

Cloud Robotics: Using Google Maps for NIFTi Robot Localization

Author: Tomas Nouza
Supervisor: Michal Reinstein

summer internship at CMP at CTU FEL [1]
from July 12 to September 16 2011 (5 weeks)

1 Abstract

This document concerns cloud services and intends to bring a 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 favorable to distribute computation power to other devices to allow robots to use low-voltage processors and hence extend the battery life. For this purpose, the Robot Operating system (ROS) [2] is an ideal solution. The ROS is an open source, meta-operating system for robots developed by Willow Garage [3] 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 [4].

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 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 Abstract.....	1
2 Introduction.....	3
3 Theory and Methodology and Resources.....	4
3.1 Resources.....	4
3.2 Kmllexport node.....	5
3.2.1. User documentation.....	5
3.2.2. Technical documentation.....	5
3.3 Googlemaps node.....	5
3.3.1. User documentation.....	5
3.3.2. Technical documentation.....	6
4 Evaluation and Results.....	7
4.1 Kmllexport node.....	7
4.2 Googlemaps node.....	8
5 Discussion.....	11
6 Conclusion.....	11
7 Appendices.....	12
A headers.....	12
kmllogger.cpp.....	12
googlemaps.py.....	12
B launch file.....	12
References.....	13

2 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 [5] 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 (victims, cars, etc.), is key for the robot operators.

The Google Maps API [6] is a powerful tool for visualizing almost everything into online maps. KML (Keyhole Markup Language) [7] is XML like format for describing geographical data. It is an international standard maintained by the Open Geospatial Consortium, Inc. (OGC) [8]. This standard is used in all Google service for geographical input.

Another useful tool is the Google Fusion Tables [9]. 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 [2]. It provides automatic management system for message-passing between nodes [11]. A node is an executable that uses the ROS to communicate with other nodes using topics [12]. 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 [3].

Another handy service is the Google Latitude [13]. 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:

1. To analyze the current state of the art regarding cloud robotics.
2. To analyze the possibilities of the Google API for cloud and web services.
3. To implement a node for processing of GPS trajectory (and for the future also detected objects) into KML files using Google standards to ensure easy and robust import to online fusion table or Google Earth application.
4. To evaluate this node on real GPS data.
5. To create a demo program, which in a robust way collects the desired trajectory data and if internet is available uploads these data to the robot's Gmail account.
6. To upload final code to the NIFTi project SVN [14].

3 Theory and Methodology and Resources

3.1 Resources

BlueBotics Unmanned Ground Vehicle (UGV) [10] 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.

Google Earth (GE) [15] 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 object we can draw into the globe [7].

Google Fusion Tables 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 [16], e.g. as how much code can be inserted into one row, but fortunately Google Fusion Tables are labeled as beta which means that they are 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.

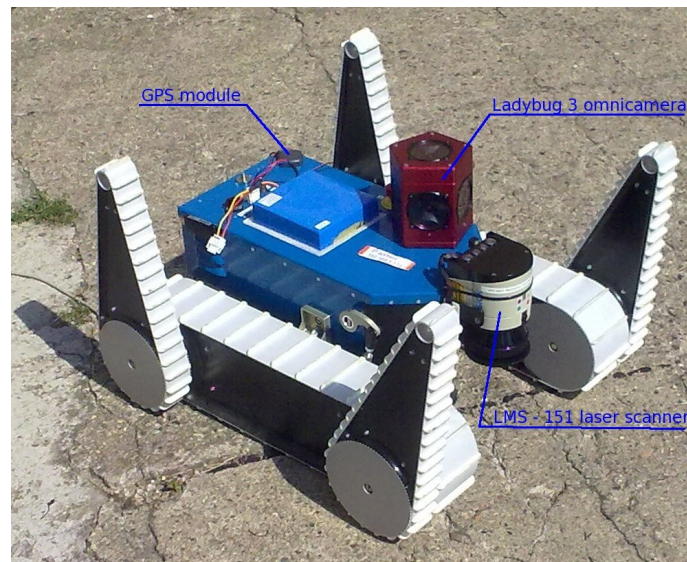


Figure 1: BlueBotics UGV [10] used in NIFTi project [5]

3.2 *Kmlexport* node

3.2.1. 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.

