Integrating Google Assistant and Pepper Robot

Greg Dumas and Shawn Greenwood
CSC 212: Introduction to Software Engineering
Spring 2022

Abstract

The Pepper robot, released by SoftBank Robotics in 2014, was developed to be a social robot with the primary function of interacting with humans. The concept holds immense promise for commercial, educational, and healthcare-related applications. The fulfilment of that promise, however, has been limited in part by Pepper's rudimentary default conversational capabilities. Contemporary "smart assistant" technologies such as Amazon Alexa, Apple Siri, Google Assistant, and Samsung Bixby can provide users with richer conversation and expanded functionality but are not readily available for use on Pepper robots. Software Development Kits (SDKs) provide one potential means to connect these smart assistants with Pepper. In this paper we prototype one such implementation using the Google Assistant SDK and the NAOqi Python SDK from SoftBank Robotics.

Introduction

Vast sums of capital and countless hours of labor have been invested in the research and development of robotics in the twenty-first century, leading to a flood of discoveries and achievements. Incredible feats of mechanical engineering have produced industrial robots with the strength to bend steel and the precision to handle eggshells; videos of "dancing" robots from Boston Dynamics demonstrate tremendous gyroscopic sophistication. The subfield of social robots, with the primary task of interacting with humans, remains a daunting challenge.

To be socially successful, a robot must effectively communicate with people. Pepper robot, as previously described, is intended for a human-facing role, but is hampered by limited conversational capacity. Our work is motivated by the goal of making Pepper a more successful social robot by integrating services that improve its capabilities to converse.

We organize the paper as follows. First, we explore related work on the development of social robotics and literature focused on Pepper robot itself. Second, we lay out the solution we developed to enable the use of Google Assistant with Pepper and discuss the technical details of our implementation. We will offer an assessment of the solution's performance and address potential avenues for future work. Finally, we offer concluding remarks.

Related works

In recent years a sizable body of literature has emerged in the field of social robots. Feil-Seifer and Matarić (2018) establishes a definition of socially assistive robots and creates a framework for understanding how and why a social robot such as Pepper might be utilized. The publications of Anghel et al (2020), Gomez-Donoso et al (2019), and Park et al (2019) illustrate the prominence of possible uses of social robots in healthcare and eldercare settings. Additional scholarship exists specifically for Pepper as well. Gardecki & Podpora (2017) examine the construction, capabilities, and limitations of both the NAO and Pepper robots. Lleonsí Carrillo (2017) provides a detailed look into the development process on Pepper robot, and Ghiță et al (2020) reflect on the use of both development tools provided by SoftBank Robotics as well as the Robot Operating System (ROS). Our work builds on this foundation by demonstrating how existing development tools can be combined with new technologies to improve the ability of a social robot, Pepper, to fulfill its intended social role.

Solution

The project's design would be dictated by a three-step process. First, a user speaks to the Pepper robot. The conversation would then be conveyed to Google Assistant for processing and response. Finally, the Google Assistant's response would be played back to the user over the Pepper robot's built-in speakers.

