University of Nevada, Reno

Speech Interface for NAASIC's Autonomous-driving Car

A thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Science in Computer Science & Engineering and the Honors Program

by

Sybille F. Horcholle

Dr. Richard Kelley, Thesis Advisor

May, 2019

Department of Computer Science and Engineering
University of Nevada, Reno

Pythia: Speech Interface for NAASIC's Autonomous-driving Car

Final Demo Report

Team #16: Sybille Horcholle, Bryson Lingenfelter, Nathan Thom

May 7th, 2019

Advisor: Richard Kelley
Instructors: Sergiu Dascalu and Devrin Lee

Contents

Abstract iv
Project Summary iv
Functionality Implemented iv
Functionality Not Implemented v
Team Contributions vi
Software Overview vii
unr_deepspeech (Python, 1 module) vii
Talker (C++, 1 module) viii
Listener (Python, 3 modules) viii

Abstract

Pythia is a speech interface for UNR's self-driving car which allows intuitive two-way communication between driver and vehicle. Users can obtain real-time information about obstacle detection and vehicle status simply by asking, and at no point needing to take their eyes off the road. The speech interface also allows the car to make announcements when critical events, such as pedestrians entering the vehicle's path, occur. By providing a simple, yet powerful and easily extensible interface to the vehicle, Pythia allows drivers to safely check that the vehicle is operating correctly.

Project Summary

The speech interface for NAASIC’s self-driving car project has been successfully implemented. This document will cover the functionality implemented, the use-cases that were not implemented and why, the contributions of each team member, and a description of the source code and its functions. Overall, the project was successful due to the team’s efforts to organize and implement the project and thanks to advisor Richard Kelley and the graduate students.

Functionality Implemented

The following use-cases were successfully implemented for Innovation Day:

1. The speech interface listens to a set of spoken commands
    - Is a pedestrian detected?
    - What’s the speed limit?
    - Where am I?
    - How much fuel do I have?
    - Repeat the announcement
    - Turn off

2. The speech interface announces certain events without being prompted
    - The current speed limit if a road sign has been seen
    - A pedestrian is detected close to the car

3. The speech interface announces if it does not understand a command
4. The speech interface can interpret multiple variations on a command, such as “do you see the sign?,” “do you see that sign?,” and “can you see the sign?”
5. The speech interface can be turned on and off by both voice command and buttons on the display
6. The speech interface is switched on and off through a terminal command
7. The speech interface translates spoken commands to readable text
8. The speech interface translates readable text to commands which can be executed by the car
9. The speech interface can synthesize text in a way interpretable by passengers (via written and oral communication)
10. The speech interface translates notifications from the car regarding sign and pedestrian detection into human-readable statements
11. The speech interface displays the command it executes to the user
12. The speech interface signals when it is hearing speech by changing color according to its status
13. The speech interface clearly displays the most recent announcement on the interface display

Functionality Not Implemented

The two main functionalities originally proposed for the project was GPS capabilities and directly controlling the car through oral input. Several use-cases involving inputting a destination by spoken command and querying for information regarding the destination were not implemented in the final design. Since GPS capabilities are available in most systems existing in a vehicle, the team decided to invest more time in the speech recognition functionality and develop a unique interface for a self-driving car, rather than recreate a common utility.

The team also originally considered giving the interface functionality to control the car, such as providing feedback if the car required more information and giving commands to initiate certain maneuvers such as changing lanes or giving control back to the driver. Due to the safety risks involved in implementing this feature and the currently limited self-driving capabilities of the car used to test the interface, the team decided to remove these use-cases from the project.

Team Contributions

Sybille Horcholle:
Sybille developed the graphical unit of the interface located in the Listener module of the source code. The TKinter GUI tool from Python was used to develop the main screen and configuration tab of the interface. Sybille developed the functionality of changing the font size of the interface, the drop-down menu to select a different voice, and modifying the wake word.

During the testing phase, Sybille tested the command functionality of the interface. The wake word was tested to ensure the interface recognized the wake word and updated the display accordingly when it was detected. Sybille also tested the command for the fuel level of the car and the command to shutdown the interface. Overall, Sybille spent approximately 15 hours in development and testing.

Bryson Lingenfelter:
Bryson expanded the interface designed in the fall semester to be activated by a wake word and be more aggressive about identify speech. He also integrated data from the mobileye and GNSS system into the interface, allowing queries about pedestrians and location, and keeping track of detected signs to report the speed limit. He tested low level features including command translation accuracy and verification of published ROS topics. Bryson spent an estimated 25 hours on the final part of the project.

Nathan Thom:

Nathan Thom developed the speech portion of the interface. Using packages from FestVox, Nathan integrated the application into the talker module of the interface to produce an audio response. In the testing phase of the project, Nathan tested the functionality of pedestrian and sign detection commands. He also verified the exit functionality of the interface, testing both the command and the interface button. Nathan worked an estimated 18 hours on the project development.

Software Overview

The complete ROS workspace for Pythia contains six packages, of which two were developed by the senior project group and four are external dependencies. The two ROS packages developed for the project are talker and listener. Additionally, a standalone package, unr_deepspeech, was released on Github. Note that almost all of the code for this package was reused for Pythia, and should not be considered separately developed code. It is provided in addition to the Pythia code because it was released separately from the Pythia code.

unr_deepspeech (Python, 1 module)

unr_deepspeech: this was the first module developed for Pythia. An open source ROS wrapper for Mozilla's Deepspeech speech-to-text code and model, featuring command comparison and spell checking. Currently maintained on Richard Kelley's Github page.

- deepspeech_node.py: core speech-to-text system. Generates text from an audio frame using a provided model for Deepspeech. If configured to do so, it will perform spell checking using a dictionary or convert the final output to the nearest match in a list of commands.
- unr_deepspeech_server.py: runs the deepspeech node as a ROS service, allowing usage and parameter setting through ROS.
- unr_deepspeech_client.py: example client for the deepspeech service. Allows mic selection and recording. Output from deepspeech is printed to the terminal.

Talker (C++, 1 module)

talker: ROS node for translating natural language text into audible speech. Advertises a ROS service which accepts a string as input and plays back speech as a side effect.

- talker.h / talker.cpp: implementation of the talker class. Acts as a wrapper for festvox and ROS's sound_play module to synthesize speech.
- talker_node.cpp: runs the talker class as a ROS node.
- talker_client.cpp: tests the talker class by calling the service then exiting.

Listener (Python, 3 modules)

Monitor: subscribes to several topics published by the vehicle to be used for event-driven alerts. Interfaces with talker to make announcements.

- monitor.py: subscribes to driverless mode status and mobileye object reporting topics. Uses these topics to announce when the vehicle enters driverless mode and when a pedestrian enters the vehicle's trajectory.

unr_deepspeech: the code for deepspeech_node and unr_deepspeech_server is used as it appears on Github and is described above. The code for unr_deepspeech_client was modified for use in pythia_client.

pythia_client:

- pythia_client.py: the main loop for Pythia. Uses code adapted from the original unr_deepspeech_client to constantly record audio, process using the deepspeech server, and generate responses using the process_speech code. This file also contains code for the GUI, which is updated as the interface receives information. Code for settings, such as interface size, current wake word, and current voice, can also be found in this file.
- process_speech.py: given text output provided by Deepspeech. Generates a response from the vehicle by determining the provided command and using vehicle data to create a response. The user can ask for current fuel level, speed, pedestrian detection information, location, and, if any signs have been seen so far, the speed limit.