EE586 Final report

Spring 2016

KineTetris

Hengyue Liu

Jason Pang

Yihao Xia

Table of Contents

Table of Figures

Abstract

Real-time hand gesture recognition affords users the ability to interact with computers in a natural and intuitive ways. Our project developed a real-time hand motion based Tetris game on TMS320DM6437 Digital Video Development Platform. Essentially the player can begin and play Tetris using hand gestures and arm motions, that are visually captured by a camera. The system is mainly including the image processing, hand gesture feature extraction, gesture recognition, game engine, and an artificial intelligence system that selects the computer block placement.

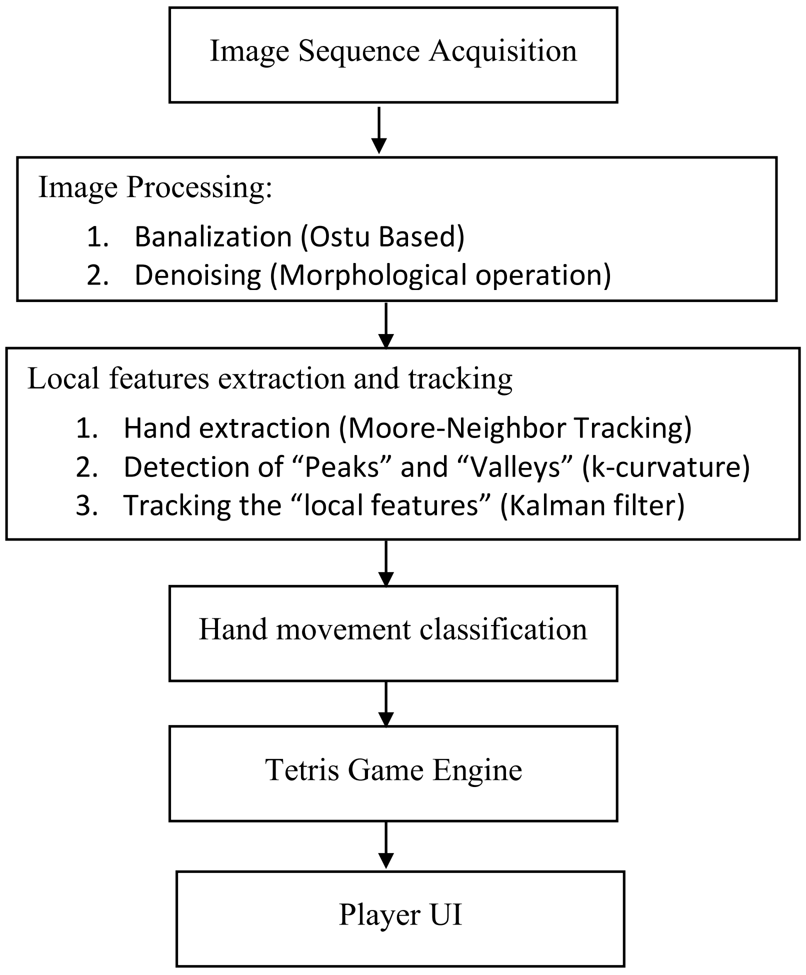


Image Sequence Acquisition
Image Processing:
1. Banalization (Ostu Based)
2. Denoising (Morphological operation)
Local features extraction and tracking
1. Hand extraction (Moore-Neighbor Tracking)
2. Detection of “Peaks” and “Valleys” (k-curvature)
3. Tracking the “local features” (Kalman filter)
Hand movement classification
Tetris Game Engine
Player UI

Fig. 1 Flowchart of the Tetris game system

1. Segmentation

1.1 Related Approach: Skin Segmentation

Skin color segmentation is widely used to detect the human faces and hands. They are based on the color discrimination between skin and non-skin pixels. Within a controlled background unlikely containing skin color tones, the skin color based segmentation works good [1][2].

Usually the skin color segmentation is in the L*a*b* space which split the color and brightness. The L*a*b* space consists of a luminosity layer L*, chromaticity-layer a* indicating where color falls along the red-green axis, and chromaticity-layer b* indicating where the color falls along the blue-yellow axis. Since all of the color information is in the 'a*' and 'b*' layers, the difference between two skin and non-skin object is able to be quantified based on the Euclidean distance metric. However, the tones of skin change according to the ambient light, shadows and the non-uniformity of image sensor. In out case, the lab environment (orange wall and warm light) sharing skin color tones makes the segmentation based on the color tone very difficult. Therefore, we use Otsu's method for segmentation.

1.2 Ostu's Method

Considering the lab environment, we found the segmentation performance is mainly influenced by the brightness, so the Otsu's method would be a feasible approach. Otsu's method is very fast to segment hand from the simple background which is showed in Fig.2 below, so we choose to use the Otsu's method for the segmentation.

Otsu' method is a nonparametric and unsupervised method [3]. It optimizes the threshold separating the foreground pixels and background pixel through minimizing the combined spread (intra-class variance) of two classes. We could define the intra-class variance as weighted sum of two pixel classes:

$$\sigma^2_{\mathrm{intra}}(t) = \omega_b(t)\sigma^2_b(t) + \omega_f(t)\sigma^2_f(t) \tag{1.1}$$

In the above equations, the $\omega_b(t)$ and $\omega_f(t)$ are the probabilities of the background and foreground separated by the threshold t. The $\sigma_b^2(t)$ and $\sigma_f^2(t)$ are the variance of the two classes. Assume the histograms are L, we have:

$$\omega_b(t) = \sum_{i=0}^{t-1} p(i) \tag{1.2}$$

$$\omega_f(t) = \sum_{i=t}^{L-1} p(i) \tag{1.3}$$

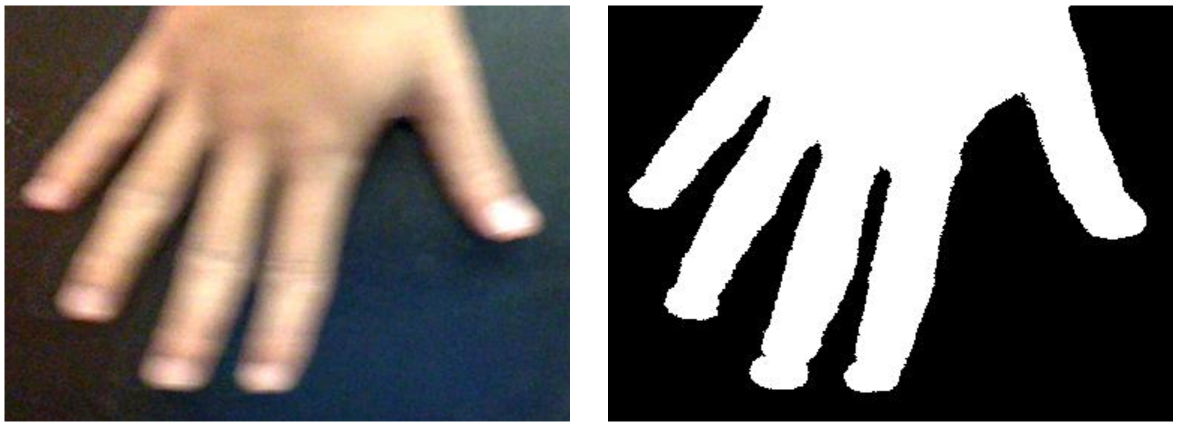
Because the problem of minimizing intra-class variance could convert to the problem of maximizing the inter-class variance, the expression of optimization is:

$$\sigma_{inter}^2(t) = \sigma^2 - \sigma_{intra}^2(t) = \omega_b(t)(\mu_b(t) - \mu_T)^2 + \omega_f(t)(\mu_f(t) - \mu_T)^2 \tag{1.4}$$

Where the μ_T is the combined mean, $\mu_f(t)$ is the mean of foreground, $\mu_b(t)$ is the mean of background and the σ^2 is the combined variance. Due to the relationship – $\mu_T = \omega_b(t)\,\mu_b(t) + \omega_f(t)\,\mu_f(t)$ – we get aim function:

$$\sigma_{inter}^2(t) = \omega_b(t)\omega_f(t)(\mu_b(t) - \mu_f(t))^2 \tag{1.5}$$

result of Otsu, skin color, skin color based k-means and so on.