3.2.2. 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).

3.3 *Googlemaps* node

3.3.1. User documentation

The ROS node *googlemaps.py* subscribes the *NavSatFix* messages published on the */mtig_node/pos_nav* topic and when the robot's position changes it is saved into the *path.tmp* file with a timestamp. When the program is terminated, it creates a new table using Google Fusion Tables (GFT) and fills this table with the collected data. At this stage, the program can only upload trajectory from the last mission. The functionality can further be easily extended¹ to uploading additional information regarding detected objects and their exact position; this concerns cars, victims and others as specified in the NIFTi project documentation.

Internet connection is needed only at the end of the mission. If the connection is fault or lost, there is a possibility to still upload the data by running the *googlemapsrecovery.py* node, which processes the data as specified in the *path.tmp* file. Note that this file is replaced every execution of *googlemaps.py*.

Accessing the data over web: Data are store at <http://docs.google.com> (login as *cturobot1* with password *jednoducheheslo*). Default visibility of the document is *private* which means that only people with access to the robot's Google account can see the data. The functionality to share the document with others or to publish the map on any web site is an integral part of the Google Documents services portfolio.

For more information about using this node see chapter 4.2.

¹ The *googlemaps* node for ROS is available in the form of first final version; few other extending functionalities are still under development .

3.3.2. Technical documentation

Parameters:

LON_ACC & LAT_ACC are placed on the line 33, 34. Their purpose is to reduce number of points logged. There is a filtering limitation implemented to a minimum position change (in degrees) for better visualization performance, i.e. to avoid a large number of overlaying objects to be visualized (LAT means latitude, LON means longitude).

Configuration:

For the correct performance, the robot's Google account login and password has to be configured on the line 15. Same line is in *googlemaps.py* and in the *googlemapsrecovery.py*. In order to change the login parameters, for example when different Google account is to be used, both source codes have to be changed.

Communication:

To communicate with the Google server authorization is required. First step is to get an authorization key, which can be obtained using HTTPS request. Here is an example in BASH [17]:

```
curl -s -d Email=cturobot1@gmail.com -d Passwd=jednoducheheslo -d
service=fusiontables https://www.google.com/accounts/ClientLogin
```

The usual Google server response is as follows:

```
SID=DQAAALAAAAA jwq3UMVDOBqClHnuL4TnnfWcSkCEMXsTE9fkIh4u7oMb_aWj3cH
VDX-N5y-mLWuQrNR139J3Ec-
GcJZkuEvNi_j01iS9vYFyVGou9BJEcX1ffn9pVqSn3LcvUApTJTquC5i5Mil04-
zxHZAeUxKotPNoWgfOehT69FvZApNdfALYq7fnMOufRua2040iTyZMDWr4cSpf5dKmk
KUvBF890oXiKDUke23996kuPblg9CQ

LSID=DQAAALAAAAAdWgZj3fs2axT33Qa7Gzv_wbME3AfHqDJqD2yZ5Sgfx1swgJTRhd
sim0BtqY8Goqgxrh9bmBWjhWhpwAqfq785z0UtNY4wf6HGk31UTTuiEqk3YtE_WN6Qq
HjV_UDISkVckCheqnhIAGxt-4icctiu9NmYGq6kuAHICoP6HoS_LEArn7t3cZCC-
VepklvuGKjfb119R2Z-jNd0g2Snc-oYam5_a0UxFzxKPCApSu_z_A

Auth=DQAAALMAAAD7aUXf3JBinvlSuZgOY9eDbVwzBYivTuSUIQZtV-
hVkloucqt favHV5jM5bljjK9UbYEjpozoiYOFJWuGa4JMBRMDvZ4tNvqk-
yO41C2SwSgutgEJFRc8Lqtbqk3UENbXL-
mqdlnzi8QSjVzgs6gat8fikrRCERB44XtS1NCJZU77FutYLi0iLMCNmpf9mIZIqd1_o
PcQ_td0JEMl3eLEgIrvOsREJDauYonKag1OYF4VfB3u0YIpcje15EOPG1mMY
```

The authorization key is the statement after string 'Auth='.

