

Cloud Robotics: Using Google Maps for NIFTi Robot Localization

Tomas Nouza
Supervised by: Michal Reinstein

January 30, 2012

Abstract

This document concerns cloud services and intends to bring more insight into the current state of the art of the map visualization services. In principle, cloud services enable distribution of computing power to many computers. Robots often have many sensors but not as much computing power to analyze all sensors in all ways we want in real-time. It is favourable to distribute computation power to other devices to allow robots to use low-voltage processors and hence extend battery life. For this purpose, the Robot Operating system (ROS) [1] is an ideal solution. The ROS is an open source, meta-operating system for robots developed by Willow Garage [2] and is considered a current state of the art among software for robots. It provides services that would be expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. More about the ROS and cloud robotics can be found in [3].

When any robot finishes his mission, it is essential to have a report covering important details regarding the performance and overall operation. The aim of this project is to provide a complete solution including:

- simple and effective sharing of reports with authorized persons
- online precise map visualization
- software background that is easily extendable to any Objects of interest to be included into the report

In regard to the points above, it is beneficial to use the state of the art public cloud services such as the Google services to solve this problem.

Contents

1	Introduction	4
2	Theory, Methodology and Resources	5
2.1	Resources	5
2.2	kmlexport node	6
2.2.1	User documentation	6
2.2.2	Technical documentation	6
2.3	Googlemaps node	6
2.3.1	User documentation	6
2.3.2	Technical documentation	7
3	Evaluation and Results	8
3.1	kmlexport node	8
3.2	Googlemaps node	9
4	Discussion	11
5	Conclusion	12
A	headers	12
A.1	kmlogger.cpp	12
A.2	googlemaps.py	12

1 Introduction

In this work we will look closer to some Google services and their possible application on the search and rescue robot developed as part of the NIFTi project. NIFTi [4] is a European project focusing on tasks in Urban Search & Rescue. It concerns human-robot cooperation in dynamic environments. Sharing of information via various interfaces is crucial for any human-robot cooperation and hence visualization of the robot's trajectory and other things, that robot has recognized during mission (cars, victims, etc.), is key for the robot operators.

The Google Maps API [5] is a powerful tool for visualizing almost everything into online maps. KML (Keyhole Markup Language) [6] is XML like format for describing geographical data. It is an international standard maintained by the Open Geospatial Consortium, Inc. (OGC) [7]. This standard is used in all Google service for geographical input.

Another useful tool is the Google Fusion Tables (GFT) [8]. It is a modern data management and publishing web application that makes it easy to host, manage, collaborate on, visualize, and publish data tables online. Data visualization can be obtained through the Google Maps.

For running any application on the robot is essential to use the ROS [1]. It provides automatic management system for message-passing between nodes [9]. A node is an executable that uses the ROS to communicate with other nodes using topics [10]. Topics are named buses over which nodes exchange the data in the form of messages. When using a computer network connected to the robot(s), the ROS automatically passes messages, so any node running on one computer can interact with all the other nodes. No code implementation is needed since it is open source package based finished solution provided by Willow Garage [2].

Another handy service is the Google Latitude [11]. It saves user current position and publishes it to other authorized users. Every upload of position is saved with a time-stamp creating an easily accessible history. Benefit is the simplicity but it needs internet connection all the time which would be problem in many locations (e.g., tunnel, building on fire, etc.). Main disadvantage is that there is no way to visualize anything more than just the position information.

The main aims of this work are:

- To analyze the current state of the art regarding cloud robotics.
- To analyze the possibilities of the Google API for cloud and web services.
- To implement a node for processing of GPS trajectory and detected objects into KML files using Google standards to ensure easy and robust export to online GFT or Google Earth application.
- To evaluate this node on real GPS data.
- To create a demo program, which in a robust way collects the desired trajectory data and if the internet connection is available uploads these data to the robot's Gmail account.

- To upload final code to the NIFTi project SVN [12].

2 Theory, Methodology and Resources

2.1 Resources

BlueBotics Unmanned Ground Vehicle (UGV) [13] on Fig. 1 is a robotic platform developed for the purpose of the NIFTi project. For its localization it can use the SICK LMS-151 laser scanner, the Point Grey Ladybug 3 omnicaamera and the MTI-G Xsens inertial unit (IMU) with a GPS module. Combination of these should provide sufficient position accuracy of the robot. At this stage, implementation of corresponding data fusion is still under development and hence the GPS module was the only source of absolute position information. Colleges from ETH Zürich developed the *mtig_node* for reading data from GPS and IMU. This node can be considered a driver layer in ROS and hence is used in this work as standard data input.

