Friedrich-Alexander-Universität
Erlangen-Nürnberg

WS 2025/26

Lab course: Human Robot Interaction

Task 6: Object Recognition

Chair of Autonomous Systems und Mechatronics
Friedrich-Alexander-Universität Erlangen-Nürnberg

Prof. Dr.-Ing. habil. Philipp Beckerle
Martin Rohrmüller, M. Sc.

Introduction

The NAO robot has two identical cameras located on the top and bottom of the robot's head. One camera provides a resolution of 640×480 at 30 frames per second and a maximum resolution of 2560×1920 at 1 frame per second, which can be used to identify objects in the robot's field of view. These enable the robot to visually perceive its surroundings. Especially for interaction purposes, it makes sense that the robot can utilize them to recognize objects, which could then be used for further applications. Choregraphe comes with a boxed object recognition function that can be used to learn and recognize objects. In contrast to the implementation in Choregraphe, the state of the art in object recognition are neural networks. In this task you should get to know the integrated object recognition function property on the one hand, but on the other hand also the possibilities through the neural networks utilizing the camera of the robot. Finally, the performance can be compared.

Task: Preparation

The objection recognition in Choregraphe does only work with the real robot as access to the cameras is needed, thus for the preperation we will focus on the implementation of object recognition with neural networks, starting with a short introduction to those methods. As in the other tasks, the programming language for that will be Python. You can either do the coding in Spyder, or in the standard editor and then run it from the terminal. ROS will not be applied in this task.

Object	Category YOLO	Category NAO
Soccerball	soccerball	soccerball
Volleyball	sports ball	volleyball
Chair 1	chair	office chair
Chair 2	chair	wood chair
Mug 1	mug	tea cup
Mug 2	mug	cat cup
Bottle 1	bottle	olive bottle
Bottle 2	bottle	syrup bottle
Scissors 1	scissors	scissors 1
Scissors 2	scissors	scissors 2
Lamp	lamp	lamp

Table 1: Objects used for the object recognition task.

Object recognition with neural networks

An artificial neural network (NN) is a computational processing systems which on a conceptual level are modeled after the functioning of biological nervous systems such as the human brain. NNs mainly consist of a large number of interconnected computational nodes (neurons) that learn from inputs. The principle of an NN is based on each neuron receiving an input and performing an operation (e.g., a scalar product followed by a nonlinear function) with the goal of further optimizing the final output. The input, usually in the form of a multidimensional vector, is loaded into the input layer and from there is passed to the hidden layers. The hidden layers then compute results from the previous layers and weigh how a stochastic change affects or improves the final result, which is called the process of learning[1]. From the raw image vectors to the final class score output, the entire network is expressed as a perceptual weighting function. The final layer contains loss functions associated with the classes that are output. Convolutional Neural Networks (CNNs) are special NNs mainly used in the field of pattern recognition in images. The network is particularly suitable for image-oriented tasks, since this architecture reduces the required parameters and allows to encode image-specific features. Thus, CNNs are mainly used in object recognition[2]. In

[1] O'Shea, K. and R. Nash. An Introduction to Convolutional Neural Networks. 2. Auflage, 26. November 2015 `https://arxiv.org/pdf/1511.08458.pdf`

[2] S. Albawi, T.A. Mohammed and S. Al-Zawi. Understanding of a convolutional neural network. In: 2017 International Conference on Engineering and Technology `https://ieeexplore.ieee.org/document/8308186`

this task, the target detection software from YOLO (You Only Look Once: Unified Object Detection in Real Time), YOLOv5 is used.

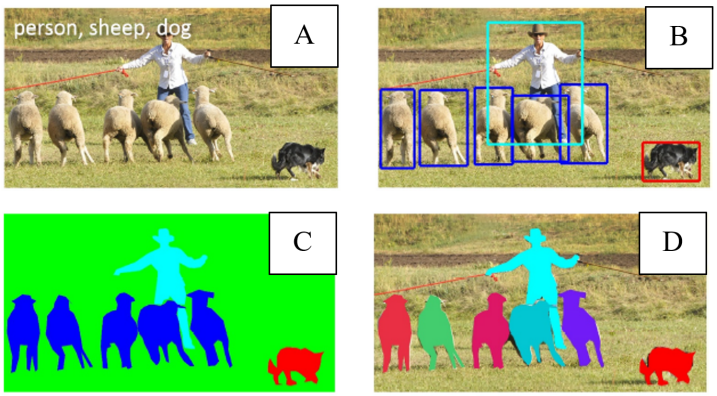

Figure 1: Representation of different outputs during object recognition: classification (A), localization (B) and segmentation (C and D) of an image.

Figure 1 shows different outputs that can be created using CNNs. The YOLO used here can generate an output as shown in (B). It creates bounding boxes around recognized objects and with that gives probabilities of the accuracy of each object recognized in the picture. A full introduction into the topic would be out of the scope of this lab task. If you like to read more, check out the linked resources and the online documentation[3].

Steps in training a CNN

Before a NN can be trained, data needs to be gathered and prepared. In the case of CNNs big amounts of data are used to allow for accurate detection, classification, and recognition. Well known dataset for object recognition are the ImageNet[4] and the COCO[5] data sets. The COCO data set consists of 330.000 images with 80 object categories. The ImageNet dataset has 1.500.000 images. As most times one does not have that many images when training a network, different methods to decrease the needed amount of images can be utilized to successfully train a NN. Data augmentation techniques and transfer learning (by utilizing freezing of layers of the NN) will be used for that.

Data augmentation

The amount of training data can be increased with generating new data from existing data. When processing images, this can mean, for example, scaling, rotating or changing the color of existing images. However, the object should remain recognizable in exactly the same way. Check out *AI Multiply*[6] for different techniques and more information. In training with YOLO, data augmentation is done automatically.

[3] `https://docs.ultralytics.com/`
[4] `https://image-net.org/about.php`
[5] `https://cocodataset.org/#home`
[6] `https://research.aimultiple.com/data-augmentation/`

Transfer learning and freezing of layers

When there is not enough data to train a whole network, transfer learning can be used, i.e., retraining a pretrained network with the new (similar) data[7].

An option to utilize the pretrained network and decrease training time is freezing layers of the network. When freezing layers, only the layers closer to the output layer will be trained, i.e., the layers that make the decision on which label to assign to the image. This technique can be used as the layers close to the input layer are more general layers that will filter the image on a broader level[8].

Neural Network

For the use of neural networks, Pytorch[9] is used in the following. This software is already installed on the Ubuntu sticks as well as on the computers in the lab. The exercises in this section are supposed to teach you the first steps to use the NN. This includes preparing the training data at the beginning. As already explained, these are then used in a next step to improve the pre-trained network. However, since the training makes high demands on the hardware, full training will not be part of the experiment and trained models are provided.

Exercise 1.
Read through the additional linked resources and answer the following questions in 2-3 sentences:

a) What is precision, recall and accuracy?

b) What is the point of freezing layers?

c) When would you use transfer learning?

d) What are methods for data augmentation?

Add the answers to the submission file.

In the following tasks you will use the neural network YOLOv5. After using Pytorch to send an image through the network for recognition, the image can be stored together with a bounding box and the probability for the respective object. Figure 2 shows the basic code structure that can be used for the following exercises. In the code, at first the network model is loaded. In this example, the model is loaded from the Internet. Alternatively, it can be loaded locally refering to the *.pt* files that are in the YOLO directory. Next, an image path is defined. Here, it is a an URL but it can also be a path to a local image. In the next line, the image is passed through the network and evaluated. The next lines output the results as follows:

- `results.print()`: Prints the classes of the recognized objects.

[7] `https://builtin.com/data-science/transfer-learning`
[8] `https://docs.ultralytics.com/tutorials/transfer-learning-froze-layers/`
[9] `https://pytorch.org/`

- `results.show()`: Shows the image with the bounding boxes created and the probability of the recognized objects.
- `results.save()`: Saves the augmented images to '.../runs/detect/'

```
import torch
import glob

# Model
# Load the pretrained model locally
model = torch.hub.load('/home/hri-lab/yolov5-master', 'custom',
    path='/home/hri-lab/yolov5-master/yolov5s.pt', source='local')
# or load the pretrained model online
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

# Images
# Path to image
img = '/home/hri-lab/zidane.jpg'
# or from webadress
img = 'https://ultralytics.com/images/zidane.jpg'

# Inference
# Evaluating image with the model
results = model(img)

# Results
results.print() # or .show(), .save(), .crop(), .pandas(), etc.
results.show()
results.save()
```

Figure 2: Execution of YOLOv5.