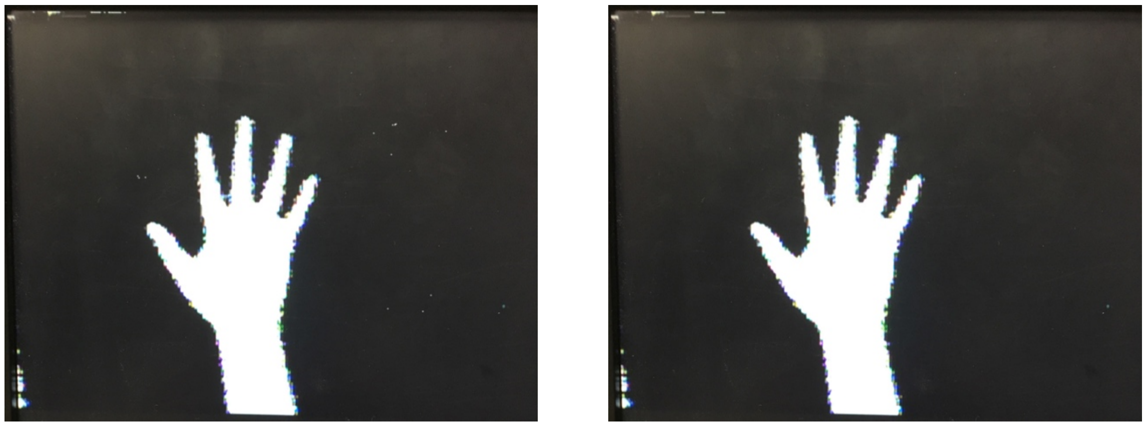


a) frame in RGB *b) results of Otsu's method*

Fig. 2 Segmentation based on Otsu's method

2. Morphological Filtering and De-noising

We employed the dilation and erosion operator with different masks. Dilation and erosion are two basic morphological operations. The dilation method is to enlarge an object uniformly in spatial extent, whereas the erosion method is to shrink an object uniformly in spatial extent. With proper dilation mask and the erosion mask, we could remove both remove the noise and smooth the edge of hand. Below is the result of morphological processed image.



a) Original binary hand image *b) result of 1 dilation + 1 erosion*

Fig. 3 Morphological processing result

3. Feature Extraction

To implement both the rotation and horizontal movement in the Tetris game, we need more features than only the Centre of Gravity(COG). Therefore, we need feature extraction method for the hand gesture classification.

3.1 Related Approach

Most intuitive approach of hand features extraction is based on the contour of hand, because the contour reflects both the hand position and orientation. The k-curvature [4] and curvature scale space (CSS) are method based on the contour information [5]. For captured image is containing a potential hand gesture, the contours of hand gesture can be extracted by edge detection. However, in CSS method, the object contour of curvature calculation is low-pass filtered. Which cannot effectively capture subtle gesture changes. Besides converting the image to 1D vector, histogram of Oriented Gradients (HOG) [6] is another method for feature extraction. It is similar to the edge orientation histograms, Main idea of it is to count occurrences of gradient orientation in local localized portion of an image. According to the Eigen vectors generated by this method. The hand gestures like up and down can be differentiated. However, when dealing with the rotation, which depends on the small shift of hand, HOG is not an effective method.

3.2 Fingertips Detection Based on K-Curvature

Comparing with other feature extraction methods, the k-curvature method extracts the "peaks" and "valleys" on the hand. The relevant angle based on this feature position can properly reflect the rotation of hand.

3.2.1 Moore-Neighbor Tracing

The first step of contour based feature extraction method is contour tracing. Contour tracing technique can trace the boundary in sequence alone the contour of objects. Sequenced contour position is a small subset of original image, and it is useful for reducing the amount of computation for further operation. In our project, the sequenced contour positions are used to detect the local features of hand including fingertips -- "peaks" and "valleys".

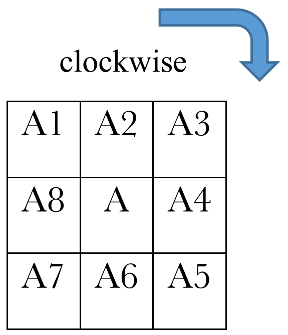


Fig. 4 Moore-neighborhood

We have implemented the Moore-neighborhood algorithm [7][8] to trace the contour of hand. For particular pixel, its Moore neighborhood pixels A1, A2, A3, A4, A5, A6, A7 and A8 shown in Fig.4 above.

The algorithm starts by scanning from the left upper corner. When reaching any pixel which is not the background point, we declare it is our "start" pixel. Then we will extract the contour by going around the pattern in a clockwise direction. After reaching any foreground pixel the we would backtrack to the previously reached background pixel A*, then continue to go around the pixel A* in a clockwise direction. Contour could be traced by run the searching operation recursively, until hitting the start pixel again. Besides, due to the noise in the frame, we employee a heuristic screening method based on length of contour to decide which one is hand.

The results of the contour tracing is showed in the Fig.5. After extracting the regions, the boundary of each region was represented as a list of pixel positions in a clockwise order. A heuristic screening of the regions based on perimeter length led to the identification of a hand.

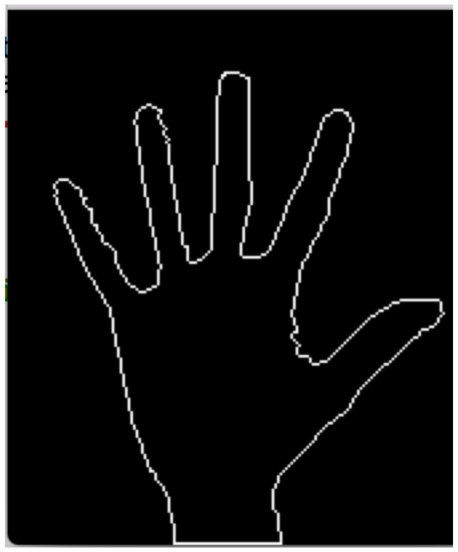


Fig. 5 result of hand contour tracing

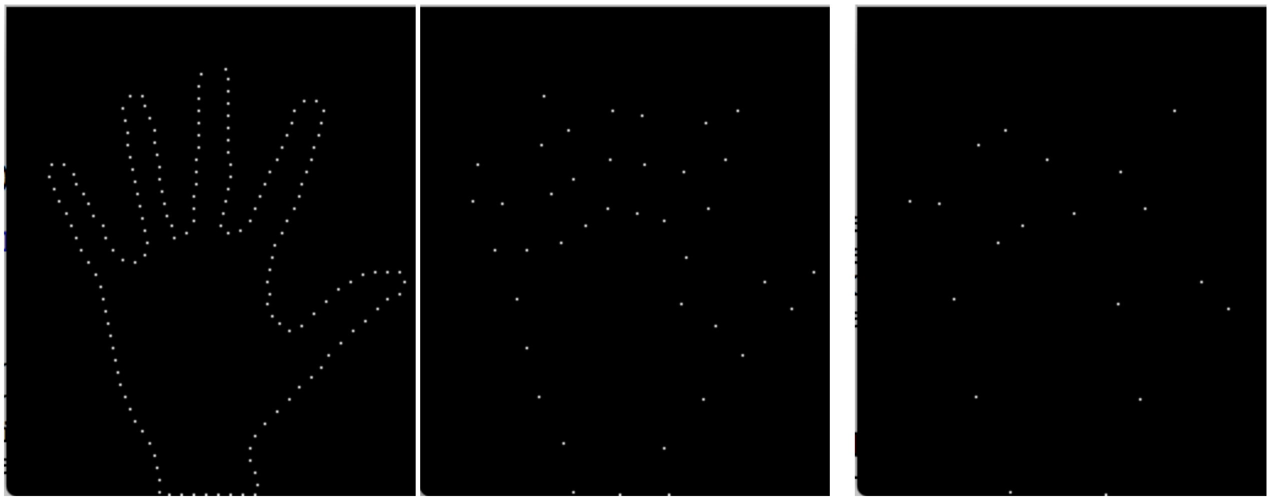
3.2.2 K-Curvature

From the contour tracing steps, the boundary of hand is represented as a list of pixel positions {(x(i), y(i))}in clockwise order. The selected contour makes it possible to further extract and analyze the local features. Within the contour sequences of the hand, the k- curvature is defined as the angle C(i) between two vectors [P (i − k); P (i)] and [P (i); P (i + k)], where k is a constant parameter. We employ the function of cosine theorem to calculate the angle C(i):

$$C(i) = \frac{\operatorname{acos}(d_c^2\text{-}d_a^2\text{-}d_b^2)}{\text{-}d_a{*}d_b} * 180/\mathrm{Pi} \tag{3.1}$$

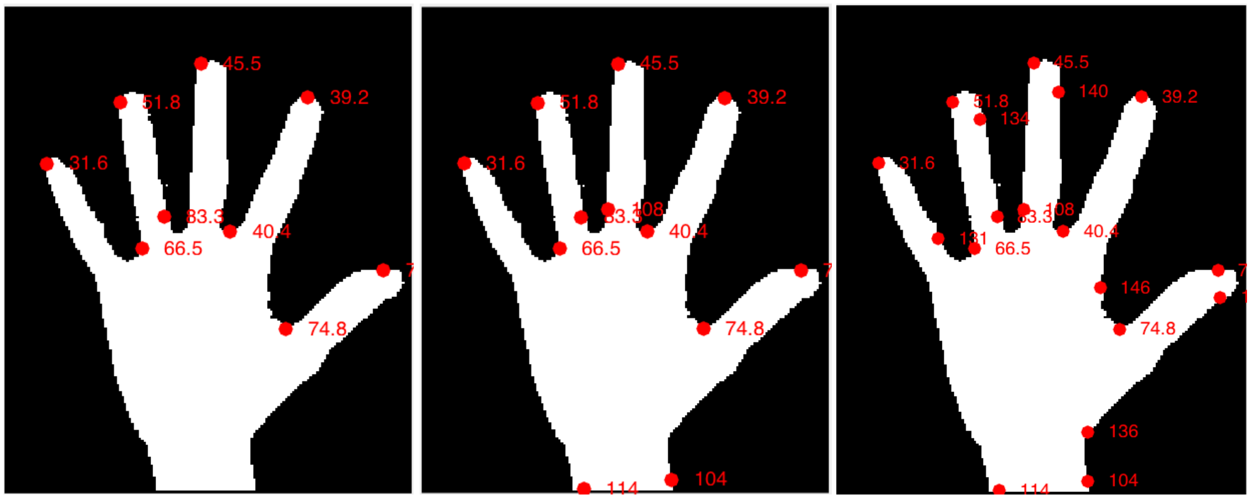
Here, the d_c is the Euclidean distance between P (i − k) and P (i + k), the d_c is the Euclidean distance between P (i) and P (i + k), the d_c is the Euclidean distance between P (i) and P (i - k), and the Pi is the circular constant.

