# Doggoborg project report - Robotics 2021

Elise van Wijngaarden
Antonio Mone
Julia Wąsala
Mick Remmerswaal
Wytze Breukel

May 28, 2021

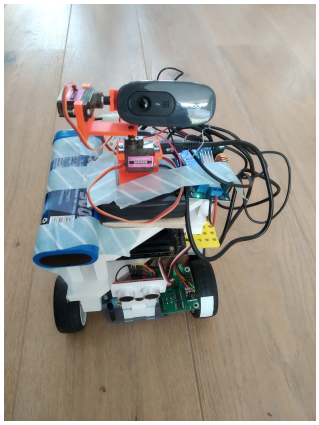

Figure 1: Doggoborg front

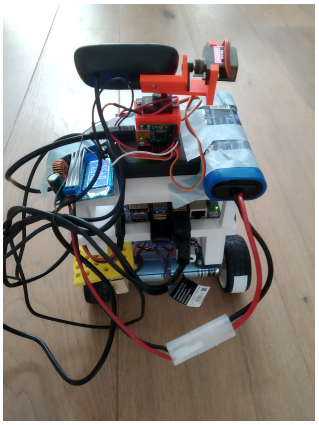

Figure 2: Doggoborg back

**Abstract**

For this project we created a dog like robot consisting of a YetiBorg and a YetsonNano. This robot had the ability to respond to it's own name, and make eye contact using it's servo mounted webcam.

## 1 Introduction and Overview

### 1.1 Novelty

The goal of the DoggoBorg was to create some sort of companion/pet for people who can't have a pet or don't want to deal with the maintenance and hassle of having one. Of course this is extra relevant in these times of social isolation. We decided to create this companion in the shape of dog due to the fact that dogs are know to be very sociable and cute. Although robot dogs are not unheard of but most of them are either focused on practical applications like Boston dynamics spot, or are more toy like. We decided to focus on the interaction between robot and owner, and gave it the ability to hear it's name and to make eye contact to simulate a living being.

### 1.2 Related work

The most famous robotic dog at the moment is `Spot`[1] by Boston dynamics, which can be used for physical applications like inspecting building. If we look at a more social applications we see that robotic dogs are

[1] `https://www.bostondynamics.com/spot`

used in for example nursing homes [1]. An interesting comparison can be made with the NAO[2] Robot although this is humanoid in shape it also has the same goal of making a social connection with the user via face and audio recognition.

# 2 Design

The final robot consists of a number of components:

- Yetiborg v2 [3]
- ZeroBorg [6]
- UltraBorg [4]
- Jetson Nano [2]
- Webcam with Microphone
- Webcam mount with 2 servo motors
- US sensor [5]

These webcam and jetson nano were mounted on the robot using a self-designed 3D printed component. The design can be seen in Figure 3. The green platform is the platform placed directly on the yetiborg, the red one is placed on top of the green platform. The requirements for this component were:

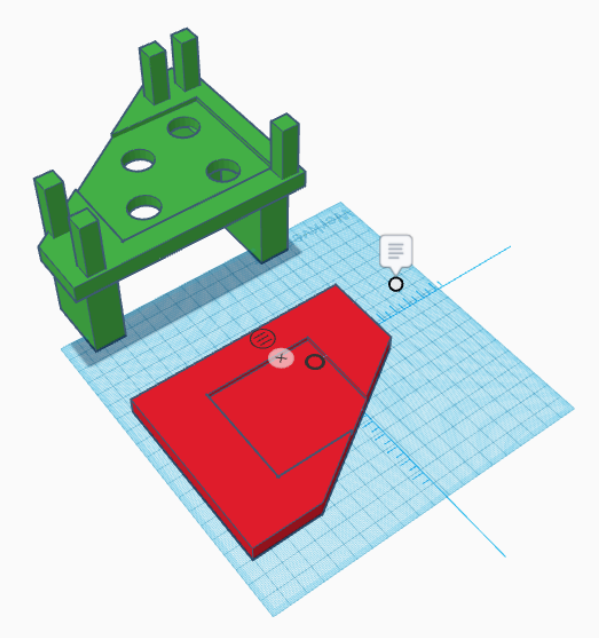

Figure 3: Printed component design

- the legs fit over the screws sticking out of the wheelplate of the yetiborg;
- the component would not interfere with any existing hardware on the yetiborg such as the camera and US sensor;
- the lower platform would have space for the jetson nano, oriented in a way that the cables to and from the jetson nano would be at the back of the yetiborg;
- the platform on top would hold the webcam mount;
- the construction needed to be stable enough such that the yetiborg could ride around safely.

# 3 Implementation

## 3.1 Audio recognition

Audio recognition was done by implementing one of the many possible speech recognition libraries available. The choice was between Google Cloud Speech library and the CMUSphinx library. We speculated that the robot may need to perform offline, therefore we chose the CMUSphinx library.

After experimentation with the library we found that the original dictionary was not able to comfortably recognize the word 'Dog'. We therefore created our own library consisting of four words: 'Hey', 'Hi', 'Dog' and 'Doggo'. With the four words as our corpus we used the CMU provided language model tool[3] to create a language model for our corpus.

[2] https://www.softbankrobotics.com/emea/en/nao
[3] http://www.speech.cs.cmu.edu/tools/lmtool-new.html

Implementing the language model and the created dictionary was as simple as providing the path to those files. The library offers many other ways to implement speech recognition, such as keyword search. After experimentation with those, we did not find an increase in the times the word 'Dog' was recognized and used the language model in the final product. Since the webcam came with a builtin microphone, no external source was needed for the capture of the audio.

### 3.2 Face recognition

The moment that Doggoborg hears its own name 'Dog', it tries to find its owner. A webcam was mounted on the robot to be able to 'see'. The robot turns in a circle to try and find the owner. Every time that the robot moves, a picture is taken with the webcam which is then evaluated. The goal is to find the owner, a human face.

To find a human face in the picture, the openCV library was used which includes the cascade classifier `haarcascade_frontalface_default.xml`[4] which we used to find a face. When a face is detected, this classifier returns the $x$ and $y$ position of the face, together with the width and the height of the face in the picture. With these four values, it can be determined in what way the robot has to move in order to center the human face in the middle of the picture. An example of this would be the following: when a face is found in the upper right of the picture, the robot has to move a little to the right and the webcam has to be tilted upwards in order to 'look' straight at the face. These two movements are controlled by two servos (one in the horizontal and one in the vertical direction) on which the webcam is placed.

Once the robot has found a face and has centered in the middle of the picture that the webcam takes, the robot is moving towards the owner until it is close enough. The robot is close enough to the owner when the face has an area of 27500 pixels. While approaching the owner, the robot tries to keep the face centered by adjusting if the face moves out of the middle of its view.

### 3.3 Locomotion and communication

The Jetson Nano communicated with the yetiborg using a web server hosted on the raspberry pi, because the two were not physically connected. An API made it possible for the Jetson Nano to send commands to the yetiborg's motors using HTTP requests. The webserver code was largely based on the pre-existing example code: `yeti2Web.py`[5]. The advantage of this approach was that it made it possible to do some testing before mounting the Jetson Nano on the yetiborg. The disadvantage was that in the example code, not all of the source code was available. Because of this, we did not know what was happening exactly when a HTTP request was being processed.

The web server was augmented with two features, that were disabled later for reasons that will be explained shortly. The first feature was obstacle avoidance: the robot would spin 180 degrees when it encountered an obstacle at a distance of 30 centimeters. The second feature was so called idle behaviour: when the robot was not given any commands, it would turn left and right randomly to mimick something like animal behaviour. Both these features were disabled in the end, because we feared they might cause interference with the Jetson Nano scripts, which were the primary goals of the project. We could not find the source of the interference, because a layer of the code was missing in the example code: the webserver handler function that was called, was not the same as the one implemented in the example.

