Universität Hamburg
Computer Vision Group

Sound localization on Turtlebot

Débora Ferreira dos Santos

November 19, 2024

1 Overview

This project aims to enable audio-visual localization on a Turtlebot 2i robot, using ROS to integrate Kinect input for movement control based on the sound source localization model OG-AVC-DETR from the paper Acoustic and Visual Knowledge Distillation for Contrastive Audio-Visual Localization. This work is part of the CML project. The setup follows:

- Turtlebot 2i;
- Azure Kinect
- Laptop (cvnb3 NVIDIA GeForce RTX 4090)
  - Ubuntu 20.04; ROS Noetic; Turtlebot packages; Azure Kinect driver;

2 Robot Design

To attach the laptop to the robot base, we designed a larger acrylic platter that could be mounted onto the robot's existing structure. For reference, the structure used in previous projects is shown in Figure 2.

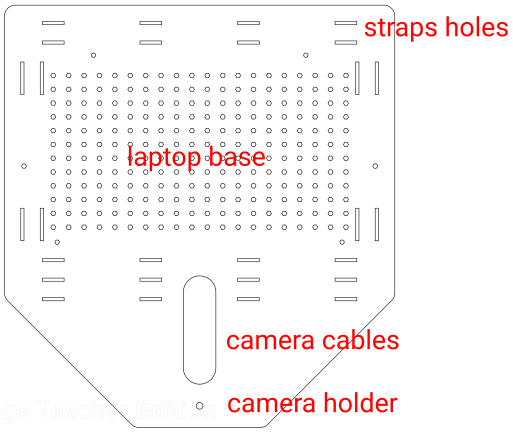
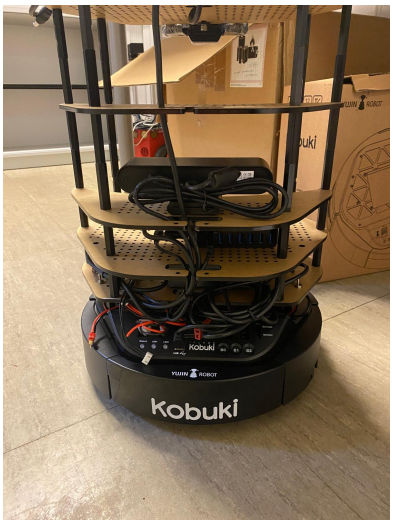


Figure 1: Plate design

The Turtlebot 2i original model CAD files can be found here. Parting from the top plate, we designed a new plate that could comport the laptop and a Kinect camera. The final files can be found here. It was manufactured from a 4 mm acrylic glass GS. This material supports drilling, which might be convenient in future projects. The camera is attached to the plate with a universal camera tripod head with ball socket to allow for adjustment of the camera angle.

(a)



(b)

(c)

Figure 2: Structure of the robot before the project started for reference .

3 Turtlebot Packages Installation

The Turtlebot2 packages are not distributed for ROS Noetic, so it must be installed manually. First, install *catkin-tools* and create a workspace

```
$ sudo apt install python3-catkin-tools
$ mkdir -p ~/turtlebot_ws/src
```

Install the packages listed in the Turtlebot ROS Wiki page by cloning in the *src* folder the newest github branch of each package. Since the CV group robot does not have and arm, the package *turtlebot_arm* is not necessary.

```
$ cd turtlebot_ws/src/
$ git clone https://github.com/turtlebot/turtlebot.git
$ cd turtlebot/
$ git checkout melodic
$ cd ..
$ git clone https://github.com/turtlebot/turtlebot_msgs/tree/indigo
$ git clone https://github.com/turtlebot/turtlebot_msgs.git
$ git clone https://github.com/turtlebot/turtlebot_apps.git
$ git clone https://github.com/turtlebot/turtlebot_interactions.git
$ git clone https://github.com/turtlebot/turtlebot_simulator.git
```

```
$ git clone https://github.com/turtlebot/turtlebot_arm.git
$ git clone https://github.com/yujinrobot/kobuki.git
$ git clone https://github.com/yujinrobot/kobuki_msgs.git
$ git clone https://github.com/yujinrobot/yujin_ocs.git
```