This key allows communicating with the GFT like with a common SQL server. The key must be placed in HTTPS request header as follows:

```
Authorization: GoogleLogin auth=<your_key>
```

Here is a python example for better convenience:

```
import re, httpplib, urllib

params =
"Email=cturobot1@gmail.com&Passwd=jednoducheheslo&service=fusiontab
les"
conn = httpplib.HTTPSConnection('www.google.com')
```

```

conn.request('POST', '/accounts/ClientLogin?' + params, params)
response = conn.getresponse()
data = response.read()
if response.status == 200:
    authRE = re.compile('uth=\S+')
    key = authRE.search(data).group(0)
    print('a'+key)

```

Standard usage of the developed *googlemaps.py* node is based on the assumption that robot has internet connection at the start of the mission. First it tries to contact Google account server to get an authorization key. New key is saved to *.key* file if obtained. When robot compiles HTTPS header, it looks into *.key* file for authorization key. Generally there is no problem, if robot doesn't obtain new authorization key because the old one is valid for about two weeks. Updating the authorization key is recommended to avoid authorization problems.

4 Evaluation and Results

4.1 *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 terminal and type:

```
roscore
```

If successful confirmation that the process[*master*] and process[*rosout-1*] has started will appear.

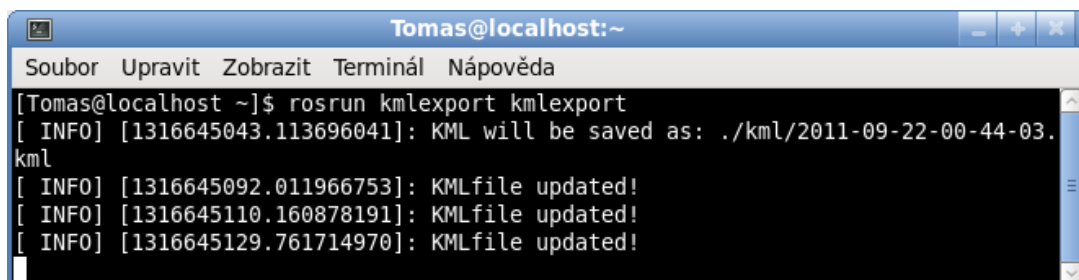
2. Roscd to *kmlexport* directory. Roscd sets the current path to the package path.

```
roscd kmlexport
```

3. Run *kmlexport*.

```
roslaunch kmlexport kmlexport
```

For every 10 positions that have pass through the move filtration (see 3.2.2. *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 ~]$ roslaunch kmlexport 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.