## 4 Results

The finished Doggoborg is shown in Figures 1 and 2. In these images the front and the back are respectively shown. To see the Doggoborg in action, a video is included in the submission. In the video, Doggoborg is called and it turns towards the owner and looks at its face.

[4] `https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php`

[5] `https://www.piborg.org/blog/yetiborg-v2-examples-web-ui`

# 5 Conclusions and Discussion

Overall the DoggoBorg worked as expected but not as reliable as we hoped, we think this was cause by the large amount of processes that needed to work together (face recognition, movement of the servos, audio recognition, locomotion etc), each of these has a small chance to fail which combined causes the Doggoborg to fail more often than we would have liked. However when all components did function, it worked completely as expected.

Audio detection worked less than admirable, the way we had to pronounce the word 'Dog' seemed to change day by day. This made testing a little bit hard. The integration worked as intended when the Doggoborg did recognize its name. Result may improve when using the Google Cloud Speech API, since the Doggoborg did not have to perform offline this is an interesting alternative.

The face recognition worked almost perfectly in a controlled setup with a static camera, but gave less accurate results in practise. We believe this was caused by a few different factors: Firstly, the lighting conditions are very variable in our testing environment, for example in the room where we were testing, there were two bright windows which caused quite contrasting lighting conditions. We tried to mitigate this by lowering the curtains and turning on the lights to create an more evenly lit environment. Also, the movement from the yetiborg caused some pictures to be blurry. We therefore decreased the turning speed. Lastly, the room we were testing in (a living room) had a lot of objects which could confuse the face recognition (ironically it even reacted to a toy dog), it even recognised itself as a face in a shiny surface. We therefore decided to remove all distracting items and cover all reflective surfaces which resulted in improved results.

The webserver offered a disadvantage: not the entire source code was available in the example code that was used, which meant that the flow of information was not immediately clear. For example, it was not obvious to us how requests would be prioritized: what would happen if the jetson nano sent a motion command to the yetiborg, while it was turning because it encountered an obstacle? For future projects, a close look needs to be taken at the yetiborg web server, or a custom one needs to be implemented. This would allow for tighter control over priorities of commands to the yetiborg motors.

Some of the physical problems we encountered were with the printed component: in the design process, a mistake was made with aligning the legs of the platform, so they had to be removed and reattached at the correct place in order to fit on the wheelplate. Another minor issue was caused by the fact that reverse motion of the yetiborg is less powerful than forward motion. Because of the weight of the complete setup, sometimes the power applied to one set of motors in forward motion, was not enough to make the other set of wheels turn backward (for example when the yetiborg had to spin). This meant that when the robot was spinning, the wheels that should have turned backward, often did not turn at all. This did not prove to be an obstacle to achieving the goal of the project, but still is a factor to be improved upon.

# References


[1] Marian R. Banks, Lisa M. Willoughby, and William A. Banks. "Animal-Assisted Therapy and Loneliness in Nursing Homes: Use of Robotic versus Living Dogs". In: *Journal of the American Medical Directors Association* 9.3 (2008), pp. 173–177. ISSN: 1525-8610. DOI: https://doi.org/10.1016/j.jamda.2007.11.007. URL: https://www.sciencedirect.com/science/article/pii/S1525861007005166.

[2] *Jetson Nano Developer Kit*. Apr. 2021. URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[3] PiBorg. *YetiBorg v2 - Getting Started*. URL: https://www.piborg.org/blog/yetiborg-v2-getting-started.

[4] *UltraBorg - PWM Servo Control w/ Ultrasonic Sensor Support*. URL: https://www.piborg.org/sensors-1136/ultraborg.

[5] *Ultrasonic Distance Sensor (HC-SR04)*. URL: https://www.piborg.org/sensors-1136/hc-sr04.

[6] *ZeroBorg Complete - Quad Motor Controller*. URL: https://www.piborg.org/motor-control-1135/zeroborg.