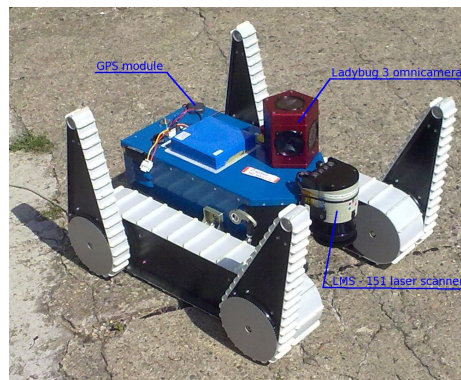


Figure 1: BlueBotics UGV [13] used in NIFTi project [4]

Google Earth (GE) [14] is a free virtual globe, map and geographical information program. It maps the Earth by the superimposition of images obtained from satellite imagery, aerial photography and GIS 3D globe. A strong feature of the GE is the ability to open and process KML files which could also be linked to other KML files that can even change dynamically. In KML there are many kinds of objects we can draw into the globe [6].

Google Fusion Tables (GFT) [8] is a web service that offers functionalities and possibilities similar to an SQL database but it has many above-standard functions. It provides means for visualizing the data with charts, plots, timelines as well as geographical maps. It has data type geometry in which KML code can be inserted. There are many limitations [15], e.g. as how much code can be inserted into one row, but fortunately GFT is labelled as beta which means that it is still under development and we expect there will be many improvements in the future. At this stage there can be inserted only 3 types of objects: *point*,

line and *polygon*. In comparison to the GE it is not as much but it is sufficient for most requirements.

2.2 kmlexport node

2.2.1 User documentation

kmlexport is a ROS node that allows almost real-time drawing of robots trajectory into a KML file, which can be opened in *Google Earth* application as shown in Fig. 3. Once started, the program creates `link.kml` file in `/kml` directory which links to the actual KML file being created. Then it listens to the `NavSatFix` message published on the `/mtig_node/pos_nav` topic and generates a new KML. After execution the *kmlexport*, it is possible to open the `link.kml` in the *Google Earth* and the actual robot's position is plotted and then refreshed every 5 seconds. To run the *kmlexport* node look at chapter 4.1.

2.2.2 Technical documentation

kmlexport can be modified using the following parameters in the source code:

- `LINK_REFRESH_FREQ` sets refresh rate for the `link.kml` in seconds.
- `KML_UPDATE_FREQ` sets the frequency of generation of new KML file (number of messages).
- `LON_ACC` & `LAT_ACC` to reduce number of points in KML file, there is a filtering to a minimum position change (in degrees, LAT means latitude, LON means longitude).
- `ROBOT_NAME` specifies robot name to be shown in title.
- `ALTITUDE` defines height of the trajectory displayed above terrain (in meters).

2.3 Googlemaps node

2.3.1 User documentation

googlemaps is a ROS node that records robot's trajectory and objects¹ that robot recognized and uploads it to the Google Fusion Tables where it can be visualized in the Google maps. As a first thing there is a focus to create a fusion table. If internet connection is not available, this is done later. Default start position is set to the 50.076652 N, 14.416872 E what is a default robot's position in CTU lab. `Odometry` messages published on the `/odom` topic are subscribed and used for robot's localization until the `GPSFix` messages published on the `/mtig_node/pos_nav2` gives a valid GPS value. Then only this `/mtig_node/pos_nav2` is used for localization but the `Odometry` messages are

¹only car detector is implemented at this stage

still used to resolve whenever robot is moving. When robot stays on one spot all GPS messages are averaged for better precision.

Anytime joystick button 4 is pressed and internet connection is available the robot's trajectory and detected object are uploaded to the GFT.

For the case of robot failure files `/tmp/cars.tmp` and `/tmp/tracetrory.tmp` are created. Using the `googlemaps_recovery_cloud.py` node content of these files can be recovered and uploaded to the GFT.

In standard situations use the `/launch/googlemaps.launch` launchfile which launches the `localization_3d/launch/car_detector_and_localization_unfiltered.launch` launch file which detects cars and calculates their position in north-east coordinates, runs the `googlemaps_cloud_joy.py` file and records following topics to a bagfile[16]:

```
/viz/camera_0/image/compressed
/viz/camera_1/image/compressed
/viz/camera_2/image/compressed
/viz/camera_3/image/compressed
/viz/camera_4/image/compressed
/viz/camera_5/image/compressed
/joy
/func_map/objects
/rosout
/odom
and all /mtig_node nodes
```

This file can be freely edited.