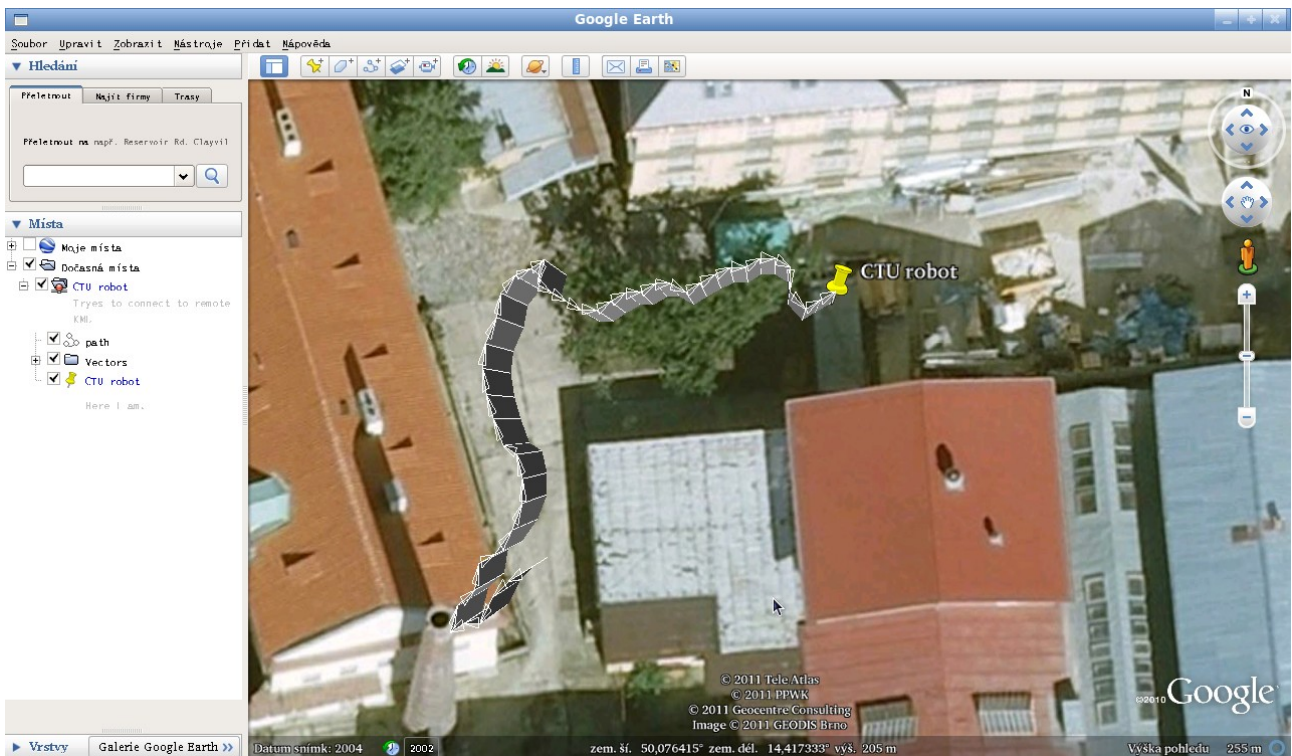


Figure 3: GPS trajectory as displayed in the Google Earth application [15] using *kmlexport*.

4.2 Googlemaps node

As a part of this project a simple demo was implemented to demonstrate the basic functionality of the *googlemaps* node. To see the simple demo, run launch file *googlemaps.launch*, that will play bagfile *2011-08-12-11-07-02.bag.active.gps.bag*:

```
roslaunch googlemaps googlemaps.launch
```

Or run the node as shown below:

1. Run *roscore* as in 4.1.
2. Run *googlemaps.py*. Fig. 4 shows standard output where *googlemaps* successfully obtained new authorization key.

```
roslaunch googlemaps googlemaps.py
```

```
Tomas@localhost:~
Soubor Upravit Zobrazit Terminál Nápověda

[Tomas@localhost ~]$ roslaunch googlemaps googlemaps.py
[INFO] [WallTime: 1316646165.875418] New authKey saved
[INFO] [WallTime: 1316646168.413998] 50.0765647888 14.417140007
[INFO] [WallTime: 1316646170.615261] 50.0765647888 14.4171504974
[INFO] [WallTime: 1316646171.861596] 50.0765724182 14.4171609879
[INFO] [WallTime: 1316646173.266180] 50.0765762329 14.4171714783
```

Figure 4: *googlemaps* running.

3. Terminate the program (e.g. Ctrl+C) to upload data to the Google Fusion Tables. Fig. 5 shows standard output during uploading procedure.


```

Tomas@localhost:~/Dokumenty/FEL/CMP/ros/googlemaps
Soubor Upravit Zobrazit Terminál Nápověda
[INFO] [WallTime: 1316170467.061093] 50.0765380859 14.4171295166
^C[INFO] [WallTime: 1316170508.727350] Writing data to fusiontable before exit..
.
[INFO] [WallTime: 1316170519.636175] Fusiontable with tableid 1515732 successfuly
ly created
[INFO] [WallTime: 1316170519.638177] Uploading path...
[INFO] [WallTime: 1316170524.624122] Path uploaded as row 1
[INFO] [WallTime: 1316170524.626364] Uploading path...
[INFO] [WallTime: 1316170528.507978] Path uploaded as row 201
[INFO] [WallTime: 1316170528.509080] Sending last position...
[INFO] [WallTime: 1316170535.504479] Last position uploaded as row 401
[INFO] [WallTime: 1316170535.505495] Everything successfully uploaded
[Tomas@localhost googlemaps]$

```

Figure 5: Terminating the node will upload data to the Google Fusion Tables [9].

