Reference Manual for the software of Image-Based Indoor Topological Navigation with Collision Avoidance for Pepper Robot*

Suman Raj Bista Belinda Ward Peter Corke

Date: 01/08/2019

1 Overview

This report provides an overview of using the software of Image-Based Indoor Topological Navigation with Collision Avoidance for Resource-Constrained Mobile Robots [1]. The source code is open-sourced and has been used for the experiments discussed in the paper [1]. The source codes for the reference image selection and image-based navigation using line segments are based on [2]. The source codes are provided "as-is" without any warranty. Before using the code, you agree to use the code at your own risk. The authors are not responsible or liable for any damages incurred using this code.

2 Installation

2.1 Source code

The source code is available on `https://github.com/suuman/pepper_navigation.git`.

2.2 Dependencies

1. MATLAB (for mapping, has to be downloaded separately).
2. OpenCV (available in ppa of Ubuntu or has to be built from source with extra modules).
3. Boost (available in ppa of Ubuntu or has to be built from source).
4. naoqi_libqi and naoqi_libqicore (ROS Packages available for Kinetic and Melodic).
5. Arpack, Lapack, Blas and SuperLU (for line segment matching, available in ppa of Ubuntu).
6. Qt5 library (Core, Widgets, Test) (for GUI, available in ppa of Ubuntu).

*The authors are with ARC Centre of Excellence for Robotic Vision and QUT Centre for Robotics, School of Electrical Engineering and Robotics, Queensland University of Technology (QUT), Brisbane, Australia.
This research was supported by the Australian Research Council Centre of Excellence for Robotic Vision (project # CE140100016) and funded by the Queensland Government under an Advance Queensland Grant.

7. Graphviz (GVC, CGRAPH, CDT) (for visualizing topological graph, available in ppa of Ubuntu),
8. Pepper Virtual Machine [3] and VirtualBox (to compile binaries that can run onboard on the Pepper Robot, has to be downloaded separately).
9. qgv and pepper_qi (custom libraries that are shipped with this repository).

2.3 Build Instructions

1. Get Source codes from the repository.

   ```
   $ git clone https://github.com/suuman/pepper_navigation.git
   ```

2. Install required dependencies (if required).

   ```
   $ sudo apt install libopencv-dev libopencv-contrib-dev libboost-all-dev
   $ sudo apt install ros-melodic-naoqi-libqi ros-melodic-naoqi-libqicore
   $ sudo apt install libarpack2-dev libblas-dev liblapack-dev libsuperlu-dev
   $ sudo apt install qt5-default
   $ sudo apt install libcdt5 libcgraph6 libcgv6 libgraphviz-dev
   ```

3. Build executables required for mapping.

   ```
   $ cd pepper_navigation/mapping
   $ ./build_linematching.sh
   ```

   The above command will build executables ***detectinesED*** and ***matchlines*** in ./linematching folder and is equivalent to the following commands

   ```
   $ cd linematching/Linematching_iso && mkdir build $$ cd build
   $ cmake ..  && make -j8
   ```

4. Build Navigation code.

   ```
   $ cd pepper_navigation
   $ ./compile.sh
   ```

   The above command will build executables ***peppernav_gui***, ***peppernav_inside***, ***peppernavigation***, and **peppernavigationoff** in ./build folder and is equivalent to the following commands.

   ```
   $ cd pepper_navigation && mkdir build $$ cd build
   $ cmake ..  && make -j8
   ```

   If the CMake options and Path has to changed from default

   ```
   $ ccmake ..  or cmake-gui ..
   ```
   then Configure and Generate Makefile.

   ```
   $ make -j8
   ```

To run the executables ***peppernav_inside*** and ***peppernavigation*** onboard on the Pepper robot, they must be compiled in the Pepper Virtual Machine. For details, please refer to the Sect. 3.4(4b).

3 Tutorial

3.1 Overview

Topological Navigation

For topological navigation, the topological map as discussed in [1] is required. The details of creating the topological map are discussed in Sect. 3.3 of this report.
The GUI version of topological navigation can be run from the executable ***peppernav_gui***. This version communicates with the Pepper robot remotely via PC and expects user input for the destination.
The Non-GUI version of topological navigation can be run from the executable ***peppernav_inside***. This version of the code is capable of running onboard on the Pepper robot if compiled on the Pepper Virtual Machine available from [3].
To get started, please refer to navmain/nav_peppergui.cpp for GUI version and navmain/nav_pepper_inside.cpp for Non-GUI version that is capable of running inside the Pepper robot.

