Head pose estimation

Jonas Vanthornhout

University of Leuven

Contents

1 Introduction

Head pose estimation in computer vision means that we try to estimate the position of a (human) head using algorithms that run on a computer.

The position of the head is useful in a couple ways. For example, given an image we can detect who is talking to whom using the position of the heads. This is based on the fact that people who talk to each other have a mutual gaze.

Also, it can help robots to understand humans. Humans tend to make movements with their head when they are talking. A good interpretation of these movements can make the robot more accurate.

Also a good head pose estimator helps computers wit extracting the essential information out of a big bunch of data. A human is very good is abstracting the useful information. When a computer detects that a human is looking at something, the computer can scan that part of the area to detect if something is happening there. The computer doesn't have to scan the other part of the area, which results in a big speed up. This concept of gaze following can also be used the other way around. For example safety systems in cars can warn the driver when he isn't looking at the road.

In this paper we use a known algorithm to estimate to pose of one's head, more specifically the pitch and yaw. First a couple of known algorithms are discussed. Then an algorithm is chosen and implemented. On the basis of the results we review the algorithm.

2 Possible methods

First we describe the current methods used for head pose estimation. This Section is a summary of "Head Pose Estimation in Computer Vision: A Survey" [5]. The main idea, advantages and disadvantages of the methods are described.

2.1 Appearance template methods

This method tries to match a given head to a set of heads with a know pose (the templates). The best match determines the pose of the head. This method, although fairly simple has couple of advantages. The set of templates can be expanded at any time. There isn't need for feature detection, this helps to keep the method simple. Unfortunately the appearance template method has also a couple of disadvantages. This method can only detect discrete pose locations. Also the head region has to be detected and localized beforehand. When there are a lot of templates the efficiency degrades, this can be solved by using SVMs. But the main problem with this method is that it is very sensitive for pairwise similarity. This means that the estimator gives the wrong pose because there is a very similar image in the database, but this image has a different pose.

2.2 Detector arrays

A detector array is a bit similar to the appearance template method. But in this method a detector for each pose is made. This solves the main problem of the appearance template method. Namely the sensitivity for pairwise similarity, this is done because this method only learns the appearance variation corresponding to the pose change. In a similar way detector arrays can do head detection, although negative nonface examples are needed. Because this method introduces a detector for each pose, the training phase is a bit more difficult. Also the number of detectors should be limited and the positive examples of one detector should be negative expamples for the other detectors.

2.3 Nonlinear regression methods

This method tries to calculate a value of the image and then uses regression methods to estimate the head pose. Because an image has a high dimensionality, the value of the image can have high dimensionality and regression in a high dimension isn't a trivial task. This can be solved by using the features of the image. This extension requires a feature detection algorithm. Nonlinear regression methods are mostly used in combination with neural networks. The resulting algorithm is then very fast and accurate. But the algorithm is still dependent on good (consistent) head localization.

2.4 Manifold embedding methods

Manifold embedding methods try to reduce the dimensionality of the image. The idea behind this approach is the following. Only a couple of dimensions are responsible for the the estimation of the head pose. These dimensions are then used to calculate the head pose. The calculation can be done with regression or template mathching. But the main problem is the reduction of the dimension. Although a human is very good in neglecting useless information, it is a nontrivial task for a computer. The reduction of the dimensionality can be done in various ways. Some of them do this in a unsupervised way. This can lead to wrong head pose estimations, because the identity of the person can be used. Other methods define eigenspaces and project the test image into the eigenspaces. The distribution of the energy over these eigenspaces is an indicator how good that eigenspace is in representing the test image.

2.5 Flexible models

The main idea of this method is to map a known model to an image. A model is graph with its nodes located on the features of the head. Because no two heads are equal, the fixation must be done in a flexible way, hence the name flexible model. A problem with this method is that some feature detector is needed. Also we must be able to overlay the graph. If some nodes of the graph doesn't have corresponding features, we need to adapt the algorithm. Another problem is that comparing the graphs is computationally expensive. This method has as main advantage that it is robust to deformation of faces.