4. Created table will appear in Google Docs [18]. In Fig. 6 is highlighted the new document.

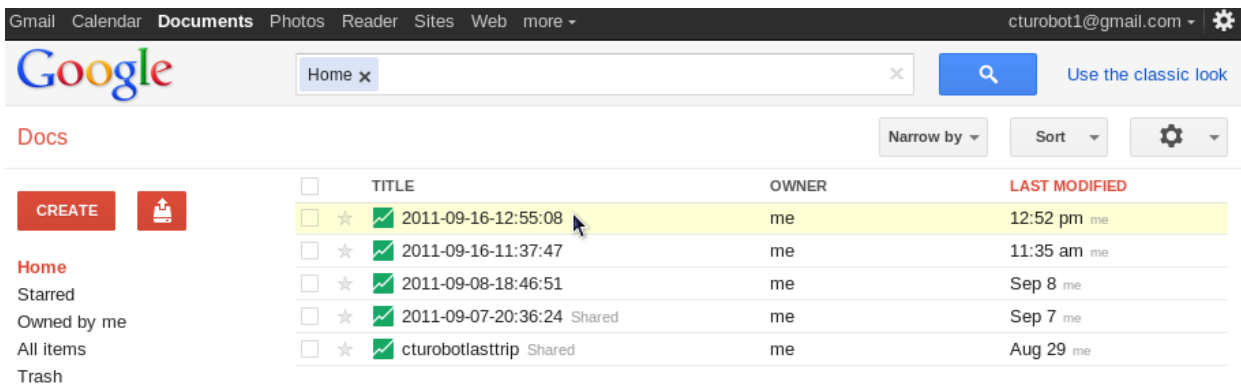


Figure 6: Google Docs interface showing the newly generated fusion table [18].

5. To see the map click to the Visualize button. Fig. 7 shows where is the button located.

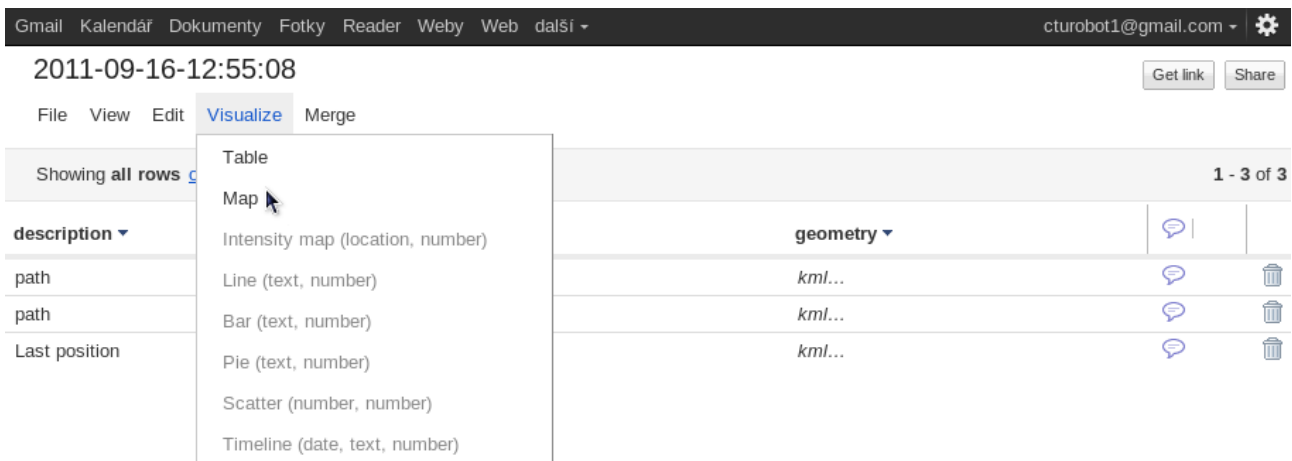


Figure 7: Example of fusion table generated using *googlemaps.py*.

