TUTORIAL 1: a first HRI prototype

Author: Rekha Raja | Assistant Professor | Department of Artificial Intelligence | IITH

Goal: The goal of this tutorial is to get familiar with the environment and program a first example of HRI with the humanoid robot Nao.

1. Webots environment and Robot Nao

Download and install webots from https://www.cyberbotics.com/
Download and extract the template code to your PC Run the system and test it.

2. NAO World and Python controller

- Open the NAO world to your Webots environment. Goto File -> Open World -> C:\Users\U679233\AppData\Local\Programs\Webots\projects\robots\softbank\nao\worlds\nao_demo.wbt
- Open the python controller to your Webots environment. Click on the ''open folder'' symbol on the controller panel in Webots and goto C:\Users\U679233\AppData\Local\Programs\Webots\projects\robots\softbank\nao\controllers\nao_demo_python folder to open **nao_demo_python.py** file.
- Click on reset simulation button. You will see that the NAO python controller is running, where the controller provide the functionalities of generating predefined movements and printing information in the console. An example of the *gps* sensor (location) is given. The main function will be called run_keyboard. For example press UP ARROW button to move robot forward, press DOWN ARROW button to move robot backward etc. Try play with different keys to check the output.

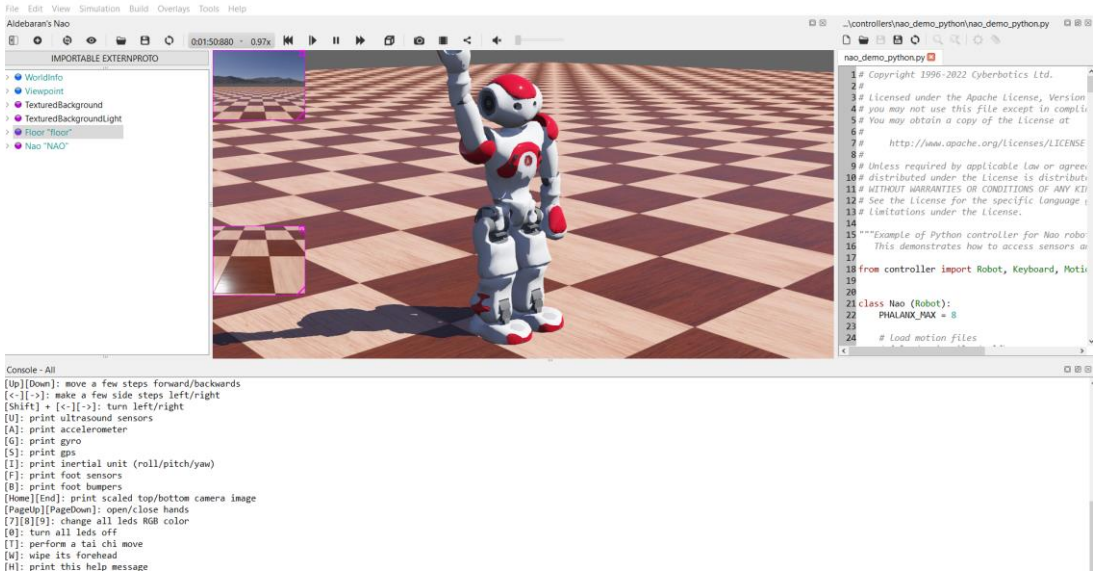


Figure 1: Webots NAO start up scenario

3. Add object (Ball) to the environment

- Add a red/green ball (with radius 0.05) using the webots interface (click on the add button in the left panel, which is used to add a new object/import an object to the environment). Click on the PROTO nodes (Webots Projects) -> objetcs -> balls -> ball (solid)

- Change the color of the ball by clicking/expanding ball on left panel -> color -> red = 1, green = 0, blue = 0, to make the ball color red.
- Change the radius of the ball by clicking on the radius and change the value to 0.05 and enter. Environment should look like Figure 2.

