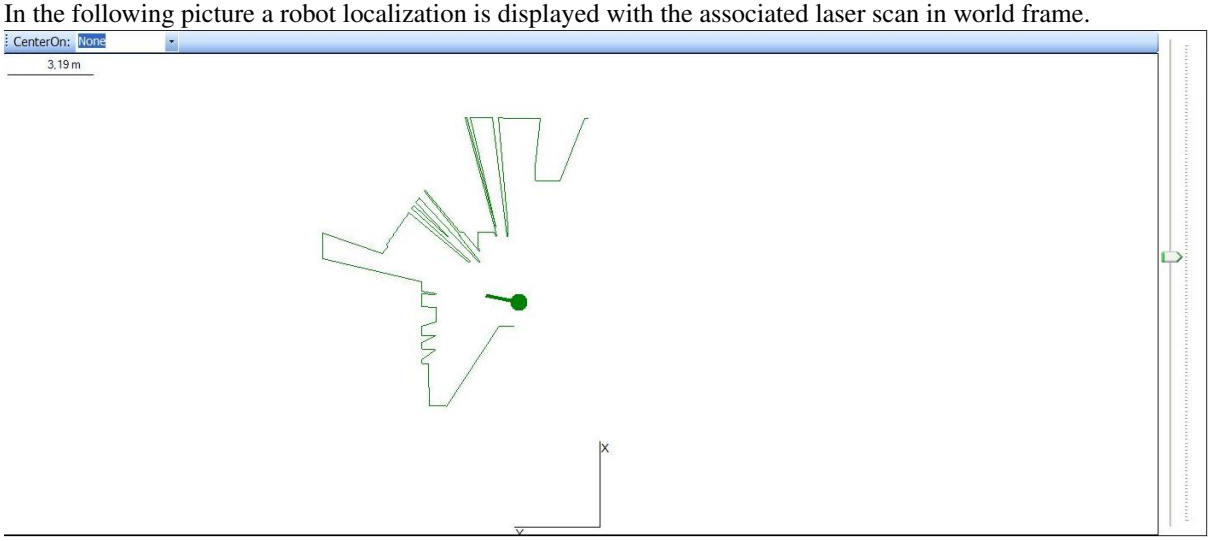
5.9.2 MapViewer

This service displays a map inside which other services can draw lines, circles or robots. It is possible to use a localization source to place these drawings in a global frame. A localization source is available once a corresponding LocalizationViewer has been created.

The services that draw these items can also choose the color in this enumeration: Blue, Black, Brown, Yellow, Red, Green and Gray.

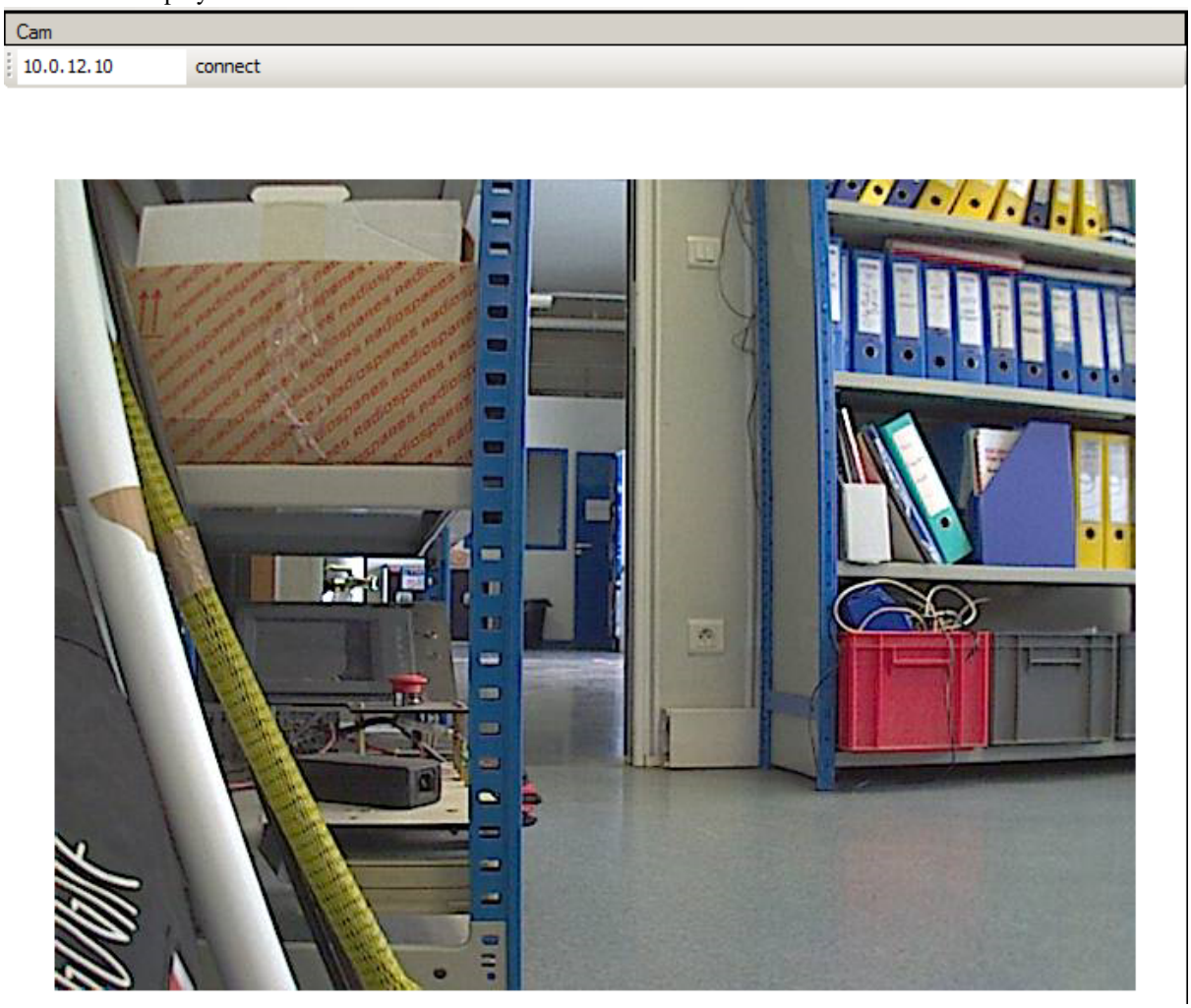
The color and the (optional) localization source to use is configured in each client service.

In the following picture a robot localization is displayed with the associated laser scan in world frame.



5.9.3 AxisCameraViewer

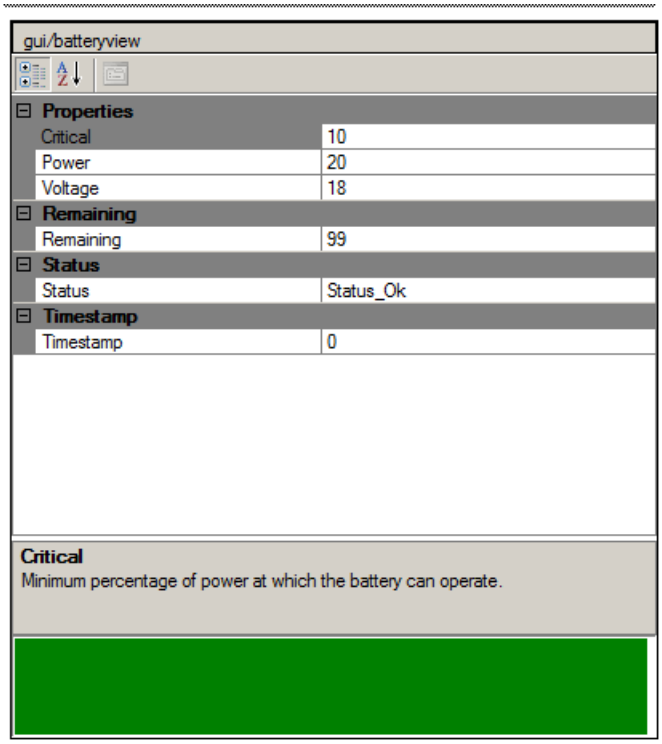
This viewer displays the video stream of the IP camera Axis.



Inside the menu bar on the top, you can enter the IP adress of the camera, then click on **Connect** button. The IP adress will be saved into configuration file of the service.

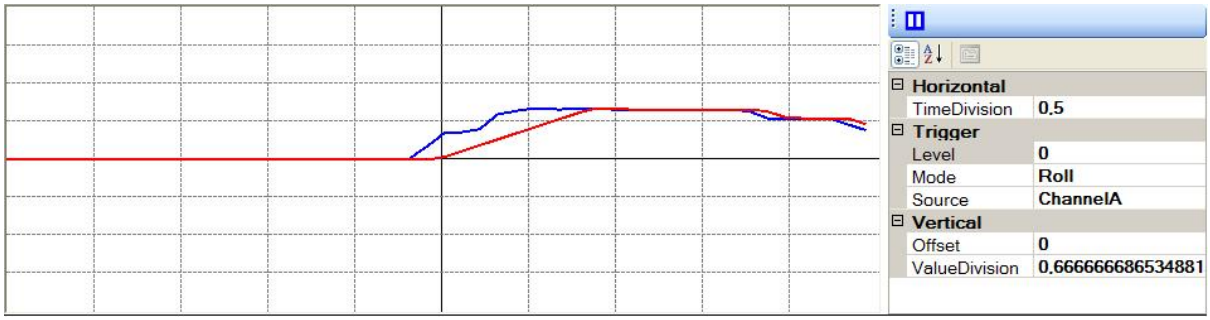
5.9.4 BatteryViewer

This service allows you to monitor battery level.



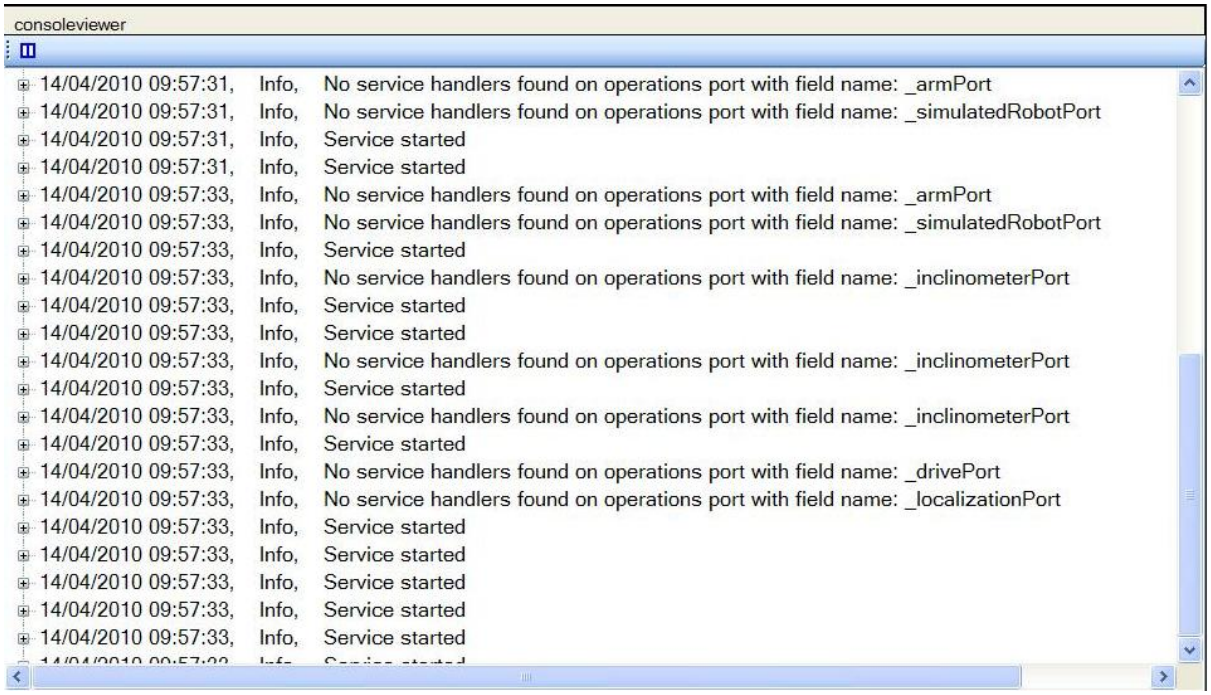
5.9.5 CarDriveViewer

This service allows you to monitor a CarDrive robot state in two oscilloscopes: one for speed and the other for steering. The blue signal is the target value (either steering or speed), and the red signal is the current value.



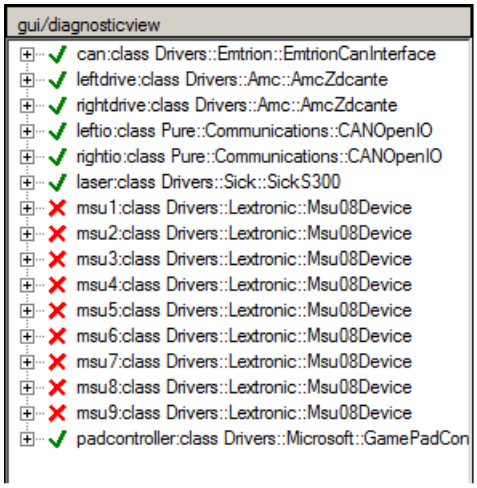
5.9.6 ConsoleViewer

This service displays the MRDS console (can typically be found at http://localhost:50000/console/output). It displays MRDS console in a window. This can be more practical than using a browser and constantly refreshing the console page.



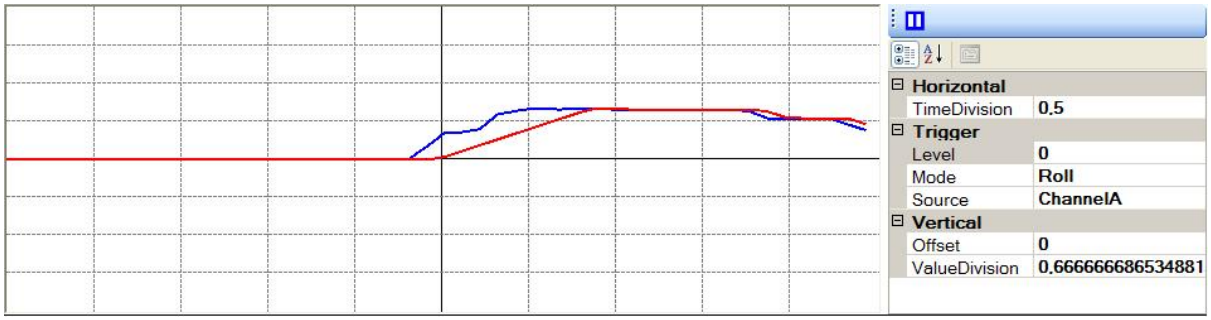
5.9.7 DiagnosticViewer

This service allows to monitor hardware status.



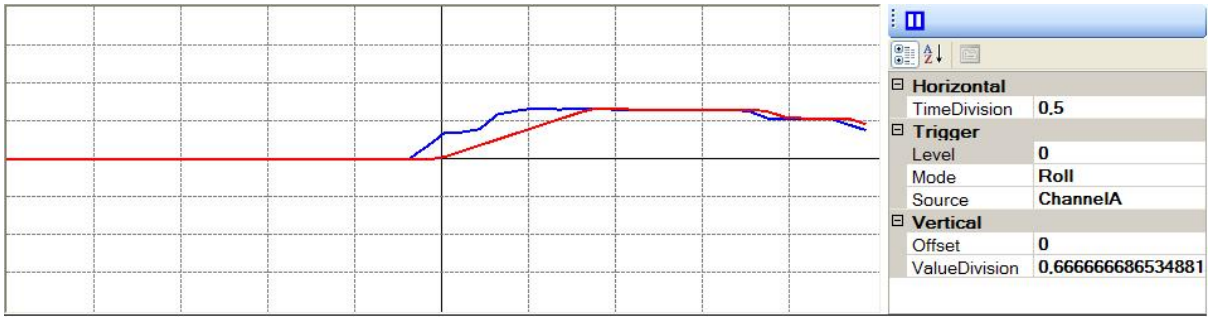
5.9.8 DifferentialDriveViewer

This service allows you to monitor a DifferentialDrive robot displaying two oscilloscopes: one for linear speed and the other for angular speed. The blue signal is the target value (either angular or linear speed), and the red signal is the current value.



5.9.9 DriveViewer

This service allow you to control and monitor a drive displaying an oscilloscope and a controller.

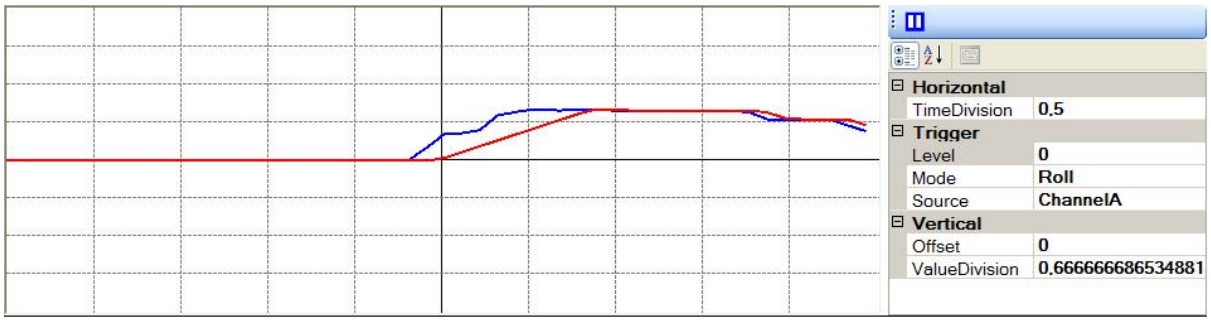


5.9.10 GPSViewer

This service displays GPS data.

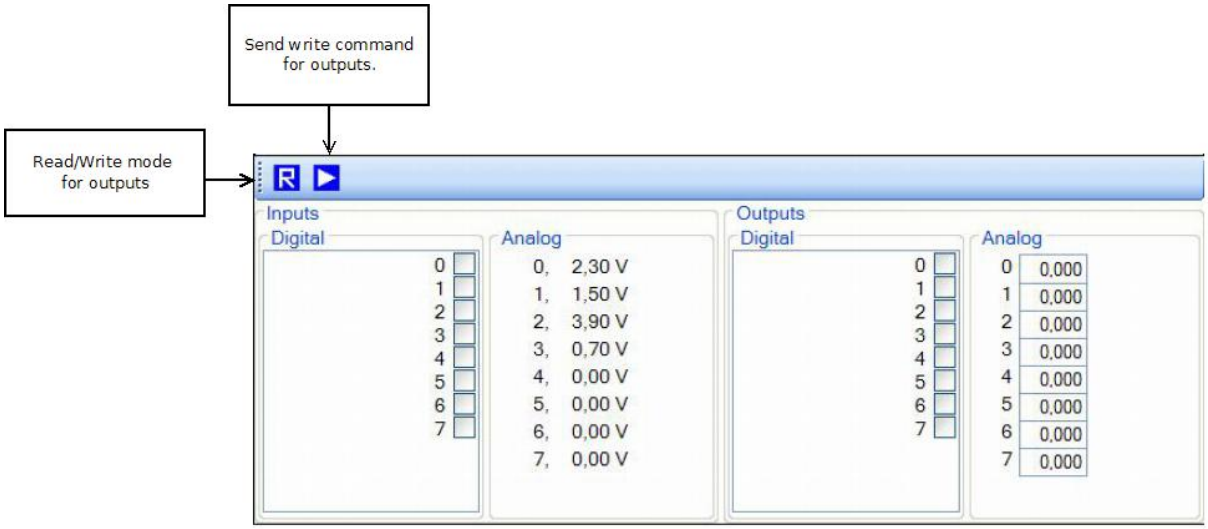
5.9.11 InclinometerViewer

This service allows you to monitor an Inclinometer displaying an oscilloscope. The blue signal is the pitch angle, and the red signal is the roll angle.



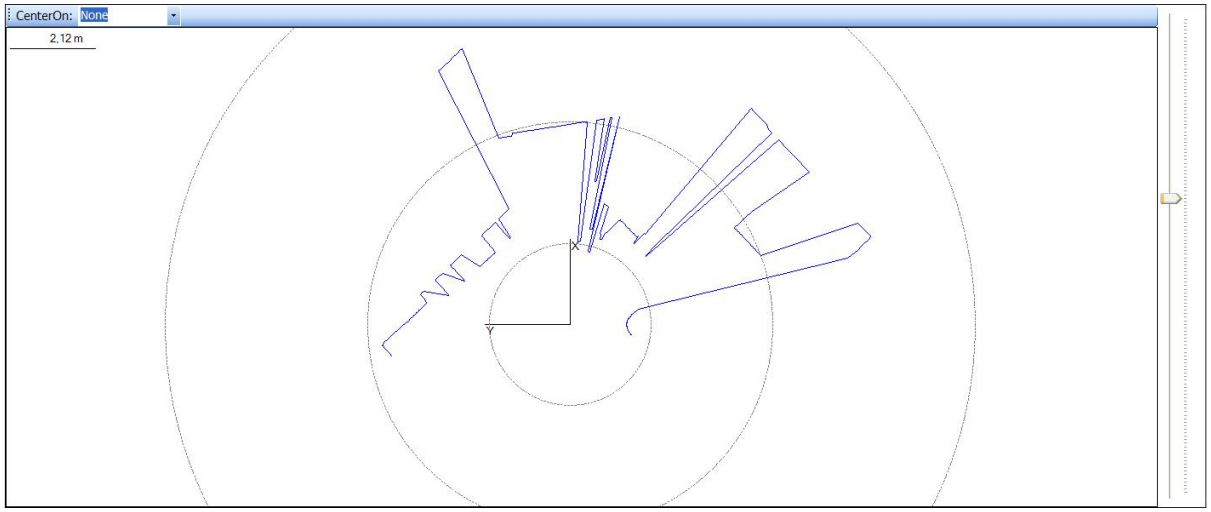
5.9.12 IOCardViewer

This service allows you to monitor a IOCard and to control outputs.



5.9.13 LaserViewer

This service displays the last laser scan in a MapViewer. It can be displayed inside the laser frame as in the following picture, or inside a global frame, based on a localization source.(See the MapViewer service)



The configuration file can be modified to:

- change the scan color.
- **choose the localization frame.**
  - "local": the laser scan will be displayed with five circles to that give distances (2m, 5m, 10m, 20m, 40m).
  - the name of a localization source.

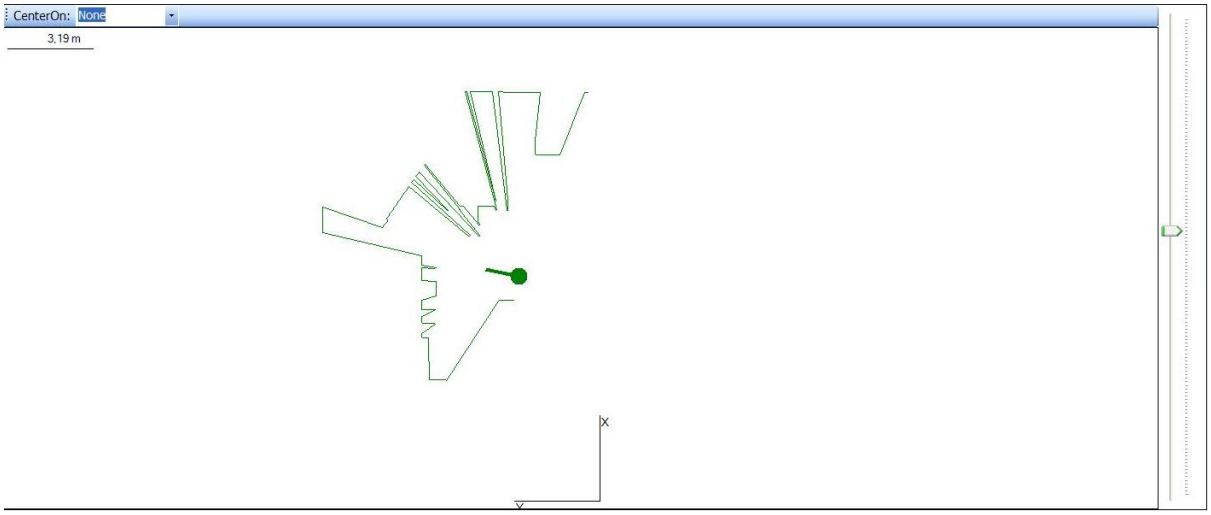
5.9.14 LocalizationViewer

This service displays a robot localization (2D position and orientation) inside a *MapViewer*. The localization can be used as reference for laser or telemeters. The robot view color can be modified using configuration file.

The frame name will be the but last one field of localization service Uri when '/' separators are used.

Ex: http://localhost:50000/robot1/localization.

The frame name will be: “robot1”. This name can be used in services that use the MapViewer to draw the robot frame based sensor information in the world frame.

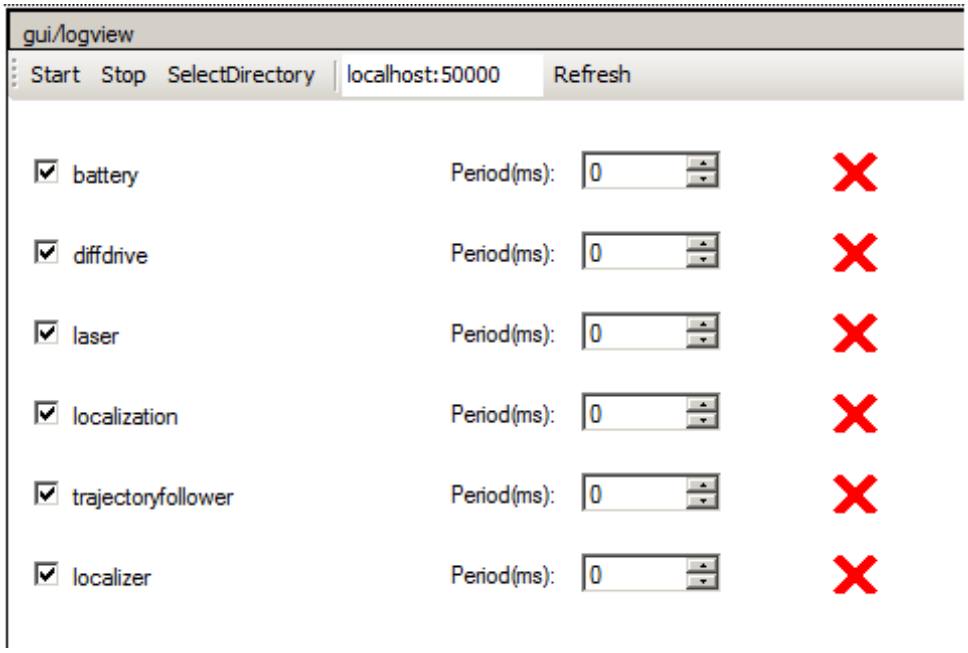


This service also adds two commands when you right-click inside a *MapViewer*:

- Reset localisation: set the localization used as partner to (x = 0,y = 0,z = 0).
- Replace localization: replace the localization used as partner by the localization of the pixel on which you right-clicked.

5.9.15 LoggerViewer

This service allows you to start or stop loggers and select the directory to store log files.



The viewer is made of a header and a body.

From header you can control loggers:

- **Start**: starts selected loggers if they are stopped. An error message will appears inside console if a start request is sent to a logger already started.
- **Stop**: stops selected loggers if they are started. An error message will appears inside console if a start request is sent to a logger already started.

- **SelectDirectory**: changes the directory where logs will be stored.
- **Refresh**: refreshes if not all the loggers that were launched appear inside the viewer (this happends because all services are not launched exactly at the same time...). Service will look for loggers into specified node (textbox between **stop** and **refresh**).

Inside the body you can see 3 columns:

- Use the left one to select the loggers you want to control.
- The middle one is the minimum period needed between two logs (use 0 if you want to log all data).
- The right one indicates if the logger is started (green) or not (red).

5.9.16 PathManagerViewer

This service displays paths contained in the PathManager service state inside a MapViewer. It also inserts a window inside the *Dashboard* that shows registered paths and allows you to tune the desired speed along the path. Speed steps are listed as child elements of the path item.

Right click on a path name to:

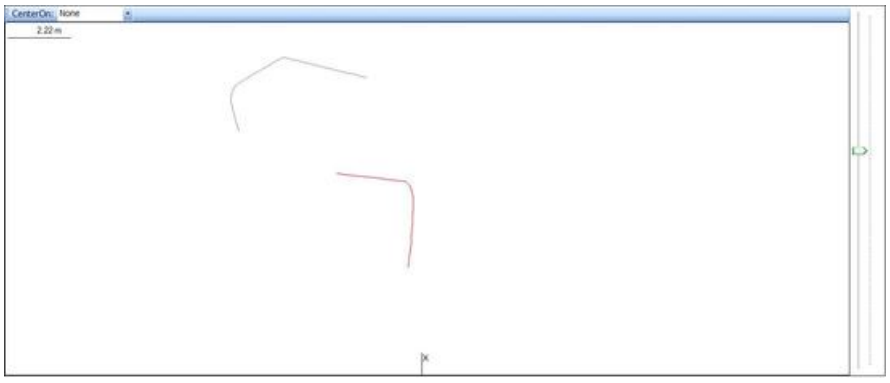
- Delete the path.
- Add a speed step.

Right click on a speed step to:

- Modify the step.
- Delete the step.

A speed step consists of a distance d and a speed s. The distance d corresponds to the distance from the beginning of the trajectory after which the speed s will be applied.

A typical way to configure these steps is to first replay the path at low speed (by setting a step at d = 0, and s = 0.1 m/s for example), and note the distances of interest (like the beginning or the end of a curve, etc) displayed in the TrajectoryFollowerViewer. You can even enter them at the same time the path is being replayed, although the changes will take effect only after a trajectory reload in the TrajectoryFollowerViewer.



5.9.17 PathRecorderViewer

This service allows you to record a path from a localization source. You just have to set the name and the length of the segments and click on the “record” button. Once the path is recorded, save it inside the *pathmanager* or delete it.

The SegmentLength field will set the minimum distance between two recorded points. This means that if the robot does not move, all the points are discarded.

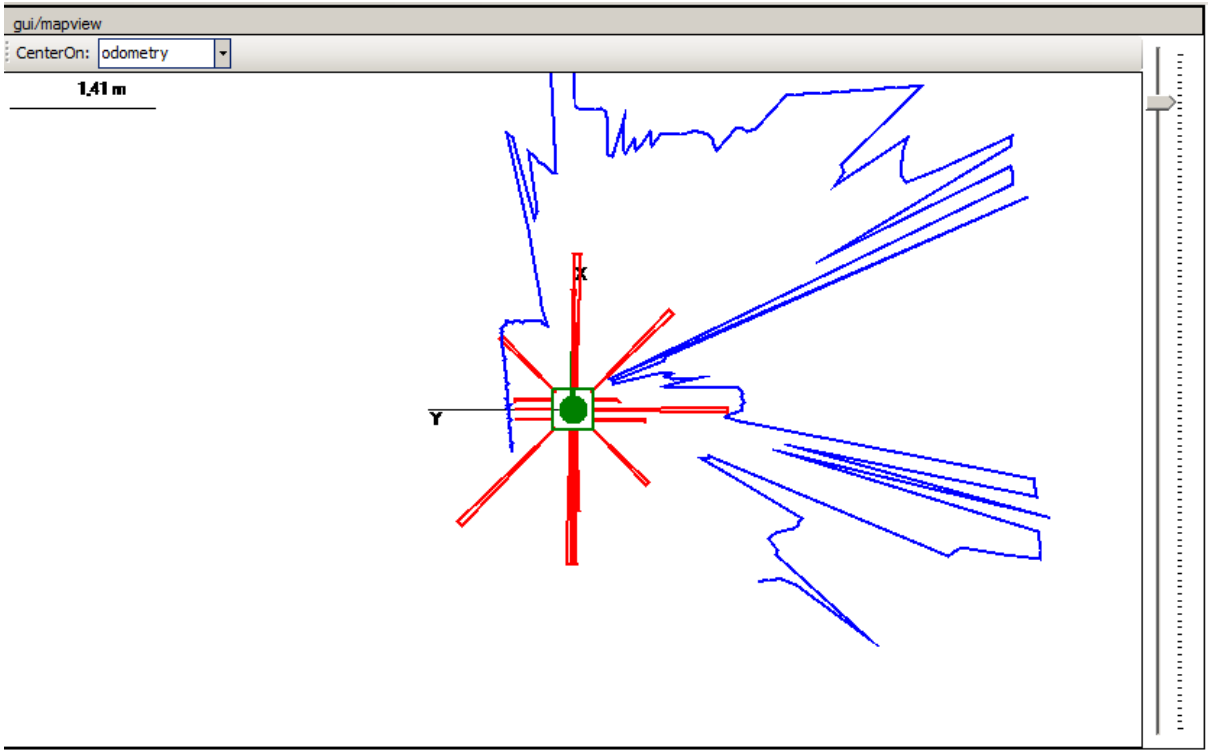
The PathName field is an identifier that must be unique. If you try to save a recorded path with a name that already exists, you will get a pop up dialog with an error message. In this case, just modify the name to be able to save the path.

5.9.18 StepViewer

This service is used to interact with step contract.

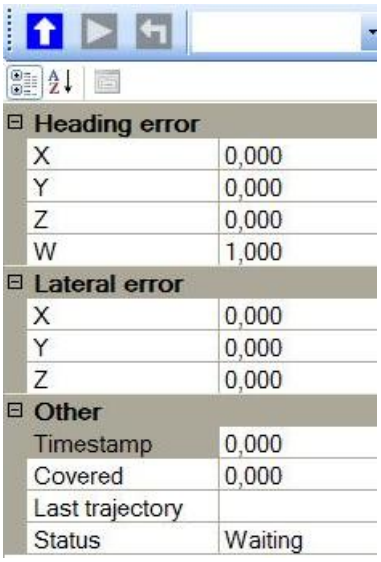
5.9.19 TelemeterViewer

This service displays telemeter data into a map view.



5.9.20 TrajectoryFollowerViewer

This service allows you to interact with a TrajectoryFollower service. It also displays two oscilloscopes to monitor lateral and heading errors.



To use it, first select a path in the upper right drop down list (If there is nothing, you can create one using the *PathRecorderViewer*, see above).

The up arrow button in the upper left corner loads the path in the trajectory follower partner.

The next button is used to start/pause the following.

The right button is used to clear the trajectory loaded (if any) in the trajectory follower service.

When the path is being replayed, you can monitor the errors in the property grid.

Note the presence of the *DistanceCovered* field, which indicates the distance covered since the beginning. This information is useful to configure the speed steps along the path.

5.10 Navigation

This package provides navigation functionalities based on Mapper, Localizer, Planner and Explorer from Karto Robotics SDK.

5.10.1 MapManager

This service is used to give access to maps created with *Mapper* to other services such as *Planner* and *Localizer*.

It loads maps' names from directory specified into its configuration files, and provides them to partners.

You can also specify the map to be used by partners, or displayed on GUI as the **CurrentMap**.

It also store **locations** associated with maps. A **location** is a specific position of the robot (x, y and theta) on the map.

5.10.2 Karto

These services wrap Karto SDK to provide its functionalities through MRDS.

Mapper

This service generates maps of the environment extracting data from laser and current localization.

Maps are .png files upgraded with methadata.

It is also possible to modify these maps to add static obstacles or unavialable arenas (see *mapedition-label*).

Localizer

This service computes robot's localization using laser data, odometry and a map. The map used is the **CurrentMap** provided by *MapManager* partner.

The odometry is extract from the *DifferentialDrive* interface.

Once a new localization is computed, this service updates current localization of robot.

Planner

The planner is used to generate paths for robots. The paths are generated directly from the map. The map used is the **CurrentMap** provided by the *MapManager* partner.

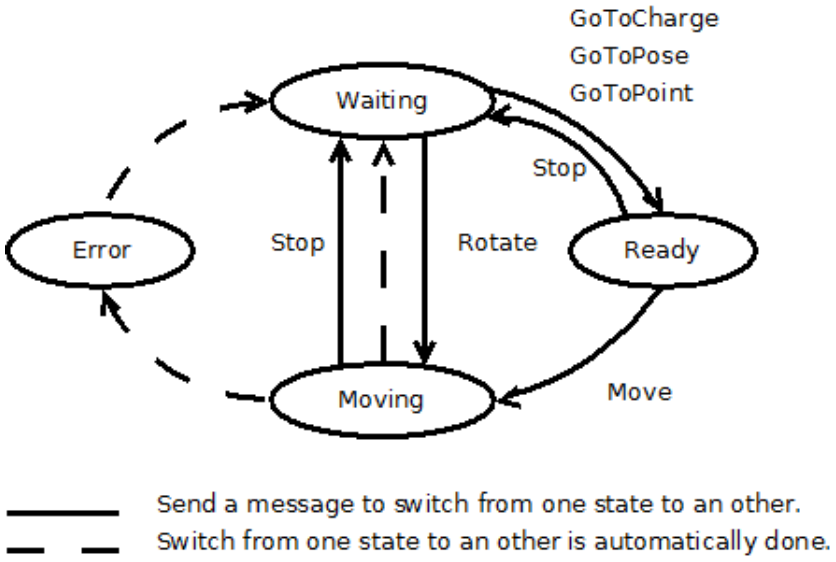
Explorer

The explorer module is used to discover a map automatically from a map partially generated.

5.10.3 DifferentialDriveMotionManager

This service manages various types of motion for robot navigation. It combines *TrajectoryFollower*, *DockingStation*, *Step* and *Planner* functionalities.

The following schema shows the relationship between messages and **CurrentStatus** of the service state. Each time that the **CurrentStatus** will change a notification will be sent.



5.10.4 ObstacleAvoidance

This service is used to bypass obstacles detected with laser during robot navigation. It uses *Planner* functionalities to request a local path and merge it with the current path used by *TrajectoryFollower* partner.

5.10.5 Gui

DifferentialMotionManagerViewer

This services is a graphical interface for *DifferentialDriveMotionManager* functionalities.

- From the combobox you can select a target location.
- Click on **Generate path**. (The path must be displayed on the *MapViewer*).
- Click on **Rotate** to make the robot rotate of selected degrees.
- Click on **Translate** to make the robot rotate of selected degrees.
- Click on **Go to charge** to send the robot to docking station. (A docking station must be registered as **Location**.
- Click on **Move** button to make the robot reach the **Location**.
- Click on **Stop** to stop the robot motion (any kind of motion).

On the bottom of the viewer you can see **CurrentRequest** and **CurrentStatus** of *DifferentialDriveMotionManager* used as partner.

It is also possible to right-click on a white area of the map displayed into the *MapViewer* and select "Generate a path" to request a path generation from current localization to the selected one.

ExplorerViewer

This service enable controling explorer function.

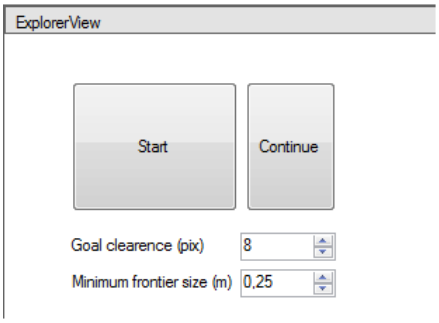
Use **Start** button to request a step forward and two rotations (for first map generation) before exploring world.

Use **Continue** button for exploring world without step forward and the rotations.

Use **Stop** button (**Start** once **Start** on **Continue** clicked) to stop the function.

FrontierSize: The minimum size of valid frontiers in meters. If the length of a frontier is smaller than this minimum, it is not considered to be a goal for exploration. **GoalClearence**: To make sure that each goal computed is reachable with a path planner, each goal is moved inside the free space at a minimum distance to any obstacle

or any frontier. This parameter sets this distance as a number of map cells (or pixels). If this number of cells is set to 0, the positions of the goals are not modified.



LocalizerViewer

This service enables resetting the function used to compute new robot's localization and set a new default localization of the robot into Localizer algorithms.

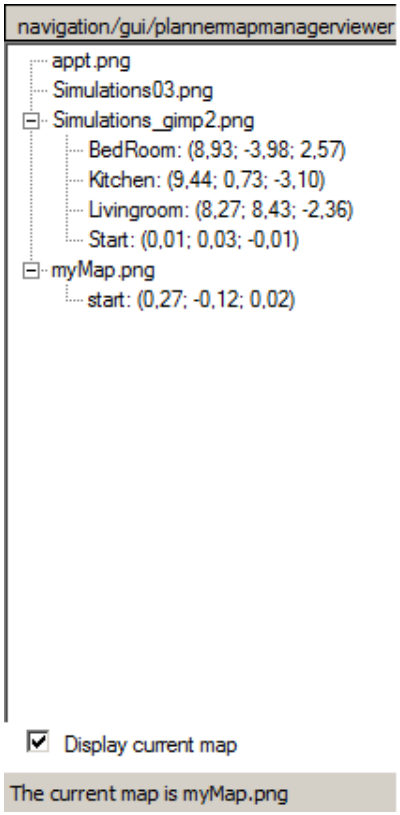
- Right click on the *MapManager* and select "Replace Localizer" this will reset the algorithms used to compute new localization and replace the robot localization by the localization of the pixel targetted by the right-click.

LocationsRecorder

This services allows to associate **Location** with the current map of the mapmanager partner.

MapManagerViewer

Graphical interface for the *MapManager*. Allows removing a **location** from a map, selecting the **CurrentMap** and displaying it inside *MapViewer* partner.



To remove a map and its locations:

- Click on a map name to select it.
- Right-clik on it and select “Erase”: this erase the map and **Locations** associated from *MapManager*. The map is not erased from hard drive so the map will appears inside the *MapManager*

at the next run. To avoid it, erase the map from hard drive manually..

To remove a location from a map:

- Click on a **Location** name to select it.
- Right-clik on it and select “Erase”.

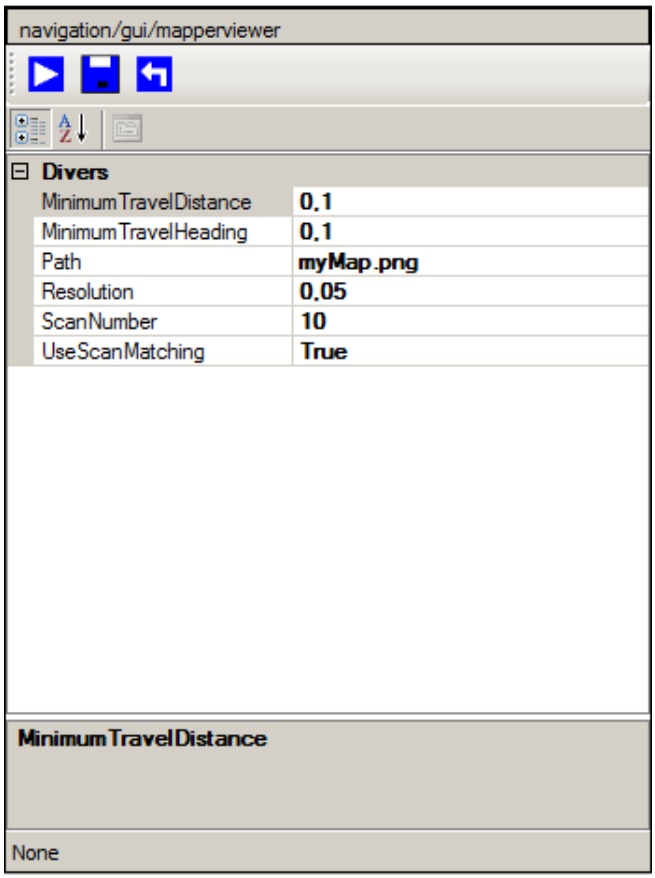
To set a map as **CurrentMap**:

- Click on a map name to select it.
- Right-clik on it and select “Set as current map.”.
- The map name of the **CurrentMap** is displayed inside the status bar on the bottom of the viewer.

If you want the **CurrentMap** to be displayed on the *MapViewer* used as partner, check the box “Display current map”.

MapperViewer

Graphical interface to generate map from laser data.



The control buttons inside the menu bar are the following:

- the left one is used to start and stop creating a map.
- the middle one is to generate a map manually.
- the right one is to reset the *Mapper*.

The parameters are the following:

- **MinimumTravelDistance** : Minimum distance to travel to take a scan in count for map creation (default : 0.1 meter)
- **MinimumTravelHeading** : Minimum angle to travel to take a scan in count for map creation (default : 0.1 radian)
- **Name**: name of the map.
- **Resolution** : resoltion of the map in meter/pixels (default : 0.05 meter/pixels).
- **ScanNumber** : Number of scans needed to generate a map (default : 10)
- **UseScanMatching**: When set to true, the mapper will use a scan matching algorithm. In most real-world situations this should be set to true so that the mapper algorithm

can correct for noise and errors in odometry and scan data. In some simulator environments where the simulated scan and odometry data are very accurate,the scan matching algorithm can produce worse results. In those cases set to false to improve results. Default value is true.

ObstacleAvoidanceViewer

Displays path generated by ObstacleAvoidance service inside a **MapViewer**.

PlannerViewer

Allows to generate path by cliking on map and send it to **TrajectoryFollower**.

5.10.6 Lua

Provides Lua interface for services.

DifferentialDriveMotionManager

- **name.GetStatus()**: returns the current service status (WAITING, READY, FOLLOWING, FOLLOWING_TO_DOCKING, STEPPING, DOCKING, UNDOCKING, CHARGING, STOPPED, DONE, ERROR).

```
status = motionmanager.GetStatus();
```

- **name.GoTo("destination")**: request the robot to reach the destination. The destination is a location associated with the **CurrentMap** of **MapManager** partner.

```
motionmanager.GoTo("Livingroom");
```

- **name.Dock()**: request the robot to reach the destination. The destination is a location associated with the **CurrentMap** of **MapManager** partner.

```
motionmanager.Dock();
```

- **name.Move()**: request the robot to move. This function must be called after .GoTo("destination") function.

```
motionmanager.Move();
```

5.10.7 Lokarria

Provides HTTP interface for services.

DifferentialDriveMotionManager

Request

Send it to retrive the state of the service interfaced.

URL	/lokarria/motionmanager
method	GET
content type	application/json

Response

Request

Send a "GoToPoint" request, returns the path generated.

URL	/lokarria/motionmanager/gotopoint
method	POST
content type	application/json

Content

Destination to reach.

```
{
    "X" = 0,
    "Y" = 0
}
```

Response

```
{
    "Points":
    [
        {
            "MaxSpeed":0.5,
            "Point":
            {
                "X":-0.018430978059768677,
                "Y":0.014290516264736652,
                "Z":0
            }
        },
        {
            "MaxSpeed":0.5,
            "Point":
            {
                "X":0.030706623569130898,
                "Y":0.023533949628472328,
                "Z":0
            }
        },
        ....
    ]
}
```

Request

Send a "GoToPose" request, returns the path generated.

URL	/lokarria/motionmanager/gotopose
method	POST
content type	application/json

Content

Destination to reach. Z coordinate represents the orientation of the robot.

```
{
    "X":0,
    "Y":0,
    "Z":0
}
```

Response

```
{
    "Points":
    [
        {
            "MaxSpeed":0.5,
            "Point":
            {
                "X":-0.018430978059768677,
                "Y":0.014290516264736652,
                "Z":0
            }
        },
        {
            "MaxSpeed":0.5,
            "Point":
            {
                "X":0.030706623569130898,
                "Y":0.023533949628472328,
                "Z":0
            }
        },
        ....
    ]
}
```

Request

Send a "Dock" request, return the path generated.

URL	/lokarria/motionmanager/dock
method	POST
content type	application/json

Content

Docking station coordinates.

```
{
    "X":0,
    "Y":0,
    "Z":0
}
```

Response

```
{
    "Points":
    [
        {
            "MaxSpeed":0.5,
            "Point":
            {
                "X":-0.018430978059768677,
                "Y":0.014290516264736652,
                "Z":0
            }
        },
        {
            "MaxSpeed":0.5,
            "Point":
            {
                "X":0.030706623569130898,
                "Y":0.023533949628472328,
                "Z":0
            }
        },
        ....
    ]
}
```

Request

Send a “Move” request.