2.6 Geometric methods

This method is based on the results of psychophysical experiments. As humans we are very good in abstracting information to guess the correct head pose. We can do this because we know which features we need to extract and what the normal ratios of these dimensions are. Geometric methods try to insert these knowledge into algorithms. First a couple of features need to be detected. Which features depends on the knowledge we want to implement. Then the ratios of distances between them are calculated (or we apply some other mathematical function). These results are compared with the results obtained in the training phase. The most similar then represents the head pose. Of course this method has the same problems as the human way of estimating the head pose. That is we can't predict the head pose with high accuracy. Also most of the known geometric methods only work well with frontal views. This is because they don't know how to adapt their feature detector. For example searching for the distance between two eyes in a profile view isn't the smartest thing to do.

2.7 Tracking methods

This method requires a stream of consuctive images. When we use a stream we have more information than just using single images. The head pose estimation is now divided into two subproblems. First we need to estimate the head pose for the first image of the sequence, this must be done with a "standard" head pose estimator. Then we need to estimate the pose for the other images in the stream. These images can use the estimation of the previous images. The way the pose of the following frames are detected are vast. For example, some features can tracked. When these feature move left we know the heads moves to the left side. Another method is to use texture mapping on a 3D model.

2.8 Hybrid methods

This method combines some of the previous method to one, hopefully better, method. The start of the tracking method is already a hybrid method. Because the first pose must be estimated in the first frame. The other frames can use the head pose estimation of the previous frames. Another way of combining the methods, is just to run both methods and take the average of the estimation. If the methods are able to give a weight to their average it is possible to calculate the weighted average and gain a better accuracy than the original methods. This works good for methods that have advantages and disadvantages that are mutually exclusive. For example some algorithms work good with far-fielded imagery while other methods work good for near-fielded imagery. When we combine these two algorithms, we have an algorithm that works in all cases.

3 Chosen implementation

The method I've chosen is the geometric method described in Subsection 2.6. I've chosen this method because of its simplicity. It just needs the position of some features. When this is known it's just some simple calculations. No complicated machine learning techniques like artificial neural networks or support vector machine are needed. This makes the algorithm very easy to understand. Also it is easy to interpret why the method predicts a certain head pose, the algorithm is thus a glassbox. Besides the algorithm is very pluggable, it is just an encoding of some information that we (humans) know.

The basis of the implemented algorithm is described in [3], but I've decided to strip down the algorithm to its essential parts. The algorithm of Gee et al. doesn't need a training phase. We can use the Bosphorus

database [7], thus we can use a training phase. This should make the algorithm more stable, because it can anticipate to noise.

3.1 Yaw

3.1.1 Rationale

The yaw detector is based on the horizontal position of the eyes in the image. First the eyes in the image are detected. Then the relative position of the eyes in the image calculated. Finally the average horizontal position is calculated. This value represent the yaw of the head. The reasoning is as follows, when the head looks right both eyes turn to the right side, when the head looks left both eyes turn to the left side. When the image is a profile view, we can only see one eye. This is extra information we can exploit.

Training phase In the training phase, the algorithm calculates the horizontal positions of the eyes and stores those values. In some cases it is possible that only one eye is detected. This can happen when the head is turned 90 degrees. The position of the undetected eye is then 0. Because it matters in what order we store the positions, we must be consistent. Therefore we try to detect which eye is the left one and which is the right one. We use the convention that we first store the position of the right eye and then the position of the left eye.

Test phase In the test phase, the position of the eyes of the test image are calculated. Every image from the training phase that doesn't have the same amount of eyes as the test image gets a very high score, which is bad. When the image has two eyes, the average position is calculated. Every image from the training phase then gets a score that is equal to the absolute difference of the averages. If way can only detect one eye in the image, we first look if we are comparing the same eye. If this isn't the fact the training image gets a very high score. When we are talking about the same eye, the score is the absolute difference of the positions of the eyes. The yaw of the image with the lowest score is the estimated yaw. When there are multiple images with the same lowest score, the estimation is the most common yaw in this set of the"best" images.

Assumptions This method makes a couple of assumptions, the position of the head in the image must be consistent. Because the head pose estimation problem is actually transformed into a feature detection problem, some algorithm must be available/developed to detect the eyes in the image. We also need a head localizator to know where the head is in the image.

3.1.2 Implementation

Training phase First the algorithm needs to be trained. In this phase the eyes of the head are detected. To do this the image is resized to 20% its original size. Then a cascade classifier [4] tries to detect the eyes in this image. When this classifier detects more than two eyes, the minimum amount of neighbours of the cascade classifier is increased. This restricts the classifier, so less eyes should be detected. We do this until maximum two eyes are remaining. When there are exactly two eyes in the image, the algorithm checks if the eyes are approximately at the same height. When one eye is completely under the other, the lowest is removed. This makes sense because eyes are located in the upper part of the head. When there are still two eyes, the algorithm checks if the eyes don't overlap. When the eyes overlap the eye with the biggest area[1] is removed. When no eyes are detected, the original image is rescaled to 30% its size and the algorithm is rerun, this done iteratively in steps of 10% until at least one eye is detected or the image is bigger than its original size. This way we gain a huge speed up when high resultion images are fed into the system. Also resizing the image helps the cascade classifier. For example freckles can be detected as an eye in a high resolution image. Finally the relative positions of the eyes are calculated. This is just calculating the midpoint along the x-axis of the rectangle that represents the eye and dividing this point by the width of the image. To avoid that we have to train the algorithm when we want to estimate the pose of a head, we store these values. Because we have two

[1] The cascade classifier returns a rectangle where the eye is detected, so it makes sense to talk about the size of the "eye".

values of each head, we need to impose an order on how to serialize them. Some possibilities are: first the biggest value and then the smallest value or first the left eye and then the right eye. The order has little impact on the accuracy and the most important thing is that the order must be consistent. Our implementation chooses the latter method because this is the most logical way and performs better (the other method is mentioned in the results). After this step the algorithm is ready to predict the yaw.

Test phase The prediction begins in the same way as the training, the relative position of the eyes in the test image are calculated. Then the positions of the test image are compared with the positions calculated in the training phase. Every image from the training phase gets a score, a low score is very good (with zero as the best score) and a high score is bad (with one million as the worst score). When the amount of eyes differ, the score is directly set to one million. When both images don't have eyes, the score is set to zero. When there is one eye in both images the score is the absolute value of the difference of the position of the eyes if both eyes are left or right eyes. When a left eye is compared with a right eye the score is one million. The case with two eyes is similar but here the position of both eyes are added before calculating the difference. We now have for every image in the training phase a score. It is possible that some image have the same lowest score. The yaw of this image is the detected yaw. When we have a tie, we calculate the most common yaw and this is the estimated yaw.

3.1.3 Results

To review an algorithm we must have some measurements to compare it with other algorithms. We've chosen four measurements.

correct The amount of yaws that are correct estimated.

accuracy The percentage of yaws that are correct estimated. The is relative version of the "correct" measurement.

average absolute error The absolute value of the difference between the detected yaw and the real yaw is calculated. This value is accumulated for every test image. Finally this value is divided by the number of images in the test set.

wrong direction The number of times the algorithm predicts a right sided yaw while the real yaw is left sided or vice versa. This measurement is important when the side is important. For example when an algorithm tries to detect if two people are talking to each other, an important property is that they are looking at each other. When the yaw detection detects a yaw that is completely at the other direction, the algorithm that uses this pose can have problems. When using this measurement we have to determine what we want to do with the frontal view. We decided to show the amount of misclassifications in the frontal view case as an apart measurement.

