2016

Ida-Louise G. Borlaug
Gunnar Charlie David Bjørk
Konstantino Z. Helders
Ole Steinar L. Skrede

TTK4851 - Robot and Humans

Group 1

Project

Facial Recognition for the NTNU Cyborg

Abstract

In this project a face recognition system that can be integrated into the NTNU Cyborg was created. This was done by using a basic open-source face recognition algorithm distributed under an open license in the OpenCV library. This was modified and built on to improve the recognition rate. The functionality to automatically detect and crop out just the face from a larger picture was added. A database-script was written to handle these images and safely store them. An interface was created where the robot asks questions of the user and the images are thereafter sorted into the database in order to create the basis for the face recognition task. All this was lastly integrated into the Robot Operating System. The algorithm is effective and works with images taken of the subject in front profile and with small angles. Side profile pictures where only one eye is shown do not work. Pictures where the subject is so far away that the face is smaller than 200×200 pixels do not work due to lack of information. The algorithm works for pictures where the subject is smiling or making a grimace most of the time, but this varies between subjects. The algorithm was effective in recognizing faces. Suggestions for further functionality to improve the system have been made.

Contents

List of Tables

List of Figures

Abbreviations used

ROS Robot Operating System[12]

API Application Programming Interface

PCA Principal Component Analysis[11]

LDA Linear Discriminant Analysis[8]

LBPH Local Binary Pattern Histogram

1 Introduction

1.1 Background

The task of this project is to implement a face recognition system for The NTNU Cyborg [14]. The NTNU Cyborg is a project in development, where the goal is to create a robot that can move around on campus and interact with people. Mounted on the cyborg is a Microsoft Kinect v.2 [6], which will function as the robots "eyes". The Kinect has a camera, the ability to output the video stream to a computer, and extract some information from the frames [7]. The NTNU Cyborg team wanted to utilize the Kinect in order to let the robot recognize people and thus be able to perform more interesting interactions. The system is going to be installed as a module in the robot's operating system, Robot Operating System (ROS) [12].

1.2 Motivation

The project was chosen based on a request from the NTNU Cyborg. After learning they needed people to implement a face recognition system for their cyborg, a decision was quickly reached to take the job. It seemed like an interesting and fun project with the possibility of gaining a lot of valuable experience implementing new functions in a working robot. This was a good fit with the current skill level and motivation of the team members.

1.3 Limitations

The team consisted of only four members, one who had little to no programming experience. This was the smallest group in the class, which limited the work capabilities for such a daunting task where the group members met for the most part only once a week. Access to the main cyborg system was limited as multiple teams worked on the same system. Access to a Kinect v.2 sensor was not gained until about halfway through the project.

1.4 Contributions

In this project a basic open-source face recognition algorithm distributed under an open license has been modified and built on to improve the recognition rate. As well as adding functionality to automatically detect and crop out just the face from a larger picture. A database-script was written to handle these images and safely store them.

An interface was created where the robot asks questions of the user and the images are thereafter sorted into the database in order to create the basis for the face recognition task. All this was lastly integrated into ROS.

1.5 Outline

- **Literature study:** What has been done in the field before. Previously tested algorithms and their results. Pros and cons of different methods for face recognition.
- **Theory:** In depth and mathematical explanation of the theoretical workings of our algorithm. Explanation of ROS.
- **Proposed solution:** Our complete system explained.
- **Implementation:** How each module was implemented and connected.
- **Results:** All tests and examples with results.
- **Discussion:** Ethical challenges with regards to privacy. Evaluation of the face recognizer.
- **Conclusion:** Conclusion and closing thoughts.
- **Further work:** Suggestions for improvements and future features.

2 Literature study

A lot of work has been done to implement face recognition in the past. Big companies like Facebook use face recognition to suggest identities of people in pictures, Microsoft use it for automatic log-in on multimedia devices like the Xbox One, and Google uses it to create automated albums[1] of your relatives and friends in Google Photos. The technology is still new, and is improved upon daily.

In addition to the big companies, there exist multiple open-source projects that experiment with face recognition, operating with an open license so everyone can use and contribute to the code in order to improve it.

[1] Only available in the US because of EU regulations. April 2016.

