Aerial Robotics Kharagpur Software Tasks Documentation

Anindya Sikdar

March 2020

Abstract— This documentation specifies the approach and results for ARL Software Team task 1 and task 3.

For task-1 I have implemented **MINIMAX algorithm**. Here the computer plays with a person and always tries to at least make a draw.

For task-2 I could only figure out the centroids of the balls but could not progress any further.

For task-3 I have done the face detection part using **Haar Cascade Classifier**.

1 INTRODUCTION

TASK-1:

In the first task we have to write an agent to play the **TIC-TAC-TOE** game using **MINIMAX Algorithm**. Here we've used the **gym-tictactoe environment**.

MINIMAX Algorithm is basically an exhaustive search of the game-space. Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the other player is also playing optimally.It is used in games such as tic-tac-toe, go, chess, Isola, checkers, and many other two-player games. Here we use it to implement our **TIC-TAC-TOE** game.

The first difficulty I faced was that I had no idea of Object Oriented Programming (OOP) and python. So, the very first thing I did was learning python and OOP. Then I moved on to understanding the **MINIMAX Algorithm**. I referred several sources and after having a thorough understanding of the algorithm I started understanding the gym environment. Implementing the algorithm in python wasn't that of a tough job.

TASK-2:

In this task we are given images of a billiard pool table with the cue and ball to be hit indicated. We have to find the pairs of collisions assuming perfectly elastic ball-ball collisions and ball-table collisions. I could only calculate the centroids of the balls initially using image moments calculation but could not progress further

TASK-3:

In this task there are two parts. In the first part we're supposed to implement an algorithm which can detect face and track its motion. In the second part we're supposed to implement a game where the user will use his face to navigate a game object.

Here I could only do the first part. It was pretty challenging to me as I didn't attended the Winter Workshop and had no idea of computer vision and openCV. So, I tried learning the basics of openCV from the documentation and then I learnt about **Haar Cascade Classifier** and used it for face detection.

2 PROBLEM STATEMENT

TASK-1:

The first task involves the implementation of the well known **MINIMAX Algorithm** which enables the agent(computer) to play with human giving the best possible move in each of his turn assuming the player also gives the best possible move in each of his turn . Here we are already given an environment to work known as the **gym-tictactoe environment** where all the necessary functions , classes , objects and variables are predefined. Our task is to complete the function **'act'** in the **minimax_agent.py**. Here we have to write the **MINIMAX Algorithm** so that that the function always returns the best possible move for the agent.

TASK-2:

The second task involves computer vision and openCV. In this task we are given images of a billiard pool table with the cue and ball to be hit indicated. We have to find the pairs of collisions assuming perfectly elastic ball-ball collisions and ball-table collisions

TASK-3:

The third task mainly belongs to computer vision and openCV. Here first we have to implement an algorithm that performs face detection . We are permitted to use openCV's inbuilt library for face detection. Then we have to track it's motion using some algorithm. Once face detection and tracking of its motion work perfectly then we have to implement a game where the user can navigate an in-game object using his face.

3 RELATED WORK

TASK-1:

The first task is pretty straight-forward. It's a kind of a classic problem in game theory. **MINIMAX Algorithm** is a well-known brute-force AI strategy in Game Theory. It's a popular algorithm used in implementation of several classic games like tic-tac-toe, go, chess, Isola, checkers, and many other two-player games. This Brute-force strategy can be optimized using **Alpha-Beta Pruning**.

TASK-2:

I could only find the centroids of the balls using image moment. For this I referred to this website –`https://www.learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp`

TASK-3

The problem of face detection was a well known and crucial problem until two eminent computer scientist Paul Viola and Michael Jones proposed `https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf` . It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. I used the Haar Cascade Classifier for face detection.

For tracking purpose I used the **Centroid Tracking Algorithm** which is again a simple and easy to implement algorithm for tracking of objects.

4 INITIAL ATTEMPTS

TASK-1:

The first thing I did in my attempt to solve the very first problem was to learn python and get a basic understanding of Object Oriented Programming. For learning Python I referred to a course in udemy –"Master Python Programming: The Complete Python Bootcamp 2020 ". After getting thorough with the python concepts I started learning the **MINIMAX Algorithm** from different sources– `https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/` ,`https://towardsdatascience.com/tic-tac-toe-creating-unbeatable-ai-with-minimax-algorithm-8` This is all I did for the 1st task before mid-semester exam. Post Mid-sems I started writing the code. Writing the code wasn't a tough thing to do as all we needed to do was to understand the code-base and complete the **act** function.

TASK-2:

For this task I found the centroids of the balls using image moments. For this I used findContours function and calculated image moments. Although I thought of an approach but I could not implement it as I just learnt openCV.

TASK-3:

The third task was kind of difficult to me as I had no idea of openCV and Computer Vision. So, before the Mid-semester examination all I did was to learn openCV from my friend's documentation of the winter workshop and get the basic idea of computer vision. Post Mid-sems I started learning face recognition and motion tracking. I studied the famous **Voila-Jones Algorithm** used for face detection. I learnt about the ML based **Haar Cascade Classifier** from `http://www.willberger.org/cascade-haar-explained/`. Then I learnt about motion tracking. Although there are several advanced algorithm to implement this , **Centroid Tracking Algorithm** is one of the simplest. For this I referred -`https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/`.

5 FINAL APPROACH

TASK-1:

There is no difference between my initial attempt and final attempt. It was important to understand the environment and all of its relevant functions,classes,objects etc. The algorithm I used for the **act** function is as follows–

```
def act(self, state, ava_actions):
    board,mark=state
    nboard = list(board[:])
    if check_game_status(nboard)<0:
        min=100
        max=-100
        action_min=ava_actions[0]
        action_max=ava_actions[0]
        if mark=='O':
            for action in ava_actions:
                nboard[action]=1
                mark=next_mark(mark)
                value , q =self.act((tuple(nboard),mark),
                [p for p in ava_actions if p!=action])
                if (value < min):
                    min = value
                    action_min = action
                nboard[action]=0          #backtrack
                mark=next_mark(mark)
            return min,action_min
        else:
            for action in ava_actions:
                nboard[action]=2
                mark=next_mark(mark)
                value ,m =self.act((tuple(nboard),mark),
                [p for p in ava_actions if p!=action])
                if (value > max):
                    max = value
                    action_max = action
                nboard[action]=0          #backtrack
                mark=next_mark(mark)
            return max,action_max
    else:
        return check_game_status(nboard),12
```

what this code does is basically for a given move of the human agent it searches the whole game space i.e it generates each possible move from the present state of the board and selects the best possible move for the current state under the assumption that the opponent always gives the optimal move. So, how does it select the best possible move? Each state is actually assigned a utility value which is an indicative of how good the move is. At any level it considers the move which has the maximum utility value for the minimax agent and for the human agent it considers the move which has the minimum utility value and returns that value and the corresponding move also. This is done in the inner if..else block of the code. So, after the whole recursion process is complete it returns the best possible move for the minimax agent.

TASK-2:

I could find the centroids of the balls using findContours and openCV. The code is given below–

```
import cv2
import numpy as np

img = cv2.imread('1.png')
img = np.asarray(img, dtype=np.uint8)
gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray_image,50,255,0)
im2, contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
for c in contours:
        # calculate moments for each contour
    M = cv2.moments(c)
     if M["m00"]!=0:
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
     else:
        cX,cY=0,0
        # calculate x,y coordinate of center
     cv2.circle(img, (cX, cY), 5, (255, 255, 255), -1)
     cv2.putText(img, "centroid", (cX, cY ),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
     255, 255), 2)
# display the image
cv2.imshow("Image", img)
cv2.waitKey(0)
```

TASK-3:

For the task 3 I used the **Haar Cascade Classifier** which is basically a ML based algorithm. In this algorithm the cascade function is trained with a large no of positive and negative images. A window gradually hovers around the given image.Each image (or the portion of the image the window covers) goes through a number of HAAR features or stages. HAAR feature is basically adjacent rectangular regions and the pixels in each of the regions are summed up and then subtracted accordingly. If it matches a certain threshold value , then it passes this stage and the other HAAR features are applied. If it doesn't passes then this image (portion of the image) is rejected and the window moves on. So, if any portion of the image satisfies all the HAAR features ,then it is considered to be the desired object. So, in this algorithm each step is basically a weak classifier and all the classifiers together make a strong classifier of the object. This very technique is used for face detection as well.

6 RESULTS AND OBSERVATION

TASK-1:

The algorithm works pretty well as I tested it a number of times. But there are some drawbacks.These are mentioned as follows–

1.The very first step takes some time to occur which is not observed in the subsequent steps. The possible reason may be the fact that the cardinality of the game-space is maximum in the beginning.

2.At any level there can be multiple states which have the same utility value but some stage may lead to victory earlier. My algorithm doesn't take account of this issue.

3.In some cases I noticed that some other move could lead to victory while the agent chooses the less optimal move which leads to a draw match.

TASK-3:

The first part is working fine as it detects the face in the image properly. However I could not do the second part of this problem i.e implementing this in a game.

7 FUTURE WORK

TASK-1:

Although the algorithm works well there are some drawbacks as I mentioned earlier. The possible remedies may be as follow—

1. The delay in the initial step may be overcome by considering a separate case of the 1st move. But I didn't do this and kept the general algorithm only.

2.To further optimize the algorithm I could've compared the depth of a particular state node and then comparing this depth value I could have derived the most optimal move. I will surely implement this in my code in the future.

3. There are cases where the agent chooses the less optimal move leading to a draw game while more optimal move was available. I could only identify one case but there may be others also. I could not find where I'm doing the mistake.

TASK-3 There is a lot of work to do in this task as I did it partly only. I am clueless about using the face detection and motion tracking in playing a game. I will require some more time learning how to do these stuffs.

8 CONCLUSION

This was really a fun journey cracking the problem statements of ARK. I learnt a lot like Python,OOP,bits of game theory and AI and of course computer vision and openCV. The 1st task was rather simple. The only difficulty was understanding the algorithm in depth and using the gym environment to write the code. This is my first time learning AI and it was really interesting. **MINIMAX Algorithm** is a very common algorithm in Game Theory and used extensively in designing classic games like Chess,Tic-Tac-Toe etc. This task was certainly a good hands-on introduction to AI and Game Theory.

The third task seemed pretty tough to me as it contained computer vision and openCV of which I had no idea before. It consists of face-detection and motion tracking. Face-detection and motion tracking are pretty important when it comes to Robotics. So, this task gave a first-time experience of computer vision which was pretty amazing.

9 REFERENCES

[1]https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

[2] https://towardsdatascience.com/tic-tac-toe-creating-unbeatable-ai-with-minimax-algorithm-8af9e52c1e7dpar

[3] Paul Viola and Michael Jones "Rapid Object Detection using a Boosted Cascade of Simple Features"

[4]https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/