Litbot documentation

Guillaume Sarrazin - Bastien Mourgue

May 2019

Contents

1 ROS introduction

ROS is a middleware which provides communications API between applications.

With ROS, an application is called a *node*. The applications can communicate in a typical server/client way, or they can send messages through topics. A *topic* is a communication canal, identified by a unique name, on which nodes can publish messages or receive messages after a subscription to the topic.

ROS uses the notion of *package*. A ROS package regroups a set of nodes and messages definitions to fulfill a certain goal.

1.1 Install and set ROS environment

To install ROS on your computer, follow this link: http://wiki.ros.org/Installation. After the installation, do not forget to source ROS setup in every terminal you want to use ROS, or write this line in your `.bashrc`:

```
$ source /opt/ros/<ros_version_name>/setup.[bash|sh|zsh]
```

1.2 Compile ROS packages

ROS uses an overlay above CMake called *catkin_make*.

A package is defined by one main CMakeLists.txt and one package.xml. Read ROS tutorial to learn how to write a CMakeLists.txt and a package.xml for catkin_make.

All the packages should be group under a common `src` directory. Let's say the tree is as follow:

```
ros_workspace
└── src
    ├── package1
    │   ├── CMakeLists.txt
    │   └── package.xml
    ├── package2
    │   ├── CMakeLists.txt
    │   └── package.xml
    └── package3
```

```
├── CMakeLists.txt
└── package.xml
```

To compile the 3 packages, you only have to type `catkin_make` in `ros_workspace` directory.

```
$ cd ros_workspace
$ catkin_make
```

It will generate 2 directories:

- `build`: it contains the intermediate build files.
- `devel`: it contains the final libraries, executables, messages headers, etc.

To use the new build packages, source the setup file located in `devel` directory:

```
$ source devel/setup.[bash|sh|zsh]
```

If you want to install the package, type:

```
$ catkin_make install
```

It will generate a third directory: `install`. Use the variable `CMAKE_INSTALL_PREFIX` to install in a specific directory:

```
$ catkin_make install -DCMAKE_INSTALL_PREFIX=<your_specific_directory>
```

1.3 Launch roscore

To manage communication between nodes, ROS needs a main server, called `roscore`, `ROS master` or `ROS server`. The first thing to do is to start roscore. For that, open a new terminal and type roscore:

```
$ roscore
... logging to /home/gsarrazi/.ros/log/1c7ff9e2-7275-11e9-a098-\
54bf6463303e/roslaunch-ursa-26924.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

```
started roslaunch server http://ursa:41667/
ros_comm version 1.14.3


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.3

NODES

auto-starting new master
process[master]: started with pid [26951]
ROS_MASTER_URI=http://ursa:11311/

setting /run_id to 1c7ff9e2-7275-11e9-a098-54bf6463303e
process[rosout-1]: started with pid [26962]
started core service [/rosout]
```

Based on your network connection, some manipulation could be requiered.

If the name of your machine is not register in the DNS database (like laptop-pollux) or if you build a local network, your server address will come back to localhost. It leads generally to communication problems between nodes if you use multiple computers. In this case, you should set the environment variable `ROS_IP` based on your IP before launching roscore. Use for example `ifconfig` command to find your IP, then type in your terminal:

```
$ export ROS_IP=<your_ip>
$ roscore
```

A good clue to know if there is a problem with roscore is to look to the `ROS_MASTER_URI` line in the message printed by roscore. It contains the address/port to use to communicate with the server. If the address is `localhost` like that:

```
ROS_MASTER_URI=http://localhost:11311/
```

you need to add configuration information for ROS.

An other solution is to ping the name of your machine. If it does not work, then you have to set `ROS_IP`.

1.4 Launch a node

For all terminals which will launch a ros node, you should:

- export `ROS_IP` environment variable if the name is not known in the DNS database. For Inria Montbonnot, machines registers in the DNS have a fixe IP. The ones not registers in the DNS have a floating IP (as laptop-pollux).

  ```
  $ export ROS_IP=<your_ip>
  ```

- export `ROS_MASTER_URI` if the terminal is not running on the same computer than roscore. Copy/paste the `ROS_MASTER_URI` line printed by roscore.

  ```
  $ export ROS_MASTER_URI=http://machine_name:11311/
  ```

- source setup.[sh/bash/zsh] of litbot project. The extension depends on the shell you are using.

  ```
  $ cd <path_to_litbot>
  $ source ros_workspace/devel/setup.[sh/bash/zsh]
  ```

1.5 ROS tutorial to learn more

If you want to learn more about ROS, you can read the following tutorial: http://wiki.ros.org/ROS/Tutorials

2 Litbot repository manipulation

2.1 Cloning and initialising the repository

The repository called *litbot* is stored in Inria gitlab at url https://gitlab.inria.fr/perception-ral/litbot. It contains a git submodule *rmp* (at url https://gitlab.inria.fr/perception-ral/rmp)A.

To initialize it, you need to clone and then to initialize the submodule.

```
$ git clone git@gitlab.inria.fr:perception-ral/litbot.git
$ cd litbot
$ git submodule update --init
```

You can find more information on git submodule on internet. For example: https://git-scm.com/book/en/v2/Git-Tools-Submodules.

2.2 Litbot repository structure

Separation of ros workspace and code The ROS related part is in directory `ros_workspace`. As a normal ros workspace, it contains a `src` directory. There is no package inside this directory but only symbolics links to the true implementation of the packages. The symbolics links can be set through the script `pre_compile_script.sh`.

All the code of `litbot` package is located in `global_workspace`.

Structure of global workspace

- `audio`: all the code related with audio: multiple sound source localization (mssl), speech recognition, dereverberation, speacher characterisation.
- `demo_config`: configuration files of the different panels for `demo_maker`.
- `dialog`: all code related to dialog. There are a look up table to associate a name with a tracked identity and code to manage a short dialog to ask people name.
- `doc`: old report for Samsung which were never really used.
- `external_independant_libraries`: contains code develop in others repositories (as `RMP`) or others libraries developed outside `litbot`.

- `fusion`: code to fusion audio sound source detection and tracked faces.
- `global_utils`: general code which could be used in all others libraries or nodes.
- `launch`: launch script. They can be used to start different nodes in one command lines. They are not used since a while because we use `demo_maker` now to start nodes.
- `msg`: ros messages for `litbot` package.
- `robot_config`: configuration files. It contains constant parameters used in `litbot` nodes as tracker, mssl, etc. The parameters can change according to the robot used. We tried to regroup all the parameters here for clarity.
- `robot_fsm`: code which manage robot behavior.
- `script`: scripts.
- `srv`: ros request/reply definitions for the services.
- `vision`: all code related to vision: face detection, position 3d estimation, appearance model (to improve the face features already computed) and the tracker.
- `visualisation`: visualisation tools. `mssl_visualisation` and `opencv_visu` are too old tools in OpenCV to draw the results of our algorithm. `qt_visu` is the new tool to see them `demo_make` node.

2.3 Git annex manipulation

Git-annex is used to version big files with git repository. But the files are not stored in the git repository as usual files. In fact they are stored separatly and git uses references to the big files.

More information are available here: https://writequit.org/articles/getting-started-with-git-annex.html or https://git-annex.branchable.com/walkthrough/.

We use git-annex in litbot for big files as neural network models, python serialize output (pickel files), etc. Each git repository can store all the git-annex files we want, but by convention, we put at least one copy on the git-annex repository located at **ursa:/local_scratch2/git-annex/litbot** and we call it **main_annex_ursa** (again by presonnal convention).

To add, remove or get files from git-annex, follow the following instructions which are a quick start on git-annex.

- First install git-annex:

  ```
  sudo apt-get update && sudo apt-get install git-annex
  ```

- Then go in litbot repository and initialise the current repository as a git-annex repository:

  ```
  $ cd litbot
  $ git annex init "a_unique_name"
  ```

  By convention, I replaced `a_unique_name` by `myLogin_nameOfTheMachine`. For example: `gsarrazi_auriga`

- Then add at least one remote git-annex repository. Here we add the git-annex repository which centralized all the git-annex files.

  ```
  $ git remote add main_annex_ursa \
          ssh://ursa/local_scratch2/git-annex/litbot
  ```

  It is supposed that your ssh access is correctly set.

- To find where are located the git-annex files:

  ```
  $ git annex whereis
  ```

- To add a file `toto.h5` to git-annex:

  ```
  $ git annex add toto.h5
  $ git commit -m "Add toto.h5 file in the git-annex"
  $ git push #if you want to push of course
  ```

- To send the file `toto.h5` to git-annex remote `main_annex_ursa`:

  ```
  $ git annex copy toto.h5 --to main_annex_ursa
  ```

- To copy the file `toto.h5` from git-annex remote `main_annex_ursa`:

  ```
  $ git annex copy toto.h5 --from main_annex_ursa
  ```

- To synchronize git-annex meta-data:

  ```
  $ git annex sync
  ```

  This command is required to get the last information about: where are located the files, if there are new files, modified files, removed files, etc.

  WARNING: this command does an automatic commit of all your modified files, even the submodules. You must be in a clean repository before executing it if you do not want your git history become dirty! Think to use `git stash` if needed.

- To get ALL git-annex files:

  ```
  $ git annex sync --content
  ```

  WARNING: this command does an automatic commit of all your modified files, even the submodules. You must be in a clean repository before executing it if you do not want your git history become dirty! Think to use `git stash` if needed.

So when you clone litbot, you will typically type:

```
$ cd litbot
$ git annex init "a_unique_name"
$ git remote add main_annex_ursa ssh://ursa/local_scratch2/git-annex/litbot
$ git annex sync
$ git annex sync --content #if you want to get all the git-annex files
$ git annex copy path_to_the_file --from main_annex_ursa #if you want to \
                                              get only the files you need
```

2.4 Important git-annex files in litbot

To see all the files present in git-annex, you can type:

```
$ git annex list
here
|main_annex_ursa
||origin
|||web
||||bittorrent
|||||
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L11_b.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L11_w.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L14_b.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L14_w.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L1_b.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L1_w.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L3_b.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L3_w.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L5_b.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L5_w.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L6_b.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L6_w.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L7_b.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L7_w.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L9_b.npy
XX___ global_workspace/audio/speaker_charac/cnn_voxceleb_weights/L9_w.npy
XX___ global_workspace/audio/speaker_charac/weights-best-8x1.hdf5
XX___ global_workspace/vision/appearance_model/model/IDE_ad_20180828.h5
XX___ global_workspace/vision/appearance_model/model/IDE_ad_20180828.pb
XX___ global_workspace/vision/appearance_model/model/IDE_ad_20180828.uff
XX___ global_workspace/vision/appearance_model/model/IDE_ad_641065.h5
_X___ global_workspace/vision/appearance_model/model/IDE_ad_783652_30.h5
XX___ global_workspace/vision/appearance_model/model/IDE_ad_783652_30_LMP.h5
XX___ global_workspace/vision/appearance_model/model/IDE_ad_783652_30_max.h5
XX___ global_workspace/vision/appearance_model/model/IDE_ad_828530.h5
XX___ global_workspace/vision/appearance_model/model/IDE_ad_828530.pb
XX___ global_workspace/vision/appearance_model/model/IDE_ad_828530.uff
XX___ global_workspace/vision/appearance_model/python/generate_database/\
```

```
                                                    default_database.pkl
```

The files which are currently required to run the demonstration are:

- for `tracker3d` with the appearance model using the siamese netwok:

  ```
  global_workspace/vision/appearance_model/python/generate_database/\
                                                      default_database.pkl
  ```

- for `speaker_charac`:

  ```
  global_workspace/audio/speaker_charac/cnn_voxceleb_weights/*
  ```

3 Litbot compilation

Litbot needs 4 differents packages to be fully compiled: `robot_com`, `nao_driver`, `lito_driver` and `litbot` packages. They will be located in `ros_workspace/src`, but only as symbolic link. The following script located in `ros_workspace` does it automatically.

```
$ cd <path_to_ros_workspace>
$ ./pre_compile_script.sh
```

If you use Ubuntu 18.04, naoqi-libqi ros package does not exists yet. Remove `nao_driver` ros package.