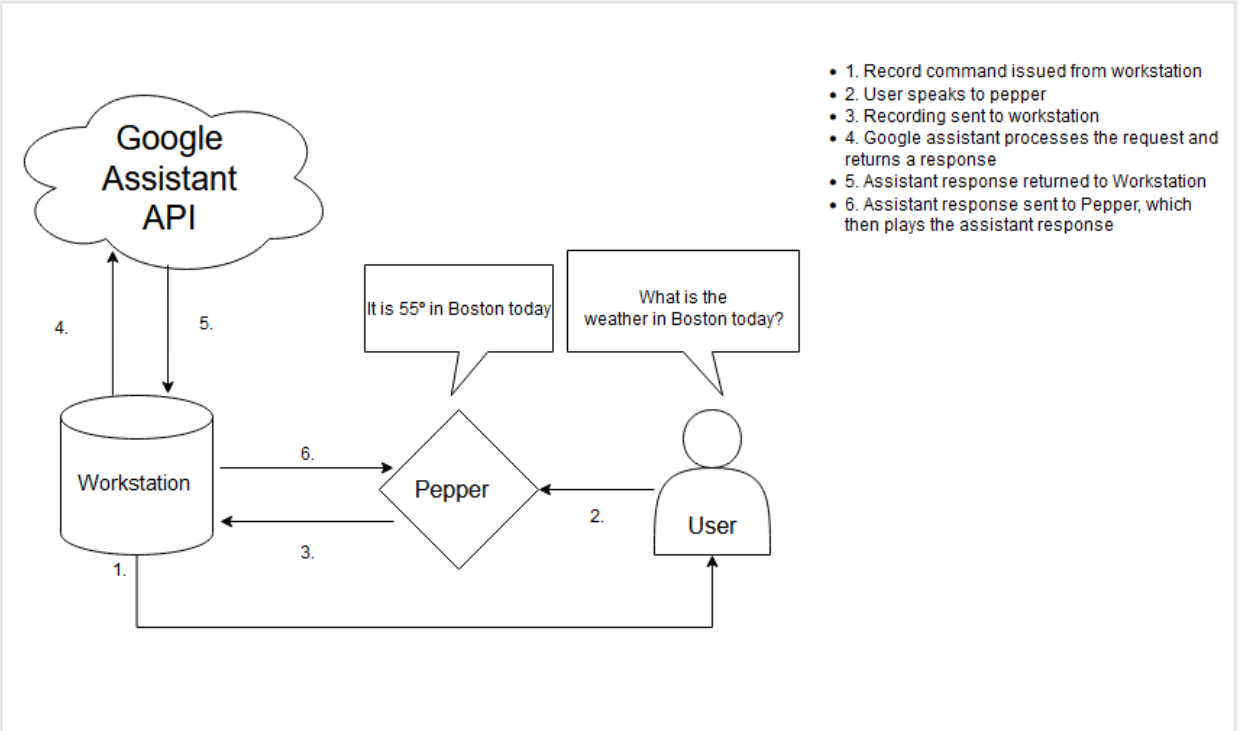

Figure 1: Project design overview

Two SDKs were crucial to the implementation. The SoftBank PyNaoqi SDK provided a means to access key features of the Pepper robot. The Google Assistant SDK allowed us to make requests and receive responses from the Google Assistant API. The technical features of each SDK will be explored in more depth in the next section.

Implementation

In this section, we describe the concrete steps taken to implement the desired functionality. The solution developed in this paper requires a Pepper robot and a separate workstation, which serves to interface between Pepper and the Google Assistant.[1]

Setup

Operating System & Development environment

For compatibility purposes with SoftBank tools, our workstation was installed with Ubuntu 16.04. In order to interface with each other, Pepper and the workstation should be connected over a local network. The workstation development environment is then configured. Before installing any software, a Python virtual development environment is set up.[2]

```
$ sudo apt-get update
$ sudo apt-get install python-dev python-virtualenv
$ virtualenv env --no-site-packages
$ env/bin/python -m pip install --upgrade pip setuptools wheel
$ source env/bin/activate
```

Figure 2: Installing a virtual environment for Python 2.7 (credit: Google Developers documentation)

Once the virtual environment is enabled, the SDKs can be set up without risk of conflicting with other Python installations on the workstation. Because the workstation was solely used for this project, the actual risk of conflicting installations was minimal. However, the authors do not assume this will always be the case and recommend the use of Python virtual environments as a general best practice.

Google Assistant SDK

First, we install software dependencies for the Google Assistant SDK.

