Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fakultät Maschinenbau und Mechatronik
Mechatronic and Micro-Mechatronic Systems

Final Project
Robotic Workshop

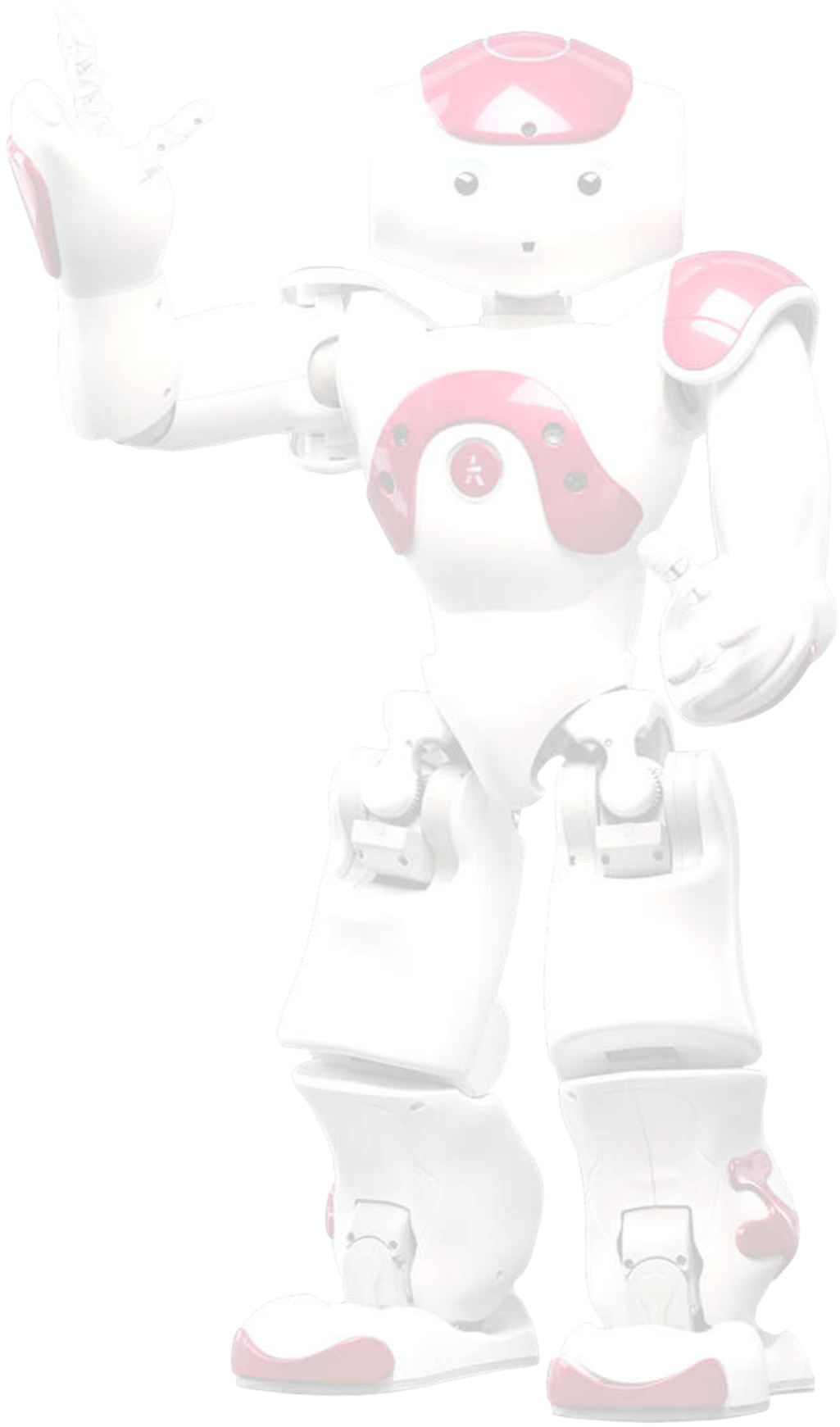
Assistance to people through facial recognition and tracking