2.1 Face Recognition Algorithms

2.1.1 Eigenfaces

One algorithm is the Eigenfaces method[15]. This method uses a holistic approach to face recognition: A facial image is a point from a high-dimensional image space and a lower-dimensional representation is found, where classification becomes easy. The lower-dimensional subspace is found with principal component analysis[11], which identifies the axes with maximum variance. This information is then used to look for similarities in other images.

However, there is a problem with this method. If the variance is generated from external sources, like light, the axes with maximum variance do not necessarily contain any useful information for recognition at all. This means a classification becomes impossible.

2.1.2 Fisherfaces

In order to solve the problem from 2.1.1 a class-specific projection with a linear discriminant analysis[8] was applied. In order to find the combination of features that separates best between classes, the linear discriminant analysis maximizes the ratio of between-classes to within-classes scatter, instead of maximizing the overall scatter[3]. The idea is simple: same classes should cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation.

This method was called Fisherfaces from the statistician Sir Ronald A. Fisher who came up with the linear discriminant analysis. The Fisherfaces method learns a class-specific transformation matrix, so they do not capture illumination as obviously as the Eigenfaces method. The discriminant analysis instead finds the facial features to discriminate between persons.

It is important to mention that the performance of the Fisherfaces method relies heavily on input data. In order to get good recognition, you need multiple pictures of the same person, in multiple types of lighting, simply because features that are dominant in good lighting might be less so in bad lighting. This means that if you train the algorithm using multiple pictures in one type of lighting/one specific angle, the algorithm might not recognize subjects in other types of lighting, or from a different angle. The more varied the training-pictures are, the surer one can be that the Fisherfaces will find the correct components in a picture.

This means that in order to get good results one needs multiple pictures of the same person, preferably in different lighting conditions and from slightly different angles.

Since the NTNU Cyborg will be moving around, asking to take pictures of people, it cannot be ensured that those conditions are met. This means another algorithm is needed.

2.1.3 Local Binary Patterns Histograms

Analysis of both above methods[9] show that in order to get the best results, one will need approximately 9 images in varying light and viewing angles.

In order to remove this need, research was concentrated on extracting local features from images, and use this information for recognition. Instead of looking at the entire picture as a high-dimensional vector, you only care about local features of the object. This is a good idea in theory, but also means that the picture will be sensitive to translation, rotation and scale. This means that an algorithm using this method will have to be robust against these things in addition to light variations.

The idea behind LBPH is to compare each pixel with its neighborhood. Choose one pixel and compare its intensity to all 8 surrounding pixels using a set threshold. Any pixel fulfilling the threshold is set to 1, the rest is set to 0. Each pixel will then be denoted by a binary number, like 10011101, describing its neighborhood in a clockwise manner. This gives 2^8 possible combinations, these are called *Local Binary Patterns*[1].

By definition the LBP operator is robust against monotonic gray scale transformations. Processing the image to remove color and applying a gray scale transformation makes the image more robust to light variations.

The LBP method makes it possible to extract very fine-grained details in images. However, using a fixed neighborhood in the method it was shown to fail at encoding details differing in scale. The method was extended to use a variable neighborhood[1].

Finally, it was proposed by Ahonen et al.[1] to divide the LBP image into m local regions, and extract a histogram from each. The spatially enhanced feature vector is then obtained by concatenating the local histograms (not merging them). These histograms are called Local Binary Patterns Histograms and can then be used for recognition while being robust to both translation and rotation.

2.2 Existing project on facial recognition with the Kinect v.2

There is exists a project that has implemented facial recognition with the Kinect v.2 sensor[13]. It is written in the C# programming language for the Windows API. It is based on the eigenvector method mentioned in 2.1.1, as well as using some built-in functionality in the Kinect sensor for skeleton detection in order to locate people before

applying face recognition. Since the NTNU Cyborg is implemented in ROS on Linux and using C++, this existing project was not used as a resource.

2.3 OpenCV

OpenCV[10] is an open source computer vision and machine learning software library. The library has more than 2500 optimized algorithms of both classic and state-of-the-art machine learning variations. It is a valuable resource for work done previously in the computer vision field, all of which is BSD-licensed such that it may be used and modified for free for both academic and commercial use. It provides a valuable resource for face recognition methods[4].

