Angry Birds Project (Vision Component)

By: Andrew Wang

ID: u4853279

Supervisor: Dr. Stephen Gould

Abstract

In this project, we focus on developing a vision module for agents playing the Angry Birds computer game. The module takes a screen-shot of the game as input and output information about objects in the game, including their position, type and shape, within a specified time constraint. To achieve this, a range of common methods in image processing and computer vision was employed. The result is a fast, reliable vision module which outputs accurate descriptions for most of the known objects.

1 Introduction

Angry Birds is a popular single-player video game. The primary goal of the game is to destroy all pigs in the scene by launching a sequence of given birds from the slingshot. The given birds may have different types, with special abilities that can be activated by tapping after the launch. Apart from the birds and pigs, there are also many other objects in the scene such as wood, ice or stone and the player should take them into consideration when deciding which action to taken. A level is completed if all pigs are destroyed and the player is awarded a score based on the number of birds used and number of objects destroyed.

The Angry Birds project focuses on developing an artificial agent which automatically plays the game. It should take as input a screen-shot of the game and output a sequence of actions which maximises the score within the given time limit. Several areas of AI, including computer vision, knowledge representation and reasoning, planning and machine learning are involved in developing such an agent.

In order to reason meaningfully about the game state and produce a plan, the agent needs to identify objects in the screen-shot, with accurate information on their location, type and shape. This project focuses on developing a vision module which does this, by combining several common techniques in image processing and computer vision.

2 Background

At the start of this project, a basic vision module was provided. When given a screen-shot, the module first compresses every pixel to a 9-bit integer, by taking the top 3 bits in each of the r, g, b components. Query for specific game objects are then performed by searching for connected pixels whose value falls in the list of possible colours for the enquired object type. Output from the

module are bounding boxes around the object, determined by the maximum/minimum x and y coordinates found during the search. The image is processed very quickly and all objects in the scene can be found in less than 0.2 seconds on each of the first 21 levels.

However, results produced from this approach are not very reliable. For example, birds in the scene are often not detected when they move (e.g. jump up or spin around). This is a serious issue since the agent will have an incorrect knowledge on the number and type of birds it has and this likely leads to an incorrect plan.

Another issue with the current system is that the objects' shape and orientation are not detected. The following are some examples of when this create a problem for the AI.

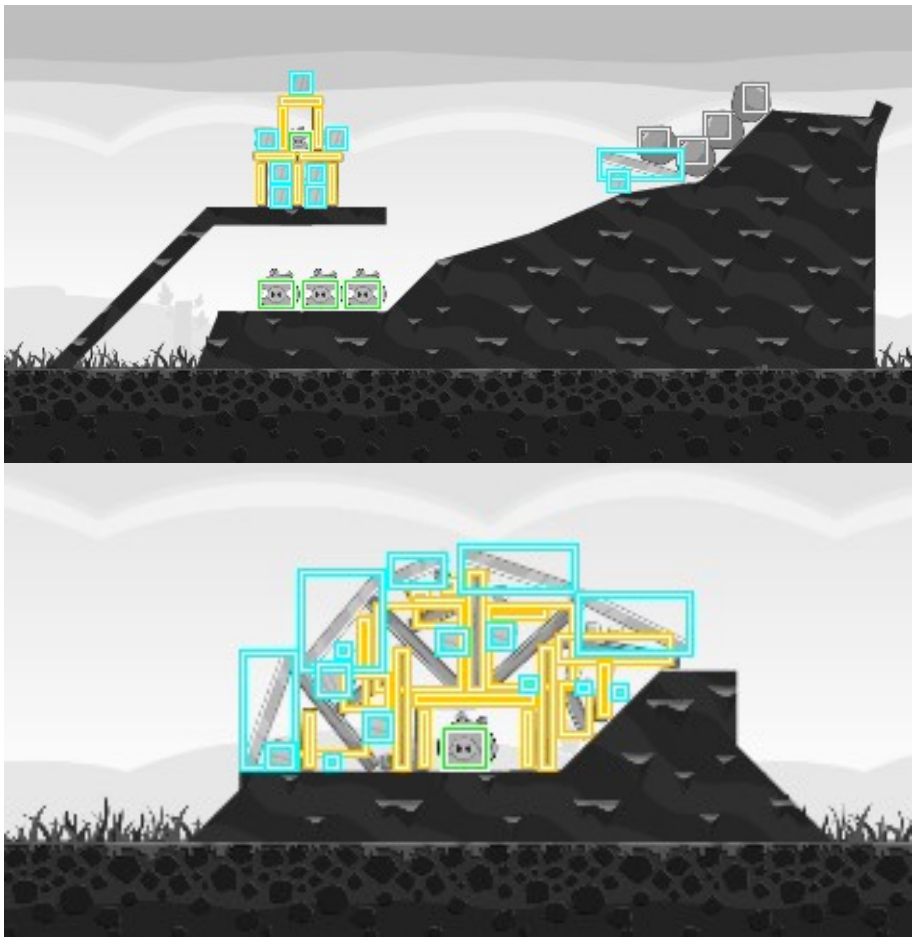

Figure 1. Examples where shape and orientation of the objects are important

In the first situation, the AI should be able to reason that the stones lying on the sloped terrain are round and attempt to make them roll down. It is not able to do that with the current approach since neither the slope, nor round stones are detected. The second example further emphases the importance of detecting object orientation, without which it is almost impossible to understand the game state.

Apart from these main issues, the existing system also sometimes detect stacked objects as one and often produce a significantly smaller bounding box than the actual object (as seen in Fig 1). All of

these make it difficult to understand the game state and reason for the agent.

Having evaluated the existing approach, it was decided a new method should be implemented. The new method aims to achieve the following objectives:

- Wood, ice and stone blocks should be detected with accurate shape and orientation
- Terrain should be detected and approximated by a polygon
- Slingshot, birds and pigs should be detected reliably and not affected by movements

3 Method

3.1 The main process

Given a screen-shot from the game, the implemented vision module find game objects in the scene in a 6 stage process, as explained below.

1. Assign a class label, such as red bird, wood or ice, to every point in the image, according to its colour
2. Use the custom defined edge detector to find edges in the image and mark them as boundary points on the label image
3. Find objects as connected-components of the same class on the label image
4. Examine contour of the connected-components to identify corners
5. Determine shape of the objects by examining their minimum and maximum bounding rectangles
6. Validate the detected objects by a set of constraints and objects passing the constraints are returned as output

3.2 Pixel Labelling

The first step taken by the vision module is pixel labelling. In this step, we attempt to determine the type of object each point in the screen-shot is most likely to be. This is done by simply examining colour of the point since most colours in Angry Birds are unique.

During implementation, a set of manually labelled training data was produced. It contains a set of points in the RGB space and their corresponding class (e.g. {160, 0, 0, red bird}), taken from a human segmented image. When a screen-shot is passed to the vision module, the Nearest-Neighbour algorithm is then used to classify each new point. That is, each point is assigned

the same class label as its closest neighbour in the training data.

The advantage of using the Nearest-Neighbour approach is that given a sufficiently large set of training data, an accurate decision boundary is produced. However, it has the issue of being very time consuming, since classifying each new point involves calculating the Euclidean distance between the new point and every point in the training data. To address this issue, a similar technique to the existing system is used. Given a new point, we first compress its colour to a 15-bit integer by taking the top 5 bits in each of the r, g, b components. The class label for each 15-bit colour is computed in an initialisation phase. Therefore, the task of classifying a new point is reduced to a table lookup operation.

Below is an example of the original image compared to its label image.

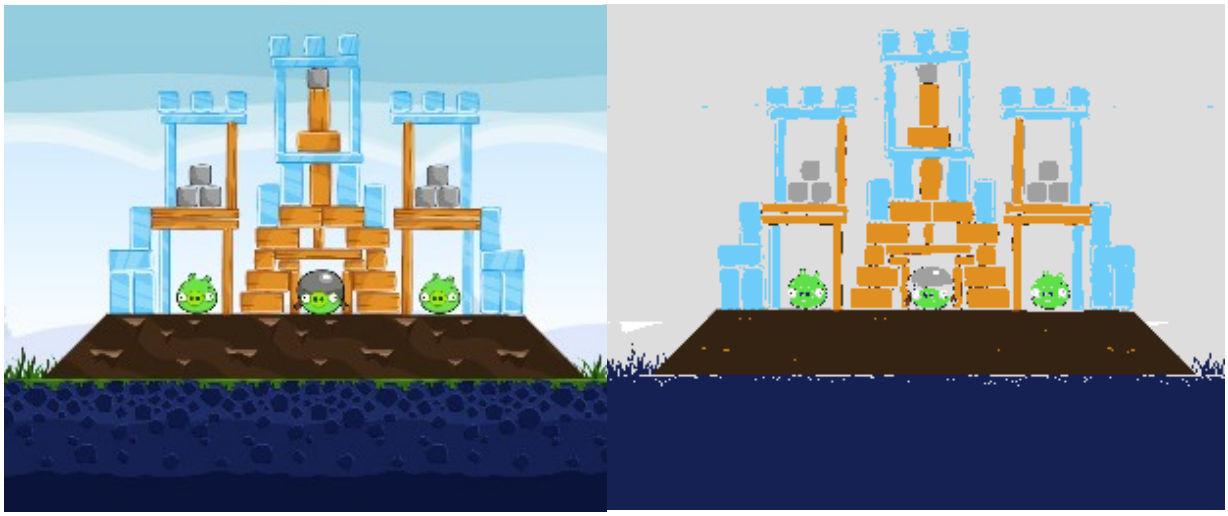

Figure 2. The original image (left) compared to its label image (right)

As shown in figure 2, the classification process correctly assign labels to most points in the image, with only a small part of the ice blocks mistaken for the background.

3.3 Edge Detection

As observed in figure 2, stacked objects of the same type often form a connected-component in the label image. This is especially the case for ice blocks. In order to separate them, a custom edge detection algorithm is implemented, using ideas from the Canny edge detector.

3.3.1 Canny Edge Detector

Despite it was developed over two decades ago, the Canny edge detector is still considered a optimal edge detector by many (Doychev, 2005). Canny edge detection identify edges in the image in a four stage process. First, the input image is smoothed by a Gaussian convolution to remove

noise. Intensity gradient is then calculated from the smoothed image in each of the horizontal and vertical direction. In this implementation, the Sobel operator, whose kernels are shown below, is used.

$$K_{Gx} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad K_{Gy} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Once the gradient in each direction has been obtained, the edge strength and orientation is calculated from them. The next step is called non-maximum suppression, which identifies local maxima in direction of the gradient. Non-maximum points are suppressed and this reduces the number of edge responses to one per edge. Finally, edge tracing with two thresholds is applied, where a point is marked as an edge if its gradient is greater than the higher threshold or if its gradient is higher than the lower threshold and it is connected to a point where gradient is high.

The implemented Canny edge detector produced results as shown below.

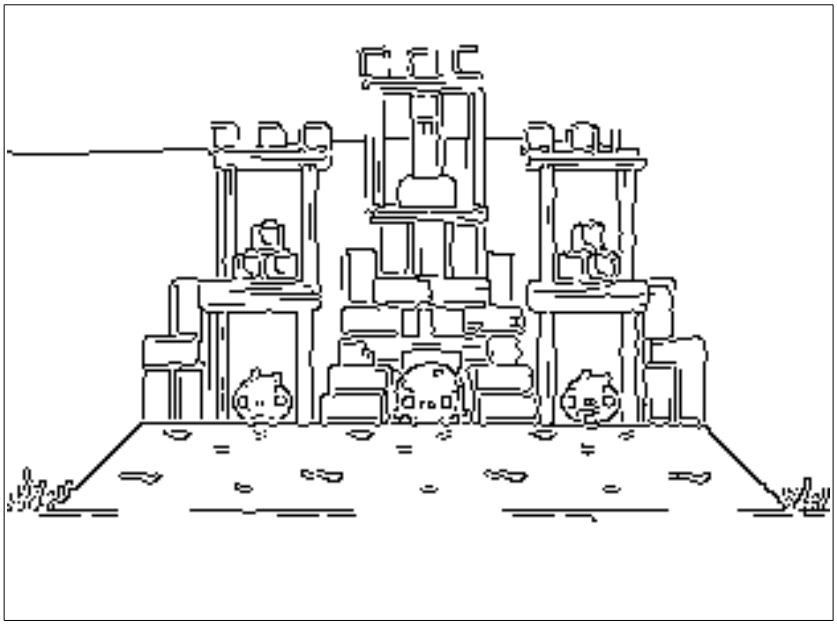

Figure 3. Result from Canny edge detection

As can be seen in figure 3, the standard approach fails to identify object boundaries in some cases, whereas in others false edges are detected inside the objects. This is because each object in Angry Bird contains colours at different intensity levels and the internal intensity change is sometimes greater than the change at boundaries. It was also noted that in many cases there were two responses to a single edge. This is is because most edges in Angry Birds are roof edges, i.e. edges where the intensity values on different sides of it are the same. This causes a problem for the Sobel operator, which compares these two sides.

3.3.2 Colour space and similarity measure

To address issues with the standard Canny edge detection, a modified approach is taken. First of all, instead of representing each point with its intensity value alone, the HSI (hue, saturation and intensity) colour space is used instead. The HSI model follows closely to the human perception of colour qualities. Hue represents the fundamental colour, saturation represents the amount of colour present and intensity represents the amount of light (Wesolkowski, 1999). The following formulae are used to calculate each of their values from the original RGB representation (Hanbury, 2008).

$$H = arctan2\left(\sqrt{3}(G - B),\ 2R - G - B\right)$$

$$S = \frac{max(R, G, B) - min(R, G, B)}{max(R, G, B)}$$

$$I = \frac{(R+G+B)}{3}$$

From this representation, it was found that while the intensity changed significantly inside each object, there was only a minimum change in hue. For example, common intensity values for ice blocks varied from 174 to 240, while the hue (in degrees) only changed from 195 to 202. Therefore, when comparing two points whose hue values are defined, the similarity measure is defined by a weighted Euclidean distance in the hue and saturation plane. Intensity based similarity measure is only used as a backup when hue is undefined (when R = G = B), as shown below.

$$D(\vec{a}, \vec{b}) = \begin{cases} \sqrt{w_h(a_h - b_h)^2 + w_s(a_s - b_s)^2}, & \text{if } a_h \text{ and } b_h \text{ defined} \\ w_v|a_i - b_i|, & \text{otherwise} \end{cases}$$

3.3.3 Edge strength

Using the similarity measure above, we define the individual edge strength of each point in the horizontal, vertical and two diagonal directions as follows. Consider a 3x3 window centred at a point, with its neighbours labelled from 1 to 7, as shown below.

2	1	8
3	0	7
4	5	6

Figure 4. Position of neighbourhood pixels

The individual edge strength of the position 0 in direction $i \in \{1, 2, 3, 4\}$ is defined by

$$G_i = D(\vec{v}_0, \vec{v}_i) + D(\vec{v}_0, \vec{v}_{i+4})$$

where $\vec{v}_i$ is vector value of the pixel in position *i.* This measure compares the point to two of its neighbours in the edge direction enquired and can effectively detect edge points when $\vec{v}_i$ and $v_{i+4}^{\rightarrow}$ are similar.

Another factor considered is that a point is more likely to be an edge in a given direction if its neighbours in the perpendicular direction are also edges. On the other hand, if neither of its neighbours are edge points, then the edge strength is likely to be caused by noise. To take the cross-correlation into account, the individual edge strength calculated are then averaged in the perpendicular direction. This removes noises and fills up holes in the detected edge.

3.3.4 Non-maximum suppression and hysteresis thresholding

One of the problems found with Canny edge detection is that edge points near corners are often disconnected, even if a very low threshold is used. The cause of this issue is that near corners, the edge orientation is often calculated incorrectly, resulting in the point being suppressed when it should not be.

To fix this, instead of calculating a single edge strength and direction for each point, the edge strength in all four directions defined earlier are kept. Non-maximum suppression and hysteresis thresholding are applied to each direction individually to produce four directional edge maps. These directional edge maps are then combined to form the final set of edge points. This process produces closed boundary around objects, at the expense of some double edge responses.

By experimenting on first 21 levels, a good set of weights for the similarity measure are found as $w_h = 400$, $w_s = 12$ and $w_i = 7$. The thresholds used for hysteresis thresholding are 125 and 300. Here is a resulting edge image from the modified approach.

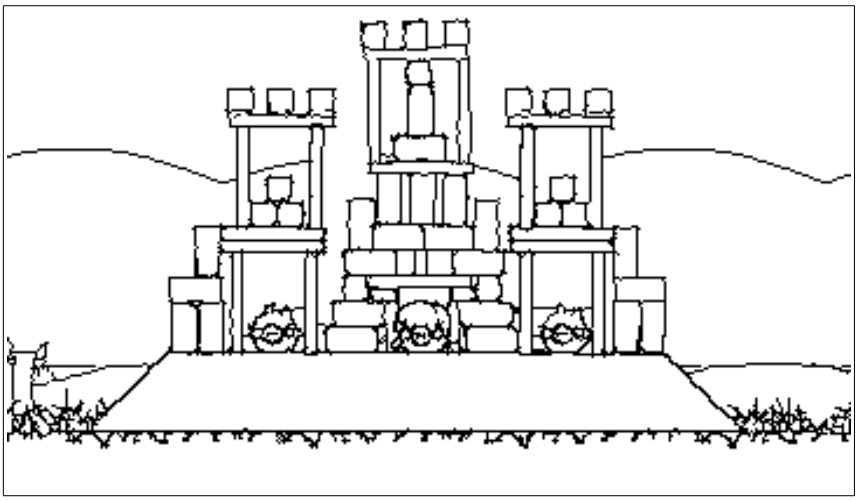

Figure 5. Edge image produced by the modified approach

As can be seen, for the terrain, wood, ice and stone, boundary points are detected properly while the internal points are not affected. The slingshot, pigs and birds, however, are over-segmented by this approach since they are drawn using a combination of colours. Therefore when marking boundaries on the label image, points labelled with slingshot, birds or pigs are skipped.

3.4 Connected-components

Having determined the class label for every point in the image and object boundaries, objects can be detected easily as connected-components. This is done by running breadth-first-search from each point in the image, treating the image as a graph, with the nodes being its pixels and two nodes are connected if and only if they are neighbours of each other and have the same class label.

During the breadth-first-search, information which will help us determine the object's shape are recorded. These include

1. Area of the object - number of points in the connected-component
2. Boundary points - points visited during the BFS but not marked as part of the object
3. Extreme points – leftmost, rightmost, topmost and bottommost points

3.5 Shape detection

As discussed earlier, obtaining accurate descriptions of the objects' shape is critical for understanding the game state and producing a good plan. Therefore, it is an important requirement for this project. The required shapes in Angry Birds include circles, rectangles and polygons. In order to detect these shapes, a combination of two methods has been used.

3.5.1 Minimum/Maximum bounding rectangles

To determine what shape the detected object is most likely to be, bounding rectangles are drawn around it in angles between 0 and 90 degrees. To determine size of the bounding rectangle around the object for an arbitrary angle θ, boundary points of the connected-component are first rotated by θ around the origin, using the following formulae

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

Once this is done, size of the bounding rectangle is simply determined by the maximum and minimum of the x and y coordinates, as shown below.

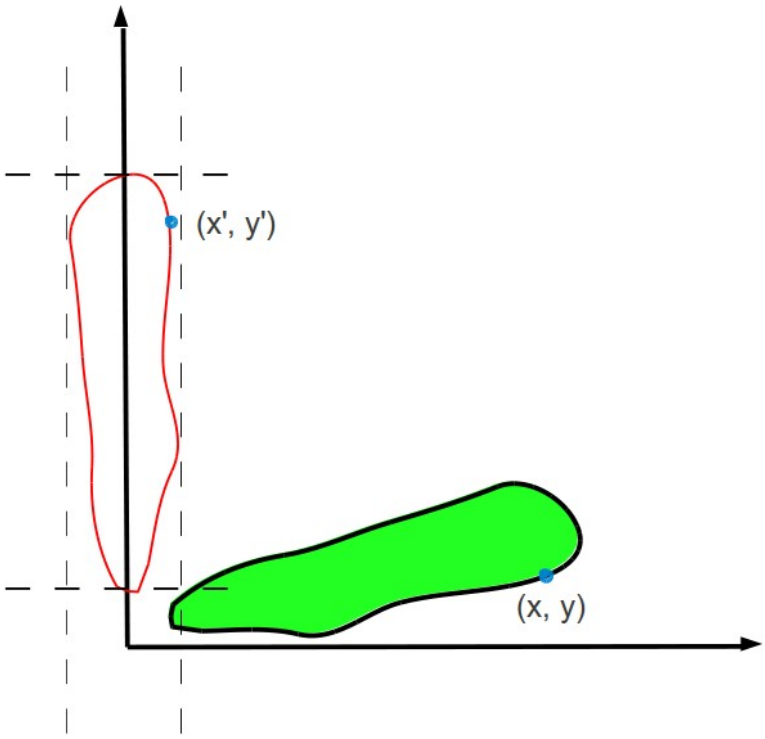

Figure 6. Rotating object boundaries to find its bounding rectangle

During this process, the minimum and maximum area observed are recorded and this information is used to determine the most likely shape of the object. Two thresholds are used to check if the object is a polygon or a circle.

1. Polygon: minimum area > actual area * t1
2. Circle: maximum area < minimum area * t2

The first test says that, if minimum area of the bounding rectangles is significantly larger than actual area of the object recorded earlier, then the object is likely to be a polygon. The polygon is then constructed by linking a list of detected corners, using a process discussed in the next section. The second check means that if the minimum and maximum areas are very close to each other, the object is likely to be a circle. Centre location of the circle is computed as an average of the extreme x and y coordinates, whereas diameter is computed as an average of its width and height.

If neither of these constraints are satisfied, then shape of the object is mostly to be a rectangle, whose size and orientation are the same as the minimum bounding rectangle found.

The general issue with this method is that all points on the object boundary has to be rotated many times. This can be very time consuming. However, bounding rectangles around an object are in fact only determined by a small portion of the boundary points, namely the corners. Therefore, contour tracing is conducted in advance to identify corners of the connected-component.

3.5.2 Contour tracing

Contour tracing is a common technique performed on digital images. The goal of contour tracing is to produced an ordered sequence of boundary points, from which general shape of the object can be extracted. For this project, the Moore-Neighbour tracing algorithm is implemented.

The general idea of the Moore-Neighbour algorithm is to scan the image from left to right (or any other direction) until a filled pixel is reached. From this pixel, we visit each of of its Moore Neighbour (8 points connected to the pixel) in a clockwise/anticlockwise direction, until a new filled pixel is found. Filled pixels are added to the boundary sequence in the order they are visited. The process continues until the starting point is reached again (Ghuneim, 2000).

Once sequence of boundary points has been produced by the Moore-Neighbour algorithm, local angle of each point is approximated by looking ahead in the sequence by a small amount *k*.

$$\theta_i = \arctan\left(\frac{y_{i+k} - y_i}{x_{i+k} - x_i}\right) \text{ for all } i <= n - k$$

where (x_i, y_i) is the *i*-th point in the boundary sequence and *n* is its length. The angle change between neighbouring points are then examined as follows.

1. We start from the first point and proceed in the sequence. For each new point, compare its angle to the previous. Direction of change, which may be clockwise, anticlockwise or no change, is found.
2. If the found direction is the same as the previous one, its magnitude is added to the accumulated change.
3. If the found direction is different from the previous one, we check that if the accumulated change is above a threshold T, in which case it is marked as a corner. Regardless of the result, the accumulated change is reset to 0.
4. The process continues until we reach end of the boundary sequence.

The tracing algorithm described above produces only a single response to each sharp corner, while remaining sensitive to small angle changes. Corners detected due to noise are then removed by joining consecutive lines which have similar angles with each other.

Example of a detected connected-component with its corners identified is shown below.

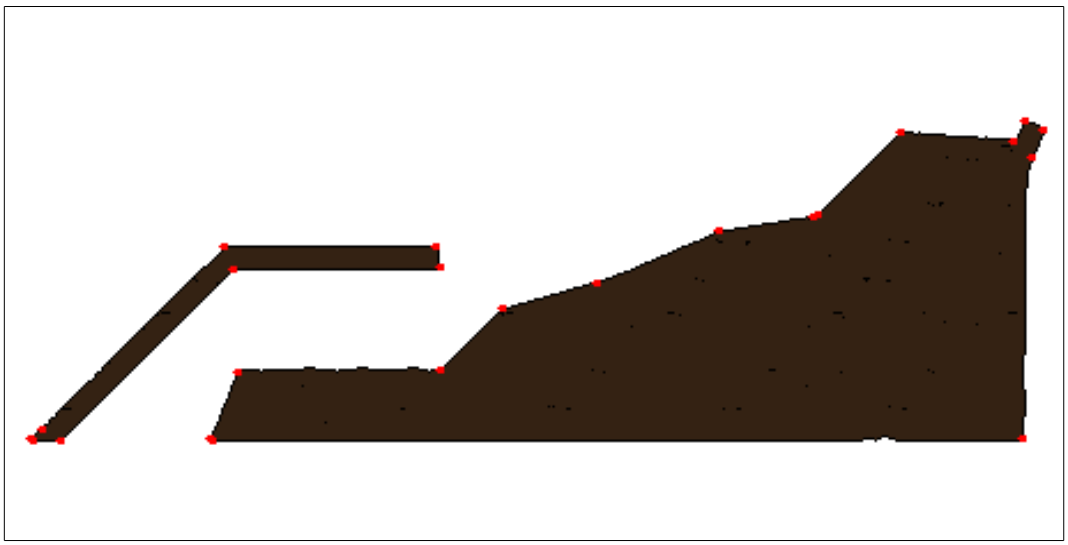

Figure 7. Detected terrain and its corners marked in red

As shown in figure 7, all corners of the terrain are properly detected, even when the angle change is very small. The detected corners are used to determine bounding rectangles of the object and are joined to form a polygon if that is the determined shape.

3.6 Validation

Once all connected-components in the scene have been detected with information on their shape, location and type. A set of final checks on size and location of the objects are applied to verify that they are in fact valid game objects.

3.6.1 Size

The first check applied to the objects is based on their size. Every known object in the game is assigned a maximum and a minimum area. Connected-components whose size fall out of this range are removed from the result. In addition, if the object is detected as a rectangle, a minimum width check is also applied to remove noise.

3.6.2 Location

The main purpose of applying the location check is to detect birds and trajectory points reliably. First of all, the assumption that birds do not overlap with each other in the x-coordinate is made. This allowed us to use very small minimum sizes for detecting birds (in fact, the minimum size for the blue bird is just 2 pixels). A similar approach is taken with detecting trajectory points, where the minimum size is defined as 1 pixel. The assumption that trajectory points start from the slingshot's launch point and are all within a maximum distance of each other is made. This resulted in a very reliable detection of of the birds and trajectory points.

4 Results

When tested on my laptop, the produced vision module find all objects in the game in less than 0.4 seconds for each of the first 21 levels. This is not considered a significant increase compared to the previous system, since taking actions in the game requires much longer.

On the other hand, quality of the produced result was significantly improved. Below are some results produced by the system.

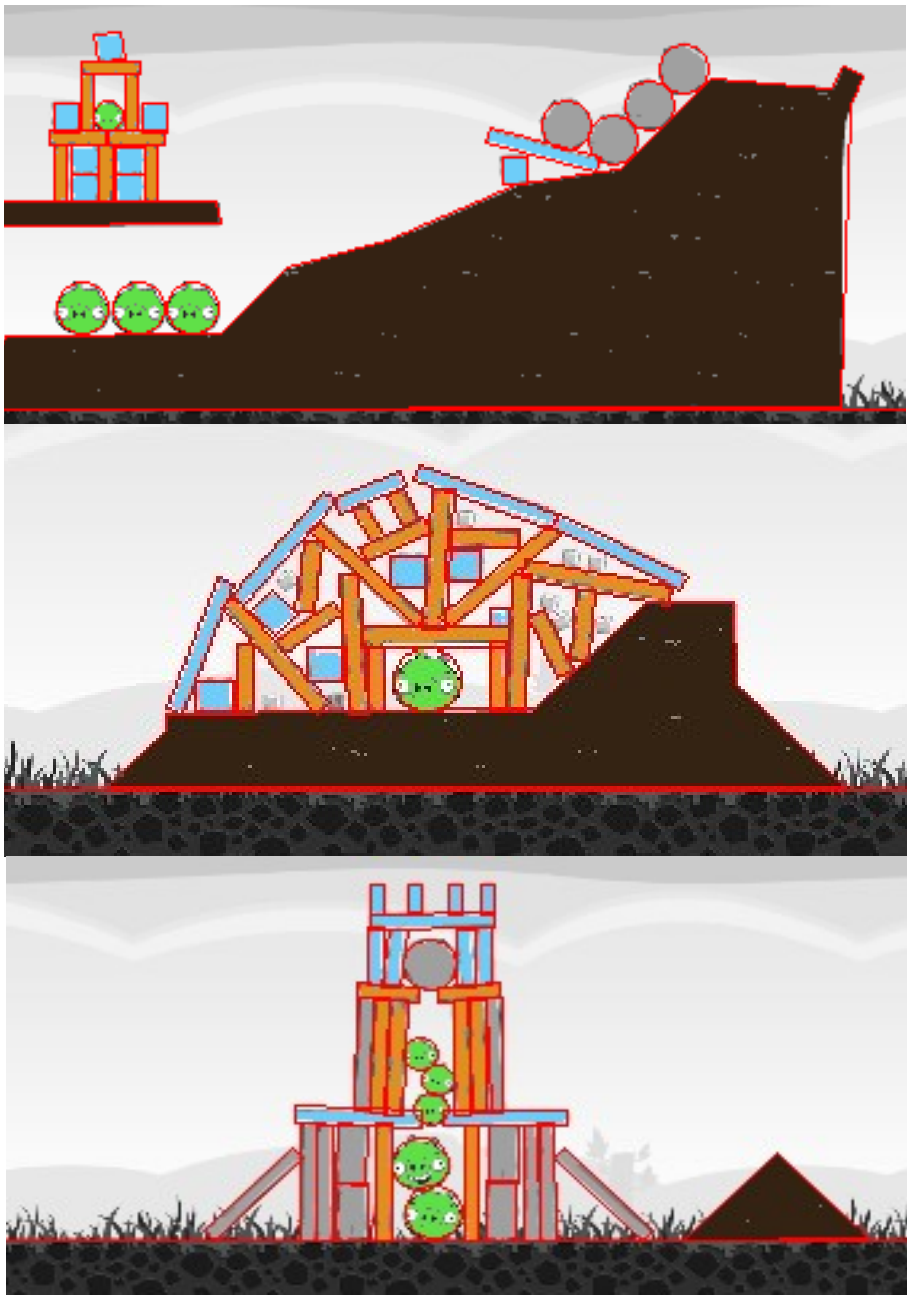

Figure 8. Sample outputs from the vision module

As can be seen above, known objects are detected quite accurately with their shape and orientation information. The only exception are small ice blocks, which are sometimes detected with an incorrect orientation or size. In addition, the produced module detects all of the red, blue, yellow,

white and black birds reliably and is not affected by the birds' movements.

5 Conclusion & Future Possibilities

In this project, a new computer vision module for agents playing the Angry Birds computer game has been produced. Given a screen-shot from the game, it is able to detect all known objects in the first 21 levels as well the the white and black birds. In addition, it is able to describe each object in the game by either a rectangle, circle or a polygon. Compared to the previous system, the results showed a significant improvement in quality of the objects detected, with only a small increase on the time requirement.

A possible future direction for this project is to extend the system to detect unknown objects as well. It was observed that the edge detector developed is able to draw closed boundaries on the image. This could be used to extract segments from the image and then mark the object as unknown if its colour is different from any known class.

6 References

F, Albregtsen, 2010, *Region & Edge Based Segmentation*, viewed at <http://www.uio.no/studier/emner/matnat/ifi/INF4300/h10/undervisningsmateriale/INF4300-2010-f03-segmentation.pdf>

Z. Doychev, 2005. *Basics of Medical Image Segmentation and Registration,* viewed at <http://campar.in.tum.de/twiki/pub/Chair/TeachingWs05SegmentationRegistrationHauptseminar/02DoychevEdgesFeaturesHandout.pdf>

A. Hanbury, 2008. *Constructing Cylindrical Coordinate Colour Spaces*, viewed at <http://muscle.ercim.eu/images/DocumentPDF/Hanbury_PRL.pdf>

A. Ghuneim, 2000. *Contour Tracing Algorithms,* viewed at <http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/alg.html>