Group 2

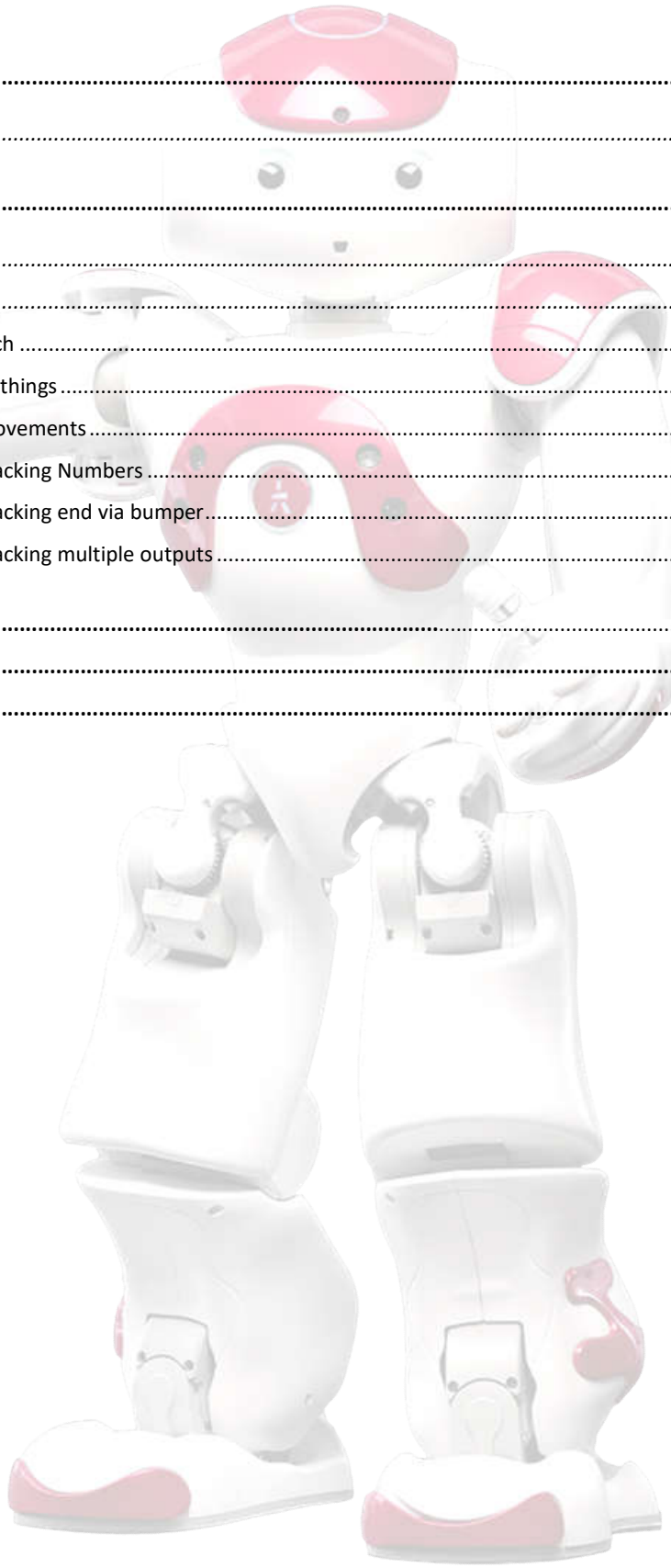
Santiago Ramos

Axel Fürholzer

Pilar Samaniego



Content



1 Task

NAO as humanoid robot has a wide field of application. However, one of the most important tasks of humanoid robots is in field of assistant for people. The propose of this project is show some of the features of NAO robot in this field. For that, in first instance the robot will interact with the user and recognize it through face recognition. Then, depending on the person the robot must be able to bring an object from the drawer of that specific person, that means the robot just follow the instruction of people stored in its database. At the end the robot will inform the person that the object was successfully delivered and interchange some words with the user.

1.1 Objectives

- Nao should recognize people's faces
- Nao should be able to find the object / Landmark corresponding to the person.
- NAO should take the corresponding box and maintain the same posture while walking to the point of return with the box in arms
- NAO should deliver the box at the destination point
- Nao should be able to grab the box and leave it by programming timelines

2 Execution

2.1 Programm

To better understand the operation of the program, a flow chart is presented below.

