Gaming Duck Hunt Using Computer Vision Techniques

Joel Meuleman
joelm@uvic.ca

ECE 471
University of Victoria
Victoria, British Columbia

Abstract

Duck Hunt is a classic arcade game where the goal of the game is to set a high score by shooting (moving the crosshair over) ducks in a scene. For a computer to play the game at a high level, it must be able to recognize ducks in the scene, and move the crosshair to these duck positions. The implemented approach was to process each frame as it was generated, locate ducks within the frame, then move the crosshair to determined duck positions iteratively. Processing of the current frame started with recoloring to grayscale and resizing to a quarter of the original frame dimensions. The difference between the current frame and the prior frame was then computed, and the resulting image then blurred and thresholded. To ensure each duck was represented as a single region in the binary image, binary dilation was used to join disconnected regions of close proximity. Continuing, the (x,y) locations of duck-like regions were calculated as the first moments of each distinct connected component. The crosshair's position was then iterated over each proposed duck location. This frame-differencing based approach had high accuracy for all level sets with static backgrounds. This solution functions with greatly reduced accuracy on dynamic scenes where random background motion is perceived as duck locations. Effects of motion could be reduced with various techniques in a future implementation of the chosen solution.

1 Introduction

The goal of the project was to design a computer vision algorithm that could score as highly as possible at the classic game *Duck Hunt*. The objective of the game is to shoot as many ducks as possible before they fly out of frame. To do this, the solution algorithm must locate ducks in the scene then move the crosshair to the predicted duck location. The challenge increases in difficulty as higher levels are reached, with added occlusion, random changes in duck velocity, random changes in duck coloring and shape, and finally dynamic scenes in the last level set.

2 Problem Interpretation and Data Challenges

'Shooting' the ducks was initially construed as two problems: locating the ducks and updating the crosshair position to where the ducks were predicted to be in the next frame. As such, the algorithm was broken into two components, duck locating and duck location predicting. This second component was later found to be not relevant, and led to a simpler revision of the demonstrated algorithm that increased the accuracy of the solution significantly.

Finding ducks in the scene, or separating ducks from the background, was the first challenge encountered. To separate a duck from the background, duck-like features needed to be identified. Familiarity with the data was important to the selection of features. First, consistency of the duck features was considered. Does the feature continue to be a good defining feature over the course of the level, over the level-set, over the set of all levels provided? Second, is it simple to find the feature within the image? Can the feature be matched quickly and simply, with small computational overhead? Thirdly, can this feature be localized within the image? This is clearly important as calculating the coordinates of ducks within the image is the end goal of the solution.

Several possible useful feature candidates in identifying a duck were determined. Duck shape was one such possible feature. All ducks will have one of several shapes, depending on wing position and direction. Shortcomings of this approach include: without some sort of duck localization, even at low resolution, each possible duck shape in multiple directions would have to be template matched by cross correlation across the entire image which becomes computationally expensive when having to match a multitude of templates and is thus a poor candidate as a real-time solution. Further, taking the gradients of the frame to template match duck shapes against is suboptimal once occlusion is introduced as the majority of the defining duck-like shape would be hidden. More challenges presented by the data included: ducks changing color frame by frame, ducks changing shape frame by frame, and ducks changing flight direction frame by frame.

The chosen 'feature' that is common to all ducks across all level variations is motion. As the backgrounds are static in nearly all levels, motion is a simple and fast way to distinguish foreground from background objects in most of the level-sets.

3 Implemented Approach

Having decided on motion in the scene as a defining feature of a duck, motion relative to the background was quantified, and duck locations were calculated from this motion. Several stages of preprocessing were done on the current frame before duck coordinates are computed. First, each frame was converted to grayscale, then resized to a quarter its original size. These first two operations are key to the solution functionality. Grayscale images are desirable as further computations only need to be applied across one channel rather than the three of RGB. Resizing of the current frame was another step to reduce the amount of computation needed in further steps, with the added benefit of compacting ducks within the frame, which made subsequent blob locating more robust to changes in duck coloring and shape. Preprocessing continues by locating motion/change between the current frame and the previous frame. To locate motion in the scene, the current frame was subtracted from the previous frame. As the only motion in a scene is due to a duck moving, by taking the difference in pixel values between the current frame and the previous frame, a region in which a duck must have moved is computed.

$$FD_t = F_t - F_{t-1} \tag{1}$$

From the region of duck motion given by frame difference, processing is continued until the ducks are reduced to a blob-like region. To assist in eliminating irregularities in the shapes of these regions of change, and to make the regions of high color contrast more consistent, a blur is applied to the frame-differenced image. Experimentally, it was found that applying a median blur with a small ($k = 3$) size kernel was most effective at smoothing the image, and was very effective at removing any interference caused by crosshair motion. Finding crosshair motion within the scene is extremely undesirable as the solution operates on the fundamental assumption that any movement within a scene is duck motion. With the image blurred, it is then thresholded to convert it to a binary image.

After the thresholding, the resulting image should be separated into two binary components, where background is reduced to 0 and foreground duck-like blobs are set to 1. Results of processing at this point typically produced roughly duck-like blobs, but often one duck would be divided into several blobs, consisting of a head region, a wing region, and sometimes a body region as seen in Figure 2c. This segmentation of ducks can be attributed to cases where duck color intensity values differed widely between wings and head colorings. Additionally, transparency played a role in grayscale intensity as the more transparent the duck, the more the background values influenced the duck's grayscale values. Experimenting with hyperparameters threshold T and median filter kernel size k was one approach to more uniformly produce one blob per duck, but a more consistent method was found to be binary dilation. Mathematically, binary dilation is defined by Equation 3 where A is a binary image and B is an arbitrary structure within the same dimension[1]. Practically, binary dilation is visualized as sliding some kernel B over an image A, where the neighborhood pixels of each non-zero pixel at the center of B are set to 1. The shape of kernel B is given by the chosen connectivity, typically 4 or 8 connectivity, where the connectivity defines the size of the neighborhood around a center pixel to dilate. This dilation algorithm may be run in several iterations to increase the area of true regions in the binary image. Binary dilation was a helpful tool to join regions of close proximity within the image.

(a) (b)

Figure 1: Result of running solution on current frame. (a) Current frame (b) Current frame with calculated duck locations