2.4 Summary

There are plenty of resources available in order to implement a face recognition system. The challenge will be to create a working module that communicates with the NTNU Cyborg such that the final system satisfies the requirements set in 1.1.

3 Theory

3.1 LBPH Algorithm

In order to satisfy the needs described in 1.1, the LBPH algorithm from 2.1.3 was chosen due to its robustness towards lighting conditions, translation, rotation and scale, together with its flexibility of not needing multiple pictures of a face in order to work satisfactory. It is also simplistic in nature to implement.

A description of the LBP operator can be given as

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c) \tag{1}$$

Here (x_c, y_c) denotes the central pixel with intensity i_c, while i_p is the intensity of the neighbor pixel. s is the sign function defined as

$$s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{else} \end{cases} \tag{2}$$

This enables the algorithm to capture very fine grained details in images.

To make sure the algorithm encodes details differing in scale, the operator is extended to use a variable neighborhood. To achieve this an arbitrary number of neighbors are aligned on a circle with a variable radius. This enables the algorithm to capture the neighborhoods depicted in figure 3.1.

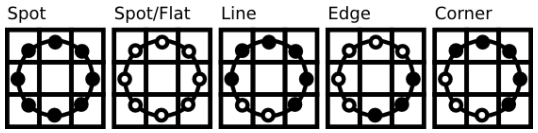


Figure 3.1: Patterns in the LBPH algortihm

For a given point (x_c, y_c) the position of the neighbor (x_p, y_p), $p \epsilon P$ can be calculated by

$$x_p = x_c + R\cos(\frac{2\pi p}{P}) \tag{3}$$

$$y_p = y_c - R\sin(\frac{2\pi p}{P}) \tag{4}$$

Where R is the radius of the circle and P is the number of sample points. If a point's coordinate on the circle does not correspond to image coordinates, the point gets interpolated. The OpenCV implementation[4] does a bilinear interpolation

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix} \tag{5}$$

Then, since the LBP operator is robust against monotonic gray scale transformations, processing the images to remove color and applying a gray scale transformation makes the images more robust to light variations. This is shown in figure 3.2.

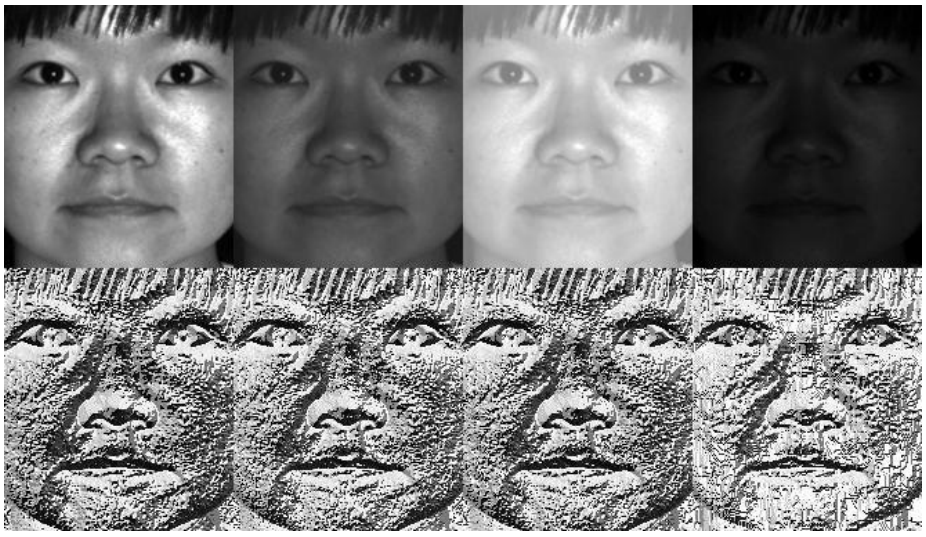


Figure 3.2: Gray scale transformation of images with local binary patterns

Finally, the LBP image is divided into m local regions, and a histogram is extracted from each. The spatially enhanced feature vector is then obtained by concatenating the local histograms (not merging them).