URL	/lokarria/motionmanager/move
method	POST
content type	application/json

Content

No content.

Request

Send a “Stop” request.

URL	/lokarria/motionmanager/stop
method	POST
content type	application/json

Content

No content.

Response

MapManager

Request

Send it to retrive the current map.

URL	/lokarria/mapmanager/map
method	GET
content type	image/png

Response

The map.

Request

Send it to retrive the parameters of the current map.

URL	/lokarria/mapmanager/parameters
method	GET
content type	application/json

Response

```
{
    "Height":438,
    "Offset":
    {
        "X":12.573829650878906,
        "Y":10.304725646972656
    },
    "Resolution":33.333335876464844,
    "Width":559
}
```

Request

Send it to retrive the locations associated with the current map.

URL	/lokarria/mapmanager/locations
method	GET
content type	application/json

Response

```
[
    {
        "Name":"BedRoom",
        "Theta":2.5675033626965611,
        "X":8.9297469812687567,
        "Y":-3.9762297215047777
    },
    {
```

```
        "Name":"Kitchen",
        "Theta":-3.1002902993660517,
        "X":9.4362780453804547,
        "Y":0.73118558235486286
    },
    {
        "Name":"Livingroom",
        "Theta":-2.3571040337780675,
        "X":8.27255535237214,
        "Y":8.4306596043614039
    },
    {
        "Name":"Start",
        "Theta":-0.014697403060978094,
        "X":0.0069280015304684639,
        "Y":0.030886232852935791
    }
]
```

Planner

Request

Request to generate a path form current position to destination.

URL	/lokarria/planner
method	POST
content type	application/json

Content

Destination to reach. Z coordinate represents the orientation of the robot.

```
{
    "X" = 0,
    "Y" = 0,
    "Z" = 0
}
```

Response

```
{
    "Points":
    [
        {
            "MaxSpeed":0.5,
            "Point":
            {
                "X":-0.018430978059768677,
                "Y":0.014290516264736652,
                "Z":0
            }
        },
        {
            "MaxSpeed":0.5,
            "Point":
            {
                "X":0.030706623569130898,
```

```
                "Y":0.023533949628472328,
                "Z":0
            }
        },
        ....
    ]
}
```

5.10.8 Logs

LogLocalizer

This service logs the localization computed by the localizer service.

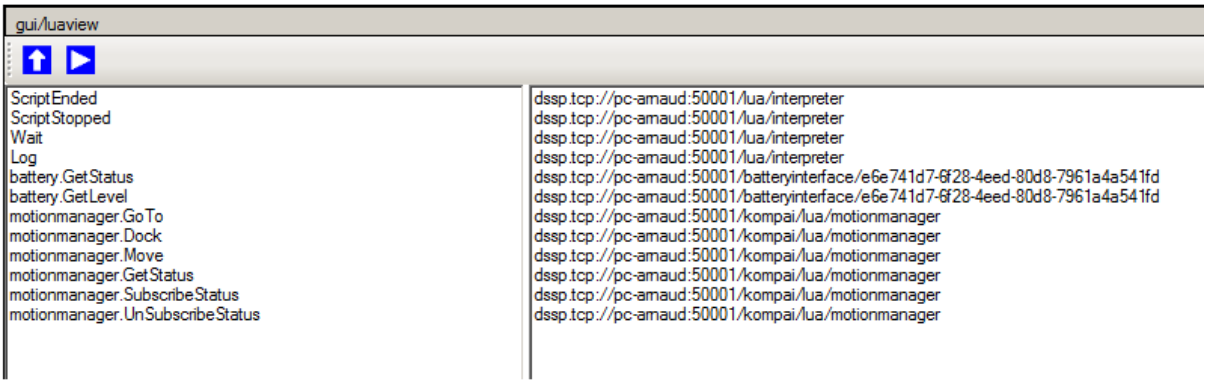
Header:

```
Timestamp(ms) X Y Theta
```

5.11 Services

5.11.1 LuaViewer

Graphic interface to load, start and stop a script. It also displays all functions registered into the interpreter.



The viewer is made of a header and a body.

From header you can:

- Load a script.
- Start / stop it.

Note: When a script is stopped an error message is displayed inside the console. Don't worry it is just because a Lua function was interrupted.

From body you can see on the left the functions registered inside Interpreter, on the right services which have registered functions.

See *Lua* to see how to use scripts.

CHAPTER

SIX

TUTORIALS

This section is a set of tutorials that will help you to start with robuBOX.

6.1 How to run an application?

6.1.1 Basic principles

There are several ways to start services with DssHost (MRDS services host application). The preferred way in robuBOX is to use manifests. All the applications provided can be launched through their manifest. For example, to start the apartment simulation (provided with **MRDS**), open a DSS command prompt and type the following command:

```
dsshost -p:50000 -manifest:"samples\config\apartment.simulation.manifest.xml"
```

> **Warning:** If you have an initialisation failure due to security settings, you can create a new securitysettings.xml file with no authentification and replace the current one. See MRDS documentation for more details: http://msdn.microsoft.com/en-us/library/bb870540.aspx

You should see the Visual Simulation Environment window appear, like the screen shot below.

Note: That it's possible to specify several manifest files, using the -manifest option several times (note here the use of the short form, -m):

```
dsshost -p:50000 -m:"samples\config\apartment.simulation.manifest.xml"
-m:"samples\config\simpledashboard.manifest.xml"
```

6.1.2 Running a real robot application

To run a robot application you need:

- the PURE manifest.
- one or various high level manifests.

The PURE manifest will start all necessary services to access the robuLAB low level controller functionalities (laser, differential drive, battery, odometry, infrared, ultrasound, diagnostic, step and trajectoryfollower). It has to be launched everytime you whant to use a real robot.

It's default path is:

```
C:\MRDS\store\applications\robulab\robot\pure\application.manifest.xml
```

For this to work, you will need to be connected to the robuLAB network (either through its WiFi access point, or through an RJ45 wire) and to know the IP adress of the PURE controller.

The adress of the PURE controller is indicated on the ID card of the robot.

To launch the PURE application:

- ckeck if the PURE server is correctly configurated
  - open *C:\MRDS\store\applications\robulab\robot\pure\server.config.xml*.
  - check if the IP adress specified between **<Host></Host>** becon match with the IP adress of the PURE controller, if not follow the next steps.
  - modify the IP adress.
  - save and quit.
- open a DSS command prompt, and write the following command:

```
dsshost -p:50000 -m:"store\applications\robulab\robot\pure\application.manifest.xml"
```

To launch a high level application such **Monitoring**, **Mapper** or **Navigation** you need to launch it with the PURE manifest as following:

```
dsshost -p:50000 -m:"store\applications\robulab\robot\pure\application.manifest.xml"
-m:"store\applications\robulab\robot\monitoring\application.manifest.xml"
```

6.1.3 Running a simulated robot application

To run a simulated robot application you need:

- a simulated environment
- a simulated robot
- one or various high level manifests.

The simulated robot will start the equivalent of the PURE Application, but in the simulation engine. Note that this application does not start a simulation environment, so we will have to specify one.

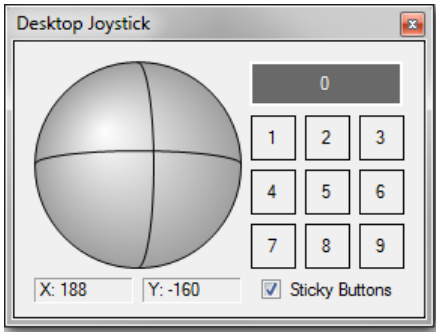
For example, to start it in the apartment environment, open a DSS command prompt, and type the following command:

```
dsshost -p:50000 -m:"samples\config\apartment.simulation.manifest.xml"
-m:"store\applications\robulab\simulated\simulation\application.manifest.xml"
```

This application starts a *JoystickDifferentialDrive* service to control easily the simulated robot. This service is compatible with the **JoystickForm** service provided by Microsoft. The **JoystickForm** service is launched by default with the simulated robuLAB.

To control the robot with do the following:

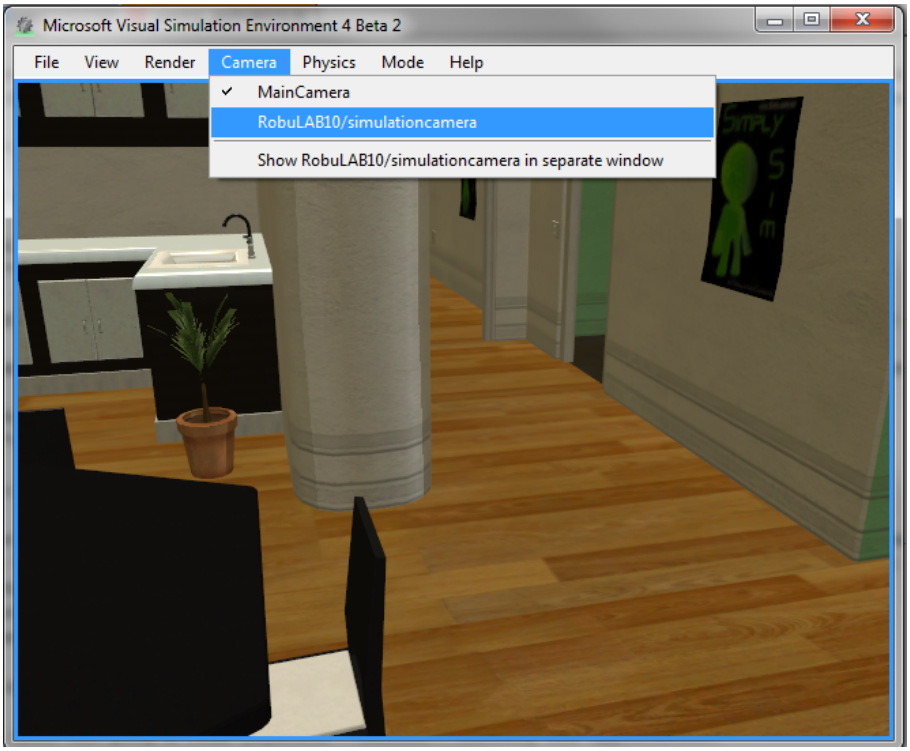
- check **Sticky buttons** check-box.
- press button 0.
- use the ball to control the robot.



Warning: Uncheck **Sticky buttons** if you whant control the robot using the GUI.

You can also plug a real joystick **before** running the application.

A camera tracking was added to robulab. You can select it from simulated environment by camera menu.



To launch a high level application such **Monitoring**, **Mapper** or **Navigation** inside a simulated environment do the following:

```
dsshost -p:50000 -t:50001 -m:"samples\config\apartment.simulation.manifest.xml"
-m:"store\applications\robulab\simulated\simulation\application.manifest.xml"
-m:"store\applications\robulab\robot\monitoring\application.manifest.xml"
```

6.2 How to generate a map?

6.2.1 Launch the Mapper application

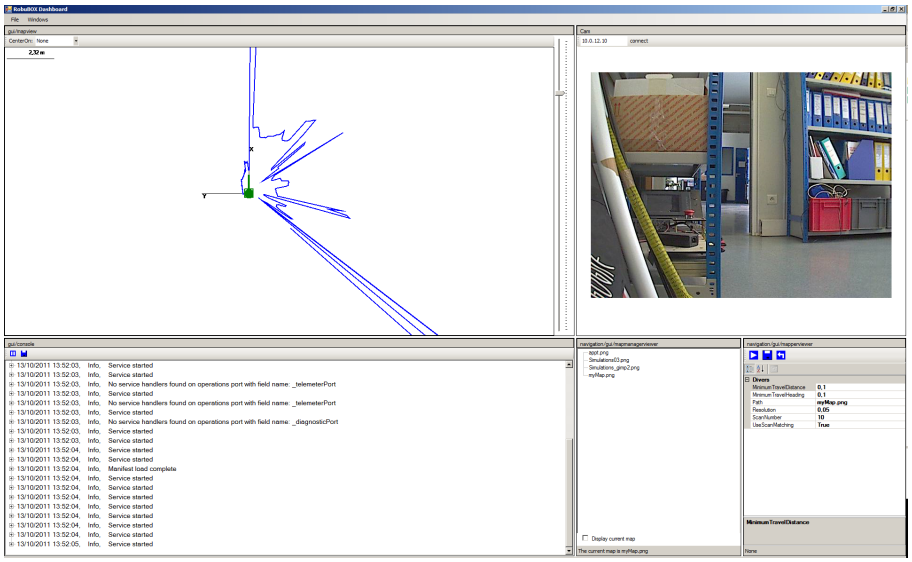
If you want to run it with the real robuLAB, start at the same time the Pure interface application:

```
dsshost -p:50000 -m:"store\applications\robulab\robot\pure\application.manifest.xml"
-m:"store\applications\robulab\robot\mapper\application.manifest.xml"
```

If you want to run it with the simulated robuLAB, start it with an environment and the simulated robuLAB:

```
dsshost -p:50000 -m:"samples\config\apartment.simulation.manifest.xml"
-m:"store\applications\robulab\simulated\simulation\application.manifest.xml"
-m:"store\applications\robulab\simulated\mapper\application.manifest.xml"
```

You should get the following graphical user interface for real robuLAB:



The GUI of this application is made of five panels.

The upper left named “gui/mapview”, is a *MapViewer*. A *LocalizationViewer* and a *LaserViewer* display data inside it.

The upper right panel is an *AxisCameraViewer* that display the video stream from the robot’s IP camera.

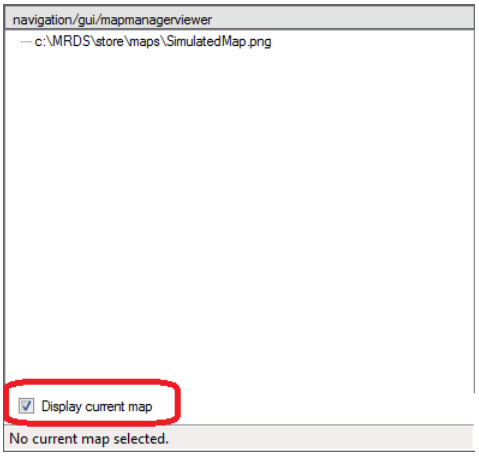
The lower left panel is a *ConsoleViewer*.

The lower middle panel is a *MapManagerViewer*.

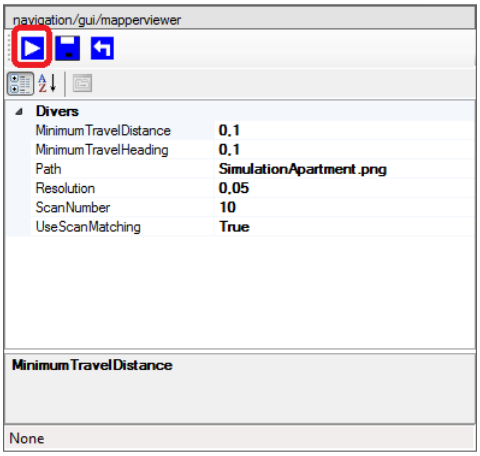
The lower right panel is a *MapperViewer*.

6.2.2 Generate the map

- using the gamepad, drive the robot to a specific location that will be the origin of the map.
- reset the robot localization: right click inside the *MapViewer* (upper left panel named “gui/mapview”) and select **Reset localization**.
- fill the parameters for the map generated inside the *MapperViewer* (except for the map name, the parameters fit for almost all situations).
  - give a unique name to your map, if a map with the same name already exists it will be overridden.
- check **Display current map** on *MapManagerViewer*.



- start generating the map from the *MapperViewer*.
  - press the upper left buton (arrow).



- drive the robot **slowly** using the game-pad until the current area is not mapped.
- Once the map is generated press the **stop** button (that have replaced the start button).

Note: The map will appear automatically inside the *MapViewer*.

Robot localization is based on odometry, there are no corrective algorithms running with this application.

Over time the robot positioning will have a higher error due to odometry.

Warning:
- drive the robot slowly especially during robot rotations and on sliding grounds.
- during the map creation **nobody** must be in front of the laser beam.

6.3 How to edit a map?

Because robot enrironment may vary, there is a need to modify the original map acquired during mapping operation in order to add or remove static obstacles and define new unavailable areas.

Because Karto writes extra data into the PNG map (resolution, offsetX, offsetY, offsetZ etc...), the map cannot simply be edited with an image editor.