6. Map contains robot trajectory and last position. Clicking on the *Get embeddable link* button will show HTML tag as in Fig. 8. This can be inserted anywhere on the web pages. Structure of the tag is logical and can be simply generated by any program or script. Table ID is obtained during creation of table. In Fig. 9 there is an example of one larger trip placed on <https://sites.google.com/site/cturobotlastrip/>

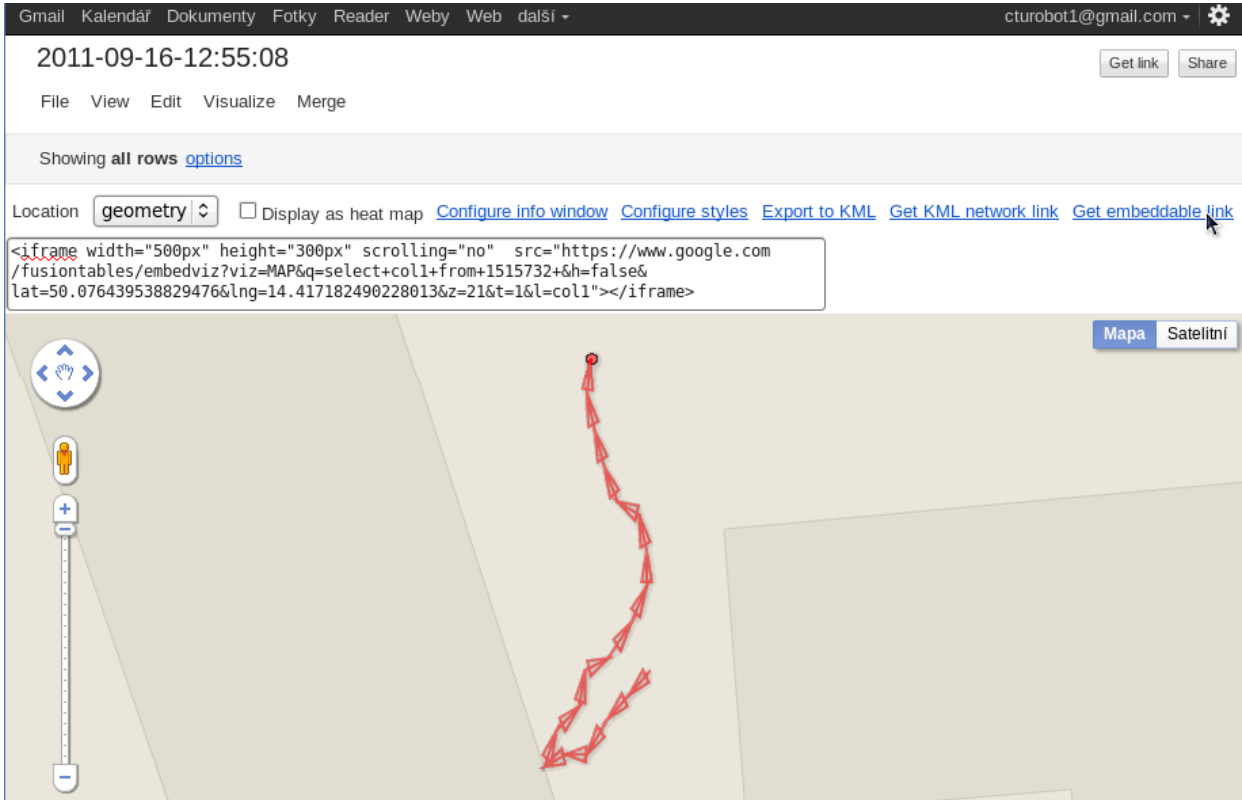


Figure 8: Visualization of the robot trajectory in Google maps.