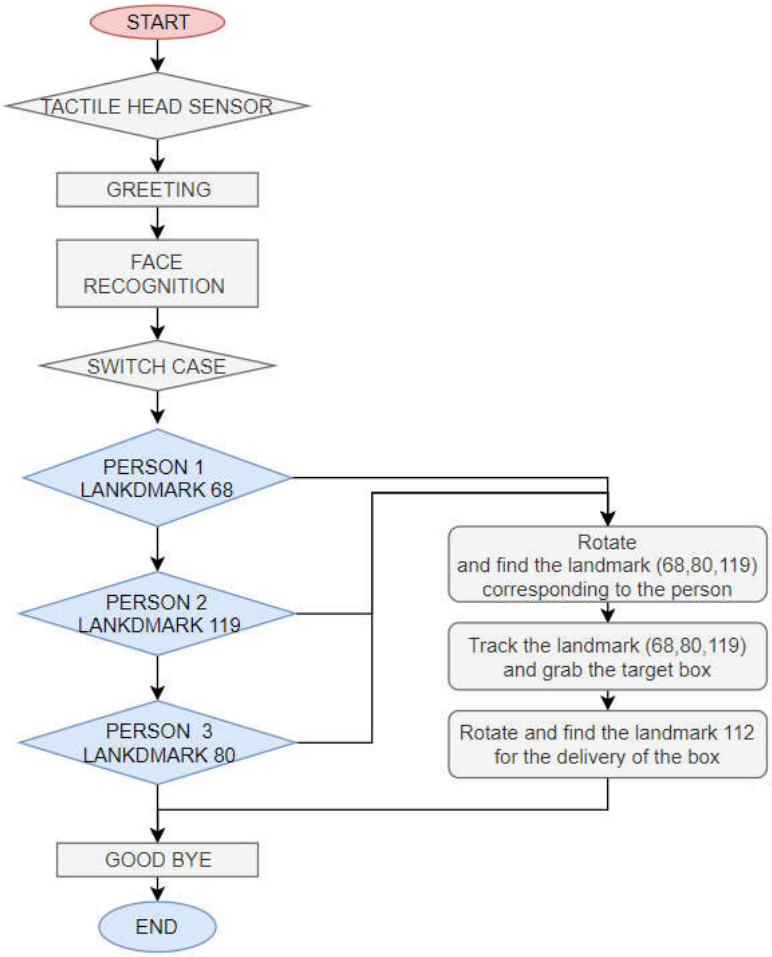


Figure 1. Flow Diagram Project

The program starts when the robot sends a signal that the Tactile head sensor was activated and greets "Hello I am NAO. Welcome to the Campus Tag" and starts its motors to stand up.

The camera is selected for facial recognition, and the robot is expected to recognize a person in order to continue with the assigned tasks.

In the workspace there will be a total of four landmarks. Three marks that correspond to each of the persons and one to define the return point. This can be seen in the following picture:

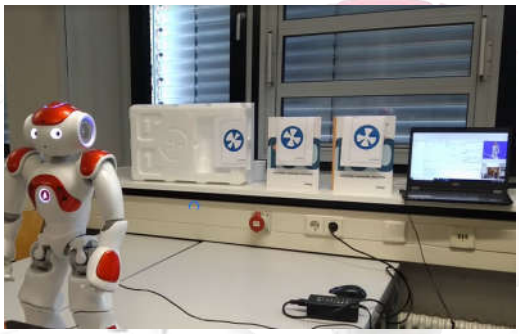


Figure 2. Workspace with Landmarks ID: 68,80,119

The ID-s of the corresponding landmarks are 68,80,119 and 112 that of the final destination. Below you can see the landmarks used

Figure 3. Landmarks ID: 68,80,119, 112

The faces of recognition is of the three members of this group, according to the corresponding landmark the robot tracks and returns the name of the person on its out.

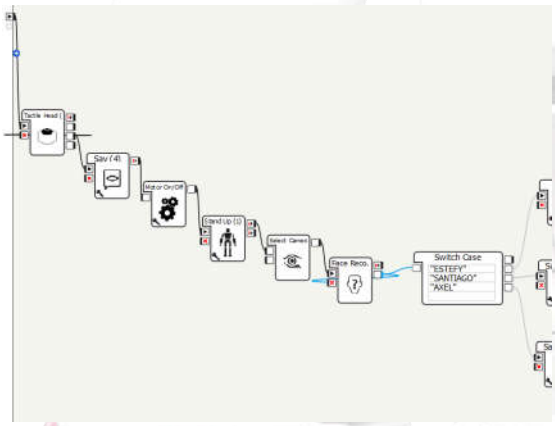


Figure 4. Choregraphe Program Begin

The robot greets the respective person and turns to search for their respective Landmark. The robot achieves its objective thanks to the distance that has been configured in the "Landmark Tacking" block

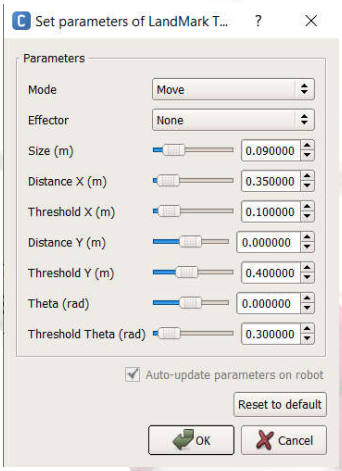


Figure 5. Settings – LankdMark tracking

If the robot previously encounters the table, the sensors of the bumpers are activated to stop tracking. Once the target is reached, an additional block "NAO Mark" is used, which gives us the ID of the landmark that the robot is currently seeing.