This documentation will first explain how to edit the map using a graphical editor and then how to preserve data from Karto.

6.3.1 Prerequisites

The following softwares are necessary:

- The Gimp (v2.6.1 tested).
- TweakPng (v 1.4.0 tested).

We assume that they are already installed on a computer.

6.3.2 Color signification

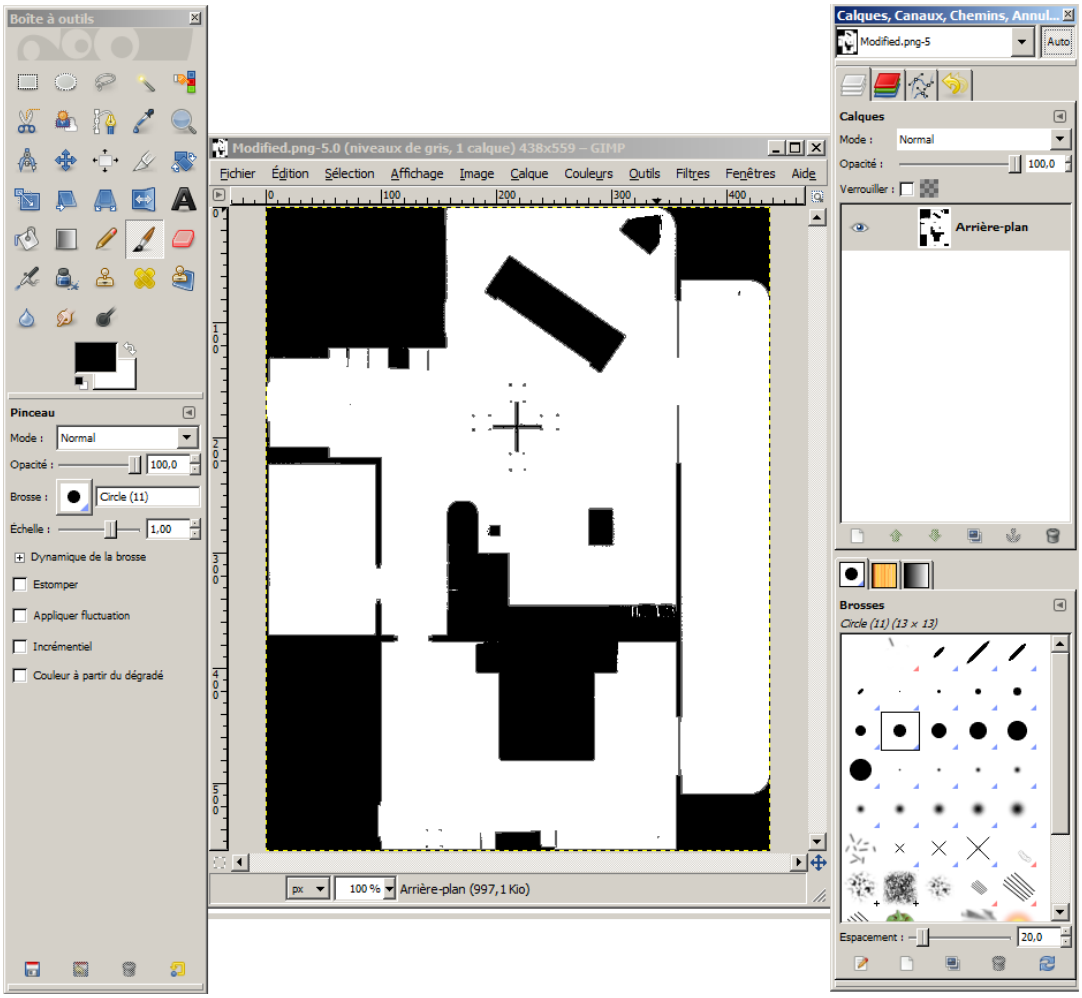
It is important to know the significations of colors of in the map:

- **White** (255.255.255): free space.
- **Grey** (100.100.100) occupied space (object).
- other colors are seen as unknown space but default unknown space is black (0.0.0).

New areas of different colors can be added to the map used by the *Planner*. By default the *Planner* generates a suggested path over free (white) space only.

6.3.3 Use The Gimp to edit the map

- Make a copy of the map you want to modify (for example: OriginalMap.png to ModifiedMap.png).
- Launch The Gimp graphical editor.
- Open the map (ModifiedMap.png) : File > Open...



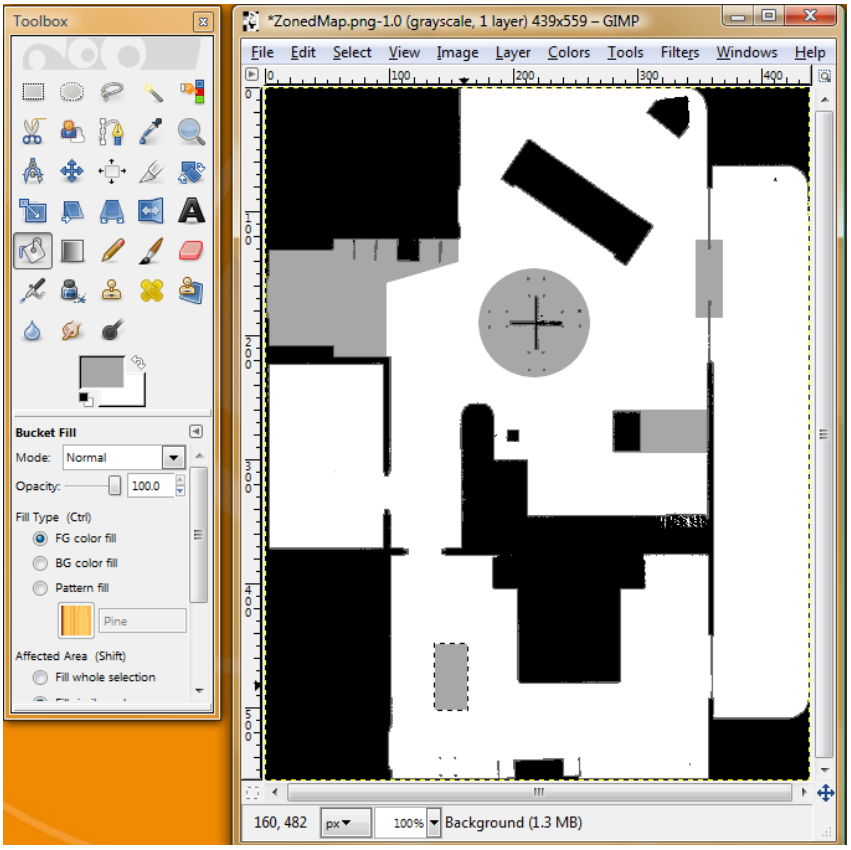
6.3.4 Add or remove obstacles

Obstacles are drawn in black so a classical tool such as the Paintbrush tool can be use to manually draw them. Conversely, use the Eraser tool in order to remove obstacles.

6.3.5 Define non going areas

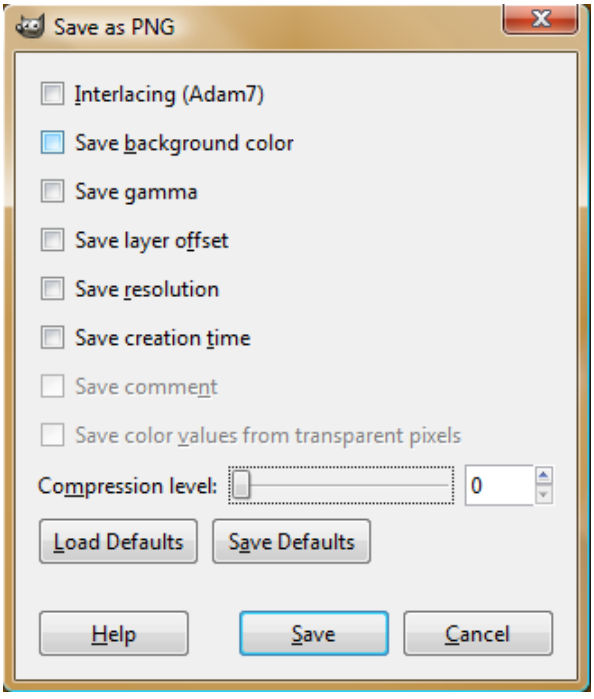
It could be interesting to define forbidden areas on the map.

- Use one of the selection tools (Rectangle, Ellipse or Free Select Tools) located at the top of the Toolbox panel
- Draw a rectangle, a circle or any other geometry to define the forbidden area
- Choose a gray scale color (avoid black color, because in this way the area will be more viewable)
- Use the Bucket Fill tool to fill the selection with the chosen color



6.3.6 Save modifications

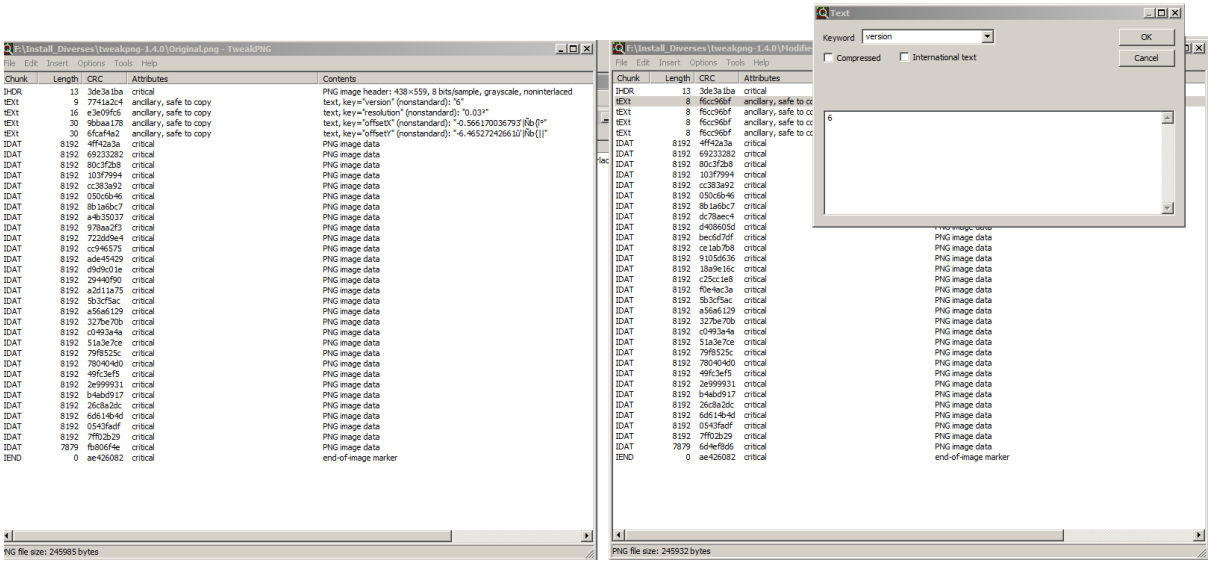
Save the modifications (File > Save As... or shift+ctrl +s). A new windows is open, uncheck all checkboxes and set the compression level to 0. save and close The Gimp.



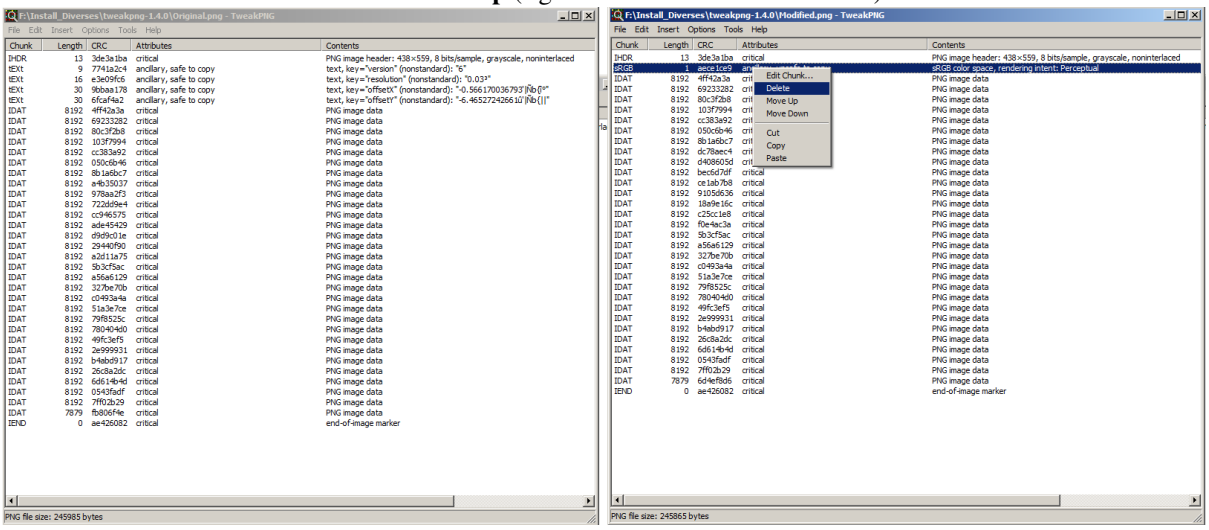
6.3.7 Preserve Karto data using TweakPng

When the map is saved, the data included by Karto are erased. This is how to add them into modified map (Modified.png).

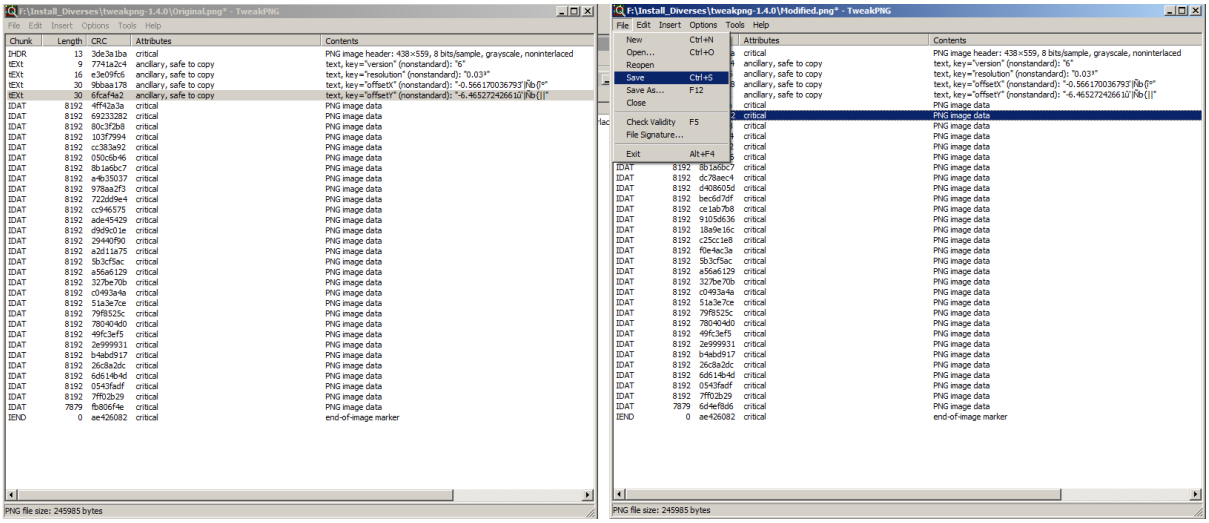
- Open tweakpng.exe
- Open original map with it (File > Open > .../OriginalMap.png).
- Open a second tweakpng window (use tweakpng.exe again, but keep the first one open), open modified map (ModifiedMap.png) with this second window.



- Delete second data line of **modified map** (right click on it and chose Delete).



- Select lines 2 to 5 of the original map, make a copy of them. Select the second line of the modified map and then make a Paste. The first 5 lines on both files should now be the same.
- Save the modified map (ctrl + s). The map directory used is “C:\MRDS\store\maps”.



- You can now use this map for the *Planner*.