Navigation over a Single Sequence of Reference Images

This version of the code requires the reference image sequence of the path that the Pepper robot has to follow. This version is similar to [2] with collision avoidance.
The executable ***peppernavigation*** performs navigation with obstacle avoidance and also can run entirely onboard on the Pepper robot.
The executable ***peppernavigationoff*** can be used to perform offline image-based localization with a reference image list.
To get started, please refer to navmain/nav_pepperonline.cpp for online navigation along the reference image sequence and navmain/nav_pepperoffline.cpp for offline localization with the reference image list.

3.2 Selection of Reference Images based on Line Segment Matching [2]

The code for the mapping is in Matlab. Make sure the line detection and matching codes have been properly built and executables ***detectinesED*** and ***matchlines*** have been placed in the correct folder i.e. in the *./linematching* folder. For selecting the reference images, we need to provide the path of the image sequence folder and the folder to store the reference images.

1. Open *selectRefImages.m*
2. Set the path of the image sequence. e.g `imseq = '../roboroom/imgs_acquired'`
3. Set the path to store the reference images. e.g `refimpath='../roboroom/ref_imgs'`
4. Run *selectRefImages.m*

The reference image folder will contain

1. Reference Images.
2. The text files with the detected line segments and their descriptors. There is one *.txt* file for each reference image.

3.3 Creating Topological Map

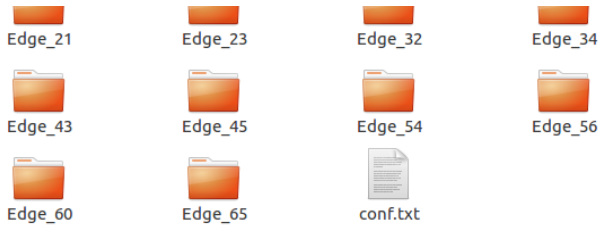
1. Organize the selected reference images in the topological map. For each edge, a separate sub-folder should be used to store the corresponding reference images. e.g. *Edge_01, Edge_02, Edge_10, ...* Please refer to *data/tmap* directory.

2. Define the edges and their link in the *genConfigFile.m* located in *mapping* directory as shown below in format <*Subfolder name*><*start node*><*end node*>.

```
edges = [
         "Edge_01" "0" "1"
         "Edge_12" "1" "2"
         "Edge_23" "2" "3"
         "Edge_34" "3" "4"
         .......
```

3. Run *genConfigFile.m* to generate *conf.txt* that contains the link between the reference images among the adjacent edges. This is essentially important for smooth turnings.

The *data/tmap* folder will look similar as below.



3.4 Topological Navigation

This section assumes that the Pepper Robot is connected to the PC with the known IP address and the topological map is available. The folder "pepper_navigation" is the base folder of the repository. The topological map is available in the *data/tmap* folder. First we define our topological map in the code (*navmain/nav_peppergui.cpp* or *navmain/nav_pepper_inside.cpp*). Steps 1, 2 and 3 explains the process of defining the topological map in *nav_peppergui.cpp* and *nav_pepper_inside.cpp*. These steps 1, 2 and 3 are a one-time process for the given topological map.
Before compiling the code, it is assumed that `CMAKE_PREFIX_PATH` for naoqi_libqi, naoqi_libqicore and OpenCV has been set correctly in the *CMakeLists.txt* file.

1. Specify the nodes and the edges the topological graph in the code (*navmain/nav_peppergui.cpp* or *navmain/nav_pepper_inside.cpp*).

```
enum nodes{
    PepperHome, Harvey, Rob1, Peter, RObb2, Cartman, Belinda,
    Manipulation, Robotics,
    N};
astar::node name[]{
          "PepperHome", "Harvey", "Rob1", "Peter", "RObb2", "Cartman", "Belinda",
          "Manipulation", "Robotics" };
```

```
astar::edge edge_array[] = {
            astar::edge(PepperHome,Harvey), astar::edge(Harvey,Rob1),
            astar::edge(Rob1,Peter), astar::edge(Peter,ROb2),
            astar::edge(ROb2,Cartman), astar::edge(Cartman,Belinda),
            astar::edge(Belinda,PepperHome),
            astar::edge(Harvey,Manipulation), astar::edge(Manipulation,Cartman),
            astar::edge(Rob1,Robotics), astar::edge(Robotics,ROb2) };
```

```
astar::cost weights[] = {        //total number of reference images in the given edge
            35, 34,
            58, 43,
            35, 57,
            12,
            21, 23,
            18, 15 };
```

2. Set display and image settings in the code (*navmain/nav_peppergui.cpp*).

```
d->saveimage(true);                                    // save images of the navigation
d->displayimage(true);                              // display navigation images on realtime
d->showfeat(false);                               // draw features used for IBVS in the image
d->setDisptime(waittime);                                 // wait for waittime > 0 (in ms).
```

3. Set online mode and Pepper robot's IP address in the code (*navmain/nav_peppergui.cpp*).

```
mode = 0;                          // mode = 0 => online mode, mode = 1 => offline mode.
opt_ip = "172.19.226.236";                              // IP address of the Pepper robot.
```

4. Compile and Run (in terminal).

   (a) **GUI Version**
   This version of the code must be executed on the external PC where the communications with the Pepper robot are done remotely via the naoqi interface.

I) To build executable *peppernav_gui* in ./build directory, run the script *compile.sh* .

```
$ ./compile.sh                        # mkdir build && cd build && cmake ..  && make -j8
```

II) To run GUI version of topological navigation, use the script *run.sh* (do not forget to set path of topological graph and
IP address of Pepper Robot).

```
$ ./run.sh                                                  # execute the code with the required arguments.
```

The above commands basically performs following operations:

```
$ ./peppernav_gui <topogarph_path> <Pepper_IP>
                                  # e.g. ./peppernav_gui ../data/tmap 172.19.226.236
```

(b) **To run onboard on the Pepper Robot.**

I) To run code onboard on Pepper Robot, it must be compiled on Pepper Virtual Machine.
Start this virtual machine on program like VirtualBox. Now, transfer the folder "*pepper_naviagtion*" that contains the codes to the Pepper Virtual Machine.

```
$ scp -P <port_no> -r <path>/pepper_navigation/ nao@localhost:<$HOME directory>
        # e.g.  scp -P 2222 -r /home/suman/pepper_navigation/ nao@localhost:/home/nao
```

II) Log in to the Pepper Virtual Machine via SSH.

```
$ ssh nao@localhost -p <port_no>                    # e.g.  ssh nao@localhost -p 2222
$ cd pepper_navigation
```

III) To build executable *peppernav_inside* in ./build directory, run *compile.sh* script in the Pepper Virtual Machine.

```
$ ./compile.sh              # mkdir build && cd build && cmake ..  && make -j8
```

IV) Transfer the entire project directory "*pepper_naviagtion*" that contains executable *peppernav_inside* from Pepper Virtual Machine to Pepper Robot.

```
$ scp -P <port_no> -r nao@localhost:<$HOME directory>/pepper_navigation <path>
                      # e.g.  scp -P 2222 -r nao@localhost:/home/nao/pepper_navigation
/home/suman/pepper_navigation
$ scp -r <path>/pepper_naviagtion nao@<Pepper_Ip>:<$HOME directory>
            # e.g.  scp -r /home/suman/pepper_navigation nao@172.19.226.236:/home/nao
```

V) Log in to Pepper Robot via SSH.

```
$ ssh nao@<Pepper_IP>                                          # e.g.  ssh nao@172.19.226.236
$ cd pepper_navigation
```

VI) To run topological navigation onboard on Pepper robot, use *run_onboard.sh* script (do not forget to set path of topological graph).

```
$ ./run_onboard.sh                                  # execute the code with the required arguments.
```

The above command basically performs following operation:

```
$ ./peppernav_inside <topogarph_path>           # e.g.  ./peppernav_inside ../data/tmap
```

3.5 Navigation over a Single Sequence of Reference Images

The code to perform the online navigation over a single path defined by the reference image list is *navmain/nav_pepperonline.cpp*. For online navigation, we have to set the correct IP address of the Pepper robot in the code (*navmain/nav_pepperonline.cpp*).

```
std::string opt_ip = "172.19.226.236";              // IP address of the Pepper robot.
```

The code must be built in the Pepper Virtual Machine and then transferred to the Pepper robot as discussed above (in Sect. 3.4(4b)) to perform online navigation onboard on the robot.

The code *navmain/nav_pepperonline.cpp* performs the offline Image-based localization in the reference image list. This offline mode uses the image sequence instead of the robot.

I) To build executables *peppernavigation* and *peppernavigationoff* in ./build directory, run the script *compile.sh*.

```
$ ./compile.sh                          # mkdir build && cd build && cmake ..  && make -j8
```

II) To run online navigation, run following command in the terminal.

```
$ ./peppernavigation <reference _images_path>
                                  # e.g. ./peppernavigation ../data/offlinetest/kfls
```

III) To run offline localization, run following command in the terminal.

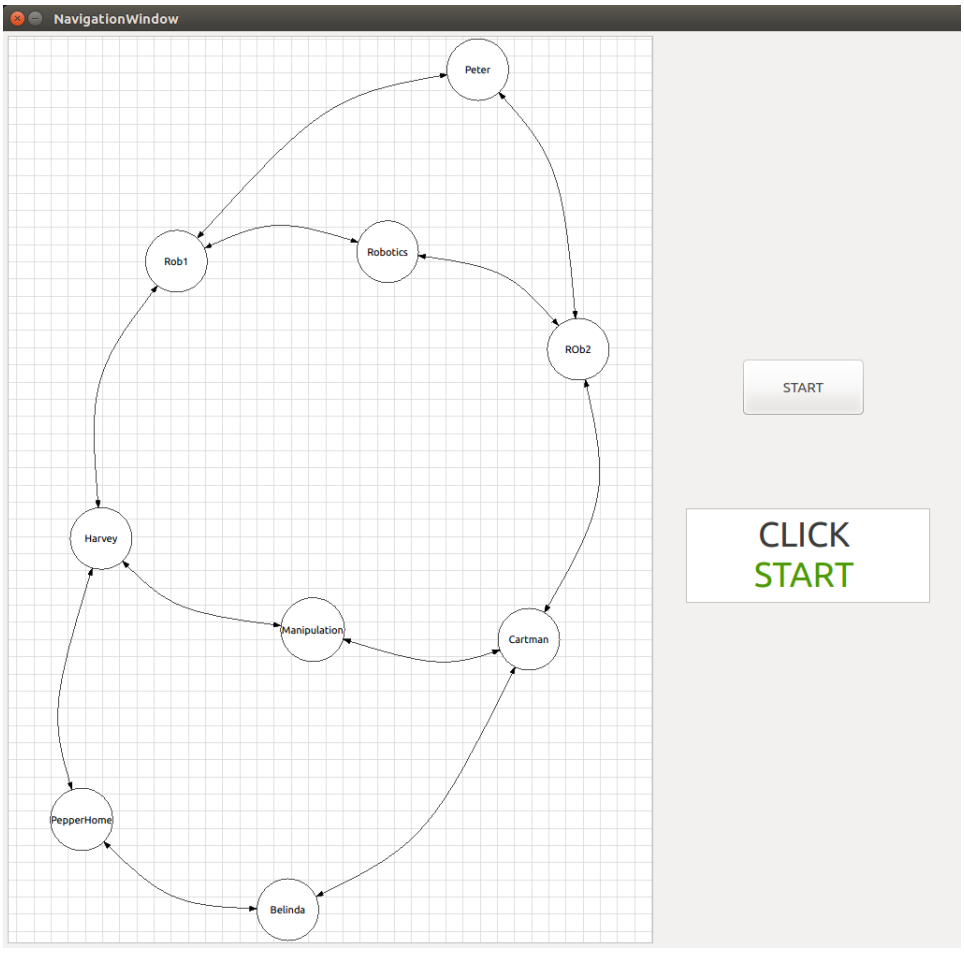
```
$ ./peppernavigationoff <reference _images_path> <navigation_sequence_path>
  # e.g. ./peppernavigationoff ../data/offlinetest/kfls ../data/offlinetest/imgs
```

4 Example using GUI Version

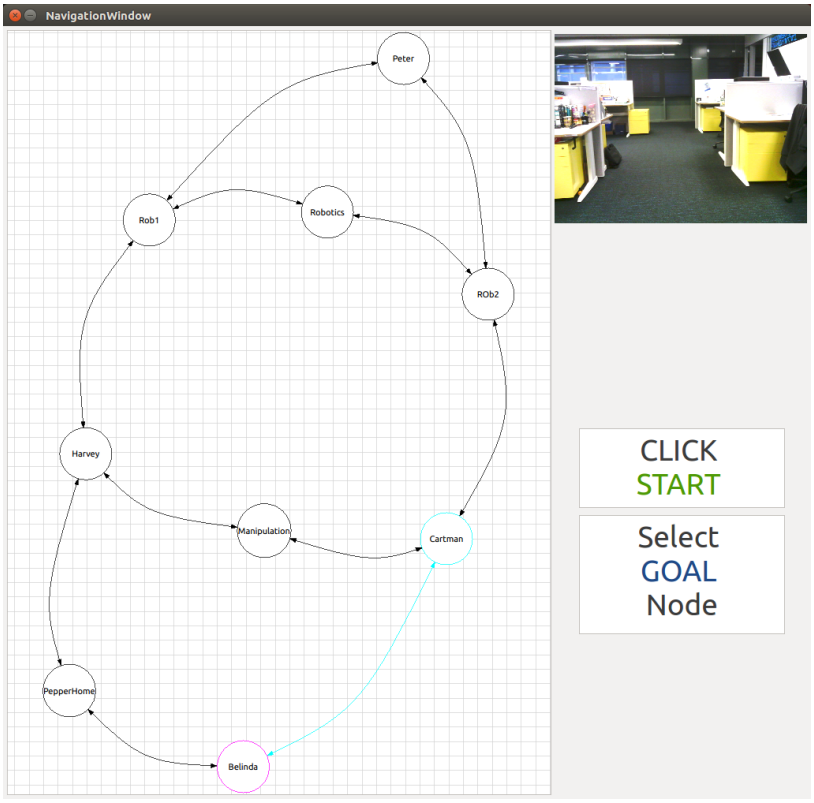
This section provides an overview of using the GUI version of topological navigation.

4.1 Performing Topological Navigation

1. When you run *peppernav_gui,* the following GUI will appear.

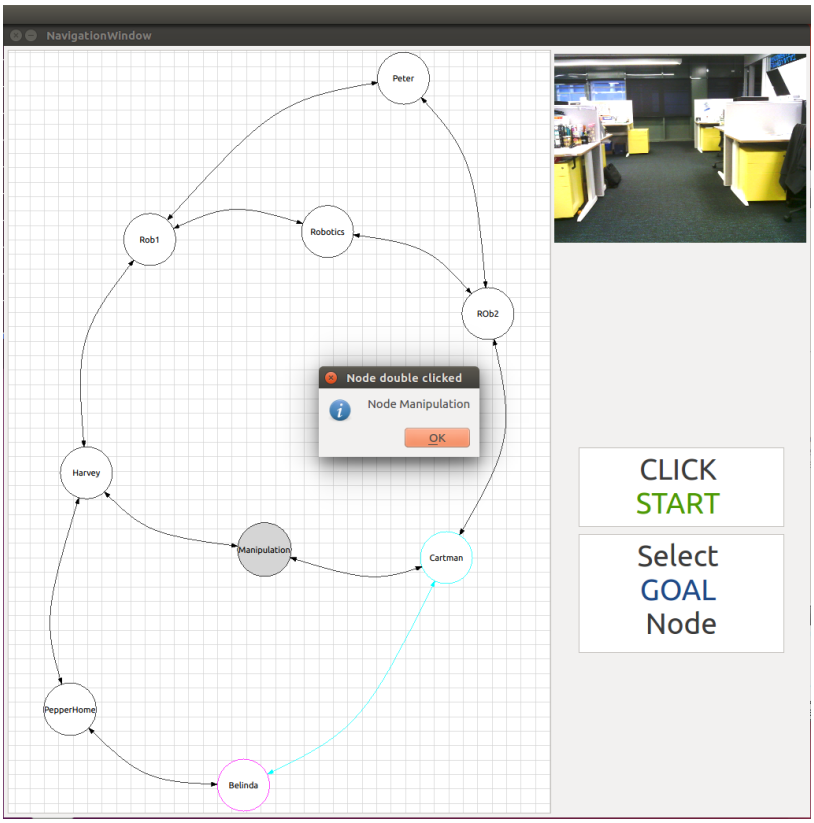


2. Press the “START” button to localize the Pepper robot globally in the topological map.

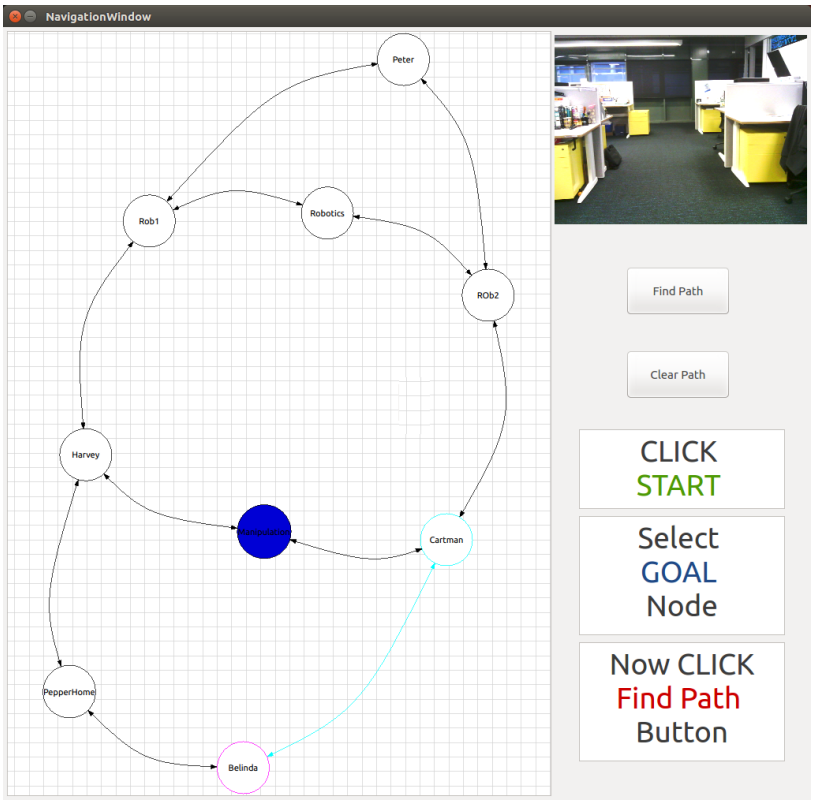


The Pepper robot is localized between the purple colored node and cyan colored node in the edge shown in cyan. The Pepper robot is facing towards the cyan colored node.