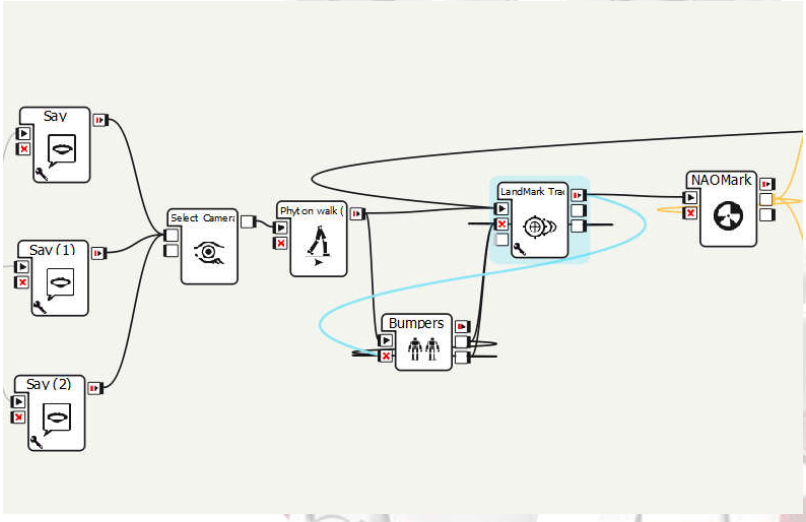


Figure 6. Choregraphe Program - Bumpers

Due to the programming problems of the program and random activation of the outputs in any instance, a comparison block "IF" is used to verify that the ID that the robot sees

corresponds to the current recognized person, otherwise without this comparison the Robot will activate any other instances of the block.

If the mark is the correct one, the Robot must pick up the box placed and turn to look for the final destination mark "ID: 112. "

Figure 7. Landmark ID: 112

A feedback is seen to the same tracking block since if it was instantiated twice it generated problems in the outputs, therefore the instantiated block contains internally two of the IDs, the lankmarks of each person and the final mark, for example: 68 and 112 or 119 and 112, etc.

Script editor

Phyton walk (4) | LandMark Tracker (2)

```
1  import time
2
3  class MyClass(GeneratedClass):
4      def __init__(self):
5          GeneratedClass.__init__(self)
6          self.tracker = ALProxy( "ALTracker" )
7          self.memory = ALProxy("ALMemory")
8          self.targetName = "LandMark"
9          self.distanceX = 0.0
10         self.distanceY = 0.0
11         self.angleWz = 0.0
12         self.thresholdX = 0.0
13         self.thresholdY = 0.0
14         self.thresholdWz = 0.0
15         self.subscribeDone = False
16         self.sizeMark = 0.0
17         self.markIds = [68, 119]
18         self.effector = "None"
19         self.isRunning = False
20
```

Figure 8. Python Code – ID Landmarks used

If the robot recognizes the landmark, the tracking begins and the NAO should take the box to the final objective. A PYTHON block was programmed in which the movement of the arms was deactivated while the robot walks with the code: "motionProxy.setWalkArmsEnabled (False, False)"

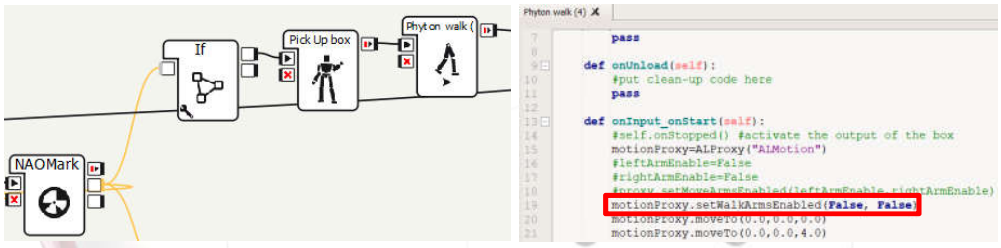


Figure 9. Choregraphe Program and Python Code Disable Arms

When the robot rotates the next landmark of the final target must be recognized (ID -112). For this, the previous trackign block was used as explained.

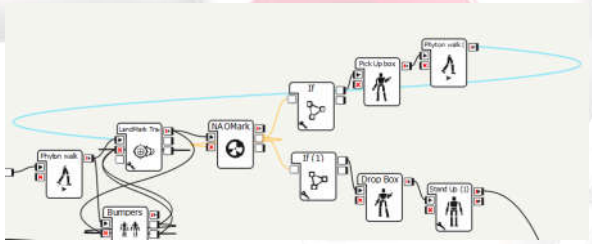


Figure 10. Choregraphe Program

And when arriving at the preselected distance the robot finally delivers the box and says goodbye to the user. For security, the robot can also be stopped with the Tactil head sensor at any moment of operation.

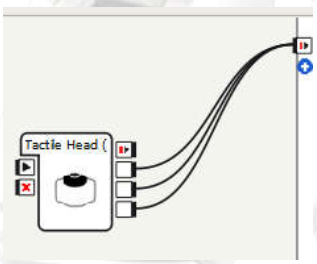


Figure 11. End of the Program

The final program is as in the following figure. There are two additional blocks that helped us to save the faces and erase the faces of people previously saved in the robot database

