CSC720 Artificial Intelligence 2

Find Bahler

Leveraging COTS components to realize realtime facial detection and recognition in *Lego: Mindstorm NXT Robots*

Benjamin Wheeler and Chung-Wei Hang

Department of Computer Science

North Carolina State University

bmwheeler@ncsu.edu, chang@ncsu.edu

May 4, 2007

Abstract

We demonstrate the ability of combining facial detection techniques with a LEGO MINDSTORMS NXT robots using Commercial Off The Shelf (COTS) components, and show the feasibility of combining this facial recognition schemes, that given proper integration may allow such "toys" to identify people in realtime. We achieved realtime facial detection with between 5 and 9 fps, and discovered that while not extremely accurate, facial recognition may be applied to returned images to "sort-of" recognize individuals.

1 Introduction

Facial recognition has been an important application of pattern classification for decades. Many classification algorithms described in textbook have been used, for example, Principle Components Analysis (PCA) and Fisher's Linear Discriminant.

To recognize faces under variations, such as lighting and poses and facial expressions, is the most challenging issue in this area. In order to deal with such variations of faces, it is helpful to extract useful features from original samples, which means to reduce high-dimensional image space to a lower dimension feature face. One famous method is linear projection, to project away variations but maintain useful features. Eigenfaces[1], Fisherfaces[1] , and Laplacianfaces[2] are three well-known methods dealing with such variations.

However, it is not easy to apply facial recognition to real-world application, since, in real cases, machines do not have the ability to detect faces or humans from what they see. These facial recognition methods cannot be implemented if given samples are normal pictures rather than faces. What we need is a way to pick "faces out of a crowd." Such a method is described by Viola and Jones[5], who proposed an efficient method of facial detection for realtime applications.

Lego Mindstorms[3] were first released as the Robotics Invention System (RIS) by the Lego Corporation in partnership with the MIT Media Laboratory in 1998 as part of the Lego Educational Products Department. It was sold in both a commercial and educational model, included a GUI-based programming interface, and provided for one of the first times affordable, programmable, and highly adaptable robots to the general public. For years these "toys" have been used both for fun and education by children and adults, to learn, teach, and just have fun with robotics. At it's heart is an "intelligence brick", essentially a mini-computer with three sensor input ports and three motor output ports. The strength of the system relied on the ability to combine specialized parts with standard Lego components to make countless possibilities achievable using the RIS.

In August 2006 Lego released the next (NXT) generation of *Lego Mindstorms*[3], providing more functionality, a bigger, faster, more capable "intelligence brick", an open source standard, and USB 2.0, bluetooth compatibility. At it's heart is a 32 bit ARM7 main microproccessor (48 MHZ) and 8 bit Atmel AVR microcontroller (4MHZ), each with onboard flash memory and RAM. It currently supports 4 sensor input ports, and three output ports, with capability for extension[6]. Currently, the full capabilities of this new platform have yet to be fully realized, with future component releases planned to add even greater possibilities. To go along with the trend we wanted to examine the possibility of integrating this platform with facial detection and recognition routines. We were motivated to investigate this possibility by a project by the RoboRealm company, marketers or computer vision software for robotics, which provided a tutorial of a LEGO MINDSTORM NXT who could identify a ball, retrieve it, and drop it at a cone.[4]

2 Facial Recognition Algorithm: Fisherfaces

The challenge of facial recognition was to recognize the face from a general view point under different illumination conditions, facial expressions, and aging effects. The idea of Fisherfaces[1] was try to project away illumination conditions, facial expressions, and aging effects, by projecting image space into feature spaces. Fisherfaces could be considered as an extension and improvement of well-known facial detection algorithm—Eigenfaces[1].

2.1 Eigenfaces

Eigenfaces used the idea of Principle Component Analysis (PCA) as follows. Consider a set of N training images $\{x_1, x_2, \ldots, x_N\}$ each in a n-dimensional image space. Each image belongs to one of c classes $\{X_1, X_2, \ldots, X_c\}$. The idea was to find a linear transformation matrix mapping from original n-dimensional image space into a m-dimensional feature space, where $m < n$.