3. Select the destination by clicking on the corresponding node.

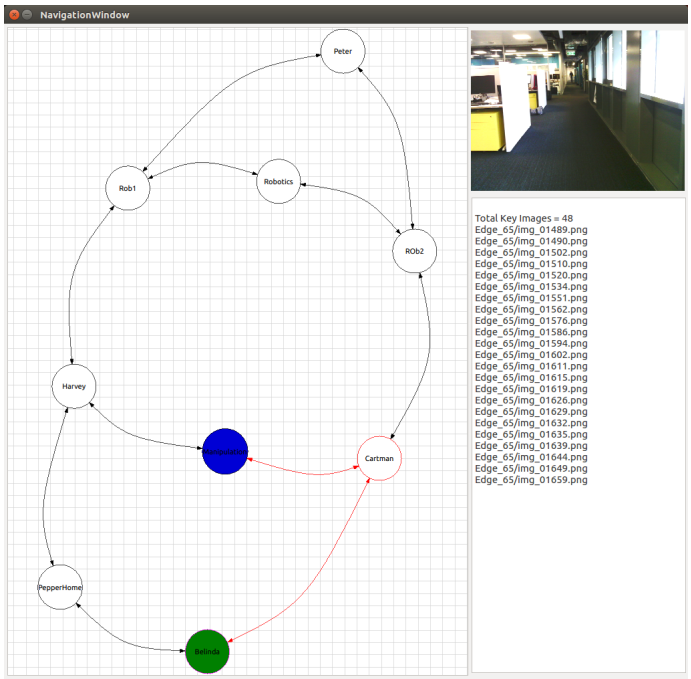


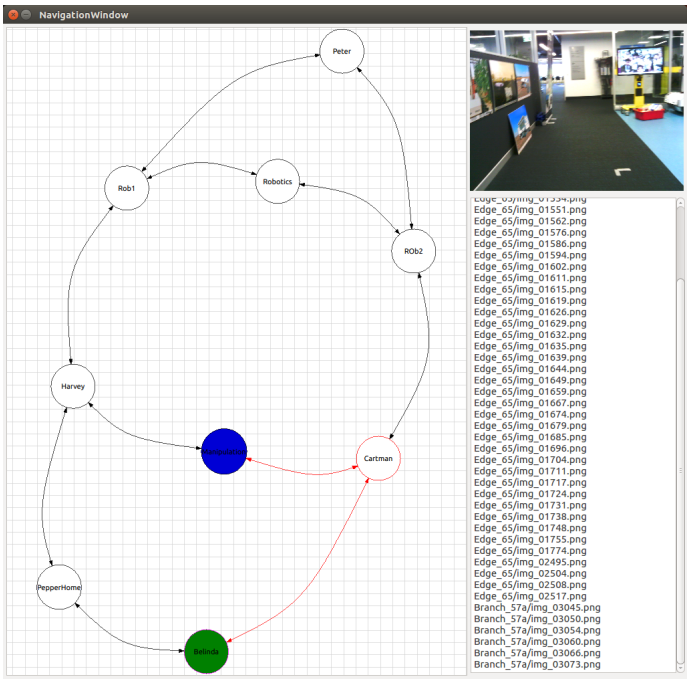
4. The destination node is shown in blue. Now select the “Find Path” button to find the optimal path using A* search.



If you select “Clear Path”, you will return to step 3.