4 Proposed solution

Our solution is to create a face recognizer module and a database module that can be plugged into ROS. This is illustrated in figure 4.1. The NTNU Cyborg already has functionality for retrieving the stream from the Kinect v.2 sensor. When the robot detects a person through the Kinect, it can pass an image to our face recognizer module. The black box represents everything in the robot between the Kinect and our modules. The system can send images to the face recognizer, which will extract useful features and then check if it looks similar to images in the database. By communicating with the database module, the face recognizer should be able to add new persons to the database when it does not recognize the person in the input image. The database should provide a list of images and their corresponding label to the face recognizer, based on the content in the database. It should also be able to retrieve the name and an image of the person, given a legitimate label.

The face recognizer module should be able to give information about the person that has been recognized back to the black box so it can take further action. When it does

not find a match it should ask if the person in the image wants to be added in the database. If they do, they should be added, and the black box should be notified that a new person has been added. If they do not, no action should be taken.

It is also important that our implementation is easy to plug into ROS, and easy to be used by other parts of the ROS system.

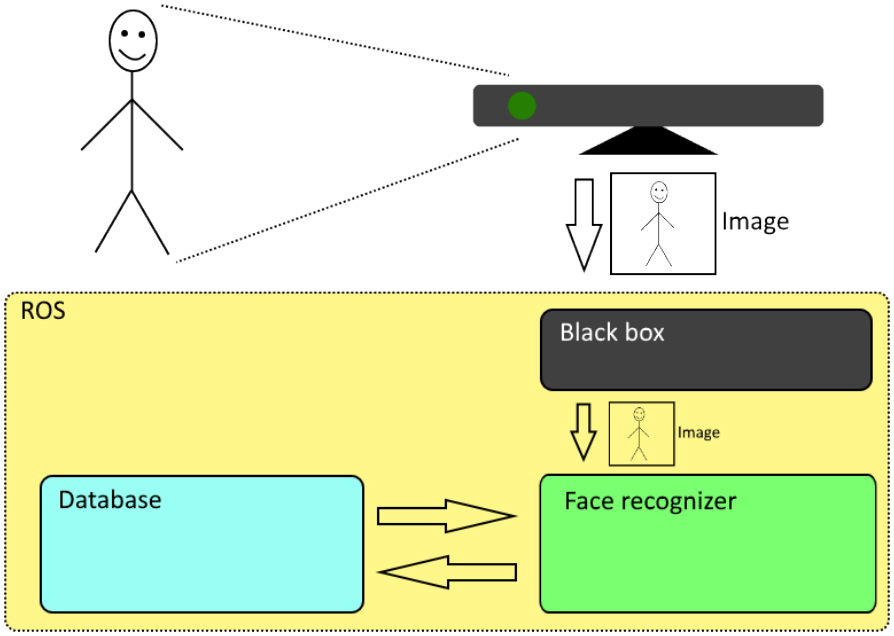


Figure 4.1: Overview of the system

5 Implementation

5.1 The database

The database is just a directory organized with sub-directories for each person. Each sub-directory contains images of the person and a text file that contains some information. The system only stores the first name, the last name and a timestamp in this text file. The names are stored so that the robot could greet a person when recognizing her/him. It would also be possible to use the timestamp to customize the greeting. For example the robot could say "Long time, no see." if it is a long time since it interacted

with the person, or it could say "So, we meet again!" if it recently interacted with a person. This is not implemented, but the possibility is there. The main reason for having a timestamp is so the robot can forget people it has not interacted with in a long time. This should serve to limit the size of the database over time. There is also a limit on the maximum number of images that should be stored. The default image limit per person is 50 images, and the default time limit before a person is deleted is roughly two years (62208000 seconds). Both these values can be modified in the Database_manager class in scripts/database_manager_server.py. In the top of this file, one can also set the path to the directory one wants to use for the database. Python is used to write the code that organizes the database (database_manager_server.py), and it is implemented as a service node for ROS. This means that it listens for requests and performs actions when called upon. Other nodes can ask the database service to add a picture of an existing person, forget old entries (this is also done automatically when adding new persons), forget a specific person, add a new person, and get the name of a person. The service returns a message that describes the success or failure of the requested operation, or the name of the requested person. After making changes to the database, the database service node always updates the .csv file that is used by the face recognition module. The .csv file contains paths to all the images in the database, and with each path there is also an associated label that corresponds to the person in the image.