$$FT_t = \begin{cases} 1, \ FD_t > T \\ 0, \ otherwise \end{cases} \tag{2}$$

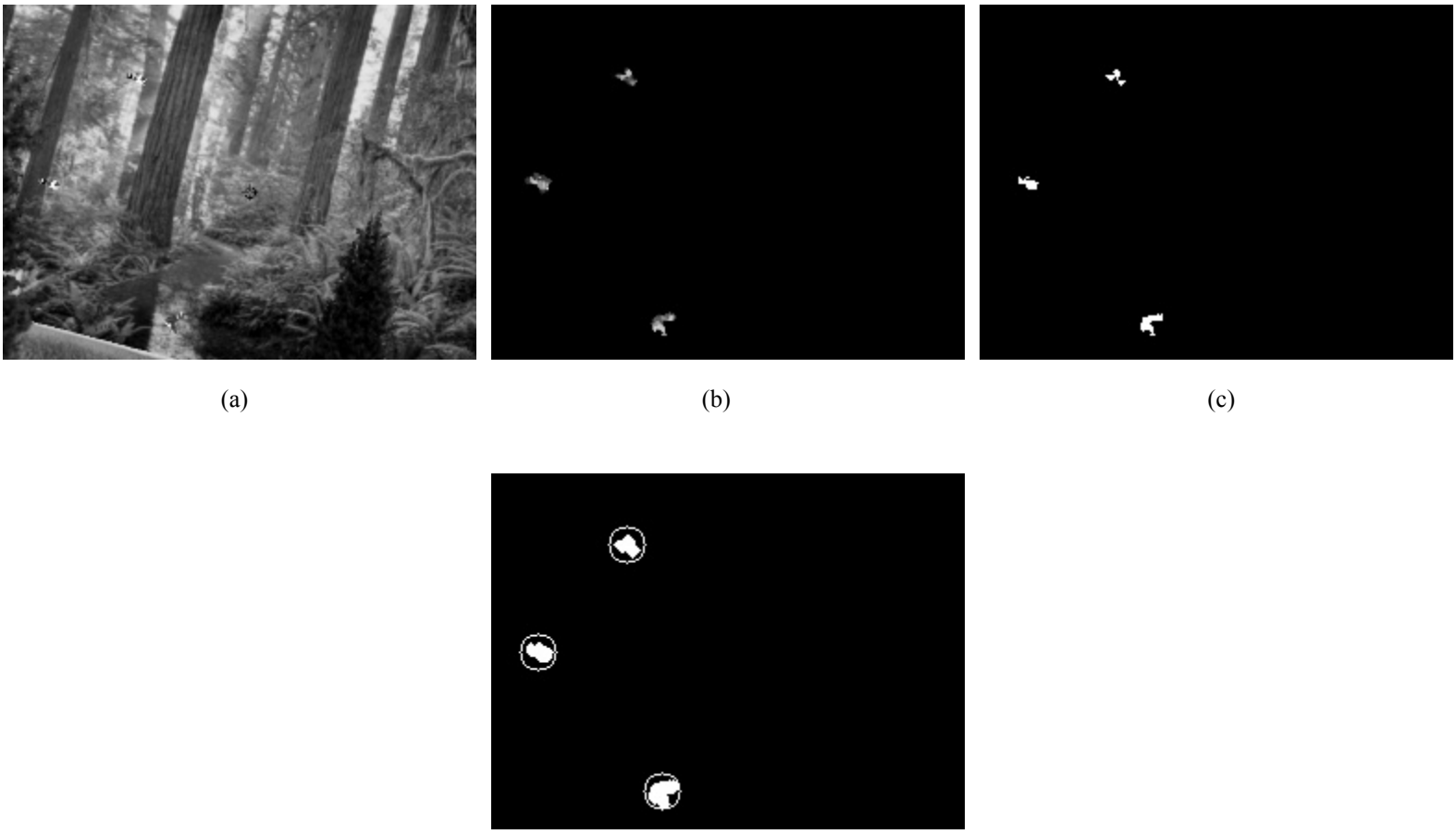
$$A \oplus B = \bigcup_{b \in B} A_b \tag{3}$$

Binary dilation with 8-connectivity and 2 iterations was found to close these small gaps between blobs incredibly effectively while preserving the duck shape of the blob. With blobs created from each duck within the scene, these blobs then had to be located within the scene. Initially, these blob centers were found by calculating the euclidean distance of binary foreground regions to background regions, then taking the maximum of these values. This method was not preferable as it was possible for a blob to have multiple points that are maximally distant from the background values, thereby producing multiple coordinates per duck-blob that technically qualified as the duck center. Instead, blob centroids were computed using OpenCV's ConnectedComponentswithStats function[2]. ConnectedComponentswithStats labels each region within a binary image, then computes the centroid of each region as centers of area (first moments)[2, 3]. The first moments of each region are computed with Equation 5. With the centroids of each blob computed, the position of each duck in the frame is now known as a list of (x,y) coordinates in the scale space of the current frame.

$$\overline{x} = \frac{1}{A} \sum_{i=1}^{n} \sum_{j=1}^{m} i \, b_{ij} \qquad \overline{y} = \frac{1}{A} \sum_{i=1}^{n} \sum_{j=1}^{m} i \, b_{ij} \tag{5}$$

The task of moving the cursor to shoot the duck within the frame was initially misconstrued and led to a needlessly complicated and less accurate algorithm as was presented in the demo. During development, the initial iteration of the duck/blob finding algorithm had significantly higher computational overhead than the current method. As such, by the time it computed duck locations, the ducks had progressed to the next frame. This implementation made some sort of velocity tracking necessary, as the cursor would have to jump to where the duck was predicted to move to be intercepted. Equation 6 calculates the velocity of each centroid at index *i*, then updates its guess on the centroid prediction in the next frame. From watching other groups' demos it was recalled that the cursor could be moved many times between duck position updates, making the logic of calculating velocity of each blob wholly unnecessary provided duck locations were computed fast enough. Another misconstruction of the performance parameters was accuracy. Accuracy was interpreted to mean crosshair movements that did or did not result in placement on a duck, whereas the actual accuracy for this task was number of ducks shot over number of ducks generated in the scene. Because of this, the presented algorithm would perform a NOOP when unsure of a duck location in an attempt to improve its accuracy scores.

$$V_i = C_{i,t} - C_{i,t-1} \qquad PP_i = P_i + V_i \tag{6}$$



(a) (b) (c)

(d)

Figure 2: Image processing and final predicted duck locations. (a) Current frame (b) Difference of current frame and prior frame (c) Threshold of median-blurred difference image (d) Centroids of dilated regions.

4 Evaluation

The demo code version and updated code version were evaluated on the tournament level set. Overall, the updated solution performed markedly better than the demo version (which ranked 8th overall) with an average accuracy increase of over 20%. The frame-differencing approach performs best on simple levels where occlusion was the only introduced challenge. The designed algorithm's major upside is its speed in locating duck candidates. By grayscaling and resizing the image, the given data of the scene from frame-differencing is significantly reduced, while still retaining data pertinent to the calculation of duck locations. All further processing was completed on binary images further reducing computational overhead. Occlusion was a trivial challenge for the motion-based approach because if any portion of a duck had moved into the visible frame, it would be detected and 'shot' at. Accuracy of the algorithm begins to drop when new duck shapes/sizes are introduced at later levels, as sometimes the computed centroids are offset from the actual duck position.

Level No.	821	822	823	824	825	826	Average Accuracy
Demo Sol	89.29%	50%	65.38%	69.23%	57.89%	31.25%	60.5%
Updated Sol	96.55%	78.57%	92.85%	92.85%	85.71%	57.14%	83.9%

Table 1: Demo solution and updated solution average accuracy over tournament level set

The unavoidable problem with the motion-based approach is that it is only applicable to static scenes. Where the scene has some dynamic element (in the provided level sets these were shadows and rain) these moving elements will be located within the scene and 'shot' at. Negative effects of motion on algorithm performance are seen in Table 1 where level 826 was a rain scene; both algorithm implementations score poorly on levels where occlusion was not the only challenge added to the scene. Accuracy could be improved on levels with dynamic scenes by taking the average or median of recent frames, as opposed to strictly differencing, to reduce the effects of random background motion.

5 Conclusion

A motion-based approach to locating ducks within the scene was an effective approach in its speed and accuracy. By taking the difference between the current frame and the previous frame, motion was localized within the image. Regions of motion were then thresholded and processed to produce duck-like blobs. Centroids of each duck-like blob were found by calculating the centers of area of each duck-like blob. The crosshair was then iterated through the positions predicted by the centroid locations. This implementation of frame-differencing to find objects within the scene worked well for several reasons. Firstly, duck locations were able to be computed quickly in each frame due to the simplicity of the algorithm. Secondly, duck locations could be found even if mostly occluded so long as some portion of the duck was visible and moving. Lastly, duck location predictions were accurate even with frame-by-frame duck augmentations provided the ducks were still moving. The motion-based approach is effective on scenes where the background is static, and has inconsistent accuracy on any level where the background is dynamic. Future work should investigate in greater detail alternative frame-differencing methods to improve accuracy in dynamic scenes.

6 References

[1] "Dilation (morphology)," *Wikipedia*, 22-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/Dilation_(morphology). [Accessed: 10-Apr-2022].

[2] "Structural analysis and shape descriptors," *OpenCV*. [Online]. Available: https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga107a78bf7cd25dec05fb4dfc5c9e765f. [Accessed: 10-Apr-2022].

[3] S. K. Nayar, "Geometric properties | binary Images - YouTube," *Youtube*, 2021. [Online]. Available: https://www.youtube.com/watch?v=ZPQiKXqHYrM. [Accessed: 10-Apr-2022].