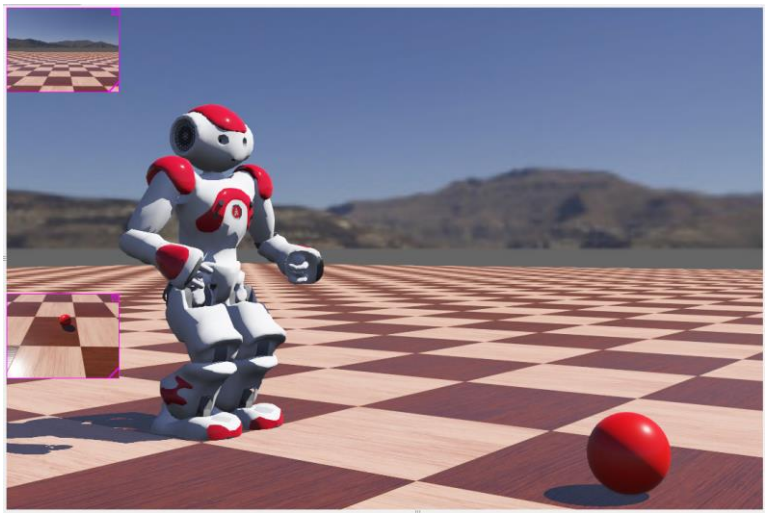


Figure 2: Football ball following scenario.

- Save the world (nao_demo.wbt) into your computer. For example, C:\Users\U679233\Documents\HRI\2023\Tutorial1\worlds
- Save the controller (**nao_demo_python.py** file) into your computer folder . For example, C:\Users\U679233\Documents\HRI\2023\Tutorial1\ controllers\nao_demo_python

- Open the world file on the right panel and add the controller file to it. It should look something like this.

  Nao {
  translation -0.0078 -0.18860153154271797 0.3056917886921413
  rotation -3. 05 -0.00072453632891 0.9999997369309203 1.5883421299327618
  controller "nao_demo_python"
  }

4. Open webcam and view Face to the environment

A function to capture the camera from the webcam and show it in the webots display interface is provided.

```
# Captures the external camera frames
   # Returns the image downsampled by 2
   def camera_read_external(self):
      img = []
      if self.ext_camera:
         print('> Starting camera')
while(True):
            # Capture frame-by-frame
            ret, frame = self.cameraExt.read()
             # Our operations on the frame come here
            img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) # From openCV BGR to RGB
```

```
    img = cv2.resize(img, None, fx=0.5, fy=0.5) # image downsampled by 2
    #display image
    cv2.imshow('image frame from webcam',frame)

    #print('press q to quit window')
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
self.cameraExt.release()
cv2.destroyAllWindows()
return img
```

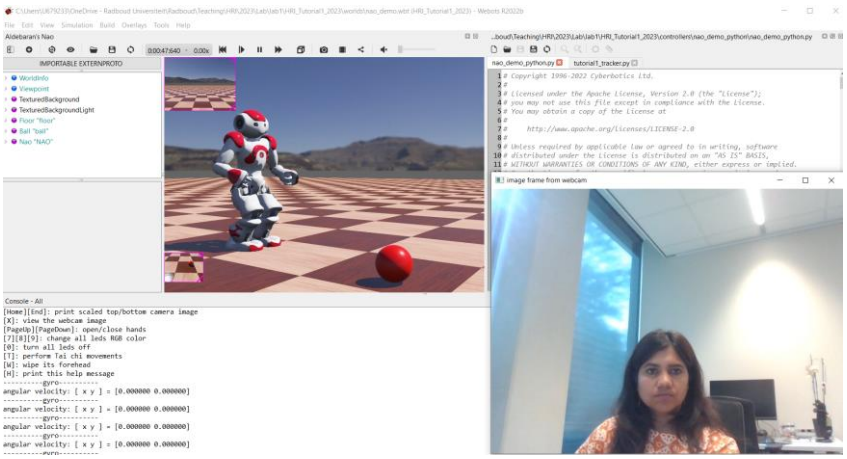


Figure 3: Viewing face through Webcam in Webots

Task-1 : Ball Following : Develop a head controller that follows the ball (with a new *run_ball_follower* main function). **Use the bottom camera of the robot**. Here you can design a closed-loop controller to maintain the head centered in the ball. Run the program and move the ball by hand to test the head movement. *You need to check the head limits. In the real world sending out of range joint angles could be fatal for the robot.*

Task-2 : Face Following :

Develop a controller (with a new *run_face_follower* main function) that moves the head towards a detected face in the camera. As the robot cameras are not capturing the image, we cannot design a closed-loop controller. Thus, we will use the centre of the face as the reference to move the head. Use the two head angles: *Yaw* and *Pitch* to move left-right and up-down when the face moves in the image. For that purpose code a function with name *look_at* with input the 2D (pixel) location of the face centre.