Exercise 2.
To begin, test the simple example shown in the Figure 2. Load the model which is provided in the YOLO directory *yolov5-master*. Alternatively, it could be downloaded. The photo to be evaluated is available in the */home* directory and can be loaded with defining the path to it. Figure out, where the results are saved.

This model (*yolov5s.pt*) comes with the YOLO package as standard and is the version pre-trained with the COCO dataset. Additionally, Pytorch allows the evaluation of multiple images at once. The following code example can be used to evaluate an entire directory:

```
...
# Images
folder_dir = 'path/to/imgs/' # path to the test images
images = []

# get the path of all individual images in folder
for img_path in glob.glob(folder_dir + '/*.png'):
   images.append(img_path)

# Inference
results = model(images)
...
```

Figure 3: Loading image directory

This creates the paths to all images in a list. This list is then passed to the model. The result is the same as before, but for several images.

Exercise 3.
Test the YOLO network which was pre-trained with the COCO dataset and re-trained using different training times (epochs) using multiple test images from our own dataset. The images are provided online in the folder *test_images*.

a) Evaluate the images with the standard model (*yolov5s.pt*).

b) Additionally test the images with each of the models trained with 50, 100 and 500 epochs, load them with the line of code

```
model = torch.hub.load('/home/hri-lab/yolov5-master', 'custom',
    path='/home/hri-lab/yolov5-master/yolov5s_epochs50.pt',
    source='local')
```

The retrained models are in the YOLO directory (as *.pt* files) as well. This line here refers to the model retrained with 50 epochs.

Have a look at the results of each run. Answer the following questions qualitatively.

c) Which objects are well recognized?

d) How do the results differ between the networks?

e) If one is performing better, why?

f) What is the difference between the pre-trained and the further trained models in terms of output classes?

Add the answers to the submission file.

You have now learned how to evaluate images using a provided model. In the following, we will now also discuss the preparation of the models, in particular the data required for this.

Creating image labels

In order to train the network, images of the objects are needed, each with the respective label of the contained objects and their position in the image. For object recognition, there is already our own dataset that was used to improve the pre-trained network and thus to create the models of the previous exercise. However, to understand the composition, you are now to prepare another part of the data set itself with the tool LabelImg.

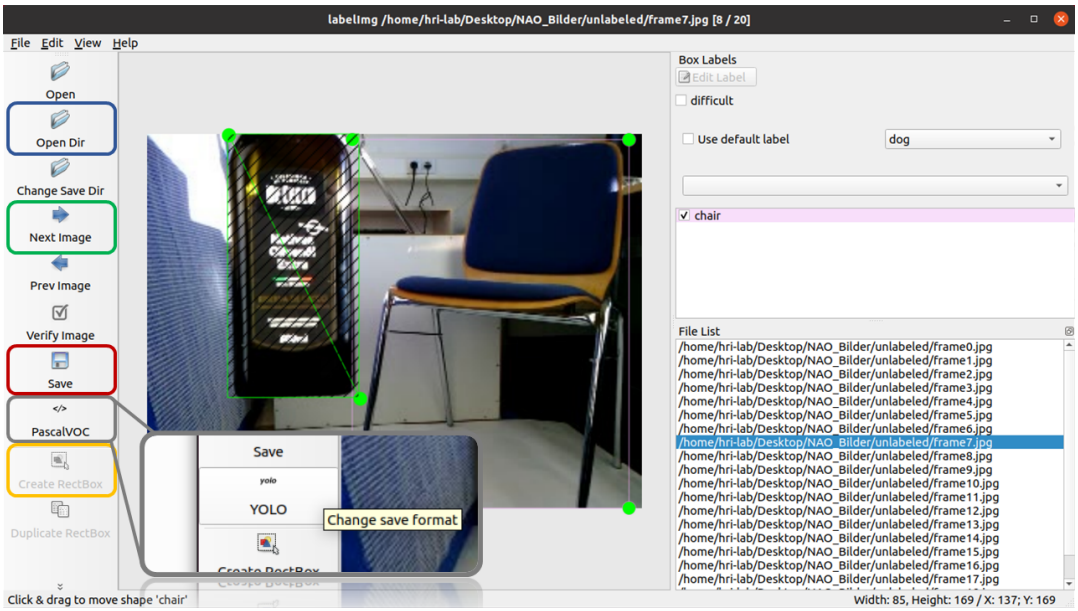

Figure 4: User interface of LabelImg.