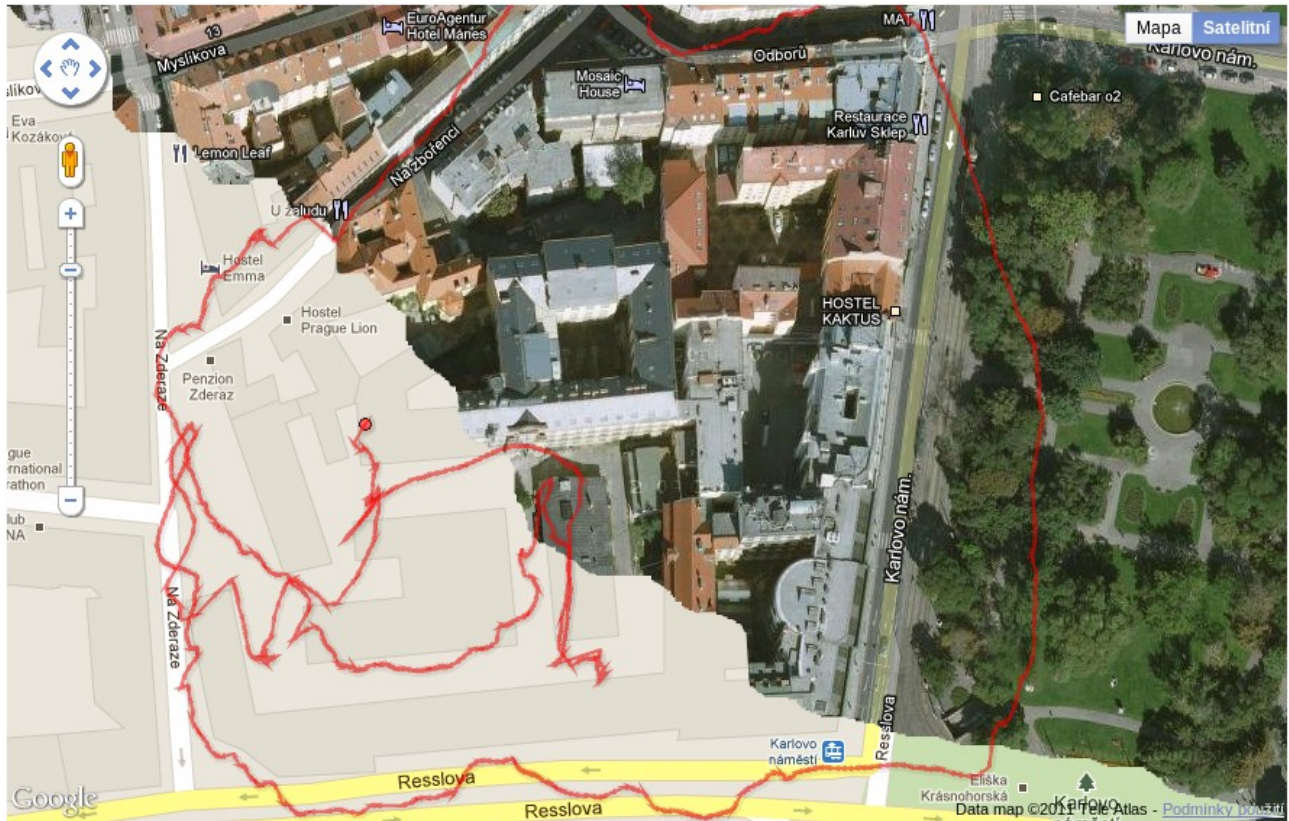


Figure 9: Visualization of robot's trip in Google maps when inserted into a web page. Satellite and normal projection were joined in GIMP [19].

5 Discussion

If *googlemaps.py* is running from a launch file, it has 15 second to exit after receiving SIGINT signal before it is forced by SIGKILL. SIGINT and SIGKILL are POSIX defined signals to process control [20]. This means if *googlemaps.py* wants to upload more than approximately 30 positions (what usually wants), node will terminates before uploading whole *path.tmp* file. At this stage Google server does not receive HTTPS requests fast enough. Solution is to use *googlemapsrecovery.py* program which reads *path.tmp* file and uploads it into Google Fusion Tables. In the future a cloud buffer will be created in this project to solve this problem.

Actual implementation of both nodes has flaw. The performance currently strongly depends on the GPS fix, which is in urban environments hard to obtain and maintain. 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. In the future, the position will be obtained using advanced estimation and data fusion methods from INS, odometry, laser scanner and GPS.

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

6 Conclusion

Two nodes for visualization of robot's trajectory were created. They can be found on the project SVN sites [14]. 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.py*, which is intended to report a publish data on the internet at the end of the mission (or 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.