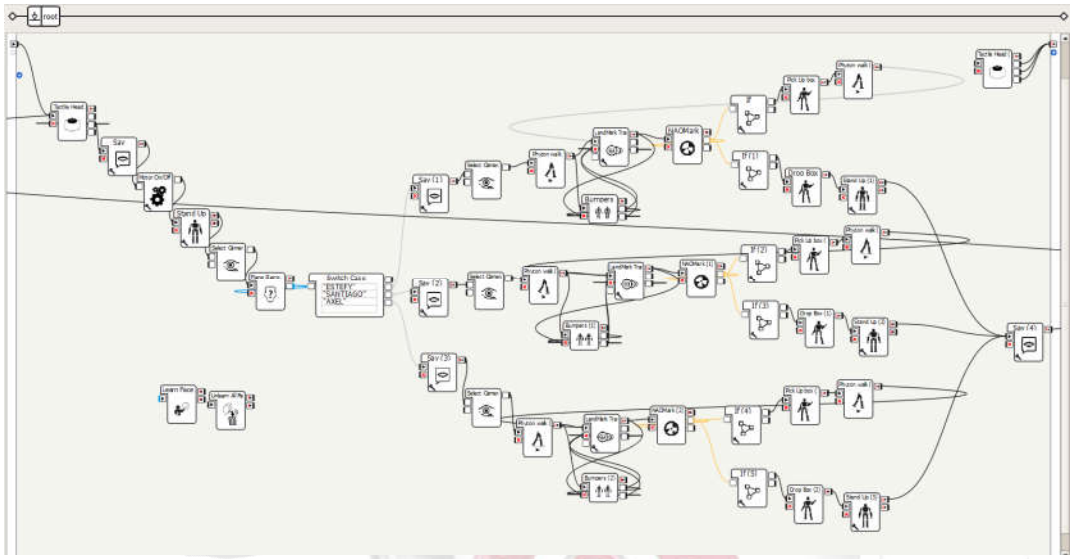


Figure 12. Program

2.2 Challenges

2.2.1 Camera Switch

There was a difference in height of the landmarks. To see both, the robot has to switch between the cameras within the program. Because it can't switch it automatically a block was implemented to change the camera used.

2.2.2 Transporting things

With its hands before its torso, NAO is already in danger of falling over. The load that he can transport is therefore nearly 0.

2.2.3 Unprecise Movements

The movements of NAO lack precision When turning 180°, in cases it turns nearly 250°.