From the definition of curvature, it is easily to figure out that along the boundary where the curvature reached a local extreme is the “local features”— “peaks” and “valleys” [6]. Local extreme is changing according to the subtle distortion of hand and the scale of the frame. There for the proper value of k and angle threshold is change according to the size of frame and different gestures. Through the tries and errors, we found that both “peaks” and “valleys” could be robustly found by setting the k = 20 and the threshold of angle to be 100 degrees. The result of “local feature” detection with different is show in the Fig. 6 and Fig. 7.



a) K = 5 *b) K = 20* *c) K = 40*

Fig. 6 The traced contour with different k value



d) Threshold = 100° *e) Threshold = 120°* *f) Threshold = 150°*

Fig. 7 The local features with different threshold of angle

3.2.3 Centroid Measurement and Tracking

Beside the "Local features" on the boundary of hand, the center of the hand is also important to act as anchor point, because in rotation gestures it is generally static. So we use it as an anchor point.

The center point of hand's region that can be easily computed from the moments of pixels in hand's region, which is defined as:

$$X_c = \frac{M_{10}}{M_{00}}, Y_c = \frac{M_{01}}{M_{00}} \quad (M_{ij} = \sum_x \sum_y x^i y^i I(x, y)) \tag{3.2}$$

Where, I(x, y) is the pixel value at the position (x,y) of the image, x and y are range over the hand's region. The position of the center point is calculated as Xc and Yc. Besides, we use the average position of the local features for the tracking of "peaks" and "valleys", The center of the hand and the average of local features are green points showed in the Fig.8. So, the overall "local features" including the "peaks", "valleys", center of hand.

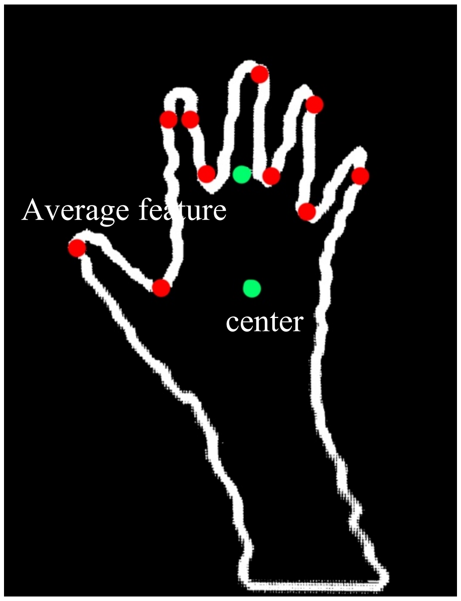


Fig. 8 real-time hand feature extraction result

4. Fingertip Tracking/Dynamic Hand Tracking

To build a robust hand tracking system, we utilized the Kalman filter to track both the local features and center of hand. For the location vector X_k of tracking point in the kth image frame, Kalman filters are used to estimate the locations $\hat{X}_{k+1}$ of "local features" on hand in the (k+1)th frame. The nearest neighbor rule to associate fingertips and valleys between frames by comparing the estimated locations $\hat{X}_{k+1}$ with the detected locations X_{k+1} in the (k+1)th image frame to achieve the robust tracking of multiple fingertips.

4.1 Kalman Based Tracking

Kalman filter solves discrete-data linear filter problem recursively [9] [10]. To estimate the state $x \in \mathcal{R}^n$, the current state is represent by difference equation:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.1}$$

with a measurement $z \in \mathcal{R}^m$ that is

$$z_k = Hx_k + v_k \tag{4.2}$$

Where w_k and v_k represent the process noise and measurement noise. They are assumed to be independent, white, and with normal probability distributions P(w) ~ N (0, Q), P(v) ~ N(0, R). The Q and R are the covariance of process and measurement respectively. Then the estimated value of current state could be by recurrent function.