Exercise 4.
Label additional data for the training data set.

a) Start the tool `labelImg` with which the images can be labelled. Open a the terminal (Ctrl+alt+t), navigate into the folder

```
$ cd ~/labelImg-master
```

and run

```
$ python3 labelImg.py
```

to open the tool.

b) Download the directory *unlabeled* which contains 20 images and open it with the button marked blue in Figure 4.

c) Switch to the mode that creates labels fitting for the YOLO network using the gray marked button.

d) With the button marked in yellow create rectangles around the objects of the classes from Table 1 you can find in the first image. The classes are already predefined in labelImg (in the folder '~/labelImg-master/data'). Safe the rectangle with the name of the object category. You can create multiple rectangles in one image. Safe the annotations with the button marked in red.

e) Continue with the rest of the ten images by using the button marked in green. At the end, verify that for each image in the folder, an additional *.txt* file has been created that contains the information about the labels and the number of the class.

So you create a *.txt* file for each image. This combination of images and labels must then be arranged for training in the structure shown in Figure 5, where the image files are in the images folder and the *.txt* files with the corresponding names must be in the labels folder.

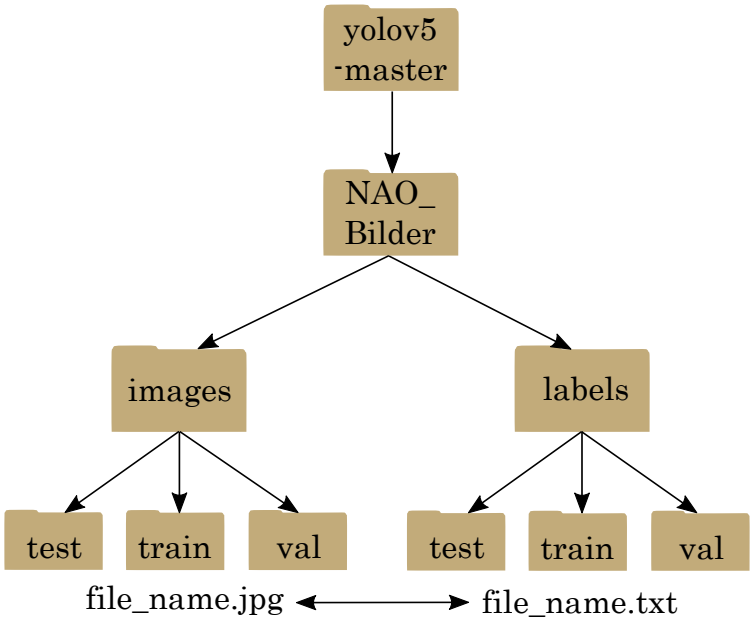

Figure 5: Structure of the training data for the YOLO network.

The rest of the data set is already available together with the label files. In the following it is described how the training can be carried out with it. For further information on the network, you can check out the website[10].

Training of the YOLO network

Training adjusts the parameters in the network, also called weights, to represent the training data. A YAML file defines the paths to the training data and the classes of the objects in the training images for Pytorch. It can be found in the YOLOv5 directory.

[10] `https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data`

Exercise 5.
Training the neural network with a dataset.

a) Copy and paste the newly labeled images and *.txt* files into the respective *train* folders in `/home/hri-lab/yolov5-master/NAO_Bilder`. This completes the data set, which is divided into training, testing and validation data.

b) Re-train the network. Use a small batch size of 4 and a number of 3 epochs for this, just to get to know the procedure here. The training can be started in *yolov5-master* from terminal with the line

```
$ python3 train.py --img 640 --batch 4 --epochs 3 --data
    NAO_Bilder.yaml --weights yolov5s.pt
```

where at first the image size is defined, as well as the batch size and the number of epochs. The previously mentioned YAML file here refers to our dataset. With the weights argument the pretrained YOLO model is used to build up on.

After the training, the results are automatically safed to '~/yolov5-master/runs/train/exp'.

c) Check out the results.

Among the results there are several graphs and figures. Was the training successful? Answer in the submission file.

So this is the procedure to improve a pre-trained neural network with an own data set and to adapt it to corresponding objects. The newly created weights, for the best and for the last training step, are saved in the folder with the results at '~/yolov5-master/runs/train/exp/weights'.

