NAO - Librarian

Semester project report

Daniil Myronov

Martin Škopek

Abstract

The goal of the semester project was to develop a client-server application for the humanoid robot NAO, which will serve as an assistant when searching for the right box (shelf) for a given book from a dynamic book catalog.

Contents

Solution

The original idea, just after choosing the topic of the semester project, was an autonomous robot walking through the space that you could walk up to, hand him a book, and the robot would place it in the correct box. This turned out to be impractical due to the NAO robot's carrying capacity, so we were content with just classifying the book and locating the correct box.

The initial design of the classifier counted on a static index of known books and their classification based only on photographs, without text processing. Similar to, for example, https://cs229.stanford.edu/proj2015/127_report.pdf.
This option proved to be a dead end, especially due to the potential performance demands and fluctuating quality of the NAO robot's cameras.

In the end, we decided to use the text on the cover and use the pre-trained easyOCR classifier (https://github.com/JaidedAI/EasyOCR) to try to get the title of the book. At this stage, instead of a static index, we decided to use the services of the publicly available Google Books API (books.google.com), it proved to be a good source of ISBNs, on the other hand, it has a problem with providing relevant categories, so we decided to use the obtained ISBNs to query to the Open Library API (https://openlibrary.org/) and then find the right box.

Client server architecture

"Client-server is a network architecture that separates a client (often an application with a graphical user interface) and a server that communicate with each other over a computer network. Client-server applications include both a client and a server. An alternative to the client-server architecture is peer-to-peer, where computers communicate with each other without a server. "
(Client–server, https://cs.wikipedia.org/w/index.php?title=Client%E2%80%93server (last visited 23.01.2023))

Due to the computational complexity of text recognition, we decided to use the NAO robot only as an agent searching the environment, and the actual recognition of books and category labels on boxes is delegated to a server with which the robot communicates using REST API:

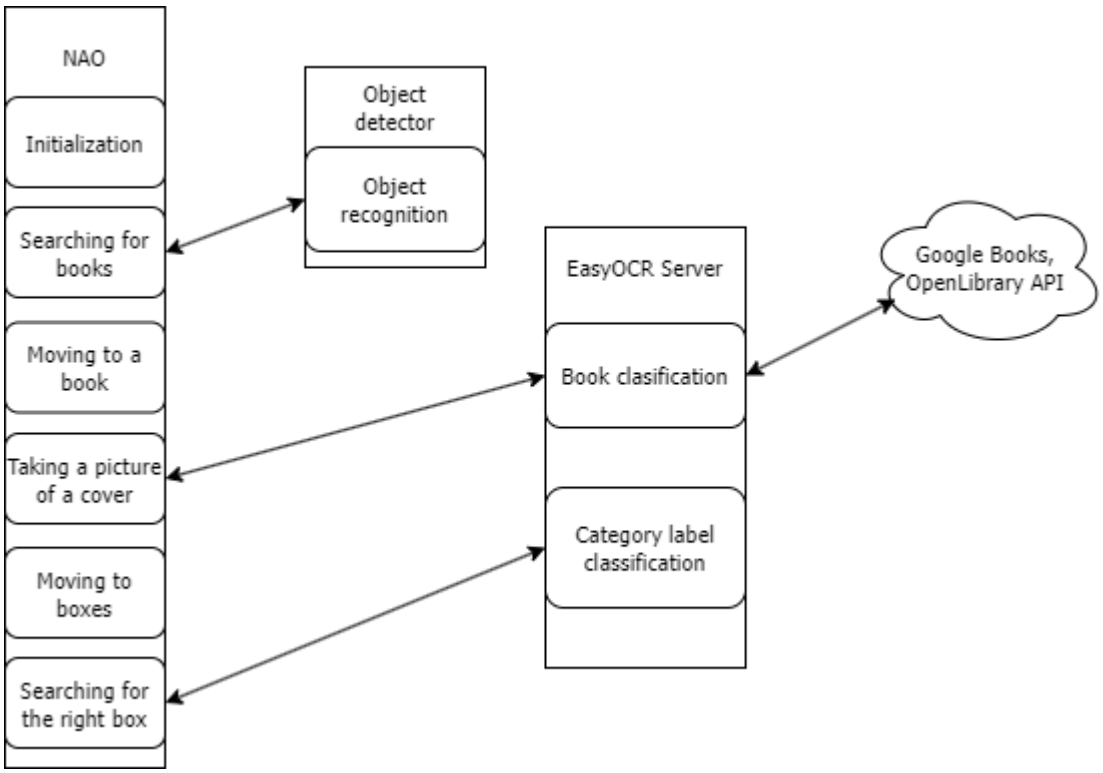

Figure 1: Project Schema (via app.diagrams.net)

Client

The purpose of the script running on the client is to let it move through the space and, if it encounters a book, to direct it to the correct box. This can be divided into several phases:

1. Initialization

File `run.py`

The user runs the program through the run.py file and enters the bot information (name or address) and server addresses in the parameters. First, the robot class itself is initialized, the server IPs are found, and we prepare the robot API.