First we defined total scatter matrix S_T,

$$S_T = \sum_{k=1}^{N} (x_k - \mu)(x_k - \mu)^T$$

where N is the number of training images, and $\mu \in R^n$ is the mean image of all training images. Then we applied Singular Value Decomposition (SVD) to find the projection matrix W_{PCA} that maximize scatters in the projected space.

$$\begin{aligned} W_{PCA} &= \arg\max_W |W^T S_T W| \\ &= [w_1 w_2 \dots w_m] \end{aligned}$$

where $\{w_i | i = 1, 2, \dots, m\}$ is the set of n-dimensional eigenvectors of S_T corresponding to the m largest eigenvalues. Thus we could use the projection matrix $W_{PCA} \in R^{n \times m}$ to do linear transformation on each image, projecting from original n-dimension image space into new m-dimensional feature space by

$$x \to y = W_{PCA}^T x$$

Finally we used nearest neighbor classifier on feature space to classify our faces.

2.2 Fisherfaces

Fisherfaces was an extension of Eigenfaces. Fisherfaces used the similar idea of reducing features by transforming images into feature spaces. The difference between Eigenfaces and Fisherfaces was that, when transforming from image space to feature space, in Eigenfaces, we maximized projected total scatter, on the other hand, in Fisherfaces, we maximized the ratio of projected between-class to projected within-class scatter. In other words, Fisherfaces tried to maximize between-class scatter and minimize within-class scatter. Fisherfaces addressed the problem that largest variances might not be good discriminations, while small variances could still be important.

Again, consider a set of N training images $\{x_1, x_2, \dots, x_N\}$ each in a n-dimensional image space. Each image belongs to one of c classes $\{X_1, X_2, \dots, X_c\}$. Fisherfaces selected projection matrix that maximized the ratio of the between-class scatter to the within-class scatter. The between-class scatter is defined as

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

The within-lcass scatter is defined as

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

where μ_i is the mean image of class X_i, and N_i is the number of training images of class X_i. Then we calculated projection matrix W_{FLD} by

$$W_{FLD} = \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|}$$

We could find the final transformation matrix W by

$$W^T = W_{FLD}^T W_{PCA}^T$$

where $W_{FLD} \in R^{(N-c) \times m}$ and $W_{PCA} \in R^{n \times (N-c)}$. Then we could do linear transformation, projecting each image from original n-dimensional image space into m-dimensional feature space by

$$x \to y = W^T x$$

Finally, we used nearest neighbor classifier on feature space to find most probable person.

3 Facial Detection Algorithm

We used an OpenCV implementation of a facial detection algorithm proposed by Viola and Jones[5]. They proposed three major steps for facial detection. First, they transformed the bitmap images into a special format called integral images, which could help us extract features efficiently. Then they used AdaBoost to determine the weights of different features. These weights represented the ability of these features to distinguish faces from non-faces. Finally, they presented an idea of cascade classifier, which discarded non-faces sub-images as many as possible and as quickly as possible.

3.1 Features

There were three kinds of features we used in facial detection: two-rectangle feature, three-rectangle feature, and four-rectangle feature. Those features were the differences between the sum of the pixels within rectangular regions.

3.2 Integral Images

In order to compute the features described earlier rapidly, we used an intermediate representation for the image, called integral images. Each location x, y of an integral image could be represented as

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

where $ii(x, y)$ is the integral image and $i(x, y)$ is the original image. We could transform from original images to integral images by scanning through all pixels once. After transforming into integral images, we could calculate any rectangular sum in four array references.

3.3 AdaBoost

Given a feature and a training set of positive and negative images, we used a variant of AdaBoost to both select the features and train the classifier. AdaBoost did this by combining a collection of weak learners to form a stronger classifier. In our case, a learner is a classifier using one single feature. We called a learner weak if it could not classify the training set correctly. We applied the idea of AdaBoost to determine the weights of each learner, in order to find a weighted combination of learners. We defined learner by

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) < p\theta \\ 0 & \text{otherwise} \end{cases}$$

where f is the feature, θ is a threshold, and polarity p indicating the direction of the inequality.

3.4 Cascade Classifier

Finally, we found faces out of a picture by transforming it into integral image, feeding all features calculated from the integral image to our cascade classifier, which was a collection of classifiers trained by AdaBoost. We constructed this cascade classifier in order to save a large amount of time. The idea was we wanted to reject as many negative sub-images as possible, while detecting all positive instances. Simpler classifiers are used to reject the majority of sub-images before more complex classifiers are called to achieve low false positive rates.

We started with a two-feature strong classifier, obtained by adjusting the strong classifier threshold to minimize false negatives in AdaBoost. Then we reduced the number of sub-images with these operations

- Evaluate the rectangular features.
- Compute the weak classifier for each feature.
- Combine the weak classifiers.

The overall form of detection process was a degenerate decision tree. A positive result from one classifier would trigger the evaluation of next classifier. On the other hand, a negative outcome at any point immediately led to rejection of the sub-image. The cascade classifier would improve the performance of detection because there were always an overwhelming majority of sub-images were negative.