Now you should still figure out what happens, when the network is now tested on some of our own images.

Exercise 6.
Create a new Python script according to Exercise 3, with which you can test the images from the folder *test_images*. Copy the new weight ('best.pt') to the folder 'yolov5-master' next to the previously used weights of YOLO. In the test script, load the new model weights with

```
model = torch.hub.load('/home/hri-lab/yolov5-master', 'best',
    source='local')
```

and do not assign a different name for the file, otherwise *hubconf.py* would have to be adapted. Run the test and save the results.

The training from the Exercise 5 was only very short. This does not yet allow you to achieve optimal results, as you may have noticed, especially since you have already used longer trained networks in Exercise 3. If you want to, you are welcome to do a longer training by yourself.

Exercise 7.
(Voluntary) Train the model with our dataset now longer, with a higher number of epochs and a bigger batch size. Test the results again with the test images.

Task: Execution

In the lab part, the objective is to first implement the object recognition that is integrated in the real NAO. Afterwards, the object recognition introduced in the preparation is tested via the neural network. Finally, the variants are compared.

Object recognition on the NAO robot

Exercise 8.
Prepare the robot for the object recognition.

a) Open Choregraphe on the lab computer.

b) Connect the computer to the "ASM-Roboter" wifi network.

c) Start the NAO robot with pressing and holding the button on the robot's chest. It will automatically connect to the network, too.

d) In Choregraphe, connect to the real robot with the *Connection* menu (name: Noah or Robin). If no robot is listed, press the button on the robot's chest once and enter the IP adress manually.

e) Open the *Video monitor* in the *View* menu of the menu bar and make sure that the *Robot view* window is also open.

f) Turn off autonomous life and click *Wake up* to stiffen the joints of the robot.

You should now see the image captured by the head camera. With this setup, the object recognition of NAO can now be used. There are several control buttons for the video monitor functions as shown in Figure 6.

Figure 6: Video monitor of Choregraphe to access the robot cameras.

To learn objects or images, they are held in front of NAO's camera and the learn button (marked blue in Figure 6) in the *Video Monitor* is pressed. This starts a countdown, at the end of which a snapshot of the image remains. Now a boundary can be drawn around the object by clicking in the photo window. If the boundary lines meet during drawing, a window opens in which a name and a side are assigned to each image. This is shown in Figure 7. With *ok*, the taken picture is stored to the database. A database can then be loaded onto the robot (yellow button in Figure 6).

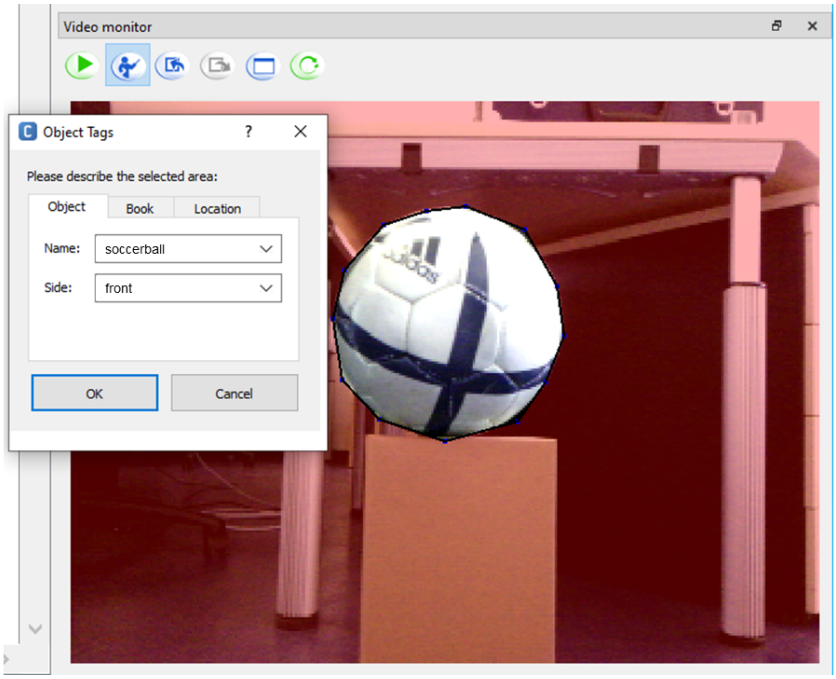

Figure 7: Marking object and assigning a label to it.

Exercise 9.
Create a new vision recognition database (button marked gray in Figure 6). Take one picture of the provided soccer ball by following the instructions above. Label it with the name "soccerball" and the side "front". Load the database to the robot.

For learning another object, respectively taking another picture, click the play button first and then the learn button. With that, a database of more objects with multiple pictures of each can be built. For that, no program is allowed to run and it is easiest to store and take the objects while NAO is in standby mode (when its autonomous life is switched off). The database can now either be exported (marked red in Figure 6) and imported later (marked green in Figure 6), or loaded onto the robot.

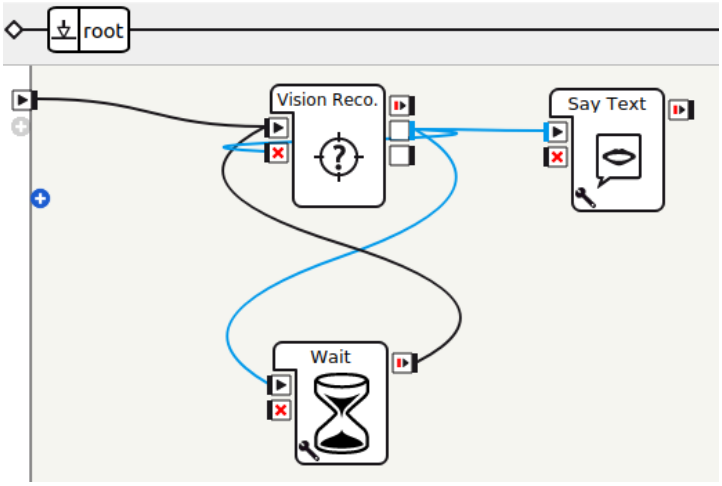

Figure 8: Vision recognition in Choregraphe.

The detection of the objects can be done with the "Vision Reco." box. NAO starts object detection as soon as the "Vision Reco" box receives a signal via the *onStart* input. As soon as NAO recognizes something known it passes on the name as a string via the output *onPictureLabel*. This textual output can then be processed by a "Say Text" or "Dialog" box as shown in Figure 8. If the robot has been taught several objects and is supposed to do more than just the recognized image, object, etc., a "Switch Case" box could be combined with the "Vision Reco." box to perform different actions when different objects are recognized. NAO's object recognition is not very strong and NAO has difficulty recognizing objects in the real world. The focus in this task is on the recognition part, which is why a simple voice output of the recognized object is sufficient here. The box in Choregraphe does not stop by itself, but must be stopped manually.

Exercise 10.
Recognise the object again.

a) Create a "Vision Reco." box from the box libraries. You can find it in Sensing > Vision > Surroundings in the library browser. Connect "onPictureLabel" to "onStop".

b) Create a "Say Text" box and connect it to the "onPictureLabel" output of the "Vision Reco." box.

c) Add a "Wait" box, connect its input to the "onPictureLabel" output of the "Vision Reco." box and its output to "onStart" as shown in Figure 8. This procedure will prevent the string output to the text box from overloading.

d) Run the program and see, if the object can be detected. If not, do Exercise 9 again, record more pictures and hold the object closer to the robot.

In contrary to a data set for training a neural network it is not very purposeful to create a data base for the NAO object recognition. This is due to the fact that the recognition can be very different for objects learned with different backgrounds and illuminations. Therefore, with the following task you should build your own database with a few of the objects from Table 1 and test it later. The description so far already mentioned how multiple objects can be integrated.

Exercise 11.
Learn the provided objects (soccerball, mug, bottle and scissors), create the object boundaries and store them in a database. Only assign names to the objects, but leave the text box for the side empty. When creating the database take at least five pictures of each object. Take them from different orientations and sides. Export the database (.vrd file) to your group's directory and upload it onto the NAO.

Now, the Vision Recognition of NAO can be applied and tested. Therefore you can reuse the Choregraphe program created in Exercise 10.

Exercise 12.
Perform experiments to recognize the objects. Try to recognize every object five

times. For each test, place the respective object in the camera's field of view and hold it there until the object is either detected or not detected for ten seconds. Draw a table with the number of times each object was correctly detected, incorrectly detected, or not detected at all.