```
$ cd <litbot_dir>/ros_workspace
$ catkin_make
```

Read README.txt at the root of litbot repository for more information.

4 Litbot runtime architecture

4.1 Main principles

The principles of `litbot` workflow can be summarise in the figure **??**.

There are the robot specific nodes which extract the raw data from the sensors. These raw data are:

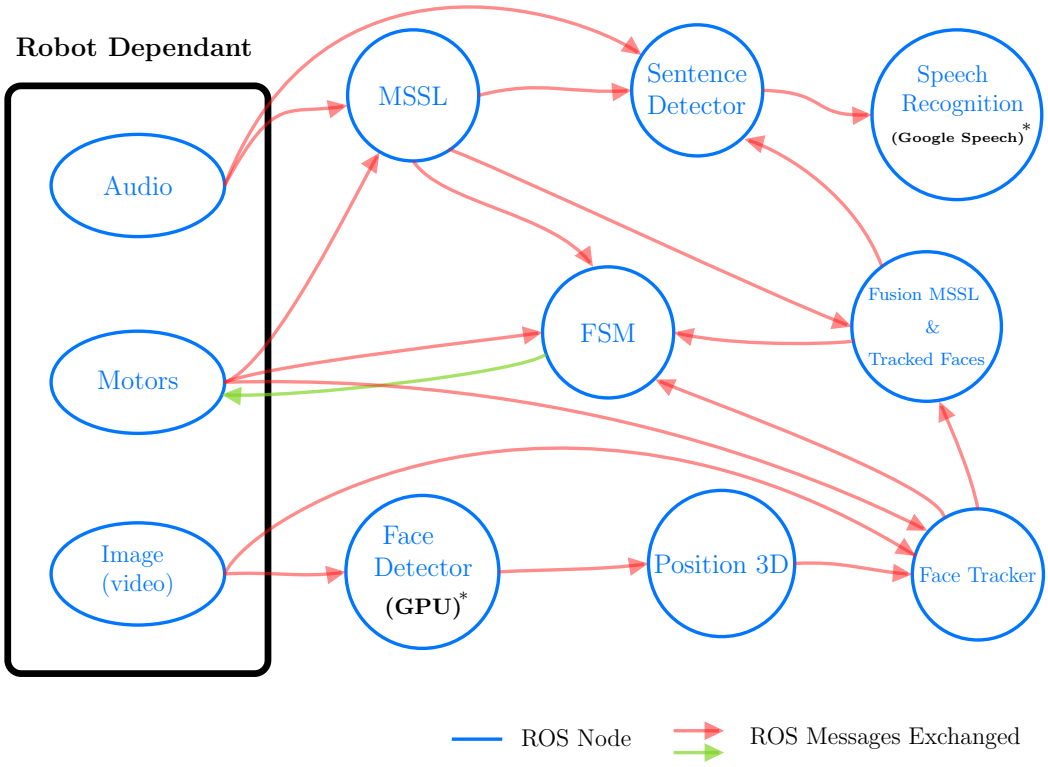

Figure 1: Global ROS Data flow

- audio raw data from 4 microphones
- mono or stereo images and their calibration
- yaw and pitch motors position

The others nodes do not communicate with the robot. When some commands should be performed, they are first send to the node handling the motors which is motors specific.

4.2 Different configurations

In the following paragraphs, different possibles nodes combinations will be shown, each combination corresponds to a specific demonstration. Of course,

once you understand them, you can mixe them as you want.

Face tracking only No audio is used in this case. The robot behavior is active in order the robot follow the person it sees: figure 2.

It is better to start:

- `face_detector_ros.py` with option `--with-face-features` in order to have a better appearance model.
- `tracker3d` with option `--appearance-model-dir` in order to increase again the robustness of the appearance model.

MSSL only No vision is used. The image grabber node can be on for demo visualisation, but it is not mandatory. The robot behavior is active in order the robot turns its head to the person it hears: figure 3.

MSSL and face tracking The robot turns its head to the person it hears and follows the person it sees: figure 4.

MSSL, face tracking and fusion The robot turns its head to the person it hears and follows the person it sees. A fusion between the person speaking and the face tracked is done. It means that the sound source detected is associated with one tracked identity.

There are 2 ways to do the fusion:

- it is done in a separate node and associates the sound source direction to the clothest tracked identity (according to the yaw angle). A threhold is used to avoid association if the sound source direction and tracked identity are too far. The ros graph is represented in figure 5.
- it is done inside the tracker node. The EM step of the tracking algorithm associates sound source directions and tracked identities. The ros graph is represented in figure 6. The tracker must be launch with the mode `VisualWithPartialAudio`.

Speech recognition When people speak to the robot, it associates the audio sentences to a tracked identity and translate it to text. The behavior is not used to avoid motors movement which could decrease the audio sentence recognition or translation. It is represented in the figure 7.

Dialog system The tracking, MSSl, speech recognition, fusion (the one you prefer) are used. We add a node which manage a small dialog system (warning: this node is not tested since a while). It asks the name of persons for which there is no name associated with their tracked identity. The ros graph is represented in figure 8 (the fusion used is the basic one).

Audio-visual tracking Audio sound source directions and detected faces are tracked in the same tracking algorithm. It allows to track person outside the field of view, based only on the speech sound they emit. The ros graph of this demonstration is represented in figure 9. Warning: the audio features which characterise the voice of a person are not robust for the moment, so identity switch when a person is outside the field of view can easily occure.

The tracker must be launch with the Mode `Full`.

5 Litbot nodes description

In this section, we will briefly describe each nodes available in litbot package.

5.1 `audio_dereverberation`

This node implements an algorithm used to remove the reverberation of the audio signal. This is based on the estimation of the convolutive transfer function (CTF) to generate the inverse filter of the room impulse response. This algorithm is adapted to a single speaker case (no overlap between speech turns). The clean speech signal is mono-channel.

The CTF is computed in the frequency domain (STFT), and the spectral representation of the audio signal is published into the topic signal_fft.

This node could be used as input of the speech recognition system.

Short descriptions of inputs and outputs:

- It takes as input:

  1. the audio signal (4 channels, 16000Hz).
     Topic: `audio_raw_16000_hz`.
     Message type: `robot_com::AudioRaw`.

- And outputs:

  - Reverberation free audio signal (1 channel, 16000Hz).
    Topic: `dereverberation/audio_raw`
    Message type: `robot_com::AudioRaw`
  - Original fft signal (4 channels, 16000Hz, 768 samples).
    Topic: `signal_fft`
    Message type: `litbot::AudioFft`
  - Dereverberate fft signal (1 channels, 16000Hz, 768 samples).
    Topic: `dereverberation/signal_fft`
    Message type: `litbot::AudioFft`

5.2 `audio_dereverberation_test`

This node is just here to test some part of the dereverberation algorithm.

5.3 `control_synchro`

This node was used to control the synchronisation between MSSl output and tracker output. It is no more used since a while. Now, we prefer check that the frame per second (fps) output of MSSL is enough high with the command: `rostopic hz -w 250 mssl`.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-h	--help			"Display the current message."
-mf	--mssl-fps	value		"Multiple sound sources localisation output rate. (Default is 125)"
-r	--robot	value	(required)	"Select the robot"
-vf	--vision-fps	value		"Multiple objects tracker output rate. (Default is 15)"

- It takes as input:
  - the tracked faces.
    Topic: `tracked_faces`.
    Message type: `litbot::TrackedFaces`.
  - the MSSL output
    Topic: `mssl`
    Message type: `litbot::MSSL`.
- Output messages on terminal. The message indicates the framerate of `tracked_faces`. If it is less than the one expected, it means the tracker or the MSSL is going slower than it should be.

5.4 `demo_maker`

Demo-maker is a GUI for managing live demonstrations of the litbot projet. It lets you configure and launch the different nodes defined in the projet, and display the output of the different algorithms. The display environment contains a scene that is composed of several panels. A panel contains a set of visual elements which are combined to display some results.

By default, the interface provides all the elements required to configure, start, stop the demonstration and manage the visualization of the scene. Some other features, disabled by default, could be add to perform specific tasks :

- Designer Features : allows user to modify the scene of the display environment : user can create new panel or remove an existing panel.
- Recording Features : allows user to generate video directly from the demo-maker GUI. The video use the scene as images input and an audio ros topic (robot microphones) as audio input.

Before starting the demo-maker, you need to define some environment variables :

- DEMOMAKER_PLUGINS_PATH : this environment variable should contained the path of the folder where the plugins libraries are stored.
- DEMOMAKER_PATH : this environment variable should contained the path of the base folder of the interface (global_workspace/visualization/qt_visu).

Nodes of the litbot project may need to be launched in a specific computer (GPU, robot drivers, ...). For that, you can start in the target computer the node demo_maker_external_launcher. When the communication between the external launcher and the demo-maker GUI is established, you can launch the node in the remote computer by selecting the name its name in the launcher configuration (field launch_device).

In the demo-maker GUI, you can found different configuration files. The .demo file contains the state of the display environment. It configures the display environment and associate each element of visualization to its region in the scene... Usually, we use the file litbot_new_debug.demo. The node configuration file sets the options of each node in order to configure the command line command used to start it. It is computer dependant because some nodes require absolute path as option.

Ros is not directly initialized by default, in order to configure the middleware using the GUI (master URI, Ros IP, Ros hostname...). These parameters are pre-set using the ros environment variable (ROS_MASTER_URI, ROS_IP, ROS_HOSTNAME) as initializer.

For a complete guide of how to configure and use the demo-maker interface, you can read the QuickStartGuide.

Short description of options:

-d or –enable-designer		"Enable designer features"
-df or –demo-file	param_value	"Load the demo configuration file when opening the application"
-h or –help		"Display the current message"
-i or –ros-auto-init		"Initialize ros when opening the application"
-nc or –node-configuration	param_value	"Load the node configuration file (.config) when opening the application"
-ns or –namespace	param_value	"Global namespace used for all the node"
-rec or –enable-recorder		"Enable recording features"

5.5 demo_maker_external_launcher

This node is a part of the demo-maker GUI. It lets user launch nodes on remote computer directly from the demo-maker.

This node should be launched in the target computer. The ROS master URI should be set before running the node. The demo_maker_external_launcher should be associated to an unique name (usually we use the name of the machine) passed in the command line. This name is the one used in the GUI to choose between the remote launchers. When the node is started, it send a message over ROS to notify that it exists and waits for an invitation from the demo-maker GUI. During this waiting state, nothing is displayed in the console. Once the connection with the demo-maker is established, you can see the message "Receive Invite from master". When the nodes are launched in the remote launcher, you can see in the console that the program start/stop a given process.

The command line used to run the node is :

$ rosrun litbot demo_maker_external_launcher name.

5.6 distance_estimation