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{4.3}$$

The difference $(z_k - H\hat{x}_k^-)$ in (4.3) is the residual. It reflects the discrepancy between the predicted measurement $H\hat{x}_k^-$ and the actual measurement z_k. The matrix K in (4.3) is chosen to be the gain that minimizes the posteriori error covariance (4.2). This minimization state can be find through setting the derivative of trace of the result to be zero. Then we have:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

In the The state estimation of Kalman filter is based on a form of feedback control. The error priori covariance of state k can be calculated through equation below:

$$P_k^- = A P_{k-1} A^T + Q \tag{4.4}$$

In our program, we employ the Kalman filter to track every points extracted from hand including "local features". Through the state update and the feedback from measurement update, the Kalman filter estimates the position of each detected point. This reduces the influence of the noise, tiny shift, and disappearing of "local features", which improve the robust of the hand gesture recognition system.

4.2 Inter-frame constraints

In order to track the "local feature" in the time sequence, we use the Euclidean distance to identify the corresponding detected points between frames. Based on the assumption that the hand would not move a very long distance suddenly between 2 frames, the new point locations in the successive frame should be matched to the nearest ones. Due to the instability of local key points detection system, the local feature points might suddenly disappear which violated the assumption, as a result of which we replace the disappearing points with the estimation results from Kalman filter.

5. Basic Gesture Classification (Stratagem of control)

Tetris game basically need commands of "left", "right", "down", and "rotations". The final step is to recognition of the gesture depending on the movement of hand [11]. (The "down" command is not included in the program but can be easily implemented.)

5.1 Horizontal movement

With the tracked positions of hand in successive frames, the average movement over the time sequence between start point A (a_x, a_y) and end point B (b_x, b_y) is defined as:

$$M_x = \frac{\sqrt{(a_x - b_x)^2}}{t_a - t_b} \tag{5.1}$$

$$M_y = \frac{\sqrt{(a_y - b_y)^2}}{t_a - t_b} \tag{5.2}$$

$$M = \frac{a_a - a_b}{t_a - t_b} \tag{5.3}$$

where a_A and a_B are the accelerations of point A and point B, t_A and t_B are the sampling time of A and B , respectively.

The horizontal or vertical trends of the hand movement make the recognition possible. Through setting positive threshold λ_p and a negative threshold λ_n, the command of the four basic gestures up, down, left, and right can be expressed as:

$$Command = \begin{cases} Left\ , M_x < \lambda_p \\ Right\ , M_x > \lambda_n \\ Up, M_y > \lambda_p \\ Down, M_y < \lambda_n \end{cases} \tag{5.4}$$

where M_x and M_y are the measured average movement on x and y axes. When the measured movement meets one of the above conditions, a corresponding basic gesture will be accepted; otherwise, the measured movement will be discarded as noise (ineffective gesture).

5.2 Rotation command

The recognition of rotation is based on the relevant angle shifts of fingertips. We set the vector from the lower left corner point to the lower right corner of the frame as the base vector(v_i) and the vectors from center of palm to the center of "local features" rotation vectors (lv_i). Then the rotation angle of each "local feature" at ith frame could be defined as:

$$\theta_i = \text{acos}\,(v_i \cdot \frac{rv_i}{|v_i||rv_i|}) \tag{5.5}$$

We define the target angle θ of rotation recognition as the accumulation of angle between rotation vector and based vector within certain time sequence.

$$\theta = \sum_{i=1}^{n} \theta_i \tag{5.6}$$

Generally, the target angle of left rotation command is less than zero, while right rotation command is greater than zero. Through setting positive threshold θ_p and a negative threshold θ_n, the command of the rotation gestures left, and right can be expressed as:

$$Rotational\ Command = \begin{cases} Left\ , \theta < \theta_n \\ Right\ , \theta > \theta_p \end{cases} \tag{5.7}$$

6. Game Engine

6.1 System Structure

The game engine of Tetris has a relatively simple structure. Random tetronimos (four-block pieces) are generated and descend in a 10x20 rectangular block field. The pieces cannot intersect with each other or move outside the field boundaries. Whenever a full horizontal line of 10 blocks is filled, that line is cleared and all lines above that are shifted down one block step. When no valid move is possible, the game ends.