6.4 How to record specific locations?

6.4.1 Launch the Navigation application

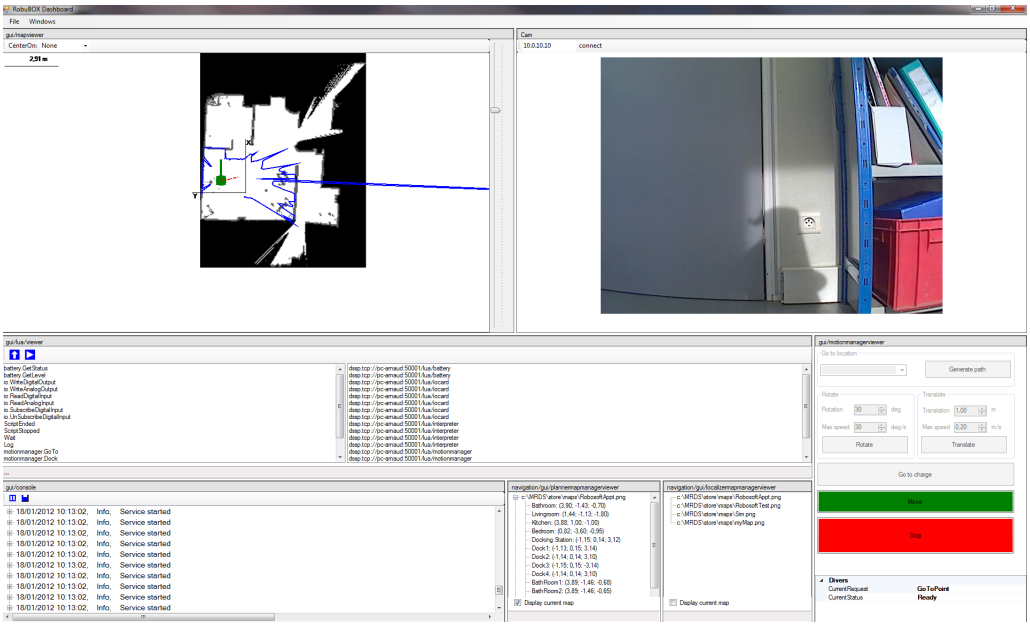
If you want to run it with the real robuLAB:

```
dsshost -p:50000 -m:"store\applications\robulab\robot\pure\application.manifest.xml"
-m:"store\applications\robulab\robot\lua\application.manifest.xml"
-m:"store\applications\robulab\robot\navigation\application.manifest.xml"
```

If you want to run it with the simulated robuLAB:

```
dsshost -p:50000 -m:"samples\config\apartment.simulation.manifest.xml"
-m:"store\applications\robulab\simulated\simulation\application.manifest.xml"
-m:"store\applications\robulab\simulated\lua\application.manifest.xml"
-m:"store\applications\robulab\simulated\navigation\application.manifest.xml"
```

You should get the following graphical user interface for real robuLAB:



The GUI of this application is made of six panels.

The upper left named “gui/mapviewer”, is an *MapViewer*. A *LocalizationViewer*, a *LaserViewer* and a *MapManagerViewer* display data inside it.

The upper right panel is a *AxisCameraViewer* that display the video stream from the robot’s IP camera.

The middle panel is a *LuaViewer*, used to run scripts.

The lower left panel is a *ConsoleViewer*.

The lower right panel is a *DifferentialMotionManagerViewer*.

The two middle lower panels are *MapManagerViewer*.

The left one named “navigation/gui/localizermapmanagerviewer” is related to the *Localizer*. The *Localizer* service will use its own **CurrentMap**.

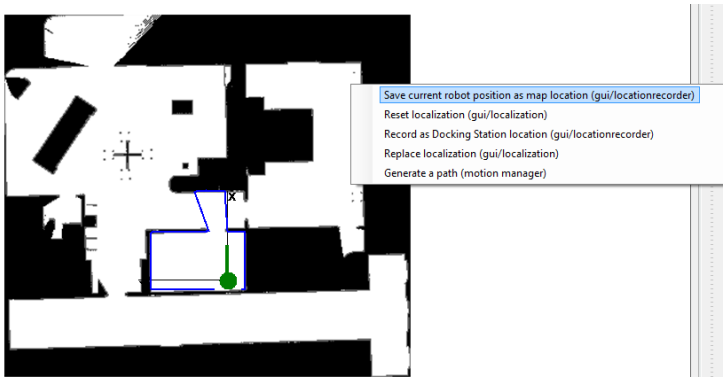
The right panel is related with the *Planner* service. The *Planner* service will use its own **CurrentMap**.

Hence, two different *MapManager* are used because a map used for the *Planner* can be a modified version of the original map used by the *Localizer*. Saved **locations** will be associated with the map from the *Planner*.

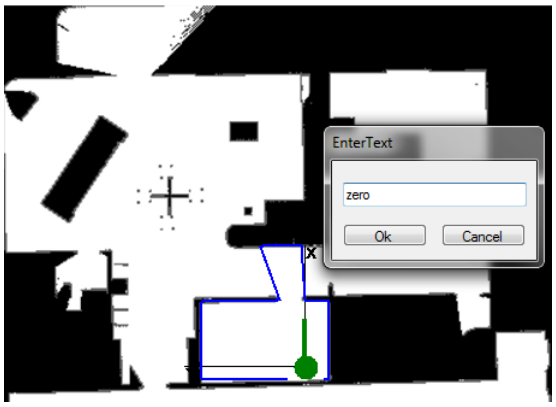
6.4.2 Save your locations

To associate **locations** with the map used by *Planner*:

- check that the robot is correctly localized inside the map (laser scans match with the map shape).
- drive the robot using the game-pad to the location, check again that the robot is correctly localized.
- right-click on the map, a panel will appear
- select **"Save current robot position as map location"**



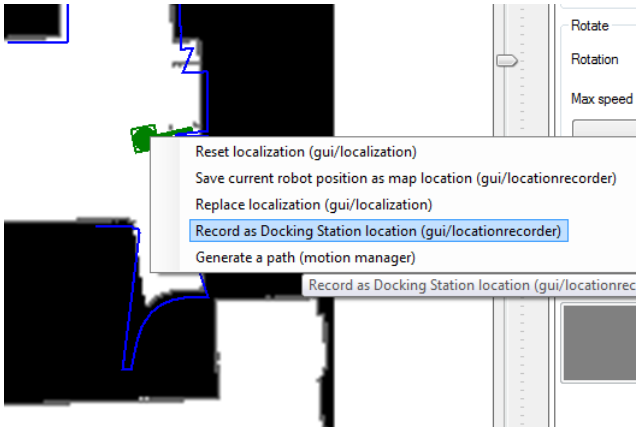
- give a name to your location using the pop-up that appears. One entered the location will be attached to the current map.



- save as many location as you need.

6.4.3 Save the DockingStation location

- verify that the robot is correctly localized inside the map (laser scans match with the map shape).
- drive the robot using the game-pad into the DockingStation, check again the that robot is correctly localized.
- right-click on the map, a panel will appear, select **"Record as docking station location"**.


Rotate
Rotation
Max speed
Reset localization (gui/localization)
Save current robot position as map location (gui/locationrecorder)
Replace localization (gui/localization)
Record as Docking Station location (gui/locationrecorder)
Generate a path (motion manager)
Record as Docking Station location (gui/locationrec

6.5 How to control the robot?

You can control the robot using a game pad or a joystick (real or simulated) and you can use *DifferentialMotionManagerViewer*, this is this control mode that will be described here.

6.5.1 Launch the Navigation application

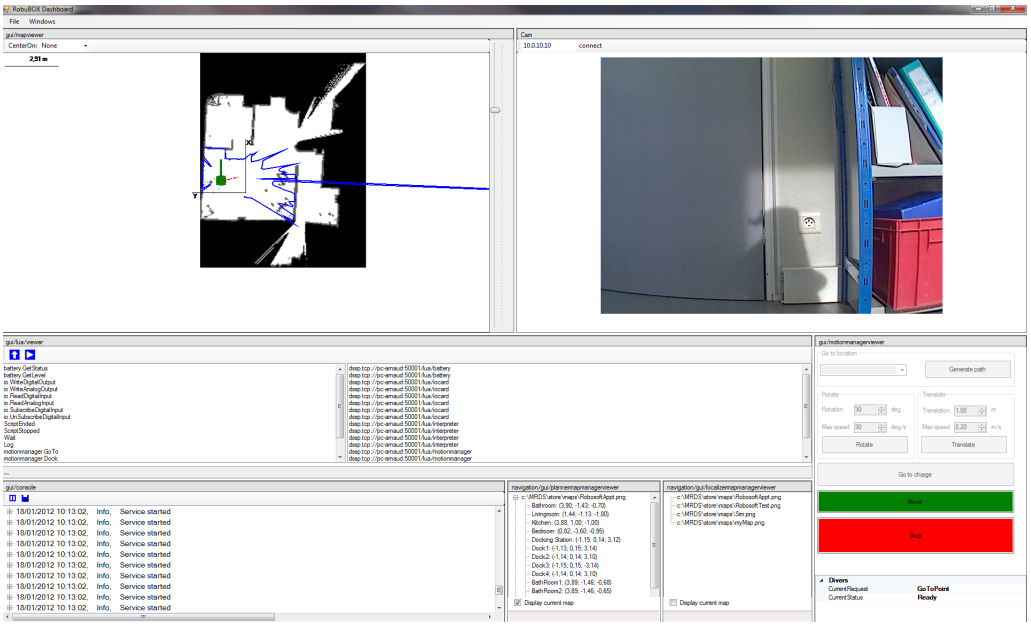
If you want to run it with the real robuLAB:

```
dsshost -p:50000 -m:"store\applications\robulab\robot\pure\application.manifest.xml"
-m:"store\applications\robulab\robot\lua\application.manifest.xml"
-m:"store\applications\robulab\robot\navigation\application.manifest.xml"
```

If you want to run it with the simulated robuLAB:

```
dsshost -p:50000 -m:"samples\config\apartment.simulation.manifest.xml"
-m:"store\applications\robulab\simulated\simulation\application.manifest.xml"
-m:"store\applications\robulab\simulated\lua\application.manifest.xml"
-m:"store\applications\robulab\simulated\navigation\application.manifest.xml"
```

You should get the following graphical user interface for real robuLAB:



The GUI of this application is made of six panels.

The upper left named “gui/mapviewer”, is an *MapViewer*. A *LocalizationViewer*, a *LaserViewer* and a *MapManagerViewer* display data inside it.

The upper right panel is a *AxisCameraViewer* that display the video stream from the robot’s IP camera.

The middle panel is a *LuaViewer*, used to run scripts.

The lower left panel is a *ConsoleViewer*.

The lower right panel is a *DifferentialMotionManagerViewer*.

The two middle lower panels are *MapManagerViewer*.

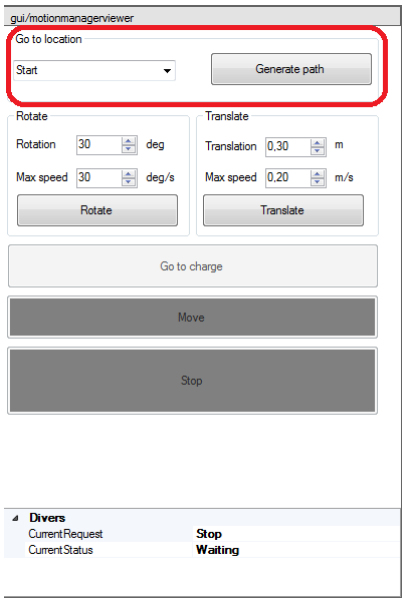
The left one named “navigation/gui/localizermapmanagerviewer” is related with to *Localizer*. The *Localizer* service will use its own **CurrentMap**.

The right panel is related with the *Planner* service. The *Planner* service will use its own **CurrentMap**.

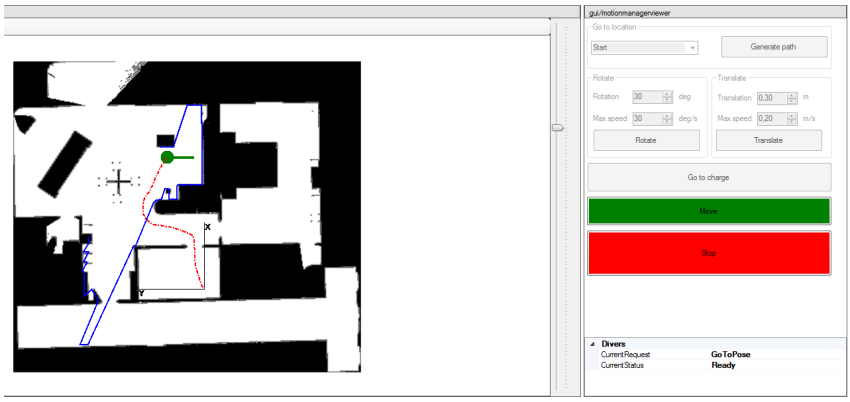
Hence, two different *MapManager* are used because a map used for the *Planner* can be a modified version of the original map used by the *Localizer*. Saved **locations** will be associated with the map from the *Planner*.

6.5.2 Reach a preset location

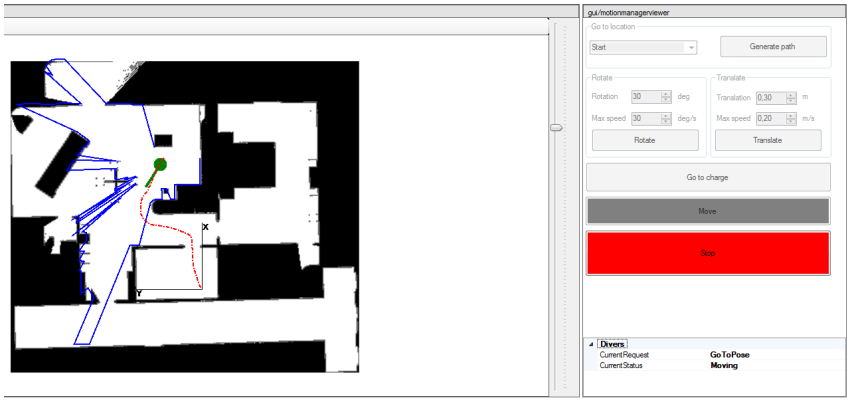
- Some locations needs to be previously registered.
- Choose the location you want to reach using the combobox



- Click on **Generate path** button, the path should appear on map and the buttons **Move** and **Stop** are unlocked (the robot is ready and waiting a confirmation).



- Click on **Move** button to make your robot follow the path, or **Stop** to cancel the request.

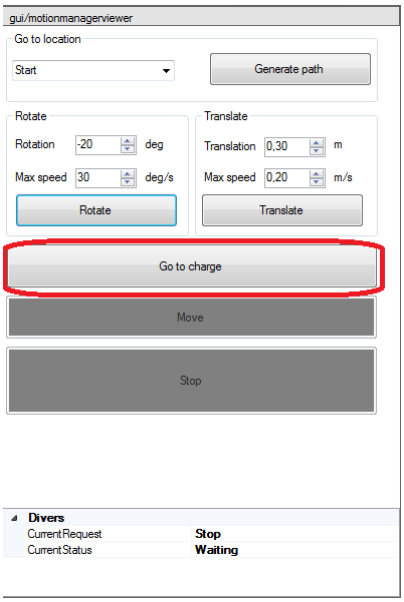


Note: If the robot is docked, it will make a backward translation of about 0.3 meters before following its trajectory.

- Press **Stop** button at any time to stop robot motion.

6.5.3 Reach the docking station

- DockingStation location must be previously registered.
- Click on **Go to charge** button.