To see an uploaded data go to the <https://docs.google.com/> and log in with the username `'cturobot1@gmail.com'` and password `'jednoducheheslo'`. Open the document with corresponding date and time of upload and click *Visualize* → *Map*.

2.3.2 Technical documentation

Parameters can be configured in the source code in the function `def __init__(self):` in class `GoogleMapsClass:` in `googlemaps_cloud_joy.py`. Mainly it is `self.startLat` and `self.startLon` which describes robot's starting position in GPS and `self.LAT_ACC` and `self.LON_ACC` which defines a minimum position change (in degrees, LAT means latitude, LON means longitude) between to points to be marked into the map similar to *kmlexport* 2.2.2. This filtering is implemented for better visualization performance, i.e. to avoid a large number of overlaying objects to be visualized.

Althought Python is an interpreted language and thus not need compilation this configuration method is wrong and in the next version will be replaced using *roscparam* [17]. Parameters will be configured in launchfile and no editing of the source code will be needed.

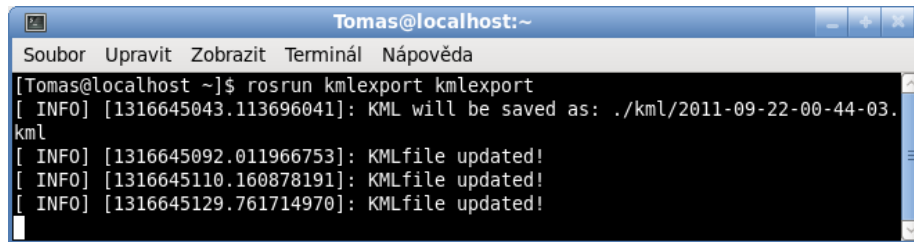
To lower the network load *googlemaps* uses cloud buffer hosted on <http://gmapscloudbuffer.appspot.com/>. Here the trajectory gets its arrow style which triples the number of points. Google account settings are defined in this application so at this stage there is no way to change Google account. This will be implemented in the future.

3 Evaluation and Results

3.1 kmlexport node

The following steps give an overview of how to execute and evaluate the developed ROS nodes:

1. Run *roscore*. Roscore is the first thing you should run when using the ROS. Open a terminal and type:
`roscore`
If successful confirmation that the `process[master]` and `process[rosout-1]` has started will appear.
2. Roscd to *kmlexport* directory. Roscd sets the current path to the package path.
`roscd kmlexport`
3. Run *kmlexport*.
`roscd kmlexport`
`roscd kmlexport`
`roscd kmlexport`
For every 10 positions that have pass through the move filtration (see 2.2.2 LON_ACC and LAT_ACC parameters) the KML file is generated and node informs about it as in the Fig. 2.



```
Tomas@localhost:~  
Soubor Upravit Zobrazit Terminál Nápověda  
[Tomas@localhost ~]$ roscd kmlexport  
[ INFO ] [1316645043.113696041]: KML will be saved as: ./kml/2011-09-22-00-44-03.kml  
[ INFO ] [1316645092.011966753]: KMLfile updated!  
[ INFO ] [1316645110.160878191]: KMLfile updated!  
[ INFO ] [1316645129.761714970]: KMLfile updated!
```

Figure 2: ROS information output to the terminal while the *kmlexport* node is running.

4. Open the `link.kml` file in *Google Earth* application. Result should look similar to Fig. 3.

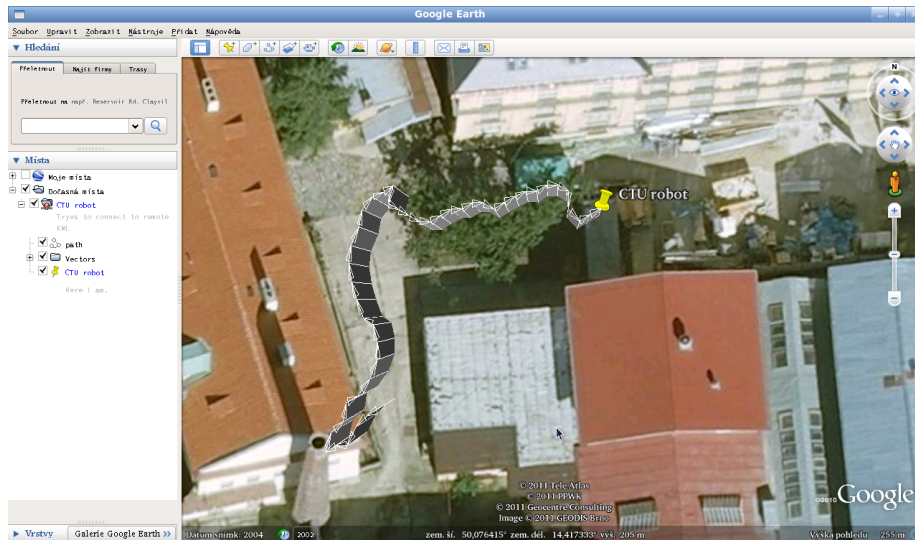


Figure 3: GPS trajectory as displayed in the *Google Earth* application [14] using *kmlexport*.

3.2 Googlemaps node

As a part of this project a simple demo was implemented to demonstrate the basic functionality of the *googlemaps* node. In the */demo* directory there are two files. A bagfile *demo.bag* which contains about 10 minutes long record of robot's trip at CTU and *demo.launch*. Launching the launchfile *demo.launch* will replay this bagfile, launch *googlemaps* node and *joystick* node. Press the joystick button 4 anytime to see the upload process. To see the simple demo:

1. Launch the */demo/demo.launch*

```
roscd googlemaps
roslaunch demo/demo.launch
```
2. Press the joystick button 4 or terminate the program (e.g. Ctrl+C) to initiate the upload process. Fig. 4 shows sample output of terminated program by Ctrl+C.
3. Created table will appear in *Google Docs* [18]. In Fig. 5 is highlighted the new document.
4. To see the map click to the *Visualize* button. Fig. 6 shows where is the button located.
5. Map contains the robot trajectory and detected cars. Clicking on the *Get embeddable link* button will show HTML tag as in Fig. 7. This can be inserted anywhere on the web pages. Structure of the tag is logical and

```

[INFO] [WallTime: 1327932855.681448] 50.076702, 14.416829; calculated from odometry; 40 positions skipped due a minimal position change
[INFO] [WallTime: 1327932856.742263] New car with id: 4 detected
[INFO] [WallTime: 1327932859.138844] 50.076702, 14.416839; calculated from odometry; 42 positions skipped due a minimal position change
[INFO] [WallTime: 1327932860.767784] 50.076702, 14.416849; calculated from odometry; 41 positions skipped due a minimal position change
[INFO] [WallTime: 1327932863.329736] 50.076702, 14.416859; calculated from odometry; 44 positions skipped due a minimal position change
^C[rosbag play-3] killing on exit
[joy-2] killing on exit
[googlemaps-1] killing on exit
[INFO] [WallTime: 1327932864.911205] Sending cars to cloud...

[INFO] [WallTime: 1327932866.553765] Data accepted
New authkey saved
5 cars uploaded

[INFO] [WallTime: 1327932866.554275] 200
[INFO] [WallTime: 1327932866.554694] Sending trajectory to cloud...
[INFO] [WallTime: 1327932867.850341] Data accepted
New authkey saved
Trajectory uploaded

[INFO] [WallTime: 1327932867.850838] 200
shutting down processing monitor...
^C . shutting down processing monitor complete
done

```

Figure 4: Terminal output of just terminated *googlemaps*.

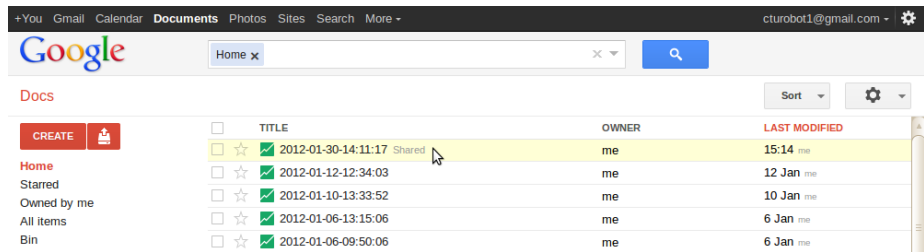


Figure 5: *Google Docs* [18] interface showing the newly generated fusion table.