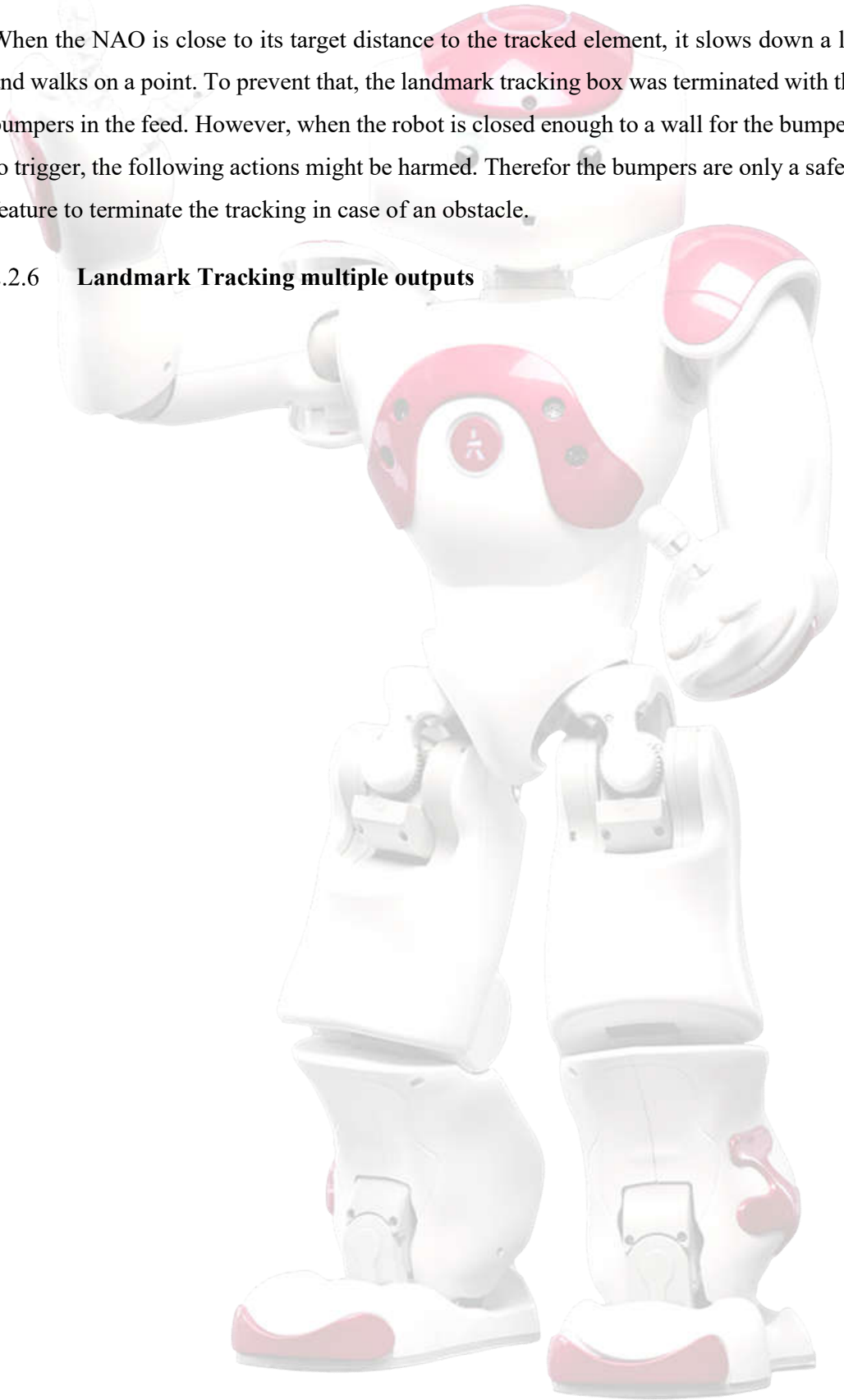
2.2.4 Landmark Tracking Numbers

The landmarks all show a unique number. The landmarks that were downloaded from the wiki contained numbers that are not included in the ALTracker by default. The problem with this block is that by instantiating it several times, the target output achieved is activated in all existing blocks, which was solved through two solutions that are explained in the following pages. So, the python script in the box was changed. All numbers were thrown out and replaced with only the ID the NAO should track at this point and with the Id of the end point.

2.2.5 Landmark Tracking end via bumper

When the NAO is close to its target distance to the tracked element, it slows down a lot and walks on a point. To prevent that, the landmark tracking box was terminated with the bumpers in the feed. However, when the robot is closed enough to a wall for the bumpers to trigger, the following actions might be harmed. Therefor the bumpers are only a safety feature to terminate the tracking in case of an obstacle.

2.2.6 Landmark Tracking multiple outputs



Landmark tracking was the biggest challenge of this task. If there are multiple landmark tracker in a program the outputs of every single one trigger when a tracking is completed. In a line of boxes with no branches but with two multiple landmark trackers there will be two boxes activated at the same time.
To proof this behaviour a simple code was written.

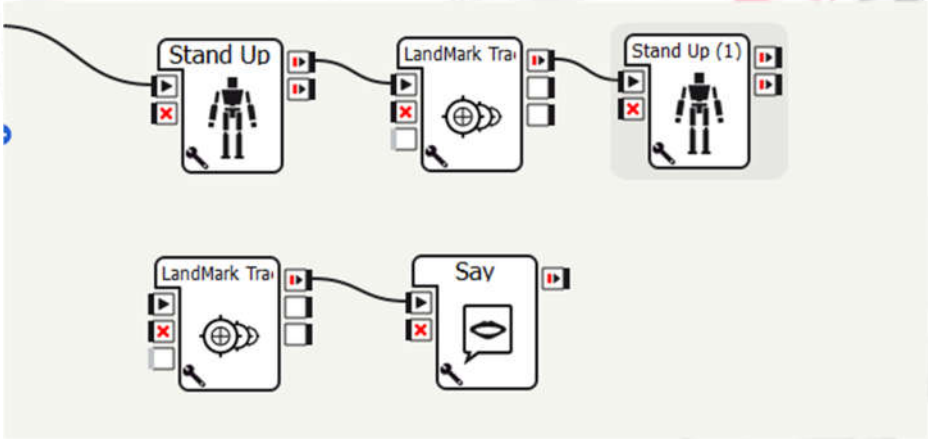


Figure 13. Choregraphe Program – Land Mark Trackign

In the code, a simple Landmark Tracking happens. The lower landmark tracking is not activated at any point but still outputs a signal, when to landmark tracking in the first line finishes. This behaviour also appears when the tracked landmarks in these blocks differ. To overcome this problem multiple solutions were tested.

The code line self.tracker.stopTracker() was implemented at different points of the code. This did either seem to have no impact at all or result in the block not working anymore.

The class self is present in both blocks. The tracker seems to be an objective of this class. If one of the tracking blocks initiated a second class (self->sel), there would be two separate trackers in the two classes. This didn't have any effect at all.

```
import time

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.trocker = ALProxy( "ALTracker" )
        self.memory = ALProxy("ALMemory")
        self.targetName = "LandMark68"
        self.distanceX = 0.0
        self.distanceY = 0.0
        self.angleWz = 0.0
        self.thresholdX = 0.0
        self.thresholdY = 0.0
```

Figure 14. Python Program Tracking block

The tracker itself is initiated in the first lines of the block. If the object tracker is initiated twice under different names, there should be two in depended functions that are running separately. There was no difference with these changes implemented.

2.2.6.1. Solution 1

So instead of a solution to the problem, a work around was implemented.