[1] Please see Appendix B for a brief overview of other attempted solutions.
[2] Note: the command “virtualenv env” creates a virtual environment with the name “env”. In future code snippets where the virtual environment is active, this is indicated by the virtual environment name being listed before the terminal prompt.

```
(env) $  sudo apt-get install portaudio19-dev libffi-dev libssl-dev
```

Figure 3: Installing Google Assistant SDK dependencies (credit: Google Developers documentation)

Using pip, the Python package manager utility, we install the latest version of the Google Assistant SDK.

```
(env) $  sudo apt-get install portaudio19-dev libffi-dev libssl-dev
```

Figure 4: Installing Google Assistant SDK using pip (credit: Google Developers documentation)

Separately, a Google Cloud Platform project must be configured to manage access and authorization to the Google Assistant API.[3] A credential authorization tool is then installed, and credentials generated. Once successfully authenticated, google-assistant-sdk can be used to send requests via audio stream, pre-recorded WAV file, or in text format. The Google Assistant service returns a response as an audio stream that can be played back or saved to a file.

NAOqi Python SDK

The NAOqi Python SDK is obtained via download from SoftBank Robotics as a .tar archive. This archive is then extracted to a directory on the workstation, and the SDK is then enabled by adding to the Python path. The SDK can be temporarily enabled via terminal command or enabled from within a Python script as shown below.

```
1  import sys
2  sys.path.append('/home/gdumas/pynaoqi/lib/python2.7/site-packages')
```

Figure 5: Adding the NAOqi Python SDK to PATH

The NAOqi Python SDK functionality can now be utilized via an import statement (line 4). In the following example, an object is created specifying the Pepper robot's IP address (line 5). This information, the robot's port number, and the NAOqi module to use are then supplied

[3] See the Google Developers page "Configure a Developer Project and Account Settings" for more detailed information. Available at https://developers.google.com/assistant/sdk/guides/service/python/embed/config-dev-project-and-account.

to an ALProxy object (line 6). In this case the “ALAudioPlayer” module is chosen, and the robot is then directed to play an audio file from a location in its local file storage (line 7).

```
4  from naoqi import ALProxy
5  pepper = '192.168.1.140'
6  audio = ALProxy("ALAudioPlayer", pepper, 9559)
7  audio.playFile("/home/nao/syn.wav")
```

Figure 6: Connecting to Pepper and playing an audio file via NAOqi Python SDK

Execution

With setup complete, we now turn to the code. The program begins in a bash script, driver.sh (see appendix A). First, an SSH connection is established between the workstation and Pepper robot. Pepper then begins recording with its microphones using the arecord utility, which captures the user’s query in a WAV file. Once recording has finished, the audio file is transferred back to the workstation via the secure copy (SCP) utility. The audio file is then used as an input to the Google Assistant SDK, which returns the Assistant response as a second WAV file called output.wav. SCP is used again, this time copying the recording of the Google Assistant back to Pepper.

The second file is a Python script, play_response.py (see Appendix A). This program begins once the audio file containing Google Assistant’s response has finished copying to the Pepper robot. First, play_response.py adds the NAOqi Python SDK to the system PATH, which as previously described allows the use of NAOqi functionality through creating proxies to modules provided by SoftBank Robotics.[4] The script then establishes a connection to Pepper using the robot’s IP address, port number, and NAOqi module name. The ALAudioPlayer module is used,

[4] The NAOqi Python API documentation states that developers can write their own NAOqi modules in Python using the ALModule(name) class, but that work falls outside the scope of this project. For more information please see http://doc.aldebaran.com/2-5/ref/python-api.html.

which enables Pepper to play the Google Assistant's response back to the user through the robot's built-in speakers.

Future works

The described implementation met the project's objective, as users could interact with Google Assistant using Pepper robot. In most tests, Pepper successfully records audio, passes the request to Google Assistant through the networked workstation, and plays back a waveform audio file of the assistant's response in a matter of seconds. However, many opportunities for further refinement and continued research remain. One limitation of the current approach is that Pepper must be cued to begin listening by terminal command on a locally networked workstation. Preferably, Pepper should be able to begin recording a user command with the use of a wake word or other signal from the end user.

A second area of improvement would be to integrate Pepper's expressions into the assistant interactions. The Pepper robot is designed with the ability to gesture using its articulated limbs, facial expressions, and LEDs. Making use of these features would create a more engaging experience for users.

Another potential avenue for future work would be to generalize the solution to work with a wider range of robots using an open source framework such as the Robot Operating System (ROS). As Ghiță (2020) notes, "NAOqi provides rich development capabilities, but its downside is that its application is limited to Softbank's robots. The developer is also tied in exploiting the [built-in] modules such as user detection, speech recognition or navigation" (pg. 4) The available modules are further constrained by SoftBank's shift in development focus - the last NAOqi release was published in October 2018. Thus, switching to an implementation

rooted in ROS could enable deployment on a wider range of robots beyond SoftBank's own product lineup, and enable use of new APIs and functionality beyond what is currently possible using the SoftBank development stack.

Conclusion

In this paper we introduced the Pepper robot, discussed its place in context of current literature on social robotics, and presented a solution which enabled Pepper to use Google Assistant. Despite several developmental dead-ends, a successful implementation was created using Bash and Python 2.7, which leveraged functionality provided by the Google Assistant and NAOqi Python SDKs. Although SoftBank Robotics has discontinued manufacturing of the Pepper robot, a substantial amount of opportunity exists for continued work with Pepper - both to refine the implementation we present here, and to explore the integration of other capabilities.

References

Anghel, I., Cioara, T., Moldovan, D., Antal, M., Pop, C. D., Salomie, I., Pop, C. B., & Chifu, V. R. (2020). Smart Environments and Social Robots for Age-Friendly Integrated Care Services. International Journal of Environmental Research and Public Health, 17(11).

Feil-Seifer, David, and Maja J. Mataric. "Defining socially assistive robotics." *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.*. IEEE, 2005.

Foster, M. E., Alami, R., Gestranius, O., Lemon, O., Niemelä, M., Odobez, J. M., & Pandey, A. K. (2016, November). The MuMMER project: Engaging human-robot interaction in real-world public spaces. In *International Conference on Social Robotics* (pp. 753-763). Springer, Cham.

Gardecki, A., & Podpora, M. (2017, June). Experience from the operation of the Pepper humanoid robots. In *2017 Progress in Applied Electrical Engineering (PAEE)* (pp. 1-6). IEEE.

Ghiță, A. Ș., Gavril, A. F., Nan, M., Hoteit, B., Awada, I. A., Sorici, A., ... & Florea, A. M. (2020). The AMIRO Social Robotics Framework: Deployment and Evaluation on the Pepper Robot. *Sensors*, *20*(24), 7271.

Gomez-Donoso, F., Escalona, F., Rivas, F. M., Canas, J. M., & Cazorla, M. (2019). Enhancing the Ambient Assisted Living Capabilities with a Mobile Robot. Computational Intelligence and Neuroscience.

Google Developers. (2021). *Introduction to the Google Assistant Service*. Google. Retrieved March 2022, from https://developers.google.com/assistant/sdk/guides/service/python

Kim, E. S., Berkovits, L. D., Bernier, E. P., Leyzberg, D., Shic, F., Paul, R., & Scassellati, B. (2013). Social robots as embedded reinforcers of social behavior in children with autism. Journal of Autism and Developmental Disorders, 43(5), 1038+.

Lleonsí Carrillo, I. (2017). Development of a teaching assistance application for SoftBank Pepper.

Park, Y.-H., Chang, H. K., Lee, M. H., & Lee, S. H. (2019). Community-dwelling older adults' needs and acceptance regarding the use of robot technology to assist with daily living performance. BMC Geriatrics, 19(1).

Pripfl, J., Körtner, T., Batko-Klein, D., Hebesberger, D., Weninger, M., Gisinger, C., ... & Vincze, M. (2016, March). Results of a real world trial with a mobile social service robot for older adults. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)* (pp. 497-498). IEEE.