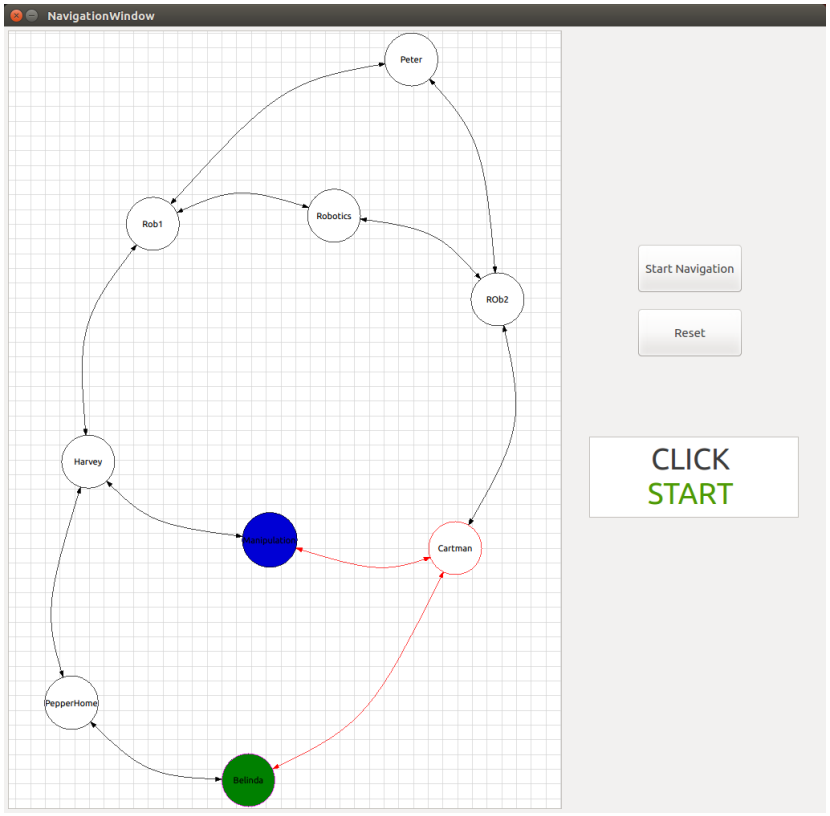
5. The optimal path is shown in red. The start node is shown in green, the destination node is shown in blue, and the intermediate node(s) is(are) shown in red. The reference images of the optimal path are retrieved from the image memory.





Once the reference image list of the optimum path has been retrieved, we are ready for the navigation.

6. Select “Start Navigation” to commence navigation.



7. During navigation, the following window appears showing the current reference images (in color) and current image (in grayscale).

8. When the Pepper robot reached the destination, the console displays the message “End of topological Navigation”, and the robot stops. Continue to Step 2 for the next destination.

4.2 Running Offline Localization

If you have the topological map and images from navigation, you can visualize localization offline. Before running the code, you have to make the following changes in the code (*navmain/nav_peppergui.cpp*).

1. Set `mode = 1;` To use with the Pepper robot, set it back to 0 (default).
2. Set `useobsavoid = false;`
3. Set `opt_ip` as the path of the folder containing navigation images.
   e.g. `opt_ip = "../data/offlinetest/imgs"`

5 Code Repository and Documentation

The repository ***pepper_navigation*** is arranged as follows:

1. **Scripts for building and executing**
   (a) Bash script to compile navigation codes: *compile.sh*
   (b) Example bash scripts to run navigation executable: *run_onboard.sh*, *run_remotePC.sh* and *run_offline.sh*.
   (c) CMake build script: *CMakeLists.txt*.
2. **navmain/**:- This folder provides the top-level interface for the navigation.
   (a) ***navigation*** (*navigation.h*, *navigation.cpp*):- Top-level interface for image-based navigation using line segments (linenav/).
   (b) ***peppernavigation*** (*peppernavigation.h*):- Top-level interface for the Pepper robot navigation that combines free-space navigation and image-based navigation.
   (c) ***pepperInterface*** (*pepperInterface.h*, *pepperInterface.cpp*):- Virtual Class with top-level virtual functions for interface with the Pepper robot or just image sequences.
      i. ***pepperRobot*** (*pepperRobot.h*, *pepperRobot.cpp*):- Derived Class to interface with the Pepper robot.
      ii. ***pepperRobotVirtual*** (*pepperRobotVirtual.h*, *pepperRobotVirtual.cpp*):- Derived Class to test ***pepperRobot*** functionalities via Virtual Pepper where images are read from the folder.
      iii. ***pepperOffline*** (*pepperOffline.h*, *pepperOffline.cpp*):- Derived Class to interface in offline mode i.e. read images from the folder and perform image-based localization only (Pepper robot not used).
   (d) ***topmapprocessor*** (topograph_processor.h):- Top-level interface for processing topological map (reads topological graph and reference images from disk for navigation).
   (e) ***astar*** (*topograph_astar.h*):- Performs A^* search in the graph. Modified code from *Boost Graph Library example: astar-cities*.
   (f) ***pepperServices*** (*pepperevents.h*):- Subscribe to the Pepper robot's internal events related to move and collision.
   (g) **navmain/maingui/**:- Interface for topological navigation. The GUI version of topological navigation runs on a remote PC. The code communicates with the Pepper robot remotely via the naoqi interface. The Non-GUI version of topological navigation is capable of running onboard on the Pepper robot if the code is compiled in the Pepper Virtual Machine.

i. ***navwindow*** (*navwindow.h*, *navwindow.cpp*, *navwindow.ui*):- Navigation with GUI control.

ii. ***navinside*** (*navinside.h*, *navinside.cpp*):- Navigation without GUI. This version can be used to run navigation onboard on the Pepper robot.