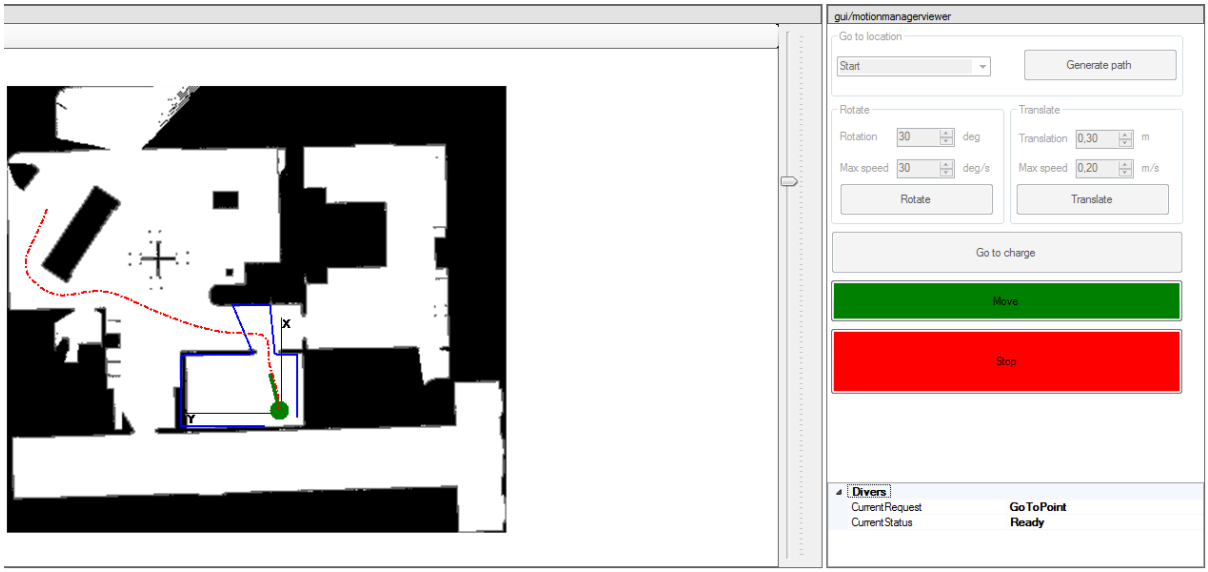


- Click on **Move** button to make your robot follow the path, or **Stop** to cancel the request.

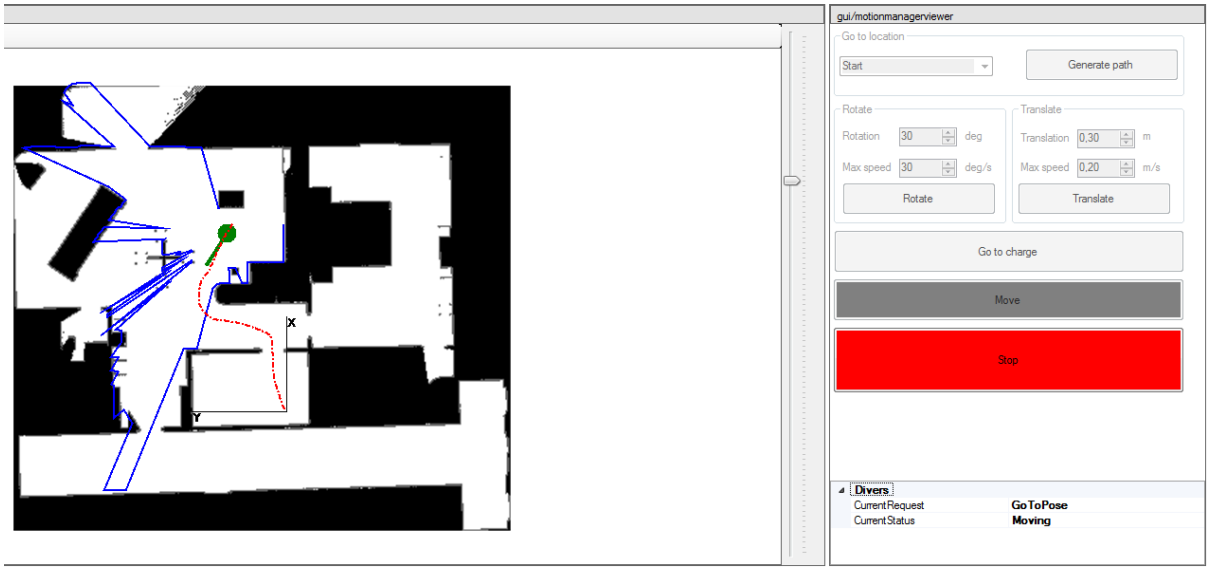
6.5.4 Reach a point on the map

- Right click on the map a menu appears.

- Choose **Generate a path (motion manager)**. The path should appear on map and the buttons **Move** and **Stop** are unlocked (the robot is ready and waiting a confirmation).



- Click on the **Move** button to make your robot following the path, or **Stop** to cancel the request.

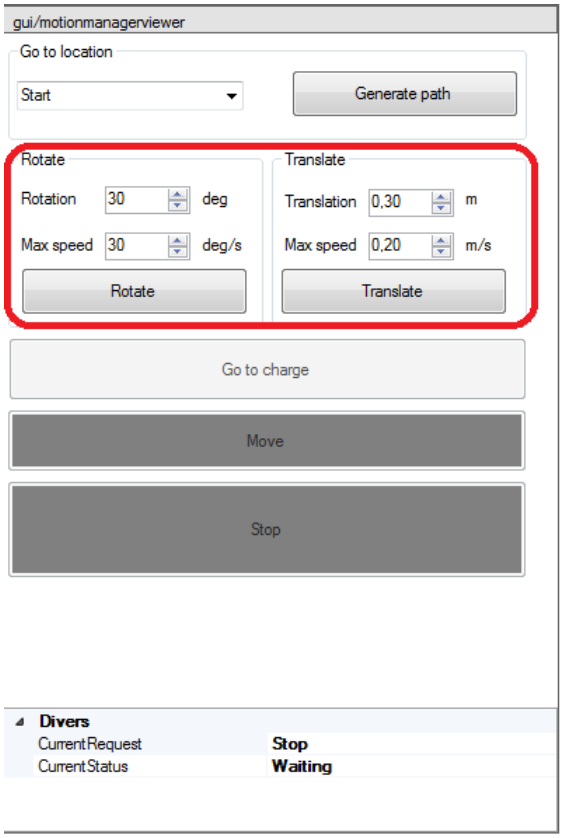


Note: If the robot is docked, it will make a backward translation of about 0.3 meters before following its trajectory.

- Press **Stop** button at any time to stop robot motion.

6.5.5 Step requests

You can request your robot to make a rotation or a translation.



- Specify the distance and the maximum speed to reach.
- Click on the dedicated button.
- Press **Stop** button at any time to stop robot motion.

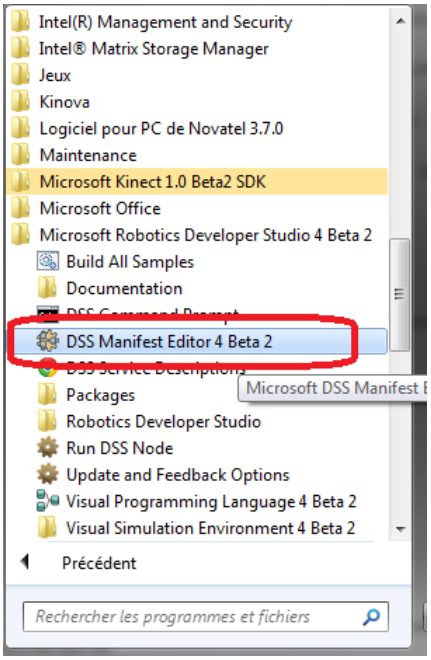
6.6 How to create a custom Dashboard?

This tutorial you will learn how to create your own dashbord.

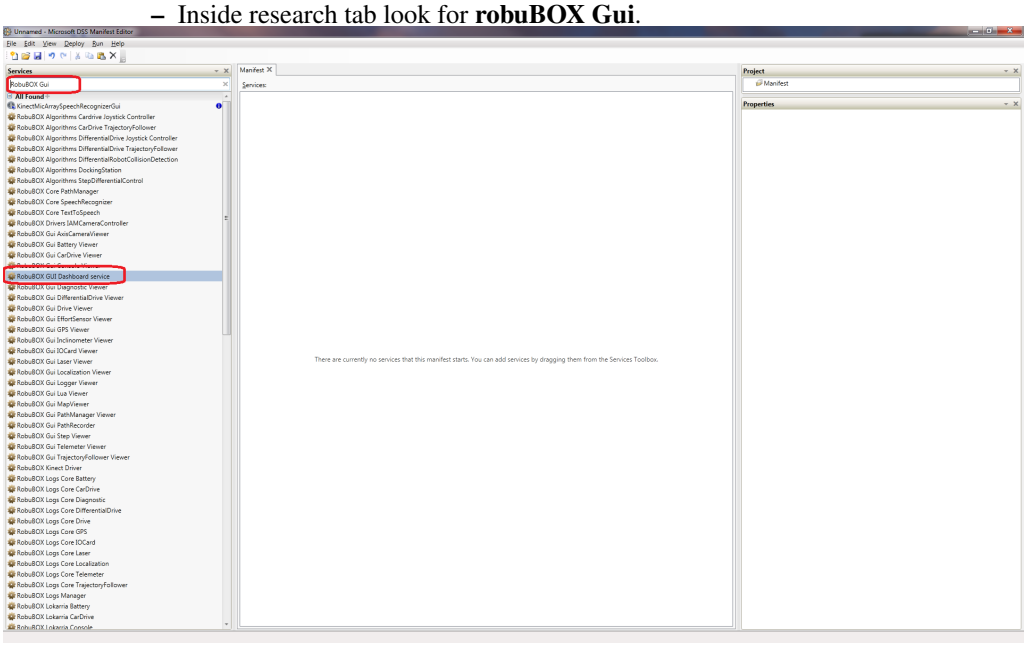
6.6.1 Create a manifest

A custom dashboard can be created using a manifest. The easiest way is to use the DSS Manifest Editor tool, provided with MRDS. For more details about manifests, consult the MDRS documentation.

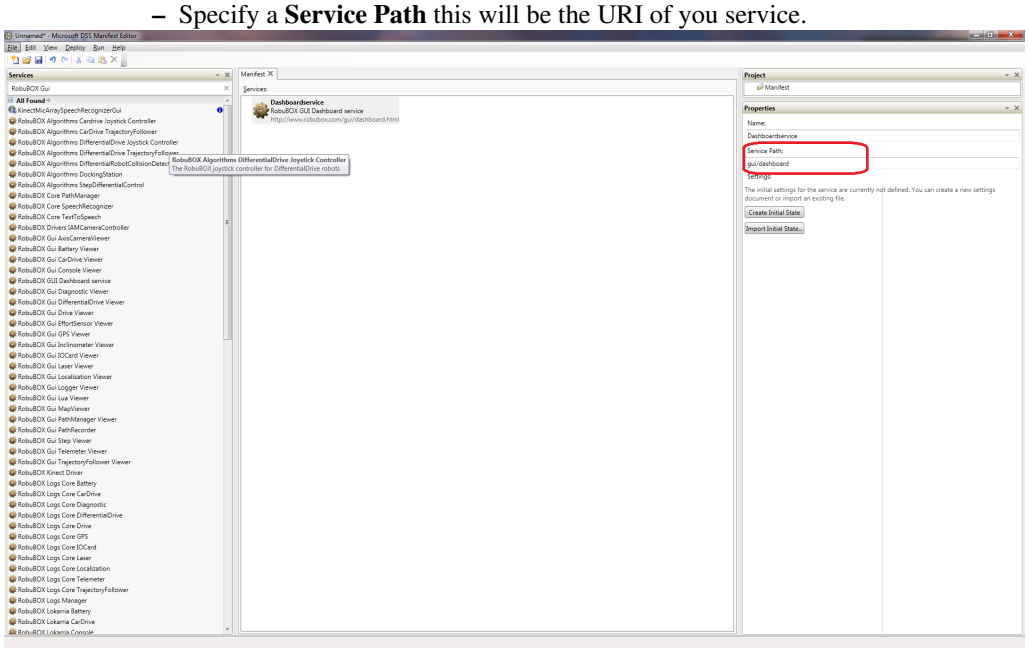
- open Start > all programs > Microsoft Robotics Developer Studio > DSS Manifest Editor



- Click on **DSS Manifest Editor**.
- Add the dashboard service.
  - Inside research tab look for **robuBOX Gui**.

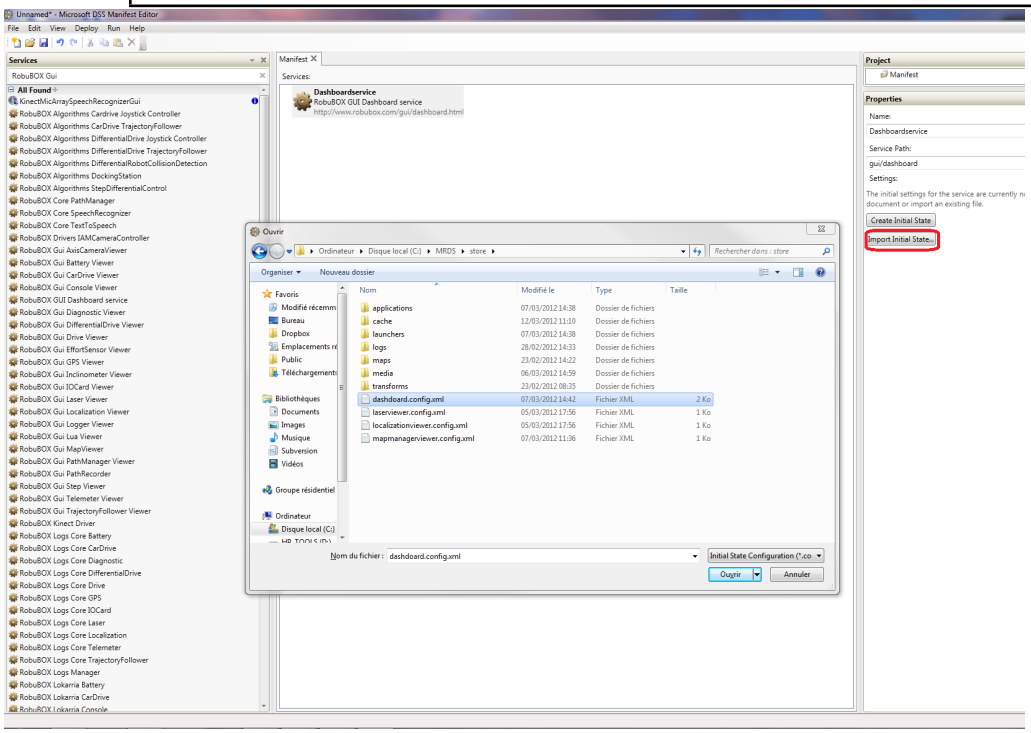


- Drag and drop the **Dashboard** service.
- Specify a **Service Path** this will be the URI of you service.



- Import the default configuration file (*c:\MRDS\store\dashboard.config.xml*).

> **Warning:** **Do not** use *Create Initial State* but import an existing one. MRDS is not able to create the state of this service.



- Add **ConsoleViewer** service.

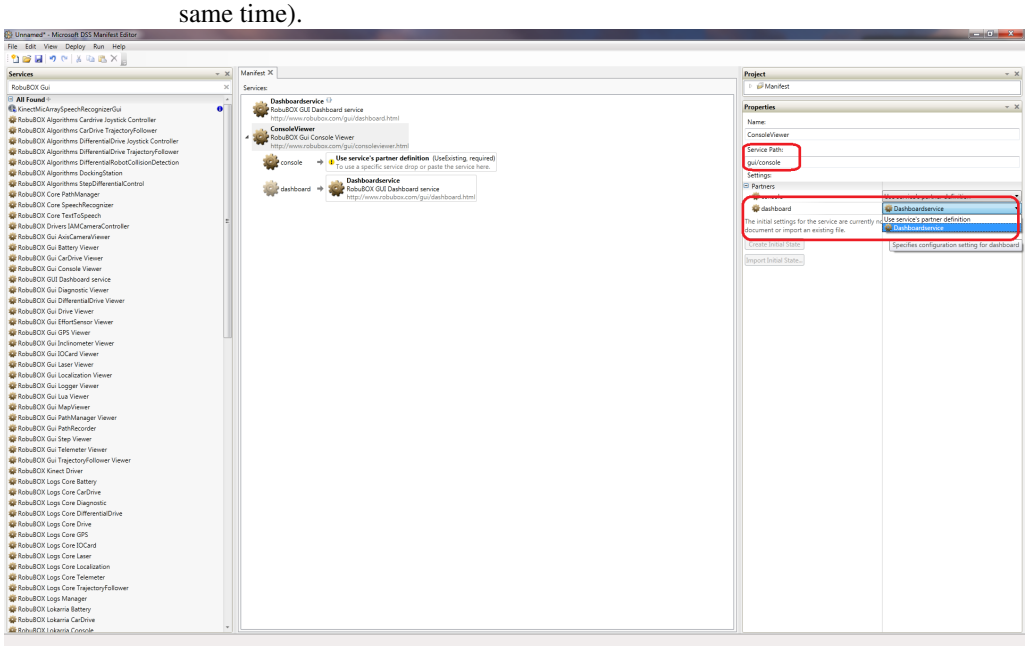
  This service is usefull for debugging, it displays all messages from the MRDS console on the GUI.