4 Software Components

4.1 OpenCV

OpenCV is an open source computer vision library released by *Intel*, by definition on *Intel*'s information page listed as "intended for use, incorporation and modification by researchers, commercial software developers, government and camera vendors." It is freely available library including a large number of Computer Vision and Image Processing functions. Versions are currently available for C++ and Python, both begin downloadable at http://4sourceforge.net/projects/opencvlibrary, with documentation and example usages available at http://opencvlibrary.sourceforge.net.

For this project we made use of their implemenation of Viola and Jone's Facial Detection algorithm, along with their pre-trained cascade classifier for frontal face detection. Also available, but not tested is a pre-trained cascade classifier for profile face detection.

4.2 VideoInput

To allow or program to grab image data directly from the camera in realtime we used the open source VideoInput library written by Theodore Watson available at, http://muonics.net/school/spring05/videoInput/. This software provides tools to automatically, or manually, recognize and connect to USB digital video input devices, allowing you to invoke a grabframe(· function to capture an image from the source, as well as tools to recognize new frames for capture, ect.

4.3 Lego Mindstorm NXT Interface Library

Written by Daniel Berger, this library provides rudimentary control and feedback between a C++ program and the NXT Intelligence brick via a Bluetooth interface. Current functionality allows on/off and speed control of motors, and access to all sensor inputs. Currently unsupported features include fine motor control, sound capability, and the ability to execute some of the more dynamic functions on the NXT Intelligence Brick, such as uploading programs.

At the time of this writing a different Library is currently under development at http://nxtpp.sourceforge.net/ which plans to allow control over the complete functionality of the NXT Intelligence Brick. The latest version, 0.3 released on 05-02-07, does supports fine motor control, but limited sensor input.

5 Hardware Components

5.1 LEGO MINDSTORMS NXT set #8527

Retail: 250 USD
Available from `www.Amazon.com`

The LEGO MINDSTORMS NXT set #8527 is manufactured by the Lego Corporation, and readily available from a variety distributors. Ours was purchased from the online superstore *Amazon.com*. The kit consists of regular LEGO TECHNIC bricks and specialized MINDSTORM NXT components, including the Intelligence Brick, Sensors, Motors, and connecting cables.

The NXT Intelligence brick acts as the "Brain" of the robot, and may be described by the following technical specifications,

- 32-bit ARM7 microcontroller
- 256 Kbytes FLASH, 64 Kbytes RAM
- 8-bit AVR microcontroller
- 4 Kbytes FLASH, 512 Byte RAM
- Bluetooth wireless communication (Bluetooth Class II V2.0 Compliant)
- USB full speed port (12 Mbits/s)
- 4 input ports, 6-wire cable digital platform w/ future expansion capability
- 3 output ports, 6-wire cable digital platform
- 100x64 pixel LCD graphical display
- Loadspeaker - 8kHz sound quality. 8-bit resolution sound channel with 2-16KHz sample rate.
- Power source: 6 AA batteries

- *specifications from LEGO MINDSTORMS User Guide*

Included with the kit are 3 motors, a sound sensor, a touch sensor, a light sensor, and a ultra-sonic sensor. Additional information and links to other MINDSTORM NXT projects and applications may be found by visiting `http://mindstorms.lego.com/`.

5.2 Camera/Reciever

Retail: 50 USD
Available at `www.raidentech.com`

We used a .25 inch active-pixel sensor, commonly known a CMOS sensor, purchased from *RadienTech Inc.* under the heading *Wireless Waterproof Night Vision Color Camera*.

- Resolution: 420 TV lines → 320x240 input image
- Power Source: 12v @ 200mAmp wired power supply, or 9-volt battery
- 1.2 GHZ reciever

It also featured 12 built in IR LED lights to realized night vision capability, which would automatically turn on when the ambient light level dropped below 3 LUX. The 1.2 GHZ receiver captured the wireless video signal, providing our analog video in component.

5.3 Digitizer

Retail: 50 USD
Available at `www.ambientweather.com`

To transform this analog video component into something our software could interpret we used a Video to USB Converter manufactured by *Hauppauge*, marketed under the name USB Live. It is available for purchase directly from the manufacture, or a number of distributors. This proved the most difficult item for us to locate, finally finding it in-stock at the online store *Ambient Weather*. It takes a component video or S-video signal, processing it into a digital image and connecting to the computer via a USB 2 interface. Using this interface it is capable or delivering frames at more than 30fps. Included software allows you to watch TV on your computer, capture video and still images, and export them into a variety of still and motion file formats. Using a open source C++ Library, *VideoInput*, we were able to grab this input for use in our program.