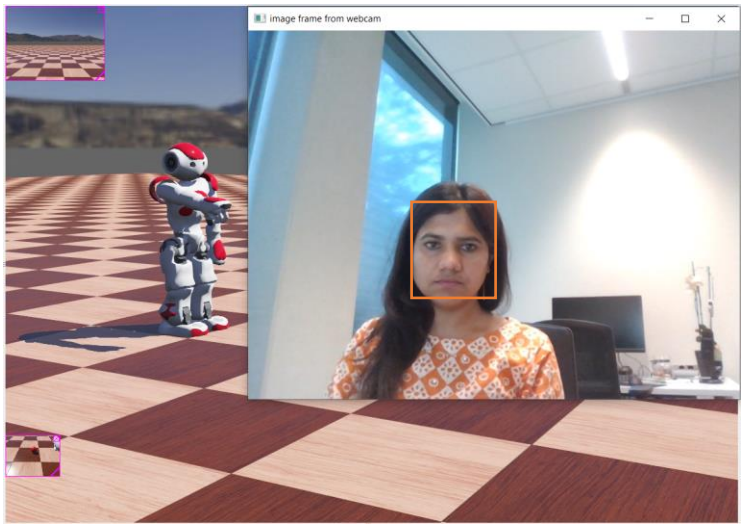


Figure 4: Face following scenario and the webcam face detection.

To detect the face use the opencv built-in function *detectMultiScale*. This algorithm uses the Haar descriptors that are stored in the file provided: *haarcascade_frontalface_default.xml*. These descriptors should be loaded in the initialization function of the controller (*__init__*)

Task-3 : High-level behavior to improve communication

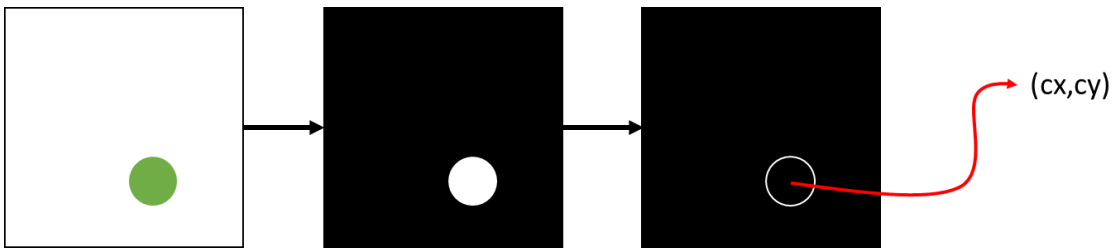
Develop a set of behaviors to make the robot react when a face and the ball appears in the image. Write it in the new *run_hri* main function. *Tip: you can use predefined movements.*

Task-4. Report, documentation and submission (Optional)

Write a report of maximum two pages explaining the designing decisions for the controller and the high-level behavior. Submit the pdf and the code in a compressed file with the full names of the team members. Only one person of the team has to submit it in Brightspace.

Instructions for the ball following exercise

Instead of the football ball, we are going to use a red/green ball to simplify the segmentation process. Below I explain the classical pipeline for object detection (with opencv) using the colour as the feature.

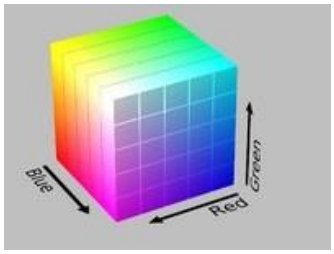


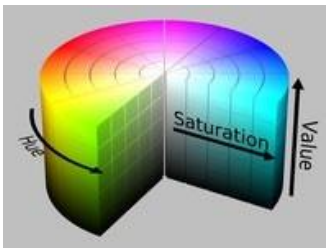
1. Colour space transformation

The first thing that you have to know is that there are multiple colour spaces and each of them is useful for a different application. The most known is the Red-Green-Blue (RGB). Usually, each of the three channels is composed as an array of bytes (8 bits). This makes that every pixel is coded with 3 values from 0 to 255 or normalized from 0 to 1. For instance:

Red: (255, 0, 0)
Green: (0, 255,0) Blue:
(0,0,255)

Tip: opencv works with the format BGR (the blue and the red channels is switched).





The RGB format is not robust to lighting changes as the three values will change drastically when we remove the light. Other Colour spaces, such as the Hue-SaturationValue (HSV) –See figure on the left– are more robust to lighting changes. Channel H encodes the pure colour. For instance 0
means red. You can play here with different colour spaces: http://colorizer.org/

Tip: usually the hue (H) is a value between 0 and 360 (like degrees), but in opencv the decided to encode it in a byte. Thus the value is divided by 2 giving a range from 0 to 180.

Therefore, a normal step in the visual segmentation pipeline is to transform the image into the colour space needed. For the face, we transform it to GRAY scale and for the ball tracking to HSV.

There are specific functions in opencv for colour transformation:
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html

2. Image thresholding

The most basic image segmentation is thresholding. We convert the original image in a pixels masks of 0s or 1s. The pixel will be 1 if it passes a predefined condition. As we know the colour of the ball (blue) we know what is the range of the hue (H) that we want to filter out. We can use the opencv function inRange:

mask = cv2.inRange(image, lower, upper)

The mask will have a 1 if it is in the range between lower and upper.

Tip: Depending on the segmentation task is useful to smooth the image before doing any gradientbased operations (e.g., edge detection, contours, etc) to remove hard changes on the image. Furthermore, in real image situations when segmenting by colour there are always artefacts and noise. The standard procedure to remove undesired blobs is to perform the erosion and dilation (overture) operators. Check the morphological operators of opencv here: https://opencv-pythontutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphologica l_ops.html

3. Blob segmentation and computing the centre of mass

We want to obtain the centre of the ball in image coordinates. For that purpose, we will use the function findContours from opencv. This function returns a list of contours. Check: https://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html

Once we have the contours we can compute the centroid of the object. If you have the ball perfectly segmented (only that blob in the image) you can just sum up all the pixels location that are 1 and then dividing by the total number of pixels that are 1. This is actually the arithmetic mean. Here, we are

going to use a more advanced method: moments. This approach works with any **greyscale** images (https://en.wikipedia.org/wiki/Image_moment). We compute the centre of mass of the blob with the function m = *cv2.moments(contour)*. This provides the contour moments that we can use to compute the centroid:

cx, cy = int(m["m10"] / m["m00"]), int(m["m01"] / M["m00"])

Note that I converted the float into an integer as the image indexes are integers. The way to access the moments in python is with a string. Check the opencv documentation for more information (https://docs.opencv.org/3.4/d8/d23/classcv_1_1Moments.html)

Tip: The m00 is the area of the contour. If it is zero it will raise an exception. Remove the blobs that are too big or too small.

4. **Using the blob location for the head controller.**

Once we have the centroid of the ball we can send it to the controller to centre the head. We can bypass the frame of reference transformations with a small hack and a proportional controller in the following form:

4.1. We compute the normalized position of the ball in the image by dividing by the size (e.g., x/width). Then we compute the error and we multiply by a positive constant (K=0.1):

dx = K* ((x / width) - 0.5)

The 0.5 makes that the range goes from -0.5 to 0.5 (instead of 0 to 1)

We compute the vertical component similarly

*Note: the robot will not move when the ball is in the centre centre of the image will give K*0.*

4.2. Finally, we subtract dx to the yaw angle of the head and check the limits of the joint before sending the angle to the joint.

Alternative solutions: there are many ways to solve the ball tracking problem. However, the conceptual idea is already in the proposed solution. For instance, we can compute the 3D location of the ball given by the environment and then project into the image plane. Another approach is, instead of thresholding, to use a ball detector.