The results are presented in table 1, the results are obtained using 20-fold cross validation on the Bosphorus database [7]. It is interesting to compare the results with a method that uses the given landmarks. For the method that doesn't make a distinction between the left and right eye, we see that the results that use the given landmarks are approximately the same use the method that does all the detection itself (method "detect (sorted)" and "landmarks (sorted)" of the table). We can conclude this method has reached its limit. But when the positions of the eyes are stored corresponding to their relative position instead of first the biggest value. We can increase the accuracy to 93.8% (method "landmarks"). Thus it is still possible to improve the algorithm. But we need to extract more information from the image. We need to know which eye is the right and which is the left. When we have two eyes, this is already done. When we can only detect one eye we need to look at other features. For example we can look at the nose. When the detected eye is at the left side of the nose we've detected the left eye and vice versa. Experimental results showed that this is more difficult then it sounds. Nose detection with a cascade classifier is difficult when the head is in profile. Therefore a more easy way is used. When the position of the eye is in the first 50% the position is placed first. When the position of the eye is in the other half, the position is stored as the second value. This method has an accuracy of 86.8% (method

"detect"). We see that this result is worse than the method that uses the landmarks. We can now say that the limiting factor is the feature detector.

method	correct	accuracy	avg. abs. err.[a]	wrong direction	wrong frontal
detect (sorted)	209	86.0%	6.3	4	9
landmarks (sorted)	214	88.1%	8.4	15	0
detect	211	86.8%	5.6	2	9
landmarks	228	93.8%	0.84	0	0
hybrid	183	75.3%	10.3	4	13
landmarks (logical)	212	87.2%	3.4	0	0
hybrid (logical)	191	78.6%	9.3	4	13

[a] average absolute error

Table 1: The results of the yaw detector using cross validation on the Bosphorus database. There are 243 samples.

When the given landmarks are used, the yaw detector predicts the yaw of 93.8% of the images correct. A natural extension is to make a hybrid. We can use the landmarks in the training phase and detect the needed features in the test phase. These results aren't very good (method "hybrid"). This is because the positions of the eyes are stored in a different way. When one of the landmarks of an eye is missing (either the inner corner or the outer corner) this missing value is interpreted as a zero value. This has the effect that the calculated position of the eye can be totally different than the real position. This isn't a very big problem if these values don't conflict with values from an other yaw. To counter this problem, a version that solves this problem is made. The hybrid is then a little bit better (method "hybrid (logical)"). But the method that only uses the landmarks makes a drop in accuracy (method "landmarks (logical)"). Apparently the miscalculation makes the algorithm perform better. Both hybrid methods perform less than the detect method. To counter this problem the landmarks and the detection should be calibrated but this is a more though problem.

3.1.4 Possible improvements & modifications

The current algorithm works only with discrete poses. But the algorithm can be adapted to predict a continuous pose. For example we can use the five best matching images and let their score influence the detected yaw.

As mentioned above can use a nose detector the analyze if an eye is a right or left eye. This will improve the algorithm, as the algorithm classifies some left eyes as right eyes and vice versa.

3.2 Pitch

3.2.1 Rationale

The idea of the pitch detector is described in [3]. First some features need to be detected. The corner of the left and right eye, the nose and the left/right corner of the mouth. Gee et al. [3] make a distinction between the nose tip and the nose base. Because detection of the nose base and nose tip are rather difficult this distinction is neglected. This has implications that the pitch can't be detected accurate anymore. But it is still possible to use the ratio $\frac{d(mouth,nose)}{d(nose,eye)}$. We will use this ratio to detect the pitch.

Training phase The mouth, nose and eye are detected with a cascade classifier [4] [2]. Then the average height of of the eyes and mouth are calculated. Finally the distance of the mouth to the nose and the distance from the nose to the eye are calculated. Those two distances are divided by each other, this number represents the pitch of the head. The reasoning behind this approach is, when the person has a "zero" pitch, i.e. the person looks straight into the camera, those two distances have a "known" ratio. When the person looks up, the image shows a bigger distance from the mouth to the nose than from the nose to the eyes. Thus the ratio increases. When the person looks down the ratio decreases. We use these variations to estimate the pitch.

Test phase When we must estimate the pitch of an image the ratio of this image is calculated. Because all the ratios are fairly close to each other a fuzzy parameter is introduced. When an image of the trainings phase has a ratio that is close to the new ratio it can vote for his pitch. How close the ratio must be is determined by the fuzzy parameter. When all images of the trainings phase are compared with the test image, every pitch has a couple of votes and the pitch with the most votes is chosen.