7 Appendices

A headers

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()*

googlemaps.py

```
def main():
```

makes initialization and subscribes listening */mtig_node/pos_nav* callbacks

```
def getKey():
```

tries to get an authorization key from Google accounts and saves it into *.key* file

```
def callback(data):
```

catches *NavSatFix* messages and if the position has changed greater LON_ACC or LON_ACC than saves it into */tmp/path.tmp* file with a timestamp

```
def writeArrow(fromLat, fromLon, toLat, toLon):
```

returns a string containing arrow in KML syntax between two points

```
def upload():
```

is called *on_shutdown()* and saves content of */tmp/path.tmp* into Google Fusion Tables

B launch file

A launch file provides running more nodes in one command and configuring each of them with parameters. For the demo to *googlemaps.py* (see 4.2.) following launch file was used:

```
<launch>
  <node pkg="roscpp" name="play" type="play" args="$(find
googlemaps)/demo/2011-08-12-11-07-02.bag.active.gps.bag"/>
  <node pkg="googlemaps" name="googlemaps" type="googlemaps.py"
output="screen" cwd="node"/>
</launch>
```

The second line runs bagfile stored in package *googlemaps* in directory *demo*. The third line runs the node *googlemaps.py* with output set to the terminal and current working directory set to place, where the *googlemaps.py* file is stored.

References

- [1] *Center for Machine Perception at Czech Technical University in Prague*. Available from (September 19, 2011): <http://cmp.felk.cvut.cz>.
- [2] *Robot Operating System*. Available from (September 19, 2011): <http://www.ros.org/wiki/ROS>.
- [3] *Willow Garage*. Available from (September 19, 2011): <http://www.willowgarage.com/pages/about-us/overview>.
- [4] *Google I/O 2011: Cloud Robotics, ROS for Java and Android*. Available from (September 19, 2011): <http://www.willowgarage.com/blog/2011/05/12/google-io-2011-cloud-robotics-ros-java-and-android>.
- [5] *Natural human-robot cooperation in dynamic environments*. Available from (September 19, 2011): <http://www.nifti.eu/mission>.
- [6] *Google Maps API Family*. Available from (September 19, 2011): <http://code.google.com/apis/maps/index.html>.
- [7] *KML Reference*. Available from (September 19, 2011): <http://code.google.com/apis/kml/documentation/kmlreference.html>.
- [8] *Open Geospatial Consortium, Inc*. Available from (September 19, 2011): <http://www.opengeospatial.org/standards/kml/>.
- [9] *Google Fusion Tables API*. Available from (September 19, 2011): http://code.google.com/apis/fusiontables/?ft_source=tour_nav&_utma=1.1220465619.1316475456.1316475456.1316475456.1&_utmb=1.1.10.1316475456&_utmc=1&_utmz=1.1316475456.1.1.utmcsr=google|utmccn=%28organic%29|utmcmd=organic|utmctr=google%20fusion%20tables&_utmv=-&_utmh=123550731.
- [10] *The new NIFTi robot platform*. Available from (September 19, 2011): <http://www.nifti.eu/news/the-new-nifti-robot-platform>.
- [11] *ROS node*. Available from (September 19, 2011): <http://www.ros.org/wiki/Nodes>.
- [12] *ROS topic*. Available from (September 19, 2011): <http://www.ros.org/wiki/Topics>.
- [13] *Google Latitude*. Available from (September 19, 2011): <http://googleblog.blogspot.com/2009/02/see-where-your-friends-are-with-google.html>.
- [14] *NIFTi SVN*. Available from (September 19, 2011): https://subversion.dfki.de/nifti/code/ros/stacks/nifti_vision/trunk/googlemaps.
- [15] *Google Earth*. Available from (September 19, 2011): <http://www.google.com/earth/index.html>.
- [16] *Using KML for Geographic Data*. Available from (September 19, 2011): http://code.google.com/apis/fusiontables/docs/developers_guide.html#KML.
- [17] *Bash*. Available from (September 19, 2011): <http://www.gnu.org/software/bash/bash.html>.
- [18] *Google Docs*. Available from (September 19, 2011): <https://docs.google.com/>.
- [19] *GNU Image Manipulation Program*. Available from (September 19, 2011): <http://www.gimp.org/>.
- [20] *Signal handling*. Available from (September 19, 2011): <http://www.yolinux.com/TUTORIALS/C++Signals.html>.