This node estimates the distance between detected faces and the camera based on the size of the detection. It was used only to control the result of the estimation, not in a demonstration scenario.

Short descriptions of options, inputs and outputs:

- It takes no option.
- It takes as input:
  - the detected faces.
    Topic: `facedetection`.
    Message type: `robot_com::FaceDetection`.
  - the camera calibration. It is needed only at the beginning.
    Topic: `camera_info`
    Message type: `sensor_msgs::CameraInfo`.
- And outputs:
  - the distance of each detected faces.
    Topic: `distance_estimation`.
    Message type: `litbot::DistanceEstimation`.

5.7 external_process_notify_end

This node is an internal process used by the demo maker when running node using the external launcher.

5.8 facedetection

This node detects faces in an image.

Per default it uses the pico face detector, but it can also use the one provided by OpenCV. It can run in real time on a cpu with pico face detector. Its drawback is that it can detect only frontal faces. The code do not handle stereo camera case.

The face detector descriptor file needed for pico is named "`facefinder`" in our repository.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-d	--display			"Display the images stream with the faces detected."
-f	--face-detector-xml	value	(required)	"Path to the face detector descriptor file."
-h	--help			"Display the current message."
-np	--no-pico			"Do not use pico facedetection. Use the default OpenCV facedetection."
-r	--robot	value	(required)	"Select the robot"

- It takes as input:
  - an image
    Topic: `image_raw`
    Message type: `sensor_msgs::Image`.
  - the camera calibration. It is needed only at the beginning.
    Topic: `camera_info`
    Message type: `sensor_msgs::CameraInfo`.
- And outputs:
  - the detected faces with the fields `face` and `upper_body` filled.
    Topic: `persons_detection`.
    Message type: `robot_com::PersonsDetection`.
  - the detected faces.
    Topic: `facedetection`.
    Message type: `robot_com::FaceDetection`.

5.9 `features_compute`

This node was used to test if class PersonFeaturesExtractorKeras works correctly. It was done only for debug.

5.10 `fusion`

This node associates the sound source direction detected (MSSL output) with the closest tracked faces identities. The yaw angle distance is used for

association. A threshold defines when 2 detections are too far to be fused.

Short descriptions of options, inputs and outputs:

- The command lines options are:

  -h --help "Display the current message."
  -r --robot value (required) "Select the robot"

- It takes as input:

  - the tracked faces.
    Topic: `tracked_faces`.
    Message type: `litbot::TrackedFaces`.
  - the MSSL output
    Topic: `mssl`
    Message type: `litbot::MSSL`.

- And outputs:

  - the list of all tracked objects which are associated with a sound source localisation (SSL) and the corresponding SSL list.
    Topic: `fusion_data`
    Message type: `litbot::FusionData`

5.11 `mssl_generate_grid`

This node lets user generate the callibration file of the mssl. It relies on the direct-path relative transfer function estimation (the same one implemented in mssl_ros node). These features are stored into a text file. The node follows the same steps than the mssl algorithm but does not use the EM algorithm and the peak detection.

The grid generator uses a set of wave files as input. Each wave file is generated from the playback of a white noise signal (sampled at 48000Hz) from the different candidates position (distance : 1m). This can be easily done using the audio callibration platform. The file name should respect the following name pattern :"X.wav" with X corresponding to the angle in degrees. The input files should contains 4 channels and have a frequency sampling of 16000Hz or 48000Hz.

If the generated grid file path already exists, the old version of the file is removed except if the option -a is activated. In this case, the new features are append at the end of the existing file.

Short descriptions of options, inputs and outputs:

-a or –append-to-file			"Append the new generated candidates at the end of the file"
-d or –file-directory	param_value	(required)	"Path to the directory which contains the white noise recordings"
-f or –grid-file	param_value	(required)	"File where the grid will be written"
-h or –help			"Display the current message"
-r or –robot	param_value	(required)	"Select the robot"

5.12 `mssl_ros`

This node implements the Multiple Sound Sources Localization (MSSL) algorithm. It computes the direct path relative transfer function (DP-RTF) of the sound based on the Convolutive Transfer Function and compares these features with a set of candidates using a recursive EM algorithm. Once the probabilities of belonging to each candidates of the grid is computed, we perform a peak detection to find the dominant peaks. A grid contains the DP-RTF features in an azimutal plan (step 5 degrees) and is specific to a robot.

The algorithm is designed to deal with multiple grids, corresponding to different elevations. In this case, the EM algorithm compares the computed features with each candidates of each grids and gets the highest probability for each azimutal direction. The output vector is composed by one of the elevation candidate for each azimutal direction. Then the classic peak detec-

tion is performed on this vector.

To activate the elevation part of the algorithm, you have to give as input the folder which contains the grids. To be valid, the grid folder should contains the different grids ordered by their elevation, and a config file. Each line of this file is composed of the name of the grid and the elevation angle relative to robot's head (in degrees).

If the elevation is not active, the algorithm will return a default elevation for each angle. This default elevation is controled by the ros parameter: `default_pitch`.

By default, the algorithm use the audio signal from the resample ROS node. It should be composed of 4 channels and sampled at 16000Hz. If you want to use a file as input instead of ROS topic, use the option -f with the wave file path. A resampling step is performed when the file is reading, but only works for a 48000Hz audio signal. If your input file has not a good sample rate, you have to use the fileplayer node. This node replaces the audio driver and publish the audio signal in the audio_raw topic (the resample node should be launched).

When the robot's head is moving, the probabilities should be align with the current head position. This is performed when the motors position are published in the motors_position topic.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-ca or –calib-audio	param_value	(required)	"Path to the audio grid calibration file/directory"
-d or –display			"Display the selected peaks in the console"
-f or –file	param_value		"Use wave file as input of the algorithm"
-h or –help			"Display the current message"
-r or –robot	param_value	(required)	"Select the robot"

- It takes as input:
  1. the audio signal (4 channels, 16000Hz).
     Topic: `audio_raw_16000_hz`.
     Message type: `robot_com::AudioRaw`.
  2. the motor's position.
     Topic: `motors_position`
     Message type: `robot_com::MotorsPosition`.
- And outputs:
  - the sound source localization results.
    Topic: `mssl`
    Message type: `litbot::Mssl`

5.13 `mssl_visual`

This node draws the probability histogram of sound source localisation. The graphical output is generated with OpenCV.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-h	--help		"Display the current message."
-r	--robot	value (required)	"Select the robot"

- It takes as input:
  - the MSSL output
    Topic: `mssl`
    Message type: `litbot::MSSL`.
- There is no output.

5.14 name_assignment

This node maintains a lookup table with the correspondence between names and tracked object identities. The correspondances are provided through a ros service.

Short descriptions of options, inputs and outputs:

- There is not command lines options.
- It takes as input:
  - the tracked faces.
    Topic: `tracked_faces`.
    Message type: `litbot::TrackedFaces`.
- It provides a service:
  - the request contains the name and its associated identity. There is nothing in the answer.
    Service name: `name_label_association`.
    Service type: `litbot::NameLabelAssociation`.
- And outputs:
  - the named tracked faces. The field `name` is completed with the corresponding name if there is one.
    Topic: `named_tracked_faces`.
    Message type: `litbot::TrackedFaces`.

5.15 position3d

This node computes the distance between the camera(s) and the detected faces. There are 2 cases:

- mono camera: based on the face detected size and a reference face size, an estimation of the distance is computed.
- stereo camera: perform triangulation to compute the distance of the detected face. Rectification of left and right image is computed. Then the face is searched in the right image (the face detection should be done on the left image in the previous node). Then triangulation is computed based on the face position in the left and right image and the stereo camera calibration.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-cc	--calib-camera	value		"Path to the camera calib file"
-h	--help			"Display the current message."
-r	--robot	value	(required)	"Select the robot"

- It takes as input:

  1. the detected faces.
     Topic: `persons_detection`.
     Message type: `robot_com::PersonsDetection`.
  2. the mono or stereo image(s).
     Topic: `image_raw` or `stereo/left/image_raw` and `stereo/right/image_raw`.
     Message type: `sensor_msgs::Image`.
  3. the camera calibration. It is needed only at the beginning.
     Topic: `camera_info`
     Message type: `sensor_msgs::CameraInfo`.

- And outputs:

  – the detected faces with the fields `face3d` and `face3d_present` filled.
     Topic: `persons3d_detection`
     Message type: `robot_com::PersonsDetection`

5.16 `robot_fsm`

This node controls the behavior of the robot. It gets the results of other nodes and mixes then to decide which motor commands it should send. It is implemented as a finite state machine.

- If there is tracked object information, the robot will follow a tracked object. It takes the first one it sees.

- If there is MSSL input, the robot will turn its head to the main peak provided by MSSL.

- If there is tracked object information, MSSL input and fusion, a combinaison of both behavior arrives. Moreover there is a weight mechanism which avoid switching to a new audio source if a followed tracked object has enough weight. To gain weight, a tracked object must be detected as speaking in the *fusion* node. A tracked object loss weight regularly if it is not speaking.

- There is also states to request names if the *dialog* node is started.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-h	--help			"Display the current message."
-nm	--no-movement			"With this option, the robot does not move."
-r	--robot	value	(required)	"Select the robot"
-vf	--vision-fps	value		"Multiple objects tracker output rate. (Default is 15)"

- It takes as input:
  - the list of all tracked objects which are associated with a sound source localisation (SSL) and the corresponding SSL list.
    Topic: `fusion_data`
    Message type: `litbot::FusionData`
  - the tracked faces completed with names by `name_assignment` node.
    Topic: `named_tracked_faces`.
    Message type: `litbot::TrackedFaces`.
  - the motors position.
    Topic: `motors_position`
    Message type: `robot_com::MotorsPosition`.
  - the MSSL output
    Topic: `mssl`
    Message type: `litbot::MSSL`.

- And outputs:

– the motors commands to follow a tracked object.
  Topic: `motors_control`
  Message type: `robot_com::MotorsCmd`

5.17 `test_coordinate_transformation`

This node is just here to test the coordinate transformation library.

5.18 `tracker3d`

The node implements a tracking algorithm developed by Yutong Ban, a former PhD student of Perception team. The node has 3 possible modes:

- `Visual`: only face tracking is performed. It is the original mode and the most tested one.
- `VisualWithPartialAudio`: face tracking is performed but audio sources detected by the MSSL are also assigned to the tracked faces inside the EM algorithm. This mode allows to do the fusion inside the tracking algorithm. The node `fusion` is no more needed.
- `Full`: face and speaker tracking is performed in the same algorithm. This mode is not very robust because the speaker features are not robust. When a speaker is outside field of view, a lot of identity switchs occur.

There is a variable inside the tracking algorithm which defines the maximum tracks it can manage: `litbot::config::MAX_TRACKED_OBJ`. It is currently set to 20. A mechanism to destroy completly old track was written a long time ago (do not mix up with the *dead process* of Yutong tracking algorithm). It was not tested recently and I think it does not work properly anymore.

To activate the appearance model based on a siamese network developped by Guillaume Delorme, the option `--appearance-model-dir` must be given. It should point to the appearance model directory:

- inside the repository, it is located in:
  `<litbot_dir>/global_workspace/vision/appearance_model`
- inside the install directory, it is located in:
  `<install_dir>/lib/python2.7/dist-packages/litbot/appearance_model`

Do not forget to set the option `--with-face-features` of `face_detection_ros.py` node to increase the robustness of the appearance model.

The node gets the calibration file through topic `camera_info` by default, but it is also possible to give it in command line.

When debugging, it is quiet hard to follow what happens inside the tracker. To ease this task, a specific log file can be created by each instance of the tracker. The file is named: `.<date>_<time>_tracker_log.txt`. To force the log output, the variable `litbot::config::WRITE_LOG` must be set to `true` inside the file `tracker_robot_config.h`.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-am	--appearance-model-dir	value		"Path to the appearance model directory"
-cc	--calib-camera	value		"Path to the camera calib file"
-h	--help			"Display the current message."
-r	--robot	value	(required)	"Select the robot"
-tm	--tracking-mode	value	(required)	"Select the tracking mode. Could be "Visual", "VisualWithPartialAudio" or "Full""

- It takes as input:
  - the mono or stereo image(s).
    Topic: `image_raw` or `stereo/left/image_raw`
    Message type: `sensor_msgs::Image`.
  - the detected faces.
    the detected faces with the fields `face3d` and `face3d_present` filled.
    Topic: `persons3d_detection`.
    Message type: `robot_com::PersonsDetection`.
  - the motors position.
    Topic: `motors_position`
    Message type: `robot_com::MotorsPosition`.

  - when the mode `VisualWithPartialAudio` is activated: the MSSL output
    Topic: `mssl`
    Message type: `litbot::MSSL`.
  - when the mode `Full` is activated: the extracted speaker features.
    Topic: `speakers_detection`.
    Message type: `litbot::SpeakersDetection,`.
  - the camera calibration. It is needed only at the beginning.
    Topic: `camera_info`
    Message type: `sensor_msgs::CameraInfo`.

- And outputs:
  - the tracked faces.
    Topic: `tracked_faces`.
    Message type: `litbot::TrackedFaces`.

5.19 `tracker_control`

This node controls the motors in order to follow tracked object. If the object is loosed, it follows the object with the smaller identity. It is an old node no more use since a while.

Short descriptions of options, inputs and outputs:

- There is no command line option.
- It takes as input:
  1. the tracked faces.
     Topic: `tracked_faces`.
     Message type: `litbot::TrackedFaces`.
  2. the motors position.
     Topic: `motors_position`
     Message type: `robot_com::MotorsPosition`.
- And outputs:
  - the motors commands to follow a tracked object.
    Topic: `motors_control`
    Message type: `robot_com::MotorsCmd`

5.20 `visualisation`

This node draws the tracking results, the peaks detected with the MSSL, and node *fusion* results on the camera image (left camera for stereo vision) and on a top view. A synchronisation is performed on data to display data with close timestamp. The graphical output is generated with OpenCV.

Short descriptions of options, inputs and outputs:

- The command lines options are:

  -h --help "Display the current message."
  -r --robot value (required) "Select the robot"

- It takes as input:
  - the mono or the left image of stereo camera.
    Topic: `image_raw` or `stereo/left/image_raw`
    Message type: `sensor_msgs::Image`.
  - the detected faces. (No more used in the code.)
    Topic: `facedetection`.
    Message type: `robot_com::FaceDetection`.
  - the detected faces with the fields `face` and `upper_body` filled.
    Topic: `persons_detection`.
    Message type: `robot_com::PersonsDetection`.
  - the list of all tracked objects which are associated with a sound source localisation (SSL) and the corresponding SSL list.
    Topic: `fusion_data`
    Message type: `litbot::FusionData`
  - the tracked faces.
    Topic: `tracked_faces`.
    Message type: `litbot::TrackedFaces`.
  - the motors position.
    Topic: `motors_position`
    Message type: `robot_com::MotorsPosition`.
  - the MSSL output
    Topic: `mssl`
    Message type: `litbot::MSSL`.

- There is no output.

5.21 main_dialog_name_system.py

This node implements a dialog system to get the name of the person present in the scene. When the robot sees a track which have no name associated to it, the dialog system start. The dialog system follows this basic scenario:

- Robot asks the name of the person.
- The speaker answers its name.
- The robot repeates the name and asks for a confirmation
- The speaker say Yes if the name is correct, No otherwise.

The dialog system can adapt to different cases.

- User response does not correspond to predefined possibilities. The robot says that it do not understand and repeat the last question.
- Another person gives the response of the question. The robot gives a warning that it is not the right speaker who answered.
- The target person goes outside the field of view of the robot. The dialog system stop directly.

User can change the language of the dialog system between french (fr-FR) and english(en-US).

The dialog system is started from the FSM and it use the results of the speech recognition system to get the answers of the speakers. The name is sent using ros services to the name lookup table node.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-l	param_value	"Language of the dialog system (fr-FR / en-US). Default : fr-FR"

- It takes as input:

  1. Speech transcription.
     Topic: `speech`.
     Message type: `litbot::SpeechResult`.

5.22 `main_speaker_charac.py`

This node provides speaker features. It extracts them from sentences detected thanks to MSSL and audio data energy. Different speaker features extractor was tested: features based on MFCC, a CNN neural network build by an intern (Caroline) and an other neural network learned on VoxCeleb and recover under python code by an other intern (Elsa). The last one is the most accurate even if the results are not very good. This poor results comes from the fact we do not use the network as expected by its creators. Normaly it should take as input a whole sentence, but we give them only a short sliding time window (to get regularly outputs).

This node needs a good GPU to run at 10/12 fps. We use to launch this node on `auriga` with GeForce GTX 1070 GPU.

To maintain a stable output rate, the time window is slided of 3 audio input buffers when we use Lito robot. With Nao, the audio output rate is less high, so the time window is slided of only 1 buffer. But if it is launched in the same time as `face_detection_ros.py`, even on different computers, we observe a decrease of output frequency for both nodes when features are extracted from a sentence.

Short descriptions of options, inputs and outputs:

- The command lines options are:

  -n --network value (required) "Path to neural network weight and biais files, or model file."
  -h --help "Display the current message."
  -r --robot value (required) "Select the robot"

  With Elsa neural network, `--network` must indicate the path to cnn voxceleb weights which is typically:

  ```
  <litbot_dir>/global_workspace/audio/speaker_charac/\
                                          cnn_voxceleb_weights
  ```

- It takes as input:
  - audio raw data after resampling to 16000 hz.
    Topic: `audio_raw_16000_hz`
    Message type: `robot_com::AudioData`.
  - the MSSL output
    Topic: `mssl`
    Message type: `litbot::MSSL`.
- And outputs:
  - the extracted speaker features.
    Topic: `speakers_detection`.
    Message type: `litbot::SpeakersDetection,`.

5.23 `face_detection_ros.py`

This node detects faces in an image.

It can detect frontal faces, but also side faces, almost rotated faces and even almost occluted faces. It must be run on a GPU to be executed in real time. Generaly we execute it on `alya` with a GeForce GTX 980 GPU. The GPU must be completly free in order it works in real time (10 fps).

To active the face features extraction, the option `-wf` must be set. It increases significantly the appearance model stability of the tracker.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-r	--robot	value	(required)	"Select the robot"
-h	--help			"Display the current message."
-wf	--with-face-features			"Active the face extraction features"

- It takes as input:
  - the mono or the left image of stereo camera.
    Topic: `image_raw` or `stereo/left/image_raw`
    Message type: `sensor_msgs::Image`.
- And outputs:

- the detected faces with the fields `face` and `upper_body` filled.
  Topic: `persons_detection`.
  Message type: `robot_com::PersonsDetection`.
- the detected faces.
  Topic: `facedetection`.
  Message type: `robot_com::FaceDetection`.

5.24 `main_sentence_activity_detector.py`

The sentence activity detector splits the input audio signal into segments. One segment corresponds to a sentence. The segment separation is based on two indicators :

- A voice activity detection
- Sources meta information (azimutal angle, id).

The voice activity detection computes the energy of the signal and compares it to a threshold. The sources labelling information comes from the multiple sound source localization (MSSL) for angle, and the fusion node (for id of of the dominant speaker). The fusion could come from the fusion node or the tracker2d node (mode VisualWithPartialAudio or Full).

By default, the sentence is composed of three parts. It starts with some silent chunks, then the interresting speech signal, and ends with other silent chunks. The sentence is considered as finished when the pause duration reach 0 second. This process is interrupted when the meta information of the current audio chunk is incompatible with the current sentence. In this case, the current sentence is notified as finished and a new one are created with the new meta information. If the sound comes from a non tracked person, then the id is set to -1.

The sentence segment only contains one channel. We select the one with the highest energy.

By default, the input audio signal comes from the audio_raw topic. But you may need to use another topic (such as the dereverberate signal). In this case, use the option -t. If you want to run multiple time the sentence activity detector, you have to change the name of the node (the one use to register to ROS server) using the option -n.

This node published the sentences one by one and does not publish anything when no sentence is detected.

Short descriptions of options, inputs and outputs:

- The command lines options are:

-r	param_value	(required)	"Select the robot."
-t	param_value		"Input audio data topic (default audio_raw)."
-n	param_value		"Name of the ros node."

- It takes as input:

  1. Audio raw signal.
     Topic: `audio_raw` or `topic given in command line`.
     Message type: `robot_com::AudioRaw`.

- And outputs:

  – Sentence segment chunk.
    Topic: `speech_audio_raw`
    Message type: `litbot::SentenceAudioData`

5.25 `main_speech_recognition.py`

This node performs the transcription of the sentence segment into text. It uses a speech recognition system (ASR). The current implementation uses Google Speech API and request an internet connection and a credentials associated to the project.

Different language is available for Google Speech ASR. We use only french and english languages.

This node uses the information contained in the SentenceAudioData message. You can run several instance of this node by setting the -t and -n options to define respectively topic and name information.

- The command lines options are:

-c	param_value (required)	"Path to google credentials."
-l	param_value	"Language used for ASR system fr-FR/en-US (default en-US)."
-t	param_value	"Additional input sentence audio data topic namespace."
-n	param_value	"Name of the ros node."

- It takes as input:

  1. Sentence audio data.
     Topic: `speech_audio_raw` or `topic given in command line`.
     Message type: `litbot::SentenceAudioData`.

- And outputs:

  – Speech transcription.
    Topic: `speech`
    Message type: `litbot::SpeechResult`

5.26 `robot_communication.py`

This node targets new developer which are looking for python code examples. It provides code to grab data from the robot, synchronise with ROS the message arrival, and send motor command to the robot.

5.27 speaker_charac_test.py

The aim of this node is to test the features extracted by `main_speaker_charac.py`. It contains different functions to do classification on the features. The ground true is given by the angle position of the speakers. This node is only used for development of `main_speaker_charac.py`.

Short descriptions of options, inputs and outputs:

- There is no command lines options.
- It takes as input:
  - the mono or the left image of stereo camera.
    Topic: `image_raw` or `stereo/left/image_raw`
    Message type: `sensor_msgs::Image`.
- And outputs:
  - the speaker audio features extracted by `main_speaker_charac.py`.
    Topic: `speakers_detection`.
    Message type: `litbot::SpeakersDetection,`.

5.28 ros_communication_example.py

This node targets new developer which are looking for python code examples. It provides code which show how to connect to topics and set callbacks inside or outside a class.

5.29 resample.py

This node resamples the audio signal at the specified frequency sampling. If the sampling rate of the audio signal is the target one, then the signal is just forwarded.

Short descriptions of options, inputs and outputs:

- The command lines options are:
  -r param_value (required) "Target output sampling rate."
- It takes as input:

1. Audio raw signal.
   Topic: `audio_raw`.
   Message type: `robot_com::AudioRaw`.

- And outputs:
  - the resampled signal.
    Topic: `audio_raw_16000_hz`
    Message type: `robot_com::AudioRaw`

6 Robots tricks

6.1 Local link connection to Lito

If you are connected in local link with Lito robot and not on Inria network, you must set correctly Lito network configuration.

On lito, type:

```
$ sudo ip ad add 192.168.1.3/24 dev eth0 valid_lft forever
```

It will set the ip adress of Lito to 192.168.1.3.

To remove this ip, type:

```
$ sudo ip ad del 192.168.1.3/24 dev eth0
```

Because this ip does not stay eternally, you can repeat automatically this command every second with `watch`:

```
$ watch -n1 "sudo ip ad add 192.168.1.3/24 dev eth0 valid_lft forever"
```

6.2 Lito drivers errors

First, to avoid some errors in drivers on lito, you must be login. For that connect a keyboard to Lito computer (unplug and replug it if it was connected at lito boot time - it generally does not see it at boot time), type `Ctrl+Alt+F1` and then login. The login is `inria` and the password is the same as the login.

If some sensor is not recognize, unplug and replug it could help. Else reboot lito computer help also.

A RMP compilation

There are 2 main part in RMP.

- The old part which uses shared memory for messages exchanges between applications.
- The new part based on ROS.

A.1 Old RMP compilation

Information on packages dependancies are available in README.txt at the root of the git repository. For the compilation, type the following command:

```
$ mkdir <rmp_dir>/buid
$ cd <rmp_dir>/build
$ cmake .. \
    -DNAOQI_SDK_DIRECTORY=<path_to_naoqisdk> \
    [-DOpenCV_DIR=<path_to_openCV>]
```

One important tools in the old RMP is the audio calibration platform. It handles the rotating circular platform and play white nose to do the robot audio calibration. It is located in: `Toolboxes/AudioCalibPlatform`.

A.2 New RMP

The new RMP part is located in `ROSWrapper`. In `ROSWrapper/src/`, there are:

- `cmake`: cmake instruction to find old RMP shared libraries. It could be used if you want to have communication between old RMP applications and ROS node.
- `lito_driver`: audio and motor driver of lito. It contains also code for old RMP drivers but it is no more compiled since a while.
- `nao_driver`: nao driver (audio, motion, video, video and face detection together).
- `recorder`: tools to record drivers messages. There are a GUI if needed.

- `robot_com`: global ROS messages definition.
- `sound`: tools useful for sound manipulation like recording in a wav, or reading a wav file and output it the audio topic.

To do a classic compilation, type:

```
$ cd ROSWrapper
$ catkin_make
```

Or read `ROSWrapper/README.txt` for more details.

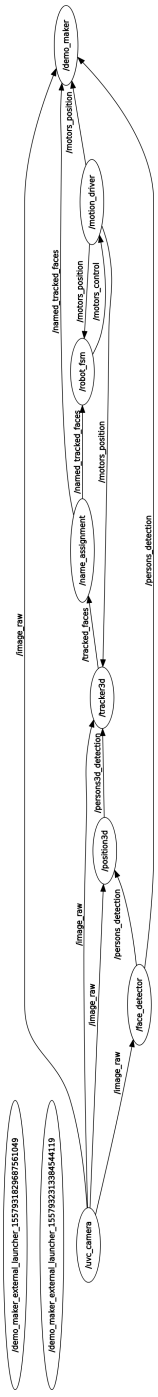

Figure 2: Tracking only

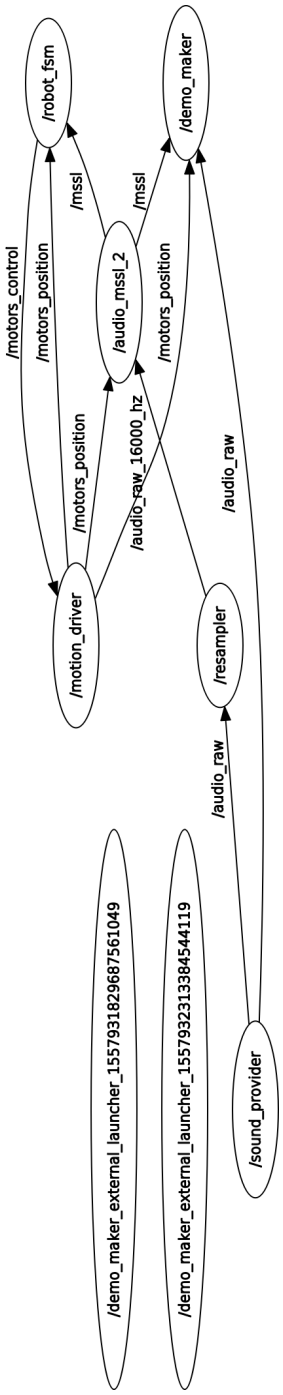

Figure 3: Audio sound source localisation only

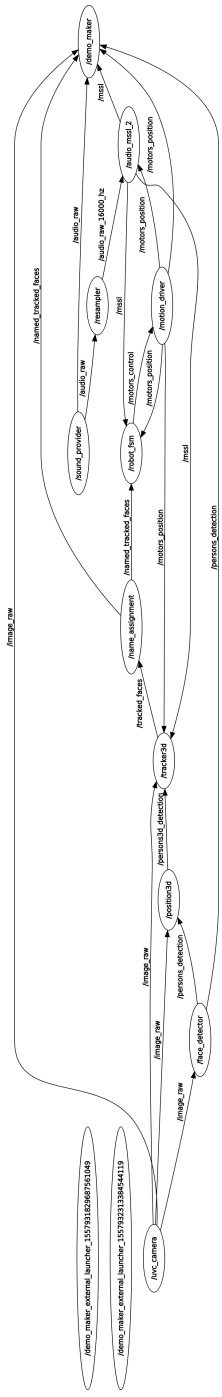

Figure 4: MSSL and face tracking

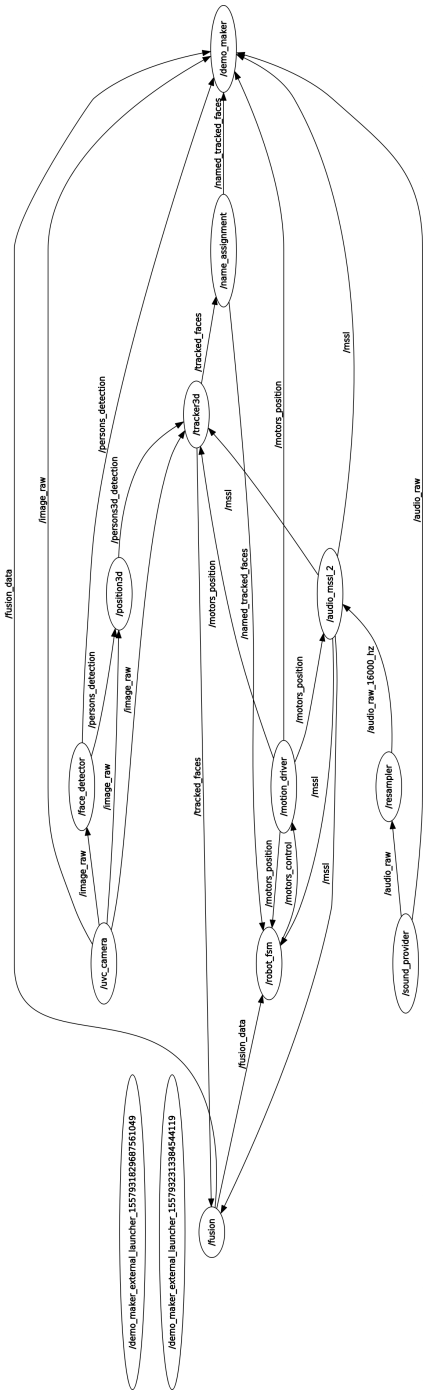

Figure 5: MSSL, face tracking and a basic fusion

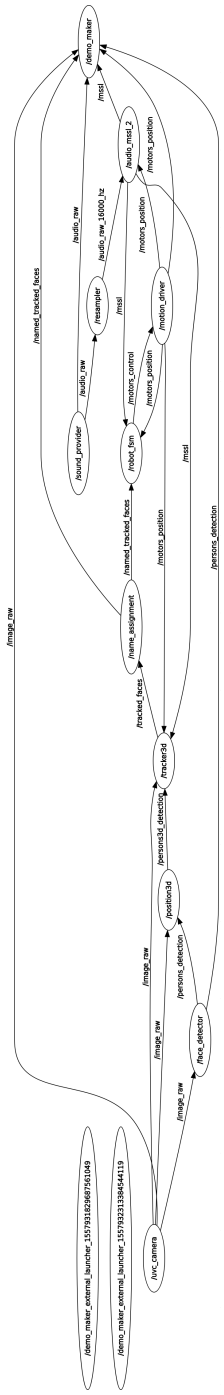

Figure 6: MSSL, face tracking and fusion inside the tracking algorithm

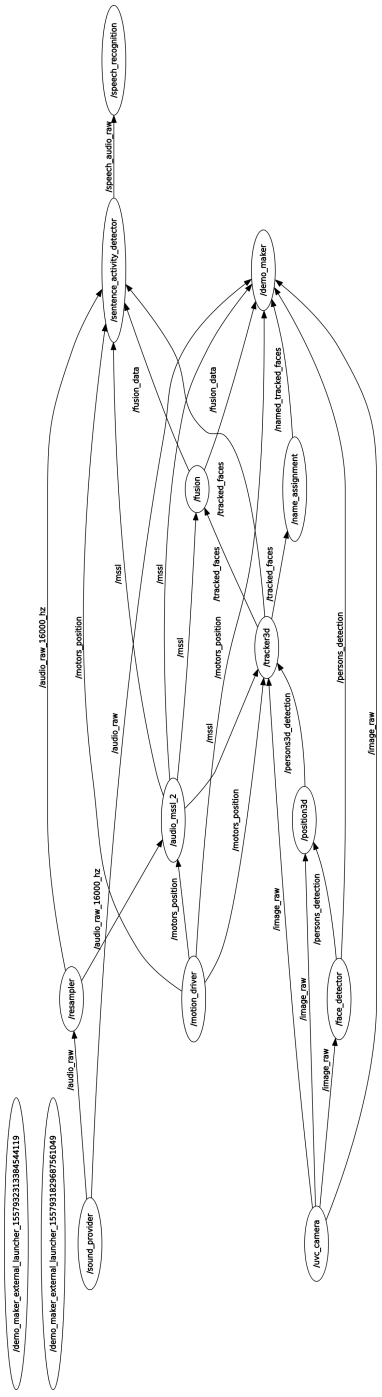

Figure 7: Speech recognition

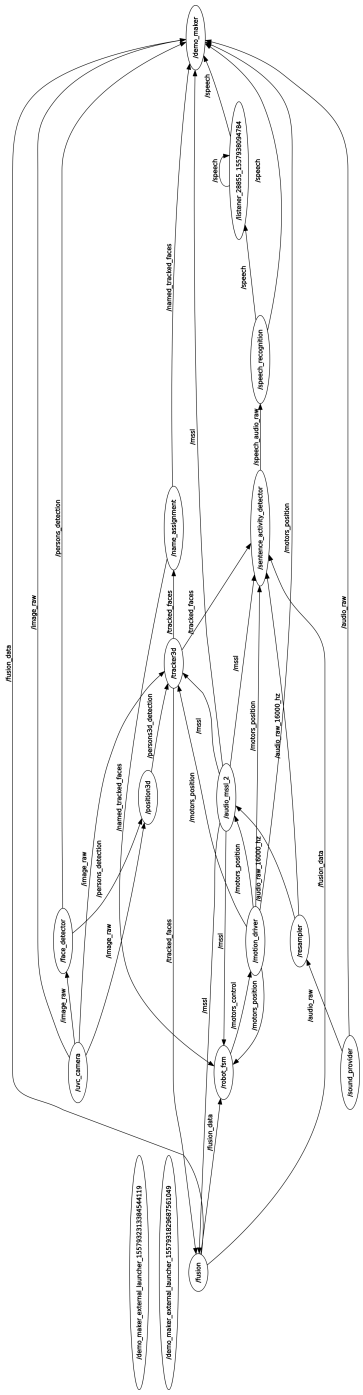

Figure 8: Dialog system

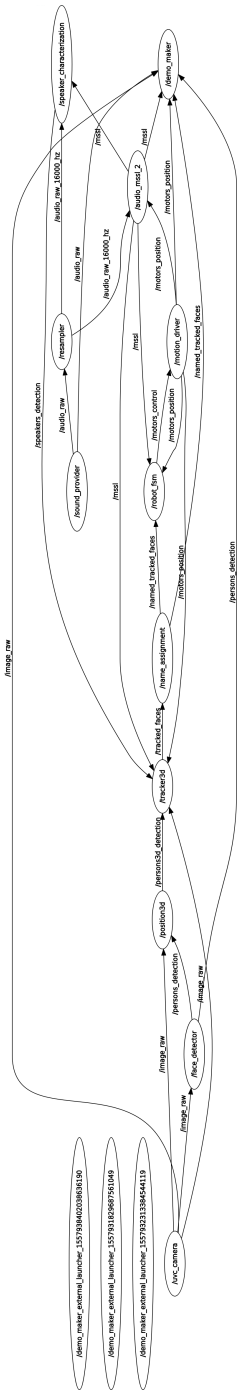

Figure 9: Audio visual tracking