Assumptions This method makes a couple of assumptions, the features eye, nose and mouth must be detectable. So a frontal view is needed. The method also asumes that the ratio is the same for every person. In general this isn't true. An important assumption that Gee et al. made is that the length of the nose was known. This is a distance that we can only measure very well with a profile view. Our algorithm doesn't need this assumption, this makes the algorithm more applicable.

3.2.2 Implementation

Training phase First we need to train the algorithm. In this phase we detect the eyes, mouth and nose. The detection of the eyes is done in the same way as described in Subsubsection 3.1.2. The detection of the mouth and nose is similar. For the nose, the image is resized to 40% its size. Then a cascade classifier [2] tries to detect one nose in the image. If more than one image is found, the amount of minimum neighbours is raised until maximum one nose is found. When no nose can be found the image is rescaled to 50% its original size. This is done till a nose is found or the image will be upscaled. The detection of the mouth is a bit more extensive. Again we use a scaled down version of the image (40% is a good start value). Because experiments showed that the mouth is a bit more complicated to find, we need to do more than just increasing the number of of minimum neighbours. First we detect every possible mouth then we delete every "mouth" that doesn't have a point in the lower quarter of the image. Then we compare every possible mouth with another possible mouth. When these two mouths overlap[2] We delete the smallest mouth. This way we can deal fairly well with false positives and we get the correct position of the mouth. When we know all these features, we calculate the vertical distance from the mouth to the nose and the vertical distance from the nose to the eyes.

Test phase The prediction starts in the same way as the training, the ratio of the distance from the mouth to the nose and the distance from the nose to the eye is calculated. This value is compared with the trained values. When the absolute difference between the test value and the trained value is lower than a certain threshold, the trained value votes for its class. In constrast to the yaw detector there is only one possible vote (you vote or you don't). The class with the most votes is the class of the detected pitch. But it is possible that two classes have the same amount of votes, when this happens we lower the threshold until it is zero or lower. When there are still classes with the same amount of votes, we raise the threshold. So in the limit the most common class is predicted.

3.2.3 Results

This method uses the same measurements as described in Subsubsection 3.1.3. But we have to define the average absolute error in another way, because we work with categorical data. Here is the new definition of the average absolute error.

average absolute error Every pitch type is mapped to an integer (U $\rightarrow$ 2, SU $\rightarrow$ 1, N $\rightarrow$ 0, SD $\rightarrow$ -1, D $\rightarrow$ -2). The absolute value of the difference between the value of the detected pitch and the value of the real pitch is calculated. This value is accumulated for every image. Finally this value is divided by the number of detections.

The results are presented in Table 2. They are obtained by using 20-fold cross validation on the Bosphorus database. Only the images that are annotated for pitch and the neutral image are used. Because the Bosphorus database also contains landmarks, we can implement the algorithm described in Gee et al. The results of this implementation are alse presented in the following table. We see that this implementation has similar results as our implementation. The algorithm that does all the detection

[2]The cascade classifier returns a rectangle where the mouth is detected, so it is straightforward to check if two mouths overlap.

itself has an accuracy of 68.3% (method "detect"). The version that uses the landmarks has the same accuracy but can better detect the neutral pose (method "landmarks").

method	**correct**	**accuracy**	**avg. abs. err.**[a]	**wrong direction**	**wrong neutral**
detect	71	68.3%	0.35	0	20
landmarks	71	68.3%	0.35	1	16
facial normal	51	49.0%	1.1	27	22
facial normal (corr.)	74	71.2%	0.29	0	14

[a] average absolute error

Table 2: The results of the pitch detector using cross validation on the Bosphorus database. There are 104 samples.

Although Gee et al. doesn't provide comparable results, we can compare our results with an implementation of their algorithm. The implementation is based on the code of [1]. A couple of remarks should be made. The algorithm uses the length of the nose, notated as L_n. To calculate this variable we need the profile view of the faces. Also the calculated angles don't show a lot of difference between upwards and downwards looking faces. Therefore we made sure that the faces looking downward had negative values. This steps assumes that we have more information. When this information wasn't available the algorithm performed very poor (method "facial normal"). When this information is available the algorithm outperformed our algorithm (method "facial normal (corr.)"). This is logical because they use more information. However they don't win by a huge margin. There are two possible reasons. The implementation makes some mistakes or the method is inherently weak.