Next, when trying to build, some packages may fail. As a quick fix, we simply deleted the *ar_track_alvar* folder inside the *ar_track_alvar* package folder. Some dependencies may also not be met. Install the extra packages as required.

```
$ git clone https://github.com/yujinrobot/yocs_msgs.git
$ git clone https://github.com/ros-perception/ar_track_alvar.git
$ catkin build --continue
```

Then source the workspace created.

```
$ echo "source ~/turtlebot_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Test the installation by running gazebo.

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

In case of error, try using gazebo with the following command

```
$ gazebo /export/home/1ferreir/turtlebot_ws/src/turtlebot_simulator/
   turtlebot_gazebo/worlds/playground.world
```

4 Azure Kinect Installation

4.1 Azure Kinect SDK

Install required packages, clone Azure Kinect SDK repository

```
sudo apt update
sudo apt install -y cmake git build-essential
git clone https://github.com/microsoft/Azure-Kinect-Sensor-SDK.git
cd Azure-Kinect-Sensor-SDK
```

Build SDK

```
mkdir build
cd build
cmake ..
make
```

Test installation running the binaries

```
k4arecorder --help
k4aviewer --help
AzureKinectFirmwareTool --help
```

In case of errors, check the Appendix A.1.

4.2 Azure Kinect ROS Wrapper

```
$ git clone https://github.com/microsoft/Azure_Kinect_ROS_Driver/tree/
   melodic
$ catkin build
$ source ~/turtlebot_ws/devel/setup.bash
4 roslaunch azure_kinect_ros_driver driver.launch
```

5 Models

5.1 EZ-VSL Model

The EZ-VSL Model was used initially to familiarize ourselves with the project, as it served as the reference for developing the final model, which is detailed in the subsequent sections. Since the EZ-VSL model is not used for the final results, its installation is not essential.

To clone the EZ-VSL repository, use

```
$ git clone https://github.com/stoneMo/EZ-VSL.git
```

While the EZ-VSL model is not used for final results, the dataset can still be utilized with the new model. Download the dataset and organize the files by placing audio files in a folder named "audio" and images in a folder named "frames".

To test the model in inference mode on the Flickr dataset, execute the following command:

```
python test.py --test_data_path /export/home/1ferreir/EZ-VSL/Dataset/ --
    test_gt_path /export/home/1ferreir/EZ-VSL/Dataset/Annotations/ --
    model_dir checkpoints --experiment_name flickr_10k --
    save_visualizations --testset 'flickr' --alpha 0.4
```

5.2 OG-AVC-DETR Model

The OG-AVC-DETR model is the actual model of interest in this project and is available on this repository. More details can be found on the associated paper.

6 Development

As a first step, we obtained a demo for the laptop only, using frames and audio obtained from webcam and microphone, as seen in https://github.com/dfsbora/og-avc. Then, we moved to the ROS framework in order to incorporate the Kinect data and control the robot base. ROS package repository available here https://github.com/dfsbora/audio_visual_localization.

6.1 Inference on laptop acquired data

The code was first modified to run on the laptop with data acquired from the laptop camera and microphone. After cloning the OG-AVC-DETR Model repository, the script datasets.py was modified and two scripts were added to the project: inference_webcam_image.py and inference_webcam_video.py.

First, the AudioVisualDatasetInference class was created in the datasets.py file. The script inference_webcam_image.py modifies the original script test_N_times.py to support inference using individual frames from the webcam. The class NeuralNetworkNode handles the model setup, data recording, and inference validation, as well as the organization and cleanup of directories used for storing temporary audio and image data. Usage:

```
$ python inference_webcam_image.py --experiment_name test_run1 [--model_dir
    checkpoints] [--recording_duration 3]
```

The script inference_webcam_video.py modifies the former one to run inference over recorded videos. The resulting video displays all fours obtained predictions side by side. Usage:

```
$ python inference_webcam_video.py --experiment_name test_run1 [--
    save_visualizations]
[--model_dir checkpoints] [--recording_duration 10.0] [--segment_duration
    1.5] [--data_dir data_video]
```

6.2 Inference on ROS framework

To acquire the Kinect data and control the robot, we use the ROS framework. The workspace of this project was already created during the installation step. The parts of interest are organized as follows