(h) ***peppernav_gui*** (nav_peppergui.cpp):- Topological navigation of the Pepper robot with GUI. The code has to be executed on the external PC connecting to the Pepper robot remotely.

(i) ***peppernav_inside*** (nav_pepper_inside.cpp):- Topological navigation of the Pepper robot without GUI. This code is capable of running onboard on the Pepper robot.

(j) ***peppernavigation*** (nav_pepperonline.cpp):- Image-based navigation of the Pepper robot along a single sequence of reference images.

(k) ***peppernavigationoff*** (nav_pepperoffline.cpp):- Image-based localization along the reference image list. This is the offline mode that uses image sequence and does not require the robot.

3. **depthnav/**:- Navigation in the drivable free-space using a 2D occupancy grid map obtained from the depth image.

   (a) To use the *depthnav* library, please refer to class ***freespacenavigation*** (*freespacenavigation.h*, *freespacenavigation.cpp).*

   (b) Other classes and functions related to *depthnav*.

   i. ***depthimagescanner*** (*DepthImageScanner.h*, *DepthImageScanner.cpp*):- Creates a 2D grid map from the depth image.

   ii. ***depth_traits*** (*depth_traits.h*): Template function to process the depth image obtained from *ros depthimage_to_laserscan* library.

   iii. ***pepperlaser*** (*pepperlaser.h*):- Defines the Pepper robot's Laser Memory Keys.

   iv. ***alpose2d*** (*alpose2d.h*, *alpose2d.cpp*)*:- libalmath* Pose2D library used to process odometry data from the Pepper robot.

4. **linenav/**:- Image-based navigation using line segments [2].

   (a) To use the original line detection and matching code based on the legacy *BIAS* library (shipped with this repository *here*), use the codes inside the linenav/edlbd/ folder. If you do not want to use the legacy *BIAS* library, the line detection and matching based on OpenCV is used. Please refer to *CMakeLists.txt* located at the base folder.

   (b) To use ***linenavigation*** as a library, please refer to *linenavigation.h* and *linenavigation.cpp*.

   (c) To display the image-based localization via ***dispnav*** class, please refer to *dispnav.h* and *dispnav.cpp*.

   (d) For the usage of ***linenavigation*** and ***dispnav***, please refer to *navigation.h* and *navigation.cpp* in navmain/ directory.

5. **mapping/**:- Reference Images selection and semi-automatic creation of a topological map for navigation.

   (a) *selectRefImages.m*:- Selection of reference images based on the line segment matching [2].

   (b) *genConfigFile.m*:- Generates configuration file that contains the link between the reference images among the adjacent edges.

6. **pepper_qi/**:- Interface with *naoqi* library. This source code has been taken and modified for the Pepper robot. This modified source code supports image acquisition from the depth camera.

7. **qgv/**:- Interactive *Qt GraphViz* display library used to display the topological graph in the GUI version of the source code.

8. **cmake/**: Contains CMake files to use *qgv* and *BIAS* libraries supplied with this repository.

9. **data/**:- Contains data for offline testing (mapping and localization).

   (a) data/offlinetest/imgs:- Contains image sequence obtained from the Pepper robot.

   (b) data/offlinetest/kfls:- Contains reference images obtained from *select_ReferenceImages.m* located in the mapping/ folder.

   (c) tmap/:- Contains topological map required for navigation.

   (d) data/tmap/conf.txt:- Configuration file that defines the topological map. This file is generated from *generate_configfile.m* located in the mapping/ folder.

10. **docs/**:- Contains reference manual and documentation.

    (a) **Source Code Documentation**:- Documentation of the source code generated by Doxygen.

    (b) **Reference Manual**:- This document.

For details, please refer to README.MD at subfolders of this repository.

The documentation of the source code of this repository generated by Doxygen is available in the docs/ folder as Pepper_Navigation_Source_Code_Documentation.pdf.

References

[1] S. R. Bista, B. Ward and P. Corke, “Image-Based Indoor Topological Navigation with Collision Avoidance for Resource-Constrained Mobile Robots” in Journal of Intelligent & Robotic Systems, vol. 102, no. 3, pp. 55, June 2021. https://dx.doi.org/10.1007/s10846-021-01390-6.

[2] S. R. Bista, P. R. Giordano and F. Chaumette, "Appearance-Based Indoor Navigation by IBVS Using Line Segments," in IEEE Robotics and Automation Letters, vol. 1, no. 1, pp. 423-430, Jan. 2016. https://doi.org/10.1109/LRA.2016.2521907.

[3] G. Suddrey, A. Jacobson and B. Ward. "Enabling a pepper robot to provide automated and interactive tours of a robotics laboratory." arXiv preprint arXiv:1804.03288 (2018). `https://bitbucket.org/pepper_qut/virtual-machine.git`.