NAO robot that imitates human movements using Mediapipe

Costilla Caballero, Salvador
Applied Robotics
Hof University of Applied Science
CDMX, Mexico
salvadorcoscab@gmail.com

***Abstract*—In this project we made a NAO robot, from Aldebaran, move imitating human movements. This movements were captured with a webcam using Mediapipe. Though this human imitation has already been done, we wanted to make it in a different way. Because most of the other projects use multiple cameras or depth sensors, such as the LIDAR sensor. The project is way far from being finished, because the robot only moves his arms and not very precisely. But we think is a good start for future work on computer vision and robotics.**

***Index Terms*—NAO, Mediapipe, Human imitation, aldebaran, robotics**

I. State of the Art

In this section we will describe different ways to move a NAO robot the Mediapipe Framework and a work related to human imitation using the NAO robot.

The NAO robot docomentation [**?**] states that there is two ways to move the NAO robot to do poses. The first one is using the angles from the joints and the second one is using inverse kinematics. Though there is more ways to move the robot, those are more for walking or to perform a predefined pose.

There is two ways to control the effectors to perform a pose, regardless we use inverse kinematics or not, which are:

- Animation methods, in which the time is fixed and there are blocking functions.
- Reactive methods, that can be used to control the robot in real time, and there are non-blocking functions.

We have various functions contained in the *ALMotionProxy* module, related to the movement by controlling the joints some of the animation methods are:

- *ALMotionProxy.angleInterpolation*, that allows you to move the robot from a specific position to another in a specific time.
- *ALMotionProxy.angleInterpolationWithSpeed*, that allows you to move the robot from a specific position to another in a specific time and speed.

One important method is:

- *ALMotionProxy.setAngles*, that allows you to move set the angles of a single joint or a set of joints of the robot.

As it was stated before the robot can be controlled using inverse kinematics as well, in this case a classical IK solver which uses only the joints of the effector chain to reach a target and a generalized IK solver (also called Whole Body control) which uses all the robot joints to reach a target. Some of the animation and reactive methods related to the inverse kinematics that we consider important are:

- *ALMotionProxy.positionInterpolation* that allows you to move the robot from a specific position to another in a specific time
- *ALMotionProxy.setPosition*, that allows you to move the robot to a specific position in a 3d space.

BlazePose [**?**], is and architecture for single person 3D pose detection based in convolutional neural networks. It is available in the Mediapipe framework. And allows you to detect poses from a video or an image. There are 33 landmarks that the BlazePose model can detect (See figure **??**). These

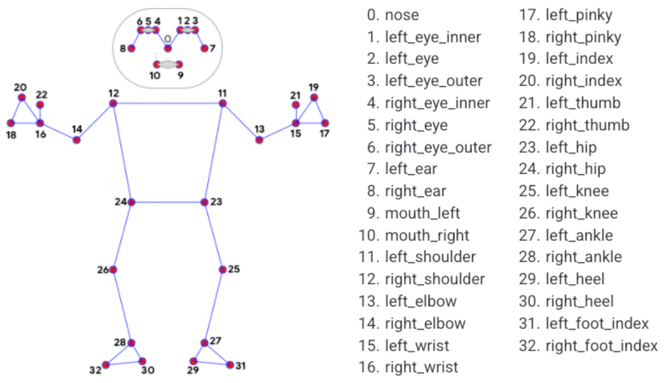

Fig. 1. Landmarks from the Mediapipe library.

landmarks can be read as a set of points in a 3D space. The Landmarker has various configuration options, such as:

- *model_complexity*, that allows you to choose between three different models, the lite, the full and the heavy. Depending on the model chosen the accuracy and the speed can increase.
- *min_detection_confidence*, that allows you to set the minimum confidence that a pose is detected.
- *min_tracking_confidence*, that allows you to set the minimum confidence that a pose is tracked.