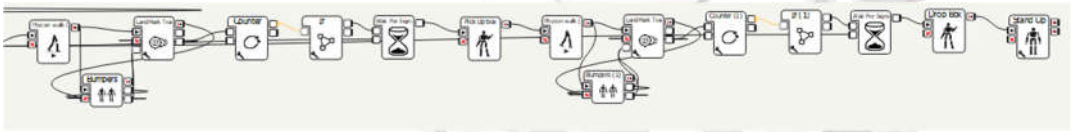


Figure 15. Choregraphe Program Solution 1

A counter and a modified if block were implemented behind the outputs of every landmark tracker. The counter adds one, every time the tracker triggers the input. The if was modified to react in case of the modulo being 0 (or 1). The bang output of the if block is only triggered every second time the tracker outputs something. Additionally, the block the signal is only transmitted to the following blocks, if the line of blocks is activated from the switch case at the beginning.

```
def onInput_onData(self, p):
    param = self.getParameter("Value to compare")
    try:
        p = float(p)
        param = float(param)
    except:
        p = str(p)
        param = str(param)
    operator = self.getParameter("Condition operator")
    if( operator == "<" ):
        self.outputTrueOrFalse( p < param )
    elif( operator == "< or =" ):
        self.outputTrueOrFalse( p <= param )
    elif( operator == "=" ):
        self.outputTrueOrFalse( p == param )
    elif( operator == "> or =" ):
        self.outputTrueOrFalse( p >= param )
    elif( operator == ">" ):
        self.outputTrueOrFalse( p > param )
    elif( operator == "≠" ):
        self.outputTrueOrFalse( p% param == 0 )
    else:
        raise Exception( "Operator not known: " + str(operator) )

def outputTrueOrFalse(self, condition):
    if( condition ):
        self.output_then()
    else:
```

Figure 16. If Block Solution 1

2.2.6.2. Solution 2

It is the solution implemented in the final program. A landmark Tracker block with just the marks of interest is created. In order to differentiate the actions for each mark, a NAO Mark detection block and an if block is placed behind the output of the tracker. The output of the if block is trigger only if the output of the NAO mark detection block correspond to the if value setting in the block, so that even if the landmark tracker activate for any mark, the if block doesn't allow the execution of its actions. As can be seen in the image the pick up action is going to be activated just when the NA-OMark sent as output the ID of the Mark for the object to be grasped that correspond with the ID inside the if block of the upper branch. The opposite case take place when the NaoMark correspond to the one inside the if block for the drop box block located in the lower branch.