Object recognition with the neural network

You have now learned how to recognize objects on the real NAO robot. In the following we will use the YOLO network for object recognition. The YOLO network has already been introduced in the preparatory tasks. The code elements are provided for the following exercises. First, the connection to the NAO robot must be established. For this the NAOqi framework is used, with which the robot can be programmed using Python. Then, you will learn how the cameras can be addressed and how images of objects can be captured. Finally, the images are evaluated with the YOLO network. Note that the NAOqi framework runs Python 2. This previous version of the programming language has some small differences in the syntax. In this exercise, however, no additional code needs to be programmed with it. On the computer in the lab there are two virtual environments, one running Python 2 (name: NAO) and one running Python 3 (name: YOLO). A virtual environment is a directory with specifically installed packages that can be activated and deactivated. Changes or installations in one environment do not affect another one. This is how different versions of packages or Python can be tested on one computer parallelly.

Connecting to the NAO's camera

In the first part, the virtual environment with Python 2 is used. Put it into operation with the following exercise.

Exercise 13.
Activate the virtual environment for using Python 2.

a) Open a new terminal and activate the virtual environment for using the Python 2 scripts for the NAO connection with

```
$ conda activate NAO
```

in which all necessary Python libraries are already installed.

b) Navigate to the folder */Desktop/NAO-files/<group_name>*, or create it if it does not already exist. Leave the terminal open for the next task. Use this folder to save the scripts.

An environment can be closed again with the command `deactivate`. The next step is to establish the connection to the NAO robot via the NAOqi framework. The code from Figure 9 can be used for this purpose.

```python
from ipaddress import ip_address
import qi
import argparse
import sys

# Exercise 15
#import time
#from PIL import Image

if __name__ == "__main__":

    ip_address_NAO = ...

    parser = argparse.ArgumentParser()
    parser.add_argument("--ip", type=str, default=ip_address_NAO,
                help="Robot IP address. On robot or Local Naoqi: use
                    '127.0.0.1'.")
    parser.add_argument("--port", type=int, default=9559,
                help="Naoqi port number")

    args = parser.parse_args()
    session = qi.Session()
    try:
        session.connect("tcp://" + args.ip + ":" + str(args.port))
    except RuntimeError:
        print ("Can't connect to Naoqi at ip \"" + args.ip + "\" on
            port " + str(args.port) +".\n"
              "Please check your script arguments. Run with -h option
                  for help.")
        sys.exit(1)

    # Function call for Exercise 15
    # main(session)
```

Figure 9: Python code for connecting to NAO robot.

This requires the IP address of the robot connected to the network. A session object is created with NAOqi that can be used to access the robot via the `connect` method. If the connection is not successful, an error message is issued.

Exercise 14.
Test the connection to the NAO robot.

a) Create a Python file *connect_NAO.py* with the code above.

b) Insert the IP address of the robot. You can get it by pressing the button on NAO's chest.

c) Run the file from the terminal with

```
$ python connect_NAO.py
```

If the script runs without an unsuccessful connection message, the connection has been established correctly.

Taking pictures of objects

The code in Figure 10 is used to access the camera and save images to the computer. This runs via a service[11] of the NAOqi framework, which is initialized and subscribes to the camera. The `getImageRemote()` function can then be used to capture the current image from the camera. In the code, a loop is applied to take multiple images with five seconds delay. The images are processed and saved as *.png* files to the computer.

[11] `http://doc.aldebaran.com/2-5/naoqi/vision/alvideodevice.html`

```
def main(session):
    """
    First get an image, then save it to a directory
    """

    # Get the service ALVideoDevice.
    video_service = session.service("ALVideoDevice")
    resolution = 2  # VGA
    colorSpace = 11 # RGB

    videoClient = video_service.subscribeCamera("python_client", 0,
        resolution, colorSpace, 5)

    #save a total of 20 images
    try:
        for i in range(0,19):
            t0 = time.time()

            # Get a camera image.
            # image[6] contains the image data passed as an array of
                ASCII chars.
            naoImage = video_service.getImageRemote(videoClient)

            t1 = time.time()

            # Time the image transfer.
            print "acquisition delay ", t1 - t0


            # Now we work with the image returned and save it as a PNG
                using ImageDraw package.

            # Get the image size and pixel array.
            imageWidth = naoImage[0]
            imageHeight = naoImage[1]
            array = naoImage[6]
            image_string = str(bytearray(array))

            # Create a PIL Image from our pixel array.
            im = Image.frombytes("RGB", (imageWidth, imageHeight),
                image_string)

            # Save the image.
            im.save("imgs/camImage"+str(i)+".png", "PNG")

            time.sleep(5) # wait 5 seconds before saving the next image

    except KeyboardInterrupt:
        video_service.unsubscribe(videoClient)
```

Figure 10: Save images from NAO camera.