In a thesis called " Mimetización de movimiento por robot antropomorfo basado en imágenes de sensor de profundidad" [**?**] the authors use the NAO robot to imitate the movements of a human using a kinect sensor. They use a client-server approach to send the data from the kinect sensor to the robot through the NAOqi API. In the client part they use the *pykinect2* library to get the data from the kinect sensor, this data are 25 landmarks that represent the human body. These data is then sent to the server part using the *socket* library. In the server part they calculate the angles between the joints and then send them to the robot using the *ALMotionProxy.AngleInterpolation* method.

II. Approach

A. Objectives

During this project we had some objectives which were:

- Solve the computer vision part, that in this case was done with Mediapipe.
- Make the robot move using the NAOqi API.

The problems were that, in order to work with the NAOqi API, we had to download and setup the SDK which is written (by the moment of this project) in Python 2.7 and the Mediapipe is only availble in Python 3.7 or higher. That is why we had to add as an objective to make the two libraries work together.

B. Methodology

The methodology that we used to solve the objectives above was to first divide the problem into modules and try to solve them separately, and then join them together and solve the problems that arose from the connection between the two modules. The main two modules were the computer vision and the robot movement.

C. Computer vision

In the beginning, the computer vision part was done very easily, because Mediapipe has a lot of usage examples and good documentation, such as the video ”AI Pose Estimation with Python and MediaPipe” [**?**], that helped us a lot on getting the landmarks and calculate some angles. On the other hand working with the NAO robot was a much more difficult, because we did not know how to start.
For the computer vision part we used the opencv library to get images from both camera and videos and then we used the Mediapipe library to get the poses from the images. We used also the opencv library to plot the landmarks that the Mediapipe library detected, so we could see how the landmarks were being detected. This landmarks conformed an object with 33 points, which represents landmarks of the human body. (See figure **??**)

D. Robot movement

Then we started working with the robot movement part, to do this we used the NAOqi API. We realized, after reading the NAOqi documentation that there is two ways to control the robot for doing poses. The first one was using angles and the second one was using inverse kinematics. We decided to use the first one because we thought that with the landmarks, obtained with Mediapipe, we could easily get the angles of each join of the robot, and also the project that we found on internet used this approach. All of this joint-angle approach is written in the *ALMotion* module of the NAOqi API. In summary the robot has various joints, and each joint can be moved using a specific angle. These joints work in a similar way as the human body joints, allowing some joints to pitch, roll, and yaw. (See figure **??**)

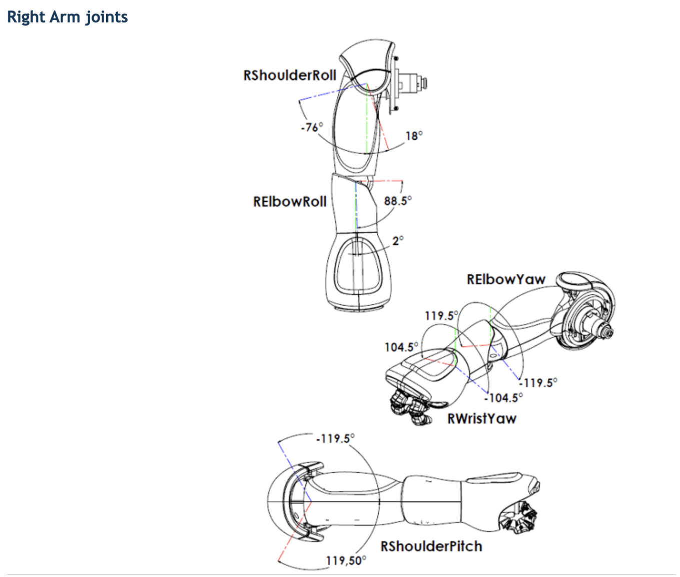

Fig. 2. Landmarks from the Mediapipe library.

E. Connection between the Computer vision and the Robot movement

After doing some testing on both parts, we had to somehow connect them. In order to do the connection between the robot and the computer vision part, we tried with *execnet*, which is a library that allows you to trigger python scripts from another python script, even with different versions of python. But we had some problems sending the data from the computer vision part to the part in which we move the robot, so we decided to try a different approach.

We decided to use sockets to send the data, as we got inspiration from a project that we found with a similiar approach, in which the author used sockets to send the data from a kinect sensor to the robot. This solved everything, about the connection between the scripts, because it uses the client-server approach, allowing me to use python 3.10 for the computer vision part that is the client and python 2.7 for the Robot movement part that is the server. We just had to do some encoding to send the data.

F. Testing the implemented code and solving problems

After changing the code that we found to work with Mediapipe and OpenCV instead of the kinect sensor. We did some testing on the functions that were already implemented in the server side, but they performed very bad, but at least, we saw that the robot moved.

Also the display of the images was very slow since the code had a delay to send the data, in order for the server to

process it. We solved this using a thread to get the data from the displayed image and send it to the server with the thread without messing up with the display.
As we said before after changing a lot the parameters and definition of the functions that were implemented before we realized that we was not sending the right data to the robot from the server, so we decided to change the approach and instead of sending the coordenates of the landmarks to the server, we decided to send the angles between them and send them directly to the server. This would help me to see what angles we was sending to the robot by showing them on the displayed image.

After we was able to see the angles that were being sent to the robot, we realized that the angles sent were not very good in the function definition, this is because the kinect sensor and Mediapipe have different coordenate systems, so we had to change the defintion of the functions.
This did not work very well, though the movements were more accurate, the robot was not imitating the movement of the hands perfectly. So we decided to change the function. Instead of using geometry we tried with vectors and the angles between them.

G. Vector approach

We saw that as we had the landmarks, that represents a point in the space, we could calculate a vector between two points (See equation **??**), and we could use the angles between the vectors to move the robot (See equation **??**). But then we realized that we could also calculate the angles between the projection of the vectors (See equation **??**) in a generated plane from a normal vector (See equations **??**,**??**,**??**). We tried this for one joint, that was the right shoulder pitch. We got the normal vector from the left shoulder to the right shoulder to generate the plane, and then we got the vectors from the right shoulder to the right elbow and from the shoulder to the right hip. We projected these vectors in the generated plane and then we calculated the angles between them.

$$\vec{v} = \vec{p_2} - \vec{p_1} \tag{1}$$

$$\cos(\theta) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\|\|\vec{w}\|} \tag{2}$$

$$proy_H \vec{v} = (\vec{v} \cdot \vec{u_1})\, \vec{u_1} + (\vec{v} \cdot \vec{u_2})\, \vec{u_2} \tag{3}$$

$$\vec{n} \cdot (\vec{p} - \vec{p_0}) = 0 \tag{4a}$$

$$\vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4b}$$

$$\vec{p_0} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{4c}$$

We implemented a function in python to do the calculations mentioned above and this function did not work well neither when we tested it, as we could see the angles on the displayed image, we realized that the Mediapipe library did not detect the z axis very well, even with a more complex model. Then we remembered that the kinect sensor has a LIDAR sensor, that allows him to detect, not only position in x and y, but also depth in a better way than the Mediapipe library. That is why we did not continue with this approach and as we did not have more time to finish the project, we decided to leave the functions that were already implemented.

III. Contribution

The main difference between my project and the others is that it has a different approach to solve the problem, relying only on one camera and machine learning to detect and recreate the human poses, instead of using a lot of sensors. This is a contribution to the state of the art, because it implies that if there were better machine learning models, the robot could imitate the human movements in a better way. There is also a contribution to the work done before in ”Mimetización de movimiento por robot antropomorfo basado en imágenes de sensor de profundidad” [**?**], because we can see the angles that are being sent to the robot, which can be useful for further research.

IV. Experiments

To do the experiments and testing we used Choreographe, which has a simulator mode, in which you can see a NAO robot and move it using the localhost as the robot IP (See figure **??**). We used this to test the robot movements, but we also used a real robot to see the movements in a real robot (See figure **??**). We had a script to try the robot movments and another script just to try the computer vision part. The union of both scripts, as we said before, was done using sockets. So we had to run two scripts in the terminal, one for the server and one for the client.

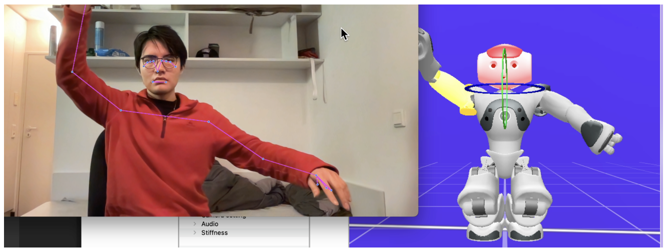

Fig. 3. NAO robot in the Choreographe simulator.

We used a webcam and videos to see the angles that are being sent to the robot. To first initaite the application you have to run the server part, using a python 2.7 interpreter. This can be done using the following command, in which you have to specify the ip address of the robot as an argument:

```
python2 server.py "ip_address"
```

Then you have to run the client part, using a python 3.10 interpreter. If you want to send the landmarks using a webcam you can use the following command:

```
python3 client.py
```

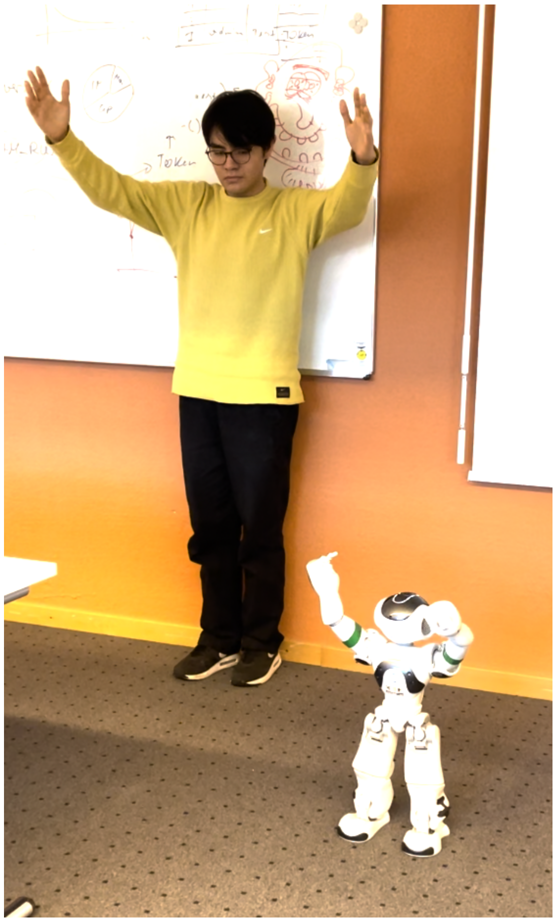
Fig. 4. NAO robot moving.

On the other hand, if you want to send the landmarks using a video you can use the following command:

```
python3 client.py "path_to_video"
```

Sometimes the robot falls and if this occurs the application might not work or would stop working, so we had to be careful about this, because sudden movements of the robot could make it fall. If the application stops working, there is no other option than to restart the application repeating the steps above.

V. Conclusion and future work

Even though the project is not finished, we think is a good start for just using computer vision to apply it in robotics.
There is a lot of work to do, such as improving the calculation of the angles or use both inverse kinematics and programming the angles to move the robot.
In order to use the legs of the robot is necessary to do further research, because we have to consider many other things such as the mass center of the robot, and take care that the movements are not too abrupt to make the robot fall.

Acknowledgment

We want to thank my proffesor, Dr. Christian Groth for helping us during the project and thank also to the Hof University of Applied Science for giving us the opportunity to work with the NAO robot, which was a great experience.

References

[1] NAOqi API, *http://doc.aldebaran.com/2-5/naoqi/motion/index.html*
[2] Mediapipe Pose Landmarking*https://developers.google.com/mediapipe /solutions/vision/pose_landmarker*
[3] I. Irigoyen, ”Mimetización de movimiento por robot antropomorfo basado en imágenes de sensor de profundidad,” Trabajo Fin de Grado, Facultad de Informática, Grado de Ingeniería Informática, Universidad del País Vasco, 2020.
[4] Nicholas Renotte - Robotics and AI, ”AI Pose Estimation with Python and MediaPipe,” YouTube, 2021. [Online]. Available: https://www.youtube.com/watch?v=06TE_U21FK4t=2109sab_channel= NicholasRenotte.