5.2 Face recognition

The face recognition used is an implementation of the algorithm described in 3.1. The algorithm was already implemented in the OpenCV 2.4.12 library[2]. This was therefore used. The algorithm takes in the path of where the .csv file is located, and the path for the image of the person that the algorithm will try to identify. The algorithm uses the .csv file to train a model that can be used for the recognition. A threshold for this model can be decided. The threshold decides how good the recognition has to be for it to be approved. If the threshold is too small it will discard recognitions that are correct, if it is too high it will find a match where there is none. This value therefore had to be chosen with care. The method used to find this value was trial and error, and in the end the algorithm worked in nearly every case. The model predicts which one of the persons in the database that gives the best match. If the match is above the threshold, the algorithm provides the label (which folder in the database) that corresponds to the person that the model thinks the person is. If the match is below the threshold the model gives out -1.

[2]This version of the OpenCV library was used because later versions did not work well with Ubuntu 12.04, which was required because of the ROS version used by the NTNU Cyborg.

5.3 ROS implementation

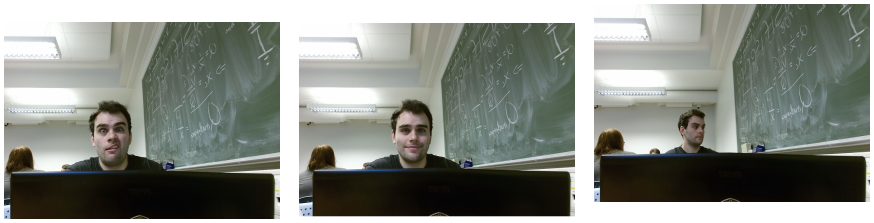
The face recognition algorithm had to be implemented in ROS [12] because the cyborg group uses this system on their robot. The face recognition algorithm was implemented as a service node in a package called facerec. The facerec package consist of the service node and the client node, also called facrec_server.cpp and facerec_client.cpp. The service node takes care of the face recognition part. It listens to a service called FaceRec. The service is used for sending requests and responses between the server and the client node. When a request is sent from the client node, the server node sends a response back. In this case it is the path to the picture and the .csv file that is sent as a request and the label from the face recognition algorithm is sent back as a response.

The client node is the node that ties our systems together. The client node is called with the path to the image of the person we want to identify. It gray scales and crops the face out of the image, and saves this new image in the path of the old one. It then sends the path to the image and the .csv file as a request over the FaceRec service. When it gets a response it checks if it is a label or a -1. If it is a label it finds an image and the name of the person in the database and asks the user if the face recognition have made the correct choice. If the response is a -1 it asks the user if it can add the person in the database. The client node therefore acts as an interface between the user and the program.

6 Results

To find out how good the system worked some tests were performed. The different input pictures given to the system were:

- Close up:
  - Smile
  - Grimace
  - Resting face
  - Side profile
- At some distance:
  - Resting face
  - Side profile



(a) Close up: Grimace

(b) Close up: Smile

(c) Long distance: Side profile

Figure 6.1: Pictures used in the test

The system was also tested with people not in the database to see that they were given the opportunity to be added, or if they were mistaken as someone else.

The pictures in these tests were taken with the Kinect 2 sensor that is used by the NTNU Cyborg team. Some examples of the pictures used in the tests can be seen in figure 6.1.

The results are found in table 6.1.

Table 6.1: Results from face recognition testing

Type of picture	Close up	Long distance
Smile	$\surd$	Not tested
Grimace	$\surd$	Not tested
Resting face	$\surd$	X
Side profile	$\surd$/X	X

It can be seen from table 6.1 that all the pictures taken "close up" return correct results, except for the side profile which sometimes returns the correct result and sometimes returns the wrong person, or gives no person found. The pictures taken at a "long distance" always return no person found. When the person in the picture was not in the database, the result was no person found nearly every time.

7 Discussion

7.1 Ethical issues of our project

In Norway there are some restrictions on how pictures and other information about persons can be used [2] [5]. In short, when storing images of a person, the consent of the subject is needed to show that image in public. A person's consent is also needed when storing information about that person's name, location and the time when the person was at a location.