  - Drag and drop the **ConsoleWiewer**.

- Specify a **Service Path**, this will be the URI of you service.

Important: When creating your manifest, you must set explicitly the Service Path of each viewer service. The path is used to save the configuration. If you don't set the paths explicitly, they will be generated by the runtime, and will be different at each startup, so you will not be able to save/load your layout configuration correctly.

- Specify the dashboard as partner (if you are running various instances of dashboard at the same time).



- Add **MapViewer** service.

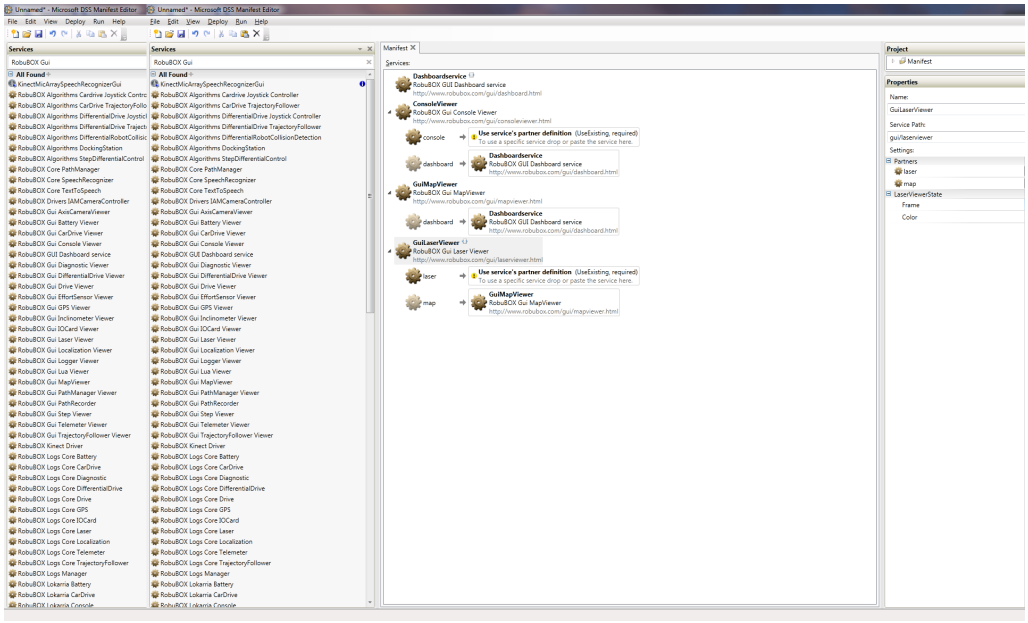
  This service is a base to display sensor data on the same frame.

  - Drag and drop the **MapViewer**.
  - Specify a **Service Path**, this will be the URI of you service.
  - Specify the dashboard as partner (if you are running various instances of dashboard at the same time).

- Add **LaserViewer** service.

  This service displays the laser scan on the **MapViewer** service.

  - Drag and drop the **LaserViewer**.
  - Specify a **Service Path**, this will be the URI of you service.
  - Create of import a configuration file.
  - Specify the **MapViewer** as partner (if you are running various instances of MapViewer at the same time).



You can add all viewers you need.

- Save your manifest inside the directory of your choice.
  - File > Save As...

For this tutorial we save it as: *C:\MRDS\store\applications\tuto\application.manifest.xml*

6.6.2 Start application

- Launch your manifest.

To launch this manifest you need to run it either with the PURE manifest or with simulated robot. Those manifests run a service that expose laser data.

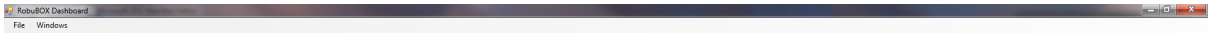
If you want to run it with the real robuLAB:

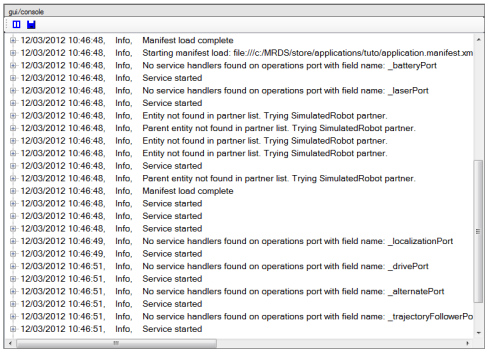
```
dsshost -p:50000 -m:"store\applications\robulab\robot\pure\application.manifest.xml"
-m:"store\applications\tuto\application.manifest.xml"
```

If you want to run it with the simulated robuLAB:

```
dsshost -p:50000 -m:"samples\config\apartment.simulation.manifest.xml"
-m:"store\applications\robulab\simulated\simulation\application.manifest.xml"
-m:"store\applications\tuto\application.manifest.xml"
```

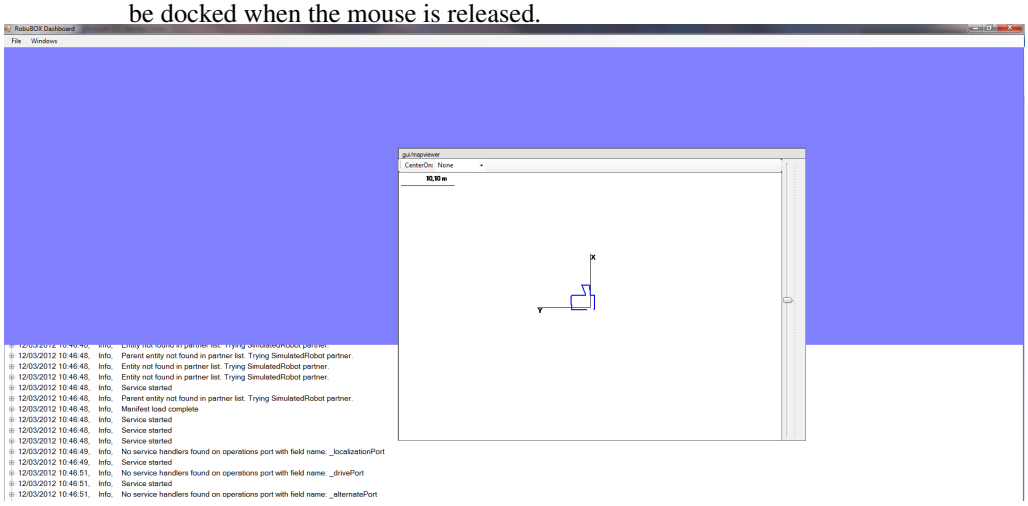
The dashboard will be launched with all viewer superposed as following:



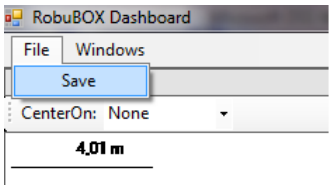


- Drag and drop the viewers.

  Place the viewers at the position of your choice. The blue area indicates where the window will be docked when the mouse is released.



- Save your configuration into Dasboard configuration file.



At each time you will want to start your custom application the positionning of the viewers will be the same.

6.7 How to script robot motions?

In this tutorial you will learn how to create a Lua script and how to start and stop it. The script can be launched from the Navigation application. This application contains a view of all functions that can be understood by the script interpreter.

6.7.1 Create a script file

- Create an empty text file with a text editor.
- Replace the extention *.txt* by *.lua*.

6.7.2 Write a funtion GoTo

- Edit the file created.
- Enter the following lines:

```
function GoTo(destination)
    motionmanager.GoTo(destination);
    status = motionmanager.GetStatus();
    while not(status == "Ready")
    do
        Wait(500);
        status = motionmanager.GetStatus();
    end
    motionmanager.Move();
    while not(status == "Waiting")
    do
        Wait(500);
        status = motionmanager.GetStatus();
    end
end
```

This function requests to generate a path, waits for the robot to became ready to follow the path and sends the **Move** request. The script will end when the robot is waiting i.e once the end of the trajectory is reached.

6.7.3 Write a funtion Dock

This fonction does the same action as the previous one but the destination is the docking station.

- Enter the following lines:

```
function Dock()
    motionmanager.Dock();
    status = motionmanager.GetStatus();
    while not(status == "Ready")
    do
        Wait(500);
        status = motionmanager.GetStatus();
    end
    motionmanager.Move();
    while not(status == "Waiting")
    do
        Wait(500);
        status = motionmanager.GetStatus();
    end
end
```

6.7.4 Write the ScriptStopped() function

This function is called if the user interrupts the script. Its function is to stop properly the robot. In this case we will just stop robot motion.

```
function ScriptStopped()
    status = motionmanager.GetStatus();
    if (status == "Moving") then
        motionmanager.Stop();
    end
end
```

6.7.5 Generate your behaviour

The following lines request the robot to travel between two points. Once the second one is reached, check the battery level. Il the level is under a limit, send the robot to charge.

```
GoTo("destination1");

Wait(3000); -- wait time in miliseconds.

GoTo("destination2");

Wait(3000);

level = battery.GetLevel();
if (level < 30) then
    Dock();

    waittime = 1000 * 60 * 60 * 4; --4h

    Wait(waittime); --wait 4h
end
```

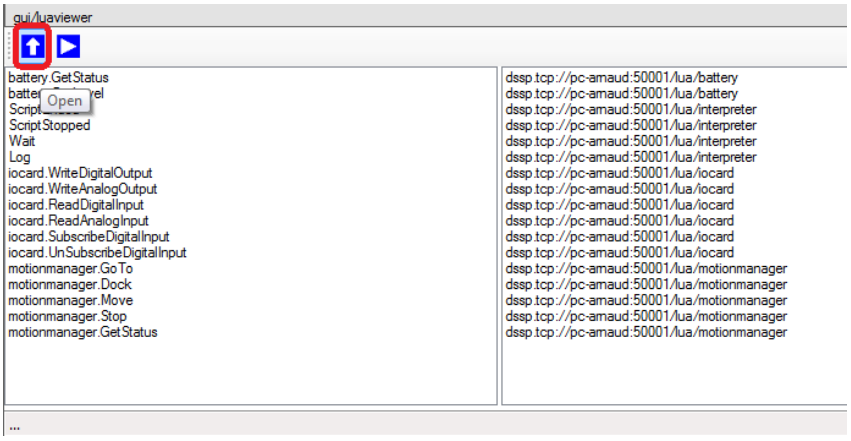
Finally your file should look like this:

```
function GoTo(destination)
    motionmanager.GoTo(destination);
    status = motionmanager.GetStatus();
    while not(status == "Ready")
    do
        Wait(500);
        status = motionmanager.GetStatus();
    end
    motionmanager.Move();
    while not(status == "Waiting")
    do
        Wait(500);
        status = motionmanager.GetStatus();
    end
end

function Dock()
    motionmanager.Dock();
    status = motionmanager.GetStatus();
    while not(status == "Ready")
    do
        Wait(500);
        status = motionmanager.GetStatus();
    end
    motionmanager.Move();
    while not(status == "Waiting")
```

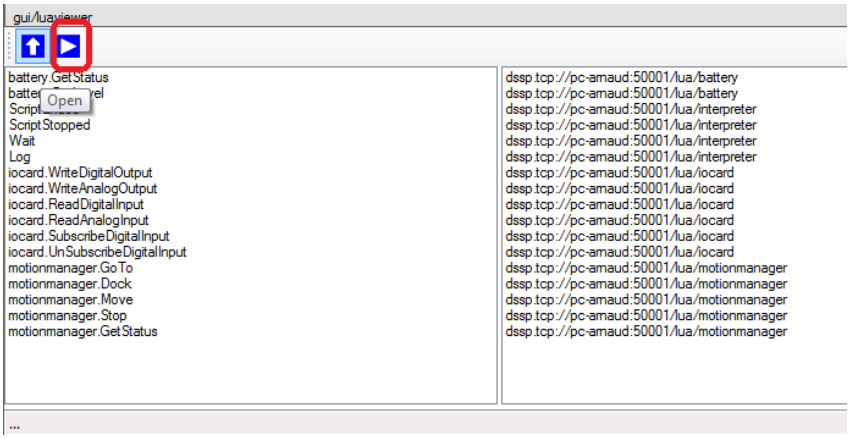
```
    do
        Wait(500);
        status = motionmanager.GetStatus();
    end
end

function ScriptStopped()
    status = motionmanager.GetStatus();
    if (status == "Moving") then
        motionmanager.Stop();
    end
end


GoTo("destination1");

Wait(3000); -- wait time in miliseconds.

GoTo("destination2");

Wait(3000);

level = battery.GetLevel();
if (level < 30) then
    Dock();

    waittime = 1000 * 60 * 60 * 4; --4h

    Wait(waittime); --wait 4h
end

ScriptEnded(); --notify to the interpreter than the script is finished.
```

6.7.6 Launch the script

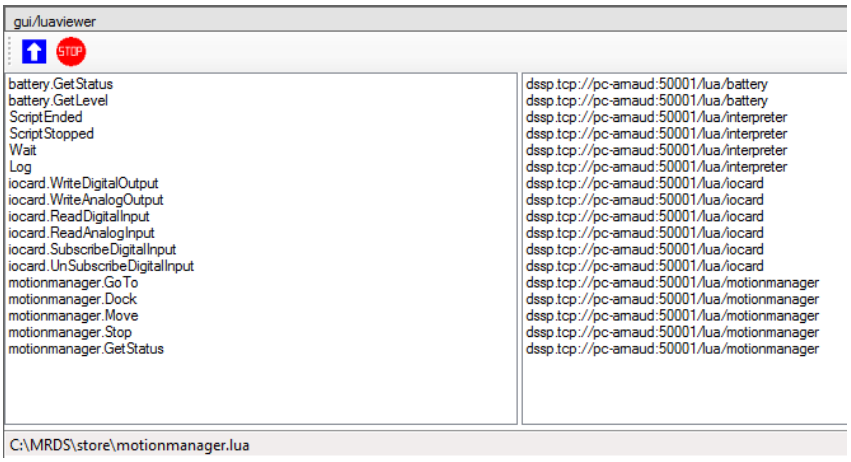
- Start navigation application.
- Click on the up arrow of the lua viewer.



- Select your script.
- Click on the right arrow of lua viewer.



The script can be interrupted by clicking on the **Stop** button form lua viewer (previously the **Start** button).



6.8 How to use HTTP API?

In this tutorial you will learn how to get back map data and control robot motions using HTTP requests.

For more details about the HTTP protocol, see Lokarria package.

All command of this tutorials are sent using curl. Curl is a command line tool for transferring data with URL syntax

- launch the navigation application with the lokarria manifest.

```
dsshost -p:50000 -t:50001 -m:"store\applications\robulab\robot\pure\application.manifest.xml"
-m:"store\applications\robulab\robot\monitoring\application.manifest.xml"
-m:"store\applications\robulab\lokarria\application.manifest.xml"
```

The default URI you want to target are:

```
http://ip_adress:50000/lokarria/navigation/mapmanager

http://ip_adress:50000/lokarria/navigation/motionmanager
```

6.8.1 Get back locations

Use this HTTP GET request to get back all locations previously saved on the map.

```
http://ip_adress:50000/lokarria/navigation/mapmanager/locations
```

You will getback all locations associated to the map under JSON format.

You can use the data returned as destinations for the robot.

6.8.2 Control the robot

- request to generate a path to reach a point without a specific orientation:

  ```
  curl -v -H "Content-type: application/json" -X POST -d {\"X\":4.68,\"Y\":-4.36}
  http://ip_adress:50000/lokarria/navigation/motionmanager/gotopoint
  ```

The data returned is the path generated.

- request to follow the path generated:

  ```
  curl -v -H "Content-type: application/json" -X POST -d {""}
  http://ip_adress:50000/lokarria/navigation/motionmanager/move
  ```

- you can stop the robot motion sendind a stop command:

  ```
  curl -v -H "Content-type: application/json" -X POST -d {""}
  http://ip_adress:50000/lokarria/navigation/motionmanager/stop
  ```