Exercise 15.
Take pictures with the NAO. Therefore, add the function from Figure 10 to *connect_NAO.py*. Add a folder *imgs* to your group's directory. Run the script again. When this is done, check out the pictures saved to the computer.

After successful execution, the practical task can now be performed. For this purpose, pictures of the provided objects are to be taken.

Exercise 16.
Take pictures of the provided objects. At the end you should have five pictures of each provided item in the *imgs* folder. To do this, you can extend the duration of the loop in the code. Alternatively, you can add a line

```
raw_input("Press ENTER to take next picture")
```

in order to wait for a keypress after taking a picture to interrupt the loop.

Evaluation of the images with YOLO

In the next step, these locally stored images are evaluated with the neural network. For this purpose, the fully trained variant is used. First, however, the virtual environment must be changed to use Python 3, which is necessary for Pytorch. This can be done using

```
$ conda activate YOLO
```

which is even possible in parallel in another terminal. In this environment Pytorch and all other necessary libraries are installed as it is on the Ubuntu stick for the preparation to this task.

Exercise 17.
Evaluate the robot pictures with YOLO.

a) Change the virtual environment to the YOLO environment.

b) Create a file *evaluate_NAO.py* and implement code to evaluate the directory with YOLO or reuse code from the preparation.

c) Insert the path to the images taken by NAO, use the model *yolov5s_epochs500.pt* and run the code.

d) Save and store the images with the results.

After execution, the results are stored in the runs folder in the path of the code.

Exercise 18.
Create another table or extend the one from Exercise 12. Add the number of successful, wrong and not recognized images of every object.

```
import torch
import os
import time

if __name__ == '__main__':

   # Model
   model = torch.hub.load('/path/to/yolov5s-master', 'custom',
       path='/path/to/yolov5s_epochs500.pt', source='local') #custom
       trained model

   print(os.getcwd())
   list_subfolders_with_paths = [f.path for f in
       os.scandir(os.getcwd()) if f.is_dir()]
   print(list_subfolders_with_paths)
   # Images
   folder_dir = '/path/to/imgs/' # batch of images
   nameSet=set()

   stop_counter = 0
   while(stop_counter < 50):
      stop_counter = stop_counter +1

      for img in os.listdir(folder_dir):
         # Inference
         fullpath=os.path.join(folder_dir, img)
         #run inference only on new images
         if os.path.isfile(fullpath) and not(img in nameSet):
            try:
               nameSet.add(img)
               results = model(fullpath)

               # Results
               #results.print()
               #results.save()
               results.show()
            except:
               pass

            stop_counter = 0

      time.sleep(0.5)
```

Figure 11: Online recognition using YOLO network

Until now, you have captured images and evaluated the entire directory. Through the connection to the NAO there is also the possibility to evaluate the images continuously. For this you will have to run both virtual environments simultaneously. With the following exercise, the code in Figure 11 can be used in the YOLO environment. It constantly

looks for new images in the *imgs* directory and evaluates them. For taking new pictures, you can reuse code as for Exercise 15 where you can also reduce the sleep time for the loop and increase the loop duration. However, the limiting factor will be the acquisition delay caused by the Wifi connection.

Exercise 19.
(Voluntary) Run the script to take pictures and the script evaluating them with the neural network at the same time. For this, you need two terminals where you activate the necessary virtual environments respectively. Run the script for connecting to the NAO first and for evaluation after that.

Comparison of the object recognition methods

You have now learned how to recognize objects with the functions integrated in the NAO on the one hand and object recognition with a neural network on the other. Surely you have already been able to experience the differences in quality. It therefore remains to conclude the task by comparing these results quantitatively.

Exercise 20.
Compare the YOLO method with the NAO method of object recognition with the previously created table. Research which object detection algorithm is running on the NAO[12].

[12] `http://doc.aldebaran.com/2-5/naoqi/vision/alvisionrecognition.html`