The implementation stores images of persons, their name, and a time stamp for when the cyborg last interacted with them. Therefore, it is important that the system always asks the user for permission to store this information. The stored information is only meant to let the system delete all information about persons it has not interacted with in a long time, and the possibility to give customized greetings to recognized persons (by using their name and possibly mentioning that the cyborg has not seen them in a long time). So the stored information is not meant to be viewed by anyone else. However, there is always the possibility that someone gets their hands on the information, and the system currently has no measures to protect the information. It should be possible to add functionality to the implemented database such that it encrypts the stored data. Both the images and the personal information could be encrypted, but the current implementation is not made with consideration of this and thus a lot of the code would probably have to be rewritten. Encrypting the images could also have a noticeable impact on the systems performance.

Since the NTNU Cyborg is supposed to wander around campus, it could be easy for anyone to gain physical access to the data it contains. Therefore, it is recommended that anyone that uses our code considers whether the lack of security is an issue, and also notes that it is required by law to ask the user for permission when storing information.

7.2 Evaluation of the face recognizer

The implementation of the database and the face recognition works separately and as a complete system in ROS, but there are some room for improvement.

The face recognition can be improved in several ways. The threshold was decided by trial and error. This method has some drawbacks and a better value for the threshold can probably be found. The threshold assigned was 60. With this threshold the face recognition works with pictures taken from the front profile. The person can then be smiling, making a grimace or have a neutral expression. Sometimes it also works when the picture is taken from the side, but here it depends on how much the face is turned. If the face is turned all the way to the side the function that crops the image will fail. This is because this function is dependent on seeing both eyes, and the face recognition will therefore also fail. If the face is turned with a small angle, and both eyes are visible the face recognizer manages to find the correct person most of the time.

At a long distance the face recognizer does not manage to find the correct person, and returns no person found. This is because the picture that is returned from the crop function has too little information. The face recognizer needs a picture that is larger than approximately 200x200 pixels to have enough information to recognize a face. The picture that is returned from the crop function is approximately 120x120 pixels

when the picture in figure 6.1c is given as input. This is not enough information and the face recognizer therefore fails, and returns no person found. This means that for the face recognizer to work, the person in the picture has to be closer than in figure 6.1c.

8 Conclusion

In this project we tried to implement a facial recognition system that can run in ROS and thus be integrated into the NTNU Cyborg project.

Using OpenCV and a simple database implementation we have made a system that runs in ROS and performs reasonably well. The implemented system also works with pictures from Kinect 2, which is what the NTNU Cyborg uses. This means that the system should be easy to integrate into the NTNU Cyborg project, but we have not been able to test this, so there could be some problems that have not been considered. Also, some work needs to be done to let the users interact with the system. When doing this, one needs to consider the ethical issues of using the face recognition system, and ensure that the system notifies its users with information according to local law.

While there is room for improvement in the system, it should work with satisfying results as it is. There are however some issues with the threshold. Finding a threshold that does not give false positives or returns no person found when the person actually was in the database was seemingly impossible. We therefore chose a threshold that would rather give false positives instead of false negatives. This was because the user then would be able to say that the program had made a wrong choice, and then add herself to the database. Since the system has not been tested in the NTNU Cyborg yet, it is bound to have some problems, and bugs, that have not been considered and discovered yet.

If the system can be integrated properly into the NTNU Cyborg, and any unforeseen problems are solved, it should provide exciting functionality to the cyborg. The ability to recognize people can make the cyborg feel more human-like and "alive" when interacted with. Hopefully, this will contribute to making the cyborg an interesting attraction on NTNU, which people find worthwhile returning to in order to see if the cyborg remembers them. Even though there might still be some work to do to utilize the full potential of the face recognition system, we believe that it is worth the effort with polishing and further developing the system, because it could have great potential in adding further interesting features to the NTNU Cyborg.