5.4 Bluetooth Dongle

Retail: 20 USD
Available at `www.tigerdirect.com`

The computer communicates with the LEGO MINDSTORMS NXT Intelligence Brick via

a Bluetooth connection. We used a Class II Bluetooth dongle purchased from *TigerDirect*. It was manufactured by the IVT Corporation, and connected to the computer via USB 2 interface. We used included IVT Corporation BlueSoleil software to manage and control the device.

5.5 Computer Base Station

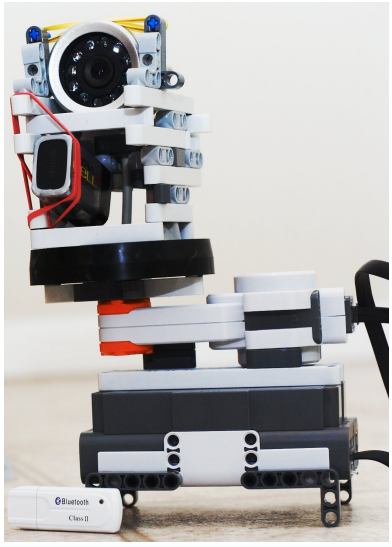
The computer base station was an HP Pavilion with AMD Athlon 64 X2 Dual Core Processor 4200+, clocked at 1GHZ, with 1.5 GB of RAM memory without separate graphics card support.

6 Implementation

Robot Design

As a proof of concept to test the feasibility of implementing facial recognition and detection with the LEGO MINDSTORM NXT we settled on a simple tower design with allowing one degree of freedom, the movement of a turret. The simplified construction reduced control complexity, and allowed for a relatively modular model allowing easy adaptation. The major limiting factors in construction were the relatively low torque of the motors, and a relatively heavy camera.

Our original choice was to construct a turret with 2 degrees of freedom to allow both a horizontal and vertical search field, but it was found NXT motors, while rated at an unloaded 350 rpm, fail to generate enough torque to allow for construction of a turret with a payload capacity sufficient to support the mechanical element required to deliver this performance with our equipment. Specifically, the combined weight of an NXT motor (for the second degree of freedom) and our camera proved too much for a single NXT motor to move in either the horizontal or vertical rotational plane. It is noted two motors combined may have been sufficient to allow horizontal rotation, but without fine motor control synchronization of such a system was infeasible. However, with only three motors currently available, a way of increasing this number is planned with release of a future product, this proves one of the major design challenges with MINDSTORM NXT set. Additionally, the use of a pinhole type "spy" camera would have allowed two degrees of freedom to be obtained, sufficiently reducing the weight to allow the greater functionality. A picture of the completed robot may be found in figure 1.

Robot.jpg 

Software/Hardware Integration

We programmed our application in C++ using Microsoft Visual Studio 5.0, using the GLUT package for OpenGL to render the current view of the Robot, the user interface, and display facial detection results. Essentially, or hardware software loop began with the Robot mounted camera, which sent an analog signal wirelessly to the 1.2 GHZ receiver. This analog signal was turned into digital signal by the USB Live Digitizer, and connected to the computer base station via a USB connection. This connection was tapped by the VideoInput library to provide frame-by-frame images to OpenCV's facial detection routine. At this point we intended to incorporate the faces returned from this OpenCV function to our facial recognition algorithm. However, porting this MatLab function to Visual Studio proved more difficult than originally anticipated and consequentially was not incorporated. At this point, we send new instructions to the NXT Intelligence Brick, using Daniel Berger's code to send instructions via a preestablished bluetooth connection using a USB bluetooth dongle. Our Hardware/Software Interface loop was thus,

Wireless Camera → Receiver → Digitizer → Computer Base Station → VideoInput Library Functions → OpenCV Library Functions → Robot Action Determination Code → NXT Interface Library Code → Bluetooth Dongle → NXT Intelligence Brick

Loop.jpg

The Robot Action Control Program

To show a viable implementation of facial detection incorporating a LEGO MINDSTORMS NXT robot, we determined the robot must not only feed video input to the computer base station, but execute action based on results from computation of said input. To achieve this, we treated the robot as a simple state machine that at any one time would be in one of four "modes",

Relaxed Robot is immobile, sending video feed, but taking no action.

Simple Search Robot actively rotates turret until a face is detected, then stops and reverts to Relaxed mode.

Persistent Search Robot actively rotates turret, continuing to naively search for more faces after the first. It continues indefinitely until user intervention.

Intelligent Search Robot's goal is to find and maintain a "facial lock" on one or more faces. It will begin by rotating the turret in naively with a one percent probability of changing turret rotational direction until a face it found. It then attempts to center the identified face by moving the turret in the appropriate direction, or stopping if the face is in the center of image. If contact is lost, it randomly chooses a direction to continue search.

The program interacts with the user through an OpenGL mouse menu, where the user is presented a dropdown menu with the option to turn facial detection on/off, turn facial image capture on/off (save the "found" faces to the program directory as consecutively named .bmp images, and the option to put the Robot in any of the above four states.

Results

6.1 Experimental Results of Facial Recognition Implementation

There were three sets of faces used in all experiments for facial recognition, training, gallery, and probe. Training faces were used for determining model parameters. We applied Fisherfaces algorithm to training faces, in order to calculate the transformation matrix W described in previous section, which was the product of project matrix W_{PCA} and W_{FLD}. Gallery faces were people who we really wanted to recognize. The number of faces for each person in gallery did not need to be as many as training, but, however, the number of faces for each person in gallery should be the same. The images in probe were the faces we wanted to recognize. The recognition system would use nearest neighbor classifier described in previous system to find the most probable person in gallery.

Training CMU PIE face database contains 66 different people. For each person, there were two different poses and 21 different light conditions. We used 56 people in CMU PIE face database. The other ten people were used for gallery and probe.

Gallery and Probe There were seven people in our gallery and probe. Two of them were from CMU PIE face database. The face images of the other people were taken using MacBook Pro built-in camera. There were four faces for each person with two different poses and slight different light conditions. We used two faces in gallery and two faces in probe.

We did many experiments with different combination of galleries, but with the same training faces from CMU PIE face database. Unfortunately, the error rate of all experiments was really high, over 90%. This error rate made our recognition not reasonable for practical use. However, we still found some interesting results.

- The combination of galleries could affect the error rate very much. If we used faces with similar light conditions and directions for all people in gallery, the error rate would be much lower. This showed that when we trained parameters of Fisherfaces, it was necessary to use faces of different people with similar light and direction conditions, to help Fisherfaces identify which features were discriminative and which were not.

- The light and direction conditions of faces in gallery should be similar to those in training faces. Because Fisherfaces would try to find best feature space by maximizing the differences between classes and minimizing the differences within classes of training faces. If the light and direction conditions of gallery faces were totally different, the feature space with respect to projection matrix would not be the most discriminative to classify these gallery faces.

- Fisherfaces was a practical algorithm for real-time application. Both training and probing were fast. It took at most 20 minutes to train over 2000 training on a normal computer. Probing took less than a second. The whole Fisherfaces was implemented in Matlab.

In general, Fisherfaces worked extremely well with CMU PIE face database, but it worked not well with practical faces captured by a camera or webcam, as long as the light and direction conditions of those faces were not too similar to the training set.

Unfortunately, we were unable to port this application from MatLab to C++, as originally anticipated, preventing us from properly implementing this feature into the Robot.

6.2 Experimental results with Facial Detection and Integration with LEGO Robot

Testing the OpenCV implementation independent of the robot on random images showed very good accuracy and fast detection at images up to 3000x3000 pixels, with an extremely low false positive rate on images with sufficient resolution. When combined with our camera, operating at 320x240 resolution, results were noticeably degraded, but encouraging. Operating at speeds of between 5 and 9 fps, it would pick up faces as small as 32x32 in the image. However, it had a tendency to misclassify objects with vertical or horizontal strips of alternate colors, i.e. a plaid couch. Even when image quality was extremely poor, it maintained a low false positive rate. Lack of fine motor control made precise action in response facial detection information difficult, resulting in our "intelligent search" mode performing more like a "drunken search", but seems to show promise given future interfaces with decreased granularity of control.

References

[1] Peter N. Belhumeur, Jo ao P. Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7), July 1997.

[2] Xiaofei He, Shuicheng Yan, Yuxiao Hu, Partha Niyogi, and Hong-Jiang Zhang. Face recognition using laplacianfaces. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 27(3), March 2005.

[3] Lego. Whats nxt? lego group unveils lego mindstorms nxt robotics toolset at consumer electronics show. http://www.lego.com.

[4] RoboRealm. Adding vision to the lego nxt. http://www.roborealm.com/tutorial/ball_picker/slide010.php.

[5] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2), 2004.

[6] Wikipedia. Lego mindstorm. http://en.wikipedia.org/wiki/Lego_mindstorm.