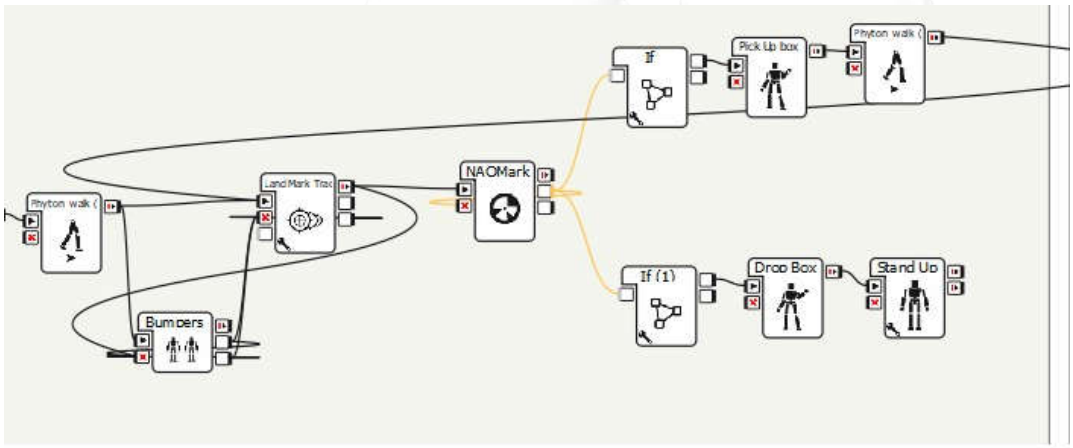
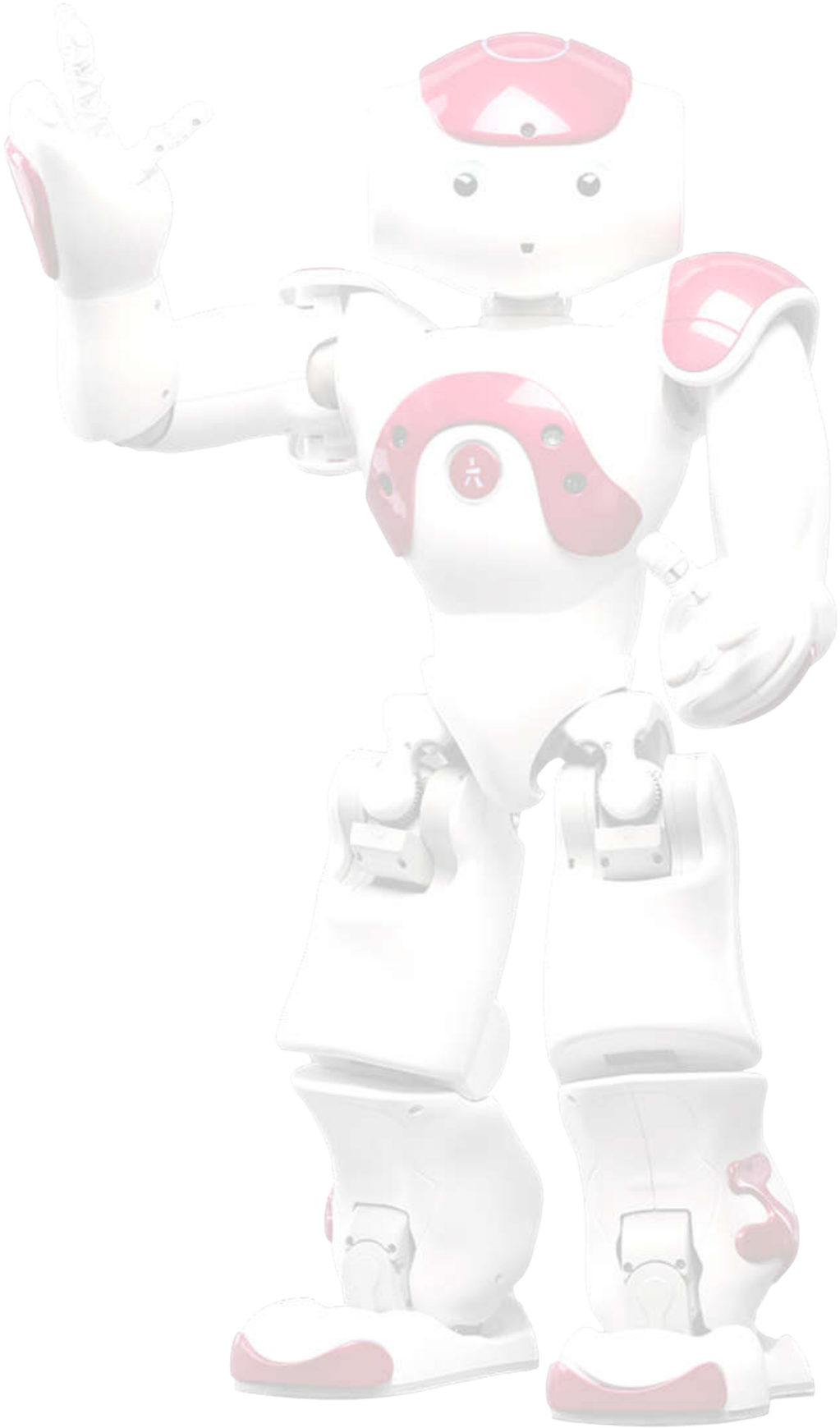


Figure 17. Solution 2



3 Results and Conclusions

Even though there are some limitations in the robot as a whole system, the robot was able to complete the task of recognize a specific person and bring the object assign to that person and deliver it in the right place. However, there is some limitation that doesn't allow to achieve the task in the expected way. The main limitation is the load capacity of NAO robot, since it is not able to hold an object with the arms in front because the center of gravity change and the robot tend to fell. Furthermore, the grasp-ing action require advance algorithms to check the position of the object and accurate movement to reach the object in any position, which is not possible with the blocks available in the Chore-ography software. For this reason, the only objective that couldn't be achieved was grasp-ing an object.

Other limitation is the repeatability of the robot and accuracy, so that the robot doesn't achieve the same target when a programmed movement is performed, and this behav-ior could be corrected implementing a control loop. On the other side, the mark track-ing task was the most challenging for the project and two solution were used to solve this problem. On the final implementation the second solution was used since require less blocks than the first alternative. In addition to the problematic of activation of the land-mark tracking block, the differentiation of the target mark is complicated. The robot is not able to differentiate the target mark when other mark is inside its visual field. For that, the marks were separate from each other, so that when the robot track one and come close to the mark, there is not interference.

As general conclusion the NAO robot has some limitations, but it is suitable to learn graphic programming language and even though the robot doesn't fill all the objec-tives of the task of this project, it solves the general idea of the task. In order to take advantage of NAO capacities, advance programming skills with Pythion is required, which allows to improve the performance of the existing blocks of the Choreographe software.

4 Bibliografía

- https://www.youtube.com/watch?v=nF2suPjTF9g
- http://doc.aldebaran.com/2-8/naoqi/trackers/trackers-sample.html
- http://doc.aldebaran.com/2-8/naoqi/vision/allandmarkdetection.html#allandmarkdetection