The reason for the weak accuracy is shown in Figure 1 for our own detector and in Figure 2 when the landmarks are used. We see that there is a lot of overlap in the classes. As comparison the data of the best method is shown in Figure 3. Although there is still overlap, we can clearly distinct the classes.

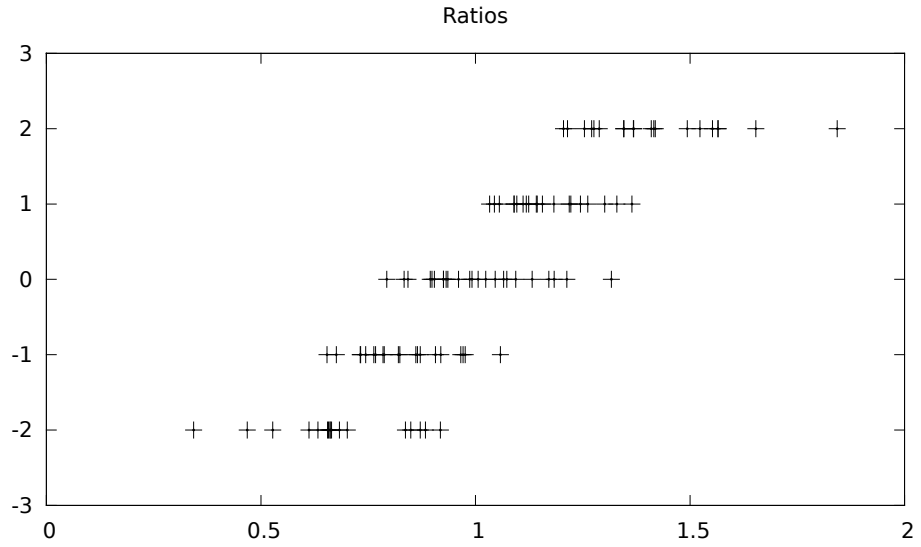

Figure 1: The ratios (x-axis) for the pitches (y-axis) using the cascade classifier.

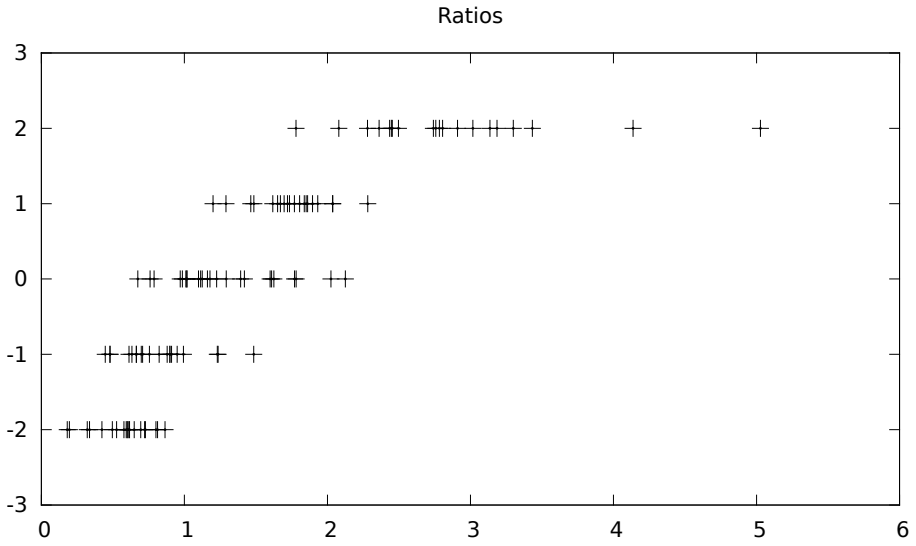

Figure 2: The ratios (x-axis) for the pitches (y-axis) using the landmarks.

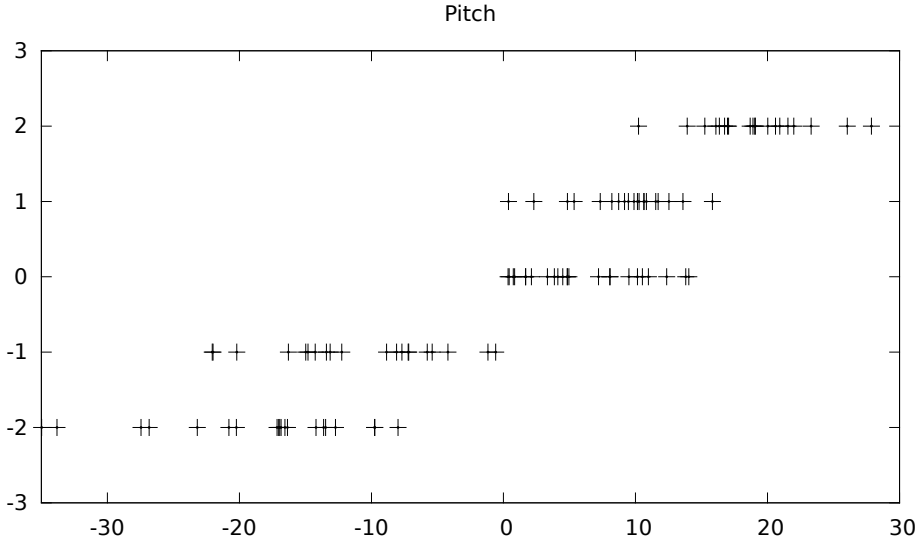

Figure 3: The ratios (x-axis) for the pitches (y-axis) using the facial normal.

3.2.4 Possible improvements & modifications

Although the dataset works with categorical data, we can estimate a continuous pitch. We can use the amount of votes to measure the influence on the pitch. We can also normalize the ratios. If we somehow know what the ratio is in a neutral front view we can calibrate the ratios. This approach needs to know what the neutral case is. Another way to calibrate is to make sure that the distance from the mouth to the eye divided by the face length follows a normal distribution. Normally this is true [6], but we can have outliers (perhaps because of some mismeasurements) and we can make sure these outliers doesn't influence the other results too much.

4 Conclusion

As presented in the results, we see that yaw detection is easier than pitch detection.

The yaw detector gives fairly good results. The chosen method, although very simple can have a high accuracy bit it needs the correct location of the needed features, i.e. the eyes. The limiting factor is the dection of these features.

The pitch detector has worse results. Here the accuracy isn't limited by the correctness of the location of the features. When we compare the results of the pitch detector with the results of a similar algorithm, we see that our algorithm isn't that bad. When we want higher accuracies we need to use other algorithms or we need to implement more knowledge in the algorithm.

References

[1] Oljira Dejene Boru. Head pose estimation using opencv. http://mmlab.disi.unitn.it/wiki/index.php/Head_Pose_Estimation_using_OpenCV.

[2] M. Castrillón Santana, O. Déniz Suárez, M. Hernández Tejera, and C. Guerra Artal. Encara2: Real-time detection of multiple faces at different resolutions in video streams. *Journal of Visual Communication and Image Representation*, pages 130–140, April 2007.

[3] A. H. Gee and R. Cipolla. Determining the gaze of faces in images. *Image and Vision Computing*, 12:639–647, 1994.

[4] Shameem Hameed. http://www-personal.umich.edu/~shameem/haarcascade_eye.html.

[5] E. Murphy-Chutorian and M. Trivedi. Head pose estimation in computer vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31 Issue 4(Pages 607-626), 2009.

[6] Pamela M. Pallett, Stephen Link, and Kang Lee. New "golden" ratios for facial beauty. *Vision Research*, 50(2):149 – 154, 2010.

[7] A. Savran, N. Alyüz, H. Dibeklioğlu, O. Çeliktutan, B. Gökberk, B. Sankur, and L. Akarun. Bosphorus database for 3d face analysis. In *The First COST 2101 Workshop on Biometrics and Identity Management (BIOID 2008)*, May 2008.