9 Further work

- **Cyborg integration**: The currently developed system works as a stand-alone module in ROS. It needs to be added to the main NTNU Cyborg and integrated with the system there.
- **Kinect integration**: Since the NTNU Cyborg team has a working integration of the Kinect 2 sensor in their ROS system, time was not spent developing this functionality here. Since the module developed in this project relies on an image of a face being available, the functionality to take a screenshot from the Kinect 2 stream that the NTNU Cyborg has available needs to be implemented. This could be done by prompting the user if it is OK that a picture of their face is taken after the user is approached by the cyborg.
- **Facebook integration**: Another project the NTNU Cyborg is currently working on is getting its cyborg online on Facebook. Here people will be able to add it as a friend. Considering this, it would be interesting to implement a feature where the cyborg could, after getting permission, access its friends pictures, and use this information to train the face recognition algorithm and recognize its Facebook friends when encountered in the real world.
- **Testing the system**: The system should be tested more when it has been completely integrated with the NTNU Cyborg system, because there might be some problems and bugs that have not been discovered yet.

References

[1] A. Ahonen, H. Abdenour, and M. Pietikäinen. *Face Recognition with Local Binary Patterns*. IEEE, 2006. URL: `http://www.ee.oulu.fi/mvg/files/pdf/pdf_730.pdf`.

[2] *Article 45c of the Norwegian Copyright Act*. URL: `http://www.lovdata.no/all/hl-19610512-002.html#45c` (visited on 04/13/2016).

[3] P. N. Belhumeur, J. Hespanha, and D. Kriegman. *Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, No. 7, 1997. URL: `http://www.cs.columbia.edu/~belhumeur/journal/fisherface-pami97.pdf`.

[4] *Face Recognition with OpenCV*. URL: `http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html` (visited on 04/13/2016).

[5] *GPS and tracking - Datatilsynet*. URL: https://www.datatilsynet.no/Teknologi/Sporing/ (visited on 04/13/2016).
[6] *Kinect*. URL: http://www.xbox.com/en-US/xbox-one/accessories/kinect-for-xbox-one (visited on 03/02/2016).
[7] *Kinect Specifications*. URL: https://en.wikipedia.org/wiki/Kinect_for_Xbox_One#Specifications (visited on 03/02/2016).
[8] *Linear discriminant analysis*. URL: https://en.wikipedia.org/wiki/Linear_discriminant_analysis (visited on 04/06/2016).
[9] A. Martinez and A. Kak. *PCA versus LDA*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 23, No. 2, 2001. URL: http://www2.ece.ohio-state.edu/~aleix/pami01.pdf.
[10] *OpenCV*. URL: http://opencv.org/ (visited on 04/13/2016).
[11] *Principal component analysis*. URL: https://en.wikipedia.org/wiki/Principal_component_analysis (visited on 04/06/2016).
[12] *Robot Operating System*. URL: http://www.ros.org/about-ros/ (visited on 03/02/2016).
[13] M. Rosack. *Sacknet.KinectFacialRecognition*. URL: https://github.com/mrosack/Sacknet.KinectFacialRecognition (visited on 03/02/2016).
[14] *The NTNU Cyborg*. URL: http://org.ntnu.no/thentnucyborg/ (visited on 03/02/2016).
[15] M. Turk and A. Pentland. *Eigenfaces for Recognition*. Journal of Cognitive Neuroscience, Vol. 3, No. 1, 1991. URL: http://www.face-rec.org/algorithms/PCA/jcn.pdf.

10 Appendices

A Code explanation

Here we explain the structure of our code and explain the most important functions. And we also mention how it fits into ROS.

A.1 The database manager

Everything is implemented in one file, except for the .srv service description files. The file contains two classes: Person and Database_manager, some methods to handle ROS requests and a main() method that is run when this program is started as a ROS service.

In the top of this class the user should specify where the database should be stored by editing the value of the 'base_path' variable.

The .srv service description files describe the methods and their parameters that a ROS service makes available. They define the request/response communication between nodes. For more on srv, see here: [srv - ROS Wiki]

Person This class defines how the info.txt file, which stores info about a person, should look like.

Code description of the Person class

Method	**Description**
to_string()	Returns the text that should be stored in the info.txt file for this person

Database_manager This class functions as the interface between the database in the file system and other processes that want to use the database in any way. It has parameters that defines the the acceptable time since a person was last interacted with before he gets deleted (n_DEFAULT_AGE_LIMIT). This value is given in seconds. The class also contains the variable that defines the maximum number of images that can be stored for each person (n_IMAGE_LIMIT), and an array that defines the accepted image formats for images in the database (accepted_image_formats).
Each person in the database is stored in his own subdirectory named sID, where ID is an unique number. When the method description mentions "id" it refers to these unique numbers, which act as an identification number for each person.

