Mapping Facial Movements to Robot Face

Miguel Salcedo, Undergraduate Student at Vassar College Class of 2014

Introduction:

As machines begin to occupy more aspects of our lives, it is sometimes necessary to have machines that mimic human-like facial expressions. Such applications include interacting with children in autism research, or testing certain aspects of human-human interaction through recreating the interaction using a robot (i.e., medical mannequins). The problem of providing realistic facial expressions requires that such expressions be synthesized on a robotic face. Due to the nature of a robot face, with its limited DOF, this leads to a loss of realism. In order to maximize the human qualities retained when synthesizing the facial expressions we introduce a control system for a 2-dimensional robotic face.

This system uses Constrained Local Models (CLM) to extract information about the configuration of the face in the input stream. CLM has been shown to be more robust and more accurate than AAM when applied to human faces, and also uses the same joint model of appearance and texture, Cristinacce et al. [2]. CLM, on the other hand, uses a different method for searching the given image. AAM uses triangular search areas across the given shape to search for the feature pints in its Point Distribution Model. CLM uses rectangular search areas around feature points that are decided by patch experts; as opposed to AAM full image search. It is these features that makes CLM more robust when used on human faces.

Using this system we hope to provide two forms of control for robotic faces. One form being real-time mapping of the input face to movements in the robot face, the other form being the creation of a database of facial expressions from which the robot face can choose to express. This system outputs facial animations rather than static expressions, therefore making them more human-like by displaying the gradual change in facial composition.

Description:

In order to create the system we had to break down the problem into parts that can be more readily handled. As in Mayer et al. [1], we decided the best way to tackle the problem would be to separate it into two phases. The first phase being Recognition has two parts, the second phase is Synthesis. The recognition phase is further broken down into untrained and trained recognition. We use the 2D shape provided in the CLM-Z tracker's Point Distribution Model (PDM), Baltrusaitis et al. [3]. This is due to the 2D nature of the robotic face we are modeling our system after. The 2D shape provided by the PDM is a matrix of 66 vertex points. These vertex points represent feature points that define the overall shape of the face.

The untrained recognition phase uses these vertex points to learn their maximum possible movement in all four (up, down, left, right) directions. The trained recognition phase simply records the movement of all vertex points in every direction for the synthesis phase. Since objects closer to the camera take up more pixels: as the distance of the object from the camera increases (z-axis) the pixel movement amount (x and y axis) increases for the same real world distance moved. In order to account for this we scale all pixel movement values based on the distance between the two inner eye corners, as defined by a learned neutral face. This neutral face is created by using the first batch of vertex point sets (batch size can be easily varied) and

the calculating the average for each vertex point after normalizing it. To normalize a vertex point (to account for relative position of the face in the input video) we define the vertex point set by using the midpoint between the inner eye corners and calculating the distance between that midpoint and each vertex point.

Once the maximum movement values have been learned and a number of 2D shapes have been normalized and recorded (after calculating movement amount from the learned neutral face), the synthesis phase can then begin to recreate the input expressions on the robot face. To do this, the synthesis phase, calculates intensity and left-right balance values for each of the defined Action Units (from the Facial Action Coding System). This definition of AU, rather than motor and servo movements, allow the system to be applied to multiple robot configurations; leaving the definition of each AU to the individual robotic face configuration.

Experiment:

To test the system we needed a virtual robot face that could perform the desired facial expressions. For this purpose we use the Source SDK and its provided Face Poser program to define facial expression animations using models created for the Half-Life 2 video game by Valve. Due to the nature of Face Poser real time control was not possible for the experiment. This is not much of a loss, since the frame rate for processing the video was extremely low (~4FPS), this would lead to very unnatural facial expressions. With this limitation in mind we set out to implement the system to work with Face Poser. For this purpose we added a FileIO part to the system that handles all file operations. FileIO handles the formatting of the file that holds all

the information on the maximum movement for each vertex point. It also handles the formatting of the facial animation file for Face Poser (vcd file format).

Face Poser provides a rich set of features for creating dialogue scenes in mods. For our purposes we focus on the facial animation aspect. Face Poser defines a set of flex points that manipulate the face image based on an intensity and balance slider (for those with a left and right aspect). These flex points are based on the Action Units defined in FACS. For our experiment we defined the synthesis phase to work with a total of 11 flex points (lid_tightener, lid_closer, inner_raiser, outer_raiser, lowerer, corner_puller, corner_depressor, part, puckerer, stretcher, head_tilt). We wanted to cover 21 DOF, but due to the lack of tracking for the eyeballs we could not provide that aspect. We also could not calculate movement of the head to the right, left, up, or down due to the high complexity of the problem. Each flex point (interchangeable with action unit) is defined, in the synthesis phase, by listing each of the vertex points that pertain to it. For each of the vertex points a group of flags is used that define details about the point. These details include: the side it is a part of (left or right), the directions it moves in when associated with this flex point (up, down, left, or right).