To implement this, the game engine consists of 6 functions, as well as the main function: *Create_Block()*, *Display_Block(int x, int y)*, *Rotate_Block(int* x, int y, int horiz)*, *Move_AI()*, *game_init()*, and *penalty(Uint32* side, int amt, int color)*. *Create_Block()* is responsible for selecting one of the 7 Tetris tetronimos at random and generating it for use in the system. *Display_Block(int x, int y)* transmits information about the current playing state to the board's display outputs. *Rotate_Block(int* x, int y, int horiz)* enables both players and the AI to change the orientation of the block they are controlling with 90 degree rotations, while keeping these blocks aligned within their expected field location. *Move_AI()* is the main AI function which we shall discuss shortly. *game_init()* initializes the playing field to the default state, and allows for restarting the game. Lastly, *penalty(Uint32* side, int amt, int color)* takes in as inputs which field (player or AI) is concerned and how many blocks to randomly add into the field as punishment for the other side performing well (for example, clearing 4 lines with one block).

6.2 Behind AI

The primary algorithm behind the *Move_AI()* function is breadth-first search (BFS). This technique traverses all possibilities (at and up to a specific tree/graph depth) to determine the optimal move [12]. It can work as long as there is some heuristic or function that can be ranked or scored. In our case, we can write an arbitrary score function for the current play field state (which we'll label depth 0), and onwards (depth 1 includes all possible play field states with the inclusion of the immediate block controlled, and depth 2 includes all of the possible play field states based on the depth 1 possibilities, followed by all possible moves with the inclusion of an incoming block) [13]. By possible play field states, each horizontal position is considered, multiplied by the number of possible orientations at that horizontal position. For example, for a 2x2 square block, there is only 1 orientation, but for the T

Tetris block, 4 orientations need to be considered at one horizontal position. Both the human and the AI are given indication of this incoming block in our implementation.

We found that a depth of 2 would be sufficient for the AI to perform well. The score was determined using a few factors. First we reward blocks placed adjacent to other blocks, on other blocks or the bottom, and along the wall, with a score increase from 0 in depth 1. Second, we punish the block based on its height placement- the higher the placement, the greater the punishment. Third, we reward any line clears significantly, since this is the ultimate tool to prevent the game from ending. Finally, we punish any moves that result in blocking off empty field spaces- these create 'holes' that prevent line clearing. However, this last heuristic is adaptive- it scales with the height of the tallest positioned block such that as the field fills, it becomes more important to clear lines rather than be concerned with blocking off lower level spaces. This is vital to prevent tower formations in the field with deep, thin canyons between, where the AI might just stack blocks to the top rather than prioritize clearing lines. This also allows the AI some chance to set up combo or multi-line clears when the field is more empty, since multi-line clears will have a significant score boost over single-line clears, but the search algorithm isn't overly skewed into clearing lines at every possible opportunity. With the scores obtained for depth 1, the same process can be repeated at depth 2 (with score changes based on the depth 1 scores instead of 0).

7. Major Challenges

One major challenge that we encountered is the memory size and operation speed. The video processing requires large amount of memory and high processing speed. In order to make the system more sensitive to intricate player movements, we first down sampled the object image reducing the complexity of image processing. Besides, by employing the Otsu' method rather than template matching, the step of hand detection is simplified. Also, considering that there are only four control gestures in the Tetris game, we did not use the multi-gesture fusion method which is more suitable to gesture classification in a large gesture set. Instead, we extract the positions of fingertips for the gesture classification, which is enough to reflect both horizontal movement and rotation in Tetris game.

Different players can perform movements at different speeds and pattern, so another challenge design control strategy which adaptive to the different gesture control styles. For example, when a user swipe right for moving one block to right, at the end of this gesture, the hands may or may not return to its original position afterwards. This kind of gesture confusion would also happen in rotation control. To reduce the influence of sub-gestures, the gesture classification is based on the cumulative value rather than abstract value of movement. Besides, when one command is running, we reset the accumulative value of other command to avoid the influence of two consecutive commands.

Another challenge is figuring out not just how to recognize proper moves, but also how to exclude unintentional moves (such as an arm movement that is part of a natural sway, or drifting position from fatigue, or ambiguous movements that could be meant for one type of move but is actually intended to signify another type of move). Various factors of the background would cause problems including changing illumination, low-quality video (tradeoff of calculation time), temporary occlusion, and background clutter.

8. Results and Conclusion

We implemented a real-time hand motion based Tetris game on TMS320DM6437 Digital Video Development Platform. One can play Tetris with hand gestures captured by the camera. Based on the hand gesture, the commands are generated accordingly, including left, right, rotation, and drop. The main Tetris game interface consists of 2 player sections, incoming blocks, current scores and live processed hand video. Fig.9 shows the game interface.

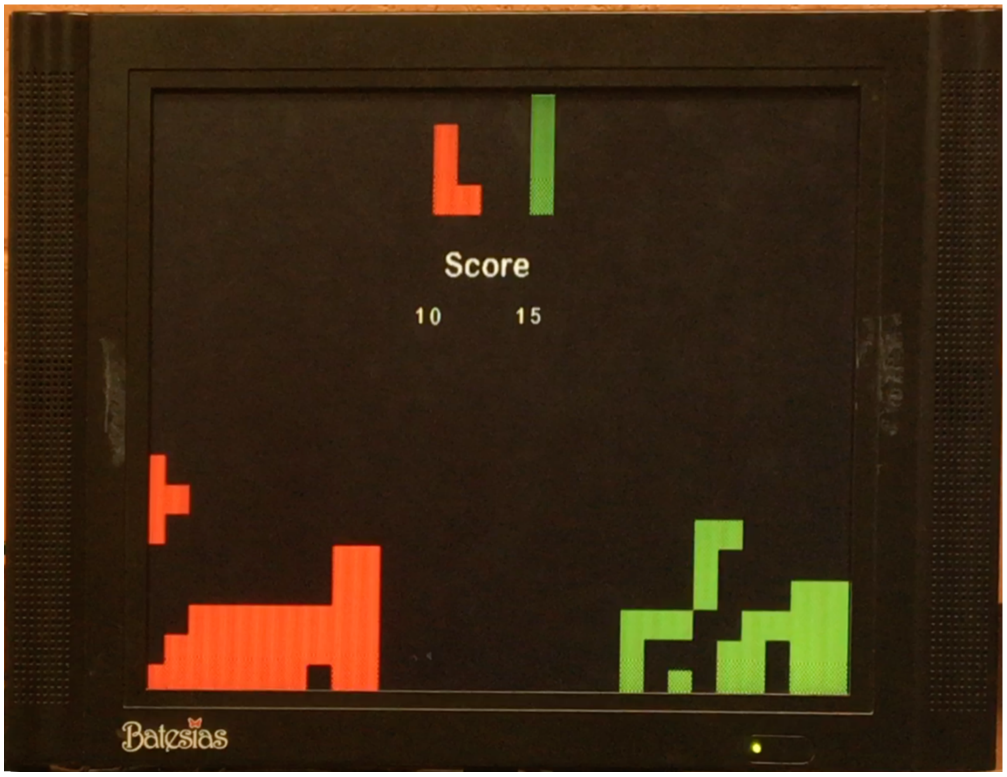


Fig. 9 The setup of game system

9. Future Works

First, due to the limited time, we did not implement the background subtraction. Background subtraction could increase the adaptability of the game system to complex and dynamically changing environment. On the other hand, the gesture classification is not robust enough since the gesture classifier is manually pre-defined and fixed without training. Since all key points are accurately located in each frame, the classification can be described as solving a two-set points matching problem. Hence, finding the affine transformation matrix between frames, then multiplying matrices to get accumulated translation and rotation would be a more reasonable way to translate features to commands.

Reference

[1] Abdel-Mottaleb, M., Elgammal, A.: Face detection in complex environments from color lmages. Proceedings of the International Conference on Image Processing (ICIP), 622–626 (1999)

[2] Alshebani, Q., Premaratne, P., Vial, P.: An Embedded Door Access Based on Face Recognition System: A Survey. To appear in (ICSPCS), 2013, Australia, (2013)

[3] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 9, No. 1, 1979, pp. 62-66.

[4] Segen, J., Kumar, S.: Fast and accurate 3d gesture recognition interface. Proceedings of the 14th International Conference on Pattern Recognition, vol. 1, 86 (1998)

[5] Chang, C.C., Chen, I.-Y, Huang, Y.-S.: Hand pose recognition using curvature scale space. IEEE International Conference on Pattern Recognition (2002)

[6] Dalal, D., Triggs, B.: Histograms of oriented gradients for human detection. IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2, (2005)

[9] Kalman, R. E. 1960. “A New Approach to Linear Filtering and Prediction Problems,” Transaction of the ASME—Journal of Basic Engineering, pp. 35-45 (March 1960).

[10] Brown R G, Hwang P Y C. Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions[J]. Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions, by Brown, Robert Grover.; Hwang, Patrick YC New York: Wiley, c1997., 1997, 1.

[11] Xie R, Sun X, Xia X, et al. Similarity matching-based extensible hand gesture recognition[J]. Sensors Journal, IEEE, 2015, 15(6): 3475-3483.

[12] Leiserson, Charles E.; Schardl, Tao B. 2010. A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers) (PDF). ACM Symp. on Parallelism in Algorithms and Architectures.

[13] Jules, Cédric. 2013. Totologic: Tetris AI explained. Daedalus Lib. http://totologic.blogspot.com/2013/03/tetris-ai-explained.html