Code description of the Database_manager class

Method	**Description**
find_used_ids ()	Used internatlly by the database manager to get control over which ids are already in use.
find_next_available_id ()	After having found the used ids with the above method, the database manager calls this method each time it needs a new id. This happens when it wants to insert new persons into the database.
forget_expired_entries ()	This method checks the timestamp of the entries in the database and removes every entry that is too old, as specified by the age limit variable.

Method	Description
forget_person_by_id (id)	Takes in an id, and deletes the corresponding entry in the database.
store_person (img, person)	Takes the path to an image, and a Person object as parameters. Then a new entry (subdirectory) is made in the database, and the image and info about the person is stored.
get_person_by_id (id)	Retrieves and returns the name of the person with the given id in the database.
show_image (id)	Displays an image of the person with the specified id. Used only for demonstration purposes.
add_picture_of_person (img, id)	Takes the path to an image and an id as parameters. Creates a copy of the image found at the specified path and stores it in the database under the entry for the person with the specified id.
update_timestamp (id)	Updates the timestamp of the entry with the specified id to the current time. This method should be called by the database manager every time it calls a method that is considered as interaction with a person.
create_csv ()	Updates the .csv file to correctly represent the database. Should be called by the database manager every time it modifies images in the database. The .csv file contains entries with the paths to all the images in the database and the corresponding id for each image. The face recognizer relies on this file, and uses it to look up images for training and recognizing.
get_next_available_ img_number (dir_path)	Images for an entry are stored with the name NUM.IMG_FORMAT, where NUM is a number and IMG_FORMAT is the format of the image. The method takes the path to a directory as parameter. Then it looks through the specified directory to find the next avilable number. Since there is a maximum limit of images per person, this method might return -1 if it discovers that this limit is reached.

ROS service methods The remaining methods are used to handle ROS requests. They only call on the methods mentioned above and returns a status string, so they should be pretty self explanatory and are not discussed in more detail.

main() The main() method is run when the database manager is started as a ROS service. It checks that the base_path variable is set, crates an instance of the Database_manager class that can handle incoming requests, and initiates and publishes the offered services.

A.2 The face recognizer

The face recognizer is implemented in the facerec_server file. The facerec_client file contains some methods to connect the face recognizer with the database manager and sends inputs to the service that the facerec_server file is listening to. In the top of the main() function the user should specify where the database should be stored by editing the value of the 'csv_file_path' variable. This should be the same as in the database_manager file.
The .srv service description the communication between the facerec_server file and the facrec_client file.

facerec_server This is where the face recognition is done. In the face_rec function there is a variable called threshold, this is the value that needs to be changed if the threshold needs to be changed.

Code description of the face recognition server

Method	**Description**
read_csv (const string& filename, vector<Mat>& images, vector<int>& labels, char separator $= ';'$)	Reads the .csv file created by the database manager and creates two vectors. One with all the images from the database, and one with the corresponding labels.
face_rec(facerec::FaceRec::Request &req,facerec::FaceRec::Response &res)	Sets the threshold, trains the model with all the images from the database, then uses this to do the recognizing. Returns the label of the person recognized or -1 if no person is found.
main()	Is run when the facrec_server is started as a ROS service. It listens for requests, calls the read_csv function and the face_rec function and publishes the label from the face_rec function.

facerec_client This file is the interface between the user and the program. It also connects the database manger and the face recognition together.

Code description of the face recognition client

Method	**Description**
add_person_in_database(const string& path)	Sends a request to the database_manager to add a person, where the picture is located at the given path. It also asks for a name from the person that is being added.
show_image(int label)	Sends a request to the database_manager to show the image of a person with the given label.
get_person_name(int label)	Sends a request to the database_manager to give the name of the person of that given label.
crop_face(Mat frame,string image_path)	Takes the image in the given path, crops it and saves the new image in the same path.
main()	Is run when the face_client is started as ROS service. It calls the crop_face function, sends the cropped picture to the facerec_server as a request. If the label is -1 it asks for permission to add the person in the database, and calls the function add_person_in_database if the answer is yes. If a person is found, it calls the functions show_image and get_person_name and asks if it have found the correct person.