2. Waiting for the head touch

File `nao_librarian.py`, *methods* `wait_for_starting_touch`, `on_starting_touch`

The ALTouch service is used to react to the touch on the head. You can find a similar algorithm in the documentation: http://doc.aldebaran.com/2-1/naoqi/sensors/altouch.html

3. Search

Method `look_for_book`

The robot will gradually look ahead 30 or 60 degrees to the left and then try again. If it still doesn't find the book, it moves.

The book in the camera image is searched using a remote server that uses the *ALIMDetection* service.

The robot moves through the space with `look_for_book` method, continuously takes pictures with the bottom camera and with the help of the method `find_book` continuously asks the recognition server if it has found the book.

4. Book found

Methods `get_distance`, `go_to_book`, `move_to_book`, `move_with_stops`

Using the move_with_stops method, we iteratively move closer to the book to get a better camera shot. If the robot is close enough to the book, it does not make any further stops. The distance to the robot is calculated using the method `get_distance` from the file `tools.py`. It takes the center of the book in the picture and uses triangulation to determine the distance.

5. Improving the camera angle

Method `change_posture`

When the robot approaches the book, by changing the position and angles of the joints, it gets into the right position to take a photo of the book. The services used are *ALRobotPosture*, *ALMotion*.

6. Taking a photo of the cover and performing the transformation

Methods `take_book_photo`, `get_warped_image` *from the file* `perspective_warp.py`

Method `get_warped_image` uses photo transformation algorithms and tries to cut out only the part with the book from the image. Canny edge detection algorithm is used for this purpose. If exactly 4 edges are found, their intersections are calculated and the perspective of the image is changed. The perspective change algorithm is taken from https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/ .

6. Sending the cover to the server

Methods `send_photo_to_server`, `decorate_sending_photo`

The bot sends the book cover image to the server using a POST request to the subaddress */cover*. Because this process can be long, while the bot waits for a response to the request, phrases like "Um...", "I think it's..." etc. are repeated asynchronously.

7. Server response processing

Methods `send_photo_to_server`, `on_book_info_not_found`, `say_book_info`

If the server's return code is not 200, the on_book_info_not_found method is invoked and the robot returns to its original position. Otherwise, the bot will get the book title, author, and genre information from the JSON response. This information will be played by the robot.

8. We head to the boxes

Method `go_to_box_area`

Location history of the robot is stored in the position_history variable and updated every time the robot moves. The first saved position is used to return to the original position.

9. Looking for the right box

Methods `find_box`, `get_text_from_image`, `move_to_next_box`, `box_not_found_decorations`

The robot takes a picture of the box and sends it to the sub-address */category*. It retrieves the genre from the JSON response. If the box genre appears in the book genres or the book genre is "Uncategorized", the next step is taken. Otherwise, the robot moves to the side by a fixed distance and repeats the previous algorithm. If the server does not find the text in the image, the robot will announce that.

10. Box found

Methods `box_found_decorations`, `go_to_position`, `wait_for_starting_touch`

The robot will say the information about the box and return to its original position. At the same time, it warns that if the user wants to run the algorithm again, he must touch the robot's head.

Server

The server was programmed in Python and built-in *http.server* was chosen to handle communication. The server is responsible for two tasks:

1. Text recognition on boxes

When a request is made to the url "/category", the image taken by the robot is uploaded to the server, where it is subsequently pre-processed. The selection of methods for pre-processing was based on the best processing of test data without a photo from the real environment (to be found in the server directory) and subsequently corrected according to the result when working with the robot in the laboratory. The articles helped when choosing processing methods[1][2][3].

2. Recognizing the text on the book cover

When requesting to the url "/cover", the image taken by the robot is uploaded to the server and we try to find out the name of the book. In this step, the preprocessing of the image turned out to be rather counterproductive, so we decided only to rotate the image and use the nonsense package (https://github.com/casics/nostril). The most likely text we send as a query to the Google Books API. If successful, we will try to find the book by ISBN in the Open Library and return the obtained information in JSON format to the client for processing.

[1]https://towardsdatascience.com/pre-processing-in-ocr-fc231c6035a7
[2]https://nextgeninvent.com/blogs/7-steps-of-image-pre-processing-to-improve-ocr-using-python-2/
[3]https://pyimagesearch.com/2021/11/22/improving-ocr-results-with-basic-image-processing/