```
turtlebot_ws
|- src
    |- audio_visual_localization                # Main project package
        |- checkpoints                          # Model checkpoints
        |- data                                 # Collected audio/frame
        |- launch                               # Custom launch files
            |- kinect_driver_server.launch      # Launches Kinect capture
        |- src                                  # Source code directory
        |- srv                                  # Custom ROS service
        |- ...
    |- azure_kinect_ros_driver                  # ROS package for Kinect
    |- ...
```

6.2.1 Kinect data acquisition

The Kinect data acquisition is managed by the RecordKinect service. The file kinect_driver_server.launch can be launched with

```
$ roslaunch audio_visual_localization kinect_driver_server.launch
```

then the driver is initialized and the server node is started, which will then publish save the captured images and audios in the specified directory, and return a success flag to the client. The main code is responsible for requesting the data and further processing it.

6.2.2 Inference and Object Tracking

The node neural_network_node.py runs the inference using the class NeuralNetworkNode as described above. It contains some modifications to accomodate to the ROS setup: the method find_kinect_index that identifies the device index of the Kinect sensor, and and compute_angle, that are explained below. The method run controls a ROS loop that requests for the RecordKinect service, creates a dataloader based on that data, performs inference, find angle i.r.t. to the robot and publishes it as a topic.

We implemented a simple solution as a baseline for finding the orientation to move the robot to. In the method find_centroid, we start with the heatmap generated by the neural network. Then we use a threshold to find blobs, taking the largest one as the primary sound source. Finally, we calculate its centroid, and we can then estimate the angle of this source i.r.t. the robot orientation in the method compute_angle. We use this angle as a command for the robot movement.

6.2.3 Robot control

The robot base movement towards the sound source orientation is controlled by the node move_base, which uses the messages from the topic sound_source_angle published by the main node. It calculates the target orientation and control the movement of the robot until it reaches such orientation. Future improvements could include a finer control of target orientation as it uses only the first update during the whole movement, whereas the sound source localization could already have changed. The node simulated_sound_source_angle_publisher allows for testing the movement in isolation from the rest of the system by publishing angle messages. Modify the test as necessary.

6.2.4 Usage

Terminal 1 - ROS/Robot Initialization

For laptop only execution:

```
roscore
```

For turtlebot execution:

```
roslaunch turtlebot_bringup minimal.launch
```

Terminal 2 - Kinect Initialization

```
roslaunch audio_visual_localization kinect_driver_server.launch
```

Terminal 3 - Main node

```
cd turtlebot_ws/src/audio_visual_localization/src/
rosrun audio_visual_localization neural_network_node.py --experiment_name
    test_run1 --model_dir checkpoints --recording_duration 3
```

Terminal 4 - Movement control

```
rosrun audio_visual_localization move_base.py
```

If testing in isolation, use the simulated angle publisher:

```
rosrun audio_visual_localization simulated_sound_source_angle_publisher.py
```

7 Ackowledgments

Reinhard Zierke - Ubuntu installation
Niklas Fiedler - TAMS - ROS and turtlebot installation
Germán Junca - TAMS - Acrylic plate design

A Installation Errors

A.1 Azure Kinect errors

A.1.1 Error: Missing Development Libraries

```
sudo apt install -y libudev-dev
sudo apt install -y libsoundio-dev
```

A.1.2 Error: Error Finding `libsoundio`

Since `libsoundio` is not critical for basic functionality, exclude building `k4aviewer`

```
cd ~/Azure-Kinect-Sensor-SDK
mkdir build_no_viewer
cd build_no_viewer
cmake .. -DBUILD_VIEWER=OFF
make
```

A.1.3 Error: `k4arecorder` Command Not Found.

```
#Verify the presence of the k4arecorder binary
ls /usr/local/bin/k4arecorder

#If the file is missing, clean and rebuild the project:
cd ~/Azure-Kinect-Sensor-SDK/build_no_viewer
make clean
make
sudo make install

#Check if the binaries are installed correctly:
ls /usr/local/bin/k4aviewer
ls /usr/local/bin/k4arecorder
ls /usr/local/bin/AzureKinectFirmwareTool
```

A.1.4 Error: `k4arecorder` Error - Missing Library `libk4arecord.so.1.4`

```
k4arecorder: error while loading shared libraries: libk4arecord.so.1.4:
   cannot open shared object file: No such file or directory
```

```
#Verify the presence of library files:
ls /usr/local/lib/libk4arecord*

#Update the library cache
sudo ldconfig

#Verify library paths:
cat /etc/ld.so.conf
cat /etc/ld.so.conf.d/*

#Ensure /usr/local/lib is listed.

#Check and update LD_LIBRARY_PATH
echo $LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
echo 'export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH' >> ~/.bashrc
source ~/.bashrc
```

A.1.5 Error: `libsoundio1`

```
# Instructions from https://atlane.de/install-azure-kinect-sdk-1-4-on-ubuntu-22-04/

# Intall libsoundio1, a dependency for k4a-tools
# https://packages.ubuntu.com/focal/amd64/libsoundio1/download
$ wget mirrors.kernel.org/ubuntu/pool/universe/libs/libsoundio/libsoundio1_1.1.0-1_amd64.deb
$ sudo dpkg -i libsoundio1_1.1.0-1_amd64.deb
$ sudo apt install -y k4a-tools

# Copy udev rules to not have to use sudo
$ sudo cp 99-k4a.rules /etc/udev/rules.d/

# Check if the Microsoft devices are correctly recognized
$ lsusb | grep "Microsoft"
```

A.1.6 Error: `libdepthengine` not found

```
#Download and install the Microsoft APT Configuration Package:
$ wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb
$ sudo dpkg -i packages-microsoft-prod.deb

#Install Required Packages:
$ sudo apt update
$ sudo apt install -y k4a-tools libk4a1.4

#Find the Location of ‘libdepthengine.so.2.0‘:
$ sudo find / -name libdepthengine.so.2.0

#output:
#/usr/lib/x86_64-linux-gnu/libk4a1.4/libdepthengine.so.2.0

# Update the ‘LD\_LIBRARY\_PATH‘ Environment Variable:
$ echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/x86_64-linux-gnu/libk4a1.4/' >> ~/.bashrc
$ source ~/.bashrc

# Check the the Library Dependencies:
$ ldd /usr/lib/x86_64-linux-gnu/libk4a1.4/libdepthengine.so.2.0

#Run the ‘k4aviewer‘ Tool:
$ k4aviewer
```

A.1.7 Error: Device access permissions (udev rules)

```
$ sudo nano /etc/udev/rules.d/99-k4a.rules
```

Contents of `99-k4a.rules`:

```
BUS!="usb", ACTION!="add", SUBSYSTEM!=="usb_device", GOTO="k4a_logic_rules_end"

ATTRS{idVendor}=="045e", ATTRS{idProduct}=="097a", MODE="0666", GROUP="plugdev"
ATTRS{idVendor}=="045e", ATTRS{idProduct}=="097b", MODE="0666", GROUP="plugdev"
```

```
ATTRS{idVendor}=="045e", ATTRS{idProduct}=="097c", MODE="0666", GROUP="
    plugdev"
ATTRS{idVendor}=="045e", ATTRS{idProduct}=="097d", MODE="0666", GROUP="
    plugdev"
ATTRS{idVendor}=="045e", ATTRS{idProduct}=="097e", MODE="0666", GROUP="
    plugdev"

LABEL="k4a_logic_rules_end"
```

```
$ sudo udevadm control --reload-rules
$ sudo udevadm trigger
```

A.1.8 Diagnosis

```
#Diagnosing USB connection and communication issues.
$ dmesg | grep -i usb

# Testing if \texttt{k4aviewer} can access the devices.
$ k4aviewer

# Verifying that the SDK and components are correctly installed.
$ dpkg -l | grep k4a

# Ensure that \texttt{k4arecorder} works:
$ k4arecorder --list
```

A.2 Pyaudio errors

```
ERROR: Could not build wheels for pyaudio, which is required to install
    pyproject.toml-based projects

$ sudo apt install portaudio19-dev
```