To calculate the intensity for each flex point we simply averaged the amount moved by all the vertex points associated with it (separate averages for left and right side) and calculated its ratio to the average maximum movement for the respective sides. If the movement of the vertex point does not match what was defined in the synthesis phase a 0 intensity value is returned. These calculations leave us with values for the intensity of the left and right sides. We then use those intensities to calculate the left-right balance required by Face Poser. Since there is no available formula for how Face Poser calculates the intensity of the weak side of the left-right balance we needed to come up with a suitable approximation. Using the approximation that:

*Weak_intensity = (1.0 – (unfixed_balance * 2)) * strong_intensity*

We could then work using the weak and strong intensities (left and right, depends on which is more intense) to find what the unfixed balance is. The unfixed balance is simply a designation to account for the fact that Face Poser uses a value between 0 and 1.0 for the balance value, where [0, 0.5) denotes the left intensity as the strong intensity, 0.5 denotes equal intensity for left and right, (0.5, 1.0] denotes the right as the strong intensity. The unfixed balance works with size of the range and then adds 0.5 to fix the balance if the strong intensity is the right side. Now that we have the intensity and balance values we simply use FileIO to create a vcd file and open it using Face Poser to see the result.

Results:

We tested the system using 2 sets of videos. One set was used to create the max files for use by the synthesis phase; these videos were each approximately 30 seconds. The second set consisted of minute long videos to be synthesized. The first set contained 4 videos, the second contained 5. Each of the videos in the second set was synthesized using all 4 of the videos in the first set. This was done to test the effects of the determined maximum movement values on the overall accuracy of the synthesized expression.

Upon analyzing the output animations in Face Poser we discovered that despite our misconceptions the CLM-Z tracker did not track eye lid movement. This means that the vertex points defined for the eye region tracked the surrounding shape of the eye, not the eye lids. This made the lid_closer flex point useless. Another issue we ran into concerned our assumption that a 2D shape was provided for each frame of the video. We were wrong in our assumption; despite

using it to approximate the time that corresponds to each vertex point set. This means that most of the output animations are a bit shorter than the input video, making them less realistic.

Making flex point intensity default to 0 when a vertex point did not move in the correct direction resulted in a lot of time spent in the neutral position. When there was movement in the face it was fairly accurate. The lack of movement can be accounted to the interference between different flex points on the same vertex point. The eye brows appeared to be the most accurately synthesized part of the face.

Conclusion:

We introduced a system for controlling a robotic face that provided both real-time control and off-line control based around the CLM-Z facial tracker implementation by Baltrusaitis et al [3]. This system was divided into 2 phases, recognition and synthesis phase, as discussed in Mayer et al [1]. We tested the off-line control aspect using the Source SDK's Face Poser program for a virtual face. Due to misconceptions about the CLM tracker's PDM the timing of the off-line control was thrown off leading to unrealistic expressions. Real-time control could not be tested using the current implementation of the tracker because of frame rate issues and the inability to do real-time animations in Face Poser.

Future Work:

Many areas of the experiment implementation can be improved. Firstly we could improve the accuracy of the definitions of each vertex point for the flex points. This can be achieved by

analyzing a video where the desired flex animation is performed and then learning the directional movement for all vertex points. Then we filter out the points not near the area the flex point affects. Lastly we use only those vertex points that moved a significant amount. This gives us higher precision in calculating flex point intensities.

Another area for improvement would be the approximation of the time when flex point intensities occur. The best way would be to provide a timer for real-time control, while using frame counts in some manner for off-line control. Lastly, the neutral face is learned in both the untrained and the trained recognition phase. In order to maximize operability between movements in the trained and the untrained parts of the recognition phase a single definition of the neutral face needs to be defined and incorporated into an external file similar to the max file format.

Bibliography

[1] C. Mayer, S. Sosnowski, K. Kuhnlenz, and B. Radig, "Towards robotic facial mimicry: system development and evaluation", *19th IEEE International Symposium on Robot and Human Interactive Communication*, 2010.

[2] D. Cristinacce, and T. Cootes, "Feature Detection and Tracking with Constrained Local Models",

[3] T. Baltrusaitis, P. Robinson, and L. Morency, "3D Constrained Local Model for Rigid and Non-Rigid Facial Tracking",