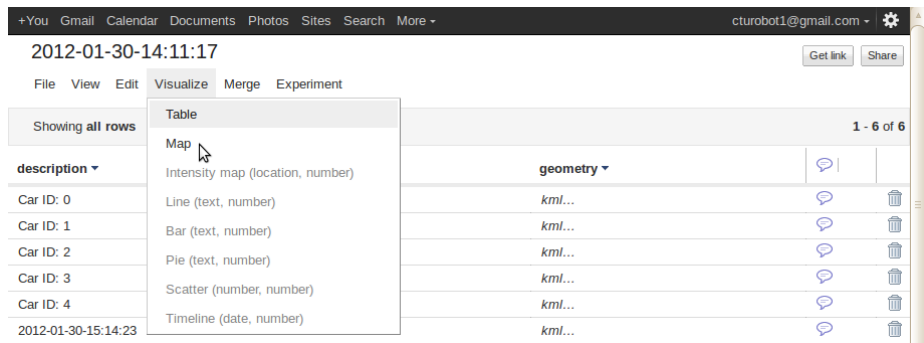


Figure 6: Example of fusion table generated using *googlemaps*.

can be simply generated by any program or script. Table ID is obtained during creation of table.

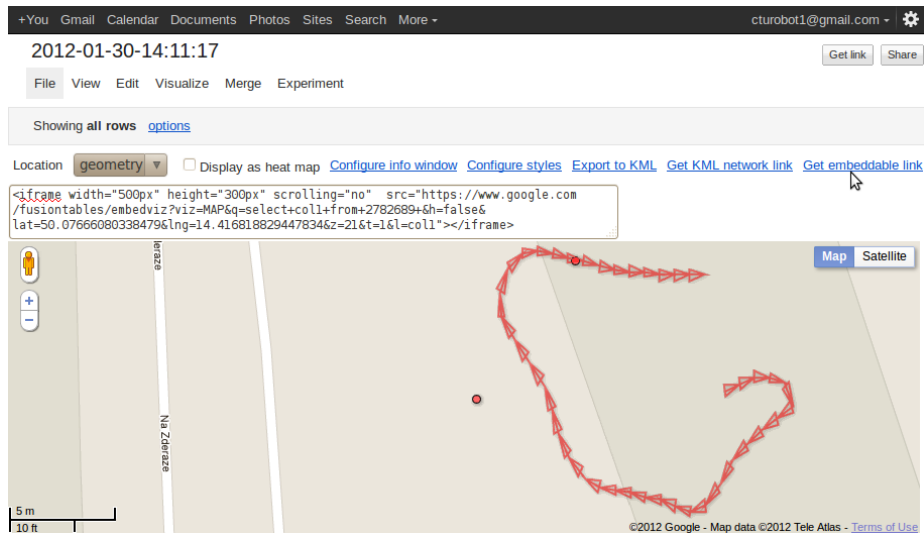


Figure 7: Visualization of the robot trajectory in *Google maps*.

4 Discussion

The actual implementation of both nodes has a flaw. The performance currently strongly depends on the GPS fix and predefined robot's starting position. GPS fix is in urban environments hard to obtain and maintain and manual input of starting position is uncomfortable. If the fix is lost or there is a signal outage, the recorded trajectory is not reliable. In buildings, tunnels and other environments, where we expect most of the robot missions to be carried out, it is usually impossible to get the GPS fix. To avoid dependency on known starting position there is a possibility to buffer every odometry data and recalculate the starting position after first GPS fix. But there will be still problem in case of no GPS fix during whole experiment (e.g. indoor all time).

In the future, the position will be obtained using advanced estimation and data fusion methods from INS, odometry, laser scanner and GPS. There is also an ambition to compare actual robot camera images with downloaded images from Google Street view (GSV) [19] and from its rotation and translation calculate robot's position. Nevertheless it should work only in environment mapped in GSV.

The actual implementation processes only robot's trajectory and detected cars. Next step will be processing and online marking of various detected objects to the map like victims and many more.

5 Conclusion

Two nodes for visualization of robot's trajectory were created. They can be found on the project SVN sites [12]. The first one is the *kmlexport*, which has ability to show actual robot's position, trajectory and other things in *Google Earth* application right during the missions. The second one is the *googlemaps*, which is intended to report and publish data on the internet at the end of the mission or whenever the joystick button 4 is pressed (and when the internet connection is available). The data published are stored in *Google Documents* using the concept of fusion tables and can be shared this way (as any other Google document). Also there is a possibility to add generated Google map to any web page. Specific implementation then depends on the web sites and web server services.

A headers

A.1 kmllogger.cpp

```
int main(int argc, char **argv)
```

creates /kml and /tmp directories, creates time based name of current KML file, subscribes listening /mtig_node/pos_nav callbacks

```
void writeKml(double curLon, double curLat)
```

generate KML file from path.tmp and vectors.tmp file

```
void linkKML(char *name)
```

creates link.kml which links to currently created KML file

```
int writeArrow(double fromLon, double fromLat, double toLon, double toLat)
```

draws an arrow between two points to vectors.tmp file

```
void callback(const sensor_msgs::NavSatFix::ConstPtr& msg)
```

catches NavSatFix messages and if the position has changed greater LON_ACC or LON_ACC than it calls writeArrow()

A.2 googlemaps.py

```
class Car:
```

an object car contains x, y, z coordinates, it's unique id and boolean value if it was uploaded

```
class GoogleMapsClass:
```

a class containing next function

```

def GPS_callback(self,data):
a callback from /mtig_node/pos_nav2 topic

def GPS_nextPosition(self,latitude,longitude,coordCount=1):
checks if position has changed enough to be saved as next point, optional pa-
rameter coordCount affects only info messages

def Odom_callback(self,data):
a callback from /odom topic

def GPS_average(self):
calculates the average GPS position from data cached during robot was not
moving

def joy_callback(self,data):
a callback from /joy topic

def Car_callback(self,data):
a callback from /func_map/objects
if car detected, check if it is a new one and add it to the car list

def NE2GPSLat(self, x, y):
transforms north-easts (y, x) coordinates to latitude

def NE2GPSLon(self, x, y):
transforms north-easts (y, x) coordinates to longitude

def createTable(self):
creates new fusiontable and stores its ID to self.tableID

def upload(self):
uploads all cached data to the fusiontable

def finish(self):
uploads everything and closes backup files

```

References

- [1] Robot Operating System. <http://www.ros.org/wiki/ROS>. Accessed: 16/01/2012.
- [2] Willow Garage. <http://www.willowgarage.com/pages/about-us/overview>. Accessed: 26/01/2012.
- [3] Google I/O 2011: Cloud Robotics, ROS for Java and Android. <http://www.willowgarage.com/blog/2011/05/12/>

- [google-io-2011-cloud-robotics-ros-java-and-android](#). Accessed: 26/01/2012.
- [4] Natural human-robot cooperation in dynamic environments. [www.nifti.eu](#). Accessed: 16/01/2012.
 - [5] Google Maps API Family. <http://code.google.com/apis/maps/index.html>. Accessed: 27/01/2012.
 - [6] KML Reference. <http://code.google.com/apis/kml/documentation/kmlreference.html>. Accessed: 27/01/2012.
 - [7] Open Geospatial Consortium, Inc. <http://www.opengeospatial.org/standards/kml>. Accessed: 27/01/2012.
 - [8] Google Fusion Tables. <http://www.google.com/fusiontables>. Accessed: 16/01/2012.
 - [9] ROS node. <http://www.ros.org/wiki/Nodes>. Accessed: 27/01/2012.
 - [10] ROS topic. <http://www.ros.org/wiki/Topics>. Accessed: 27/01/2012.
 - [11] Google Latitude. <http://googleblog.blogspot.com/2009/02/see-where-your-friends-are-with-google.html>. Accessed: 27/01/2012.
 - [12] NIFTi SVN. https://subversion.dfki.de/nifti/code/ros/stacks/nifti_vision/trunk/googlemaps. Accessed: 19/11/2011.
 - [13] The new NIFTi robot platform. <http://www.nifti.eu/news/the-new-nifti-robot-platform>. Accessed: 27/01/2012.
 - [14] Google Earth. <http://www.google.com/earth/index.html>. Accessed: 27/01/2012.
 - [15] Using KML for Geographic Data. https://developers.google.com/fusiontables/docs/developers_guide#KML. Accessed: 27/01/2012.
 - [16] bagfile. <http://ros.org/wiki/Bag>. Accessed: 30/01/2012.
 - [17] rosparam. <http://ros.org/wiki/rosparam>. Accessed: 29/01/2012.
 - [18] Google Docs. <https://docs.google.com/>. Accessed: 27/01/2012.
 - [19] Google Street view. www.google.com/streetview. Accessed: 30/01/2012.