SoftBank Robotics. (2021). *Pepper the humanoid and programmable robot: Softbank Robotics*. SoftBank Robotics - Group. Retrieved March 2022, from https://www.softbankrobotics.com/emea/en/pepper

SoftBank Robotics. *Python SDK - SOFTBANK ROBOTICS DOCUMENTATION*. SoftBank Robotics - Group. Retrieved April 2022, from http://doc.aldebaran.com/2-5/dev/python/index.html

Appendix A – Project code

A1 – driver.sh

```
#!/bin/bash

# an implementation of Google Assistant on Pepper Robot

# SSH to robot
ssh nao@192.168.1.140 'bash -s' <<'ENDSSH'
  # record the audio request on Pepper
  echo "recording"
  arecord -d 5 -f S16_LE -c 1 -r 16000 -t wav query.wav &
  wait
  # close SSH connection
  exit
ENDSSH

# copy audio input to host system
scp nao@192.168.1.140:/home/nao/query.wav /home/gdumas/env/pga-test_v3/
wait

# process through GA
source /home/gdumas/env/bin/activate
googlesamples-assistant-pushtotalk --device-id pepper-f12d8 --device-model-id pepper-f12d8-pepper-puku14 -i query.wav -o syn.wav
wait

# copy GA response back to Pepper
scp /home/gdumas/env/pga-test_v3/syn.wav nao@192.168.1.140:/home/nao
wait

# play response through Pepper's speakers
python play_response.py
```

A2 – play_response.py

```
import sys

sys.path.append('/home/gdumas/pynaoqi/lib/python2.7/site-packages')

from naoqi import ALProxy
pepper = '192.168.1.140'
audio = ALProxy("ALAudioPlayer", pepper, 9559)
audio.playFile("/home/nao/syn.wav")
```

Appendix B – Attempted solutions

As alluded to in the Implementation section, the final project design was the result of an exploratory process that involved a series of trial-and-error attempts.

The first attempt at writing an app for Pepper used the SoftBank Android developer kit, named QiSDK. QiSDK is a robust SDK for writing Android apps specifically designed to be used with Pepper. Pairing this with Android Studio offered extensive control of Pepper's functionality, including several functions that would allow us to complete Pepper's functionality in this project with relative ease. The original design plan for the app involved saying a key phrase that Pepper would pick up, akin to how Google Nest uses "Hey Google" as a key phrase to indicate when a device should listen for audio input. QiSDK has a "PhraseSet" structure that allows a developer to define words and phrases that Pepper can look for when receiving and transcribing audio, and can be used to execute other functions in turn. Pepper could then be triggered to record audio using standard Android SDK functions, which would then be sent to Google Assistant or a different third-party server for processing. This approach would have enabled Pepper to read out Google Assistant's responses in its own voice rather than playing an audio file, since Google Assistant can be configured to send .json files that contain a string representation of a response instead of .wav audio files. However, this approach proved to be infeasible for the project. Despite QiSDK's final update being somewhat recent (2019), building code with Android Studio proved to be very difficult, potentially stemming from a plugin

installation bug. The other, more significant reason the Android Studio approach was infeasible was that it was impossible to enable developer mode on our specific Pepper robot. Enabling developer mode is required to push any homebrew apps to Pepper. On some robots with older Android OS versions, this is option is unavailable. SoftBank are the only ones who can update Pepper's OS, and we were unable to update our robot before the end of this project.

The second attempted design switched from a QiSDK approach to using NAOqi. Pepper would listen for a wake word or phrase, then stream recorded audio from its microphone to a webserver running on a locally networked workstation. This webserver would pass the audio to the Google Assistant API for processing, then stream the assistant response audio for Pepper to play. Audio streaming issues precluded the use of this approach during the project timeline, although the source of the error could not be identified in the time available. This design, however, directly led to the third and successful implementation which replaced the webserver with the scp secure-copy utility.