ECE 8540
Analysis of Tracking Systems

Assignment 7
Simulated Behavior-Based Robot

Vivek Koodli Udupa
C12768888

December 6, 2018

1 Introduction

This report considers the problem of modeling a behavior-based robot. Behavior based robots are robots that take action based on sensor reading and a set of prioritized actions. Behavior based robots are given a behavioral repertoire to work with dictating what behaviors to use and when to use them. Rather than building world models, behavior-based robots simply react to their environment and problems within that environment.

Behaviors are set of actions that help the robot make some decision based on the sensor reading. For example, the robot could be given the instruction to go right if there is a wall in front and go left if there is a wall in front and another wall to its right. Sensor's would help the robot detect the walls, and based on the position of the walls the robot would either go left or right.

This report describes the modeling of a simulated robot which survives in shark infested waters. The robot's behavior includes running away from the sharks, hunting for food and avoiding collision with other similar robots.

2 Methods

The simulated world consists of sharks, food and robots. The sharks consume food as well the robots to survive. The robots are modeled to run away from the sharks and consume food to survive. If the robot runs out of energy, it cannot move any further and hence turns into food for the sharks. This section describes the process of modeling the robot, "VicTron", to make survival decisions based on the position of sharks, food and other robots.

The action sequences given to VicTron are prioritized. This facilitates the robot to choose the best among the available actions. The priorities are ordered as:

- If the current energy is greater than a specified threshold then enter power mode. Power mode priorities are as follows:
  1. Run away from shark
  2. Go towards the nearest food
  3. Avoid collision with other robots
- If the robot's energy drops below the minimum energy threshold, then switch to survival mode. Survival mode priorities are as follows:
  1. Go for the nearest food
  2. Run away from sharks
  3. Sleep

2.1 Implementation

Robot starts off with an energy of 255. Each movement costs certain amount of energy. Amount of energy spent is directly proportional to the speed of the robot. The direction of motion is specified by angles in degrees.The goal is to survive as long as possible. The following variables are used to implement the above mentioned behavior.

1. FoodClosestDistance : Distance to closest food in pixels
2. FoodClosestAngle : Angle in degrees towards closest food
3. RobotClosestDistance : Distance to other closest robot, in pixels
4. RobotClosestAngle : Angle in degrees towards closest robot
5. SharkClosestDistance : Distance to closest shark, in pixels
6. SharkClosestAngle : Angle in degrees to closest shark
7. CurrentRobotEnergy : Robot's current energy
8. *RobotMoveAngle : Angles in degrees to move
9. *RobotExpendEnergy : Energy to expend in motion
10. RunFromShark : Distance to shark below which the robot starts to run away from the shark
11. MinEnergy : Minimum energy below which the robot enters survival mode

12. RunForFood : Distance to food below which the robot starts to follow and hunt that food item
13. PersonalBubble : Distance to other robot, below which the VicTron starts to avoid collision
14. prevAngle : Static variable used to simulate memory of previous movement angle
15. angleInc : Incremental angle steps

2.2 Power Mode

In this mode, the robot has sufficient energy to move faster. Hence the first priority in this mode is to avoid sharks. As soon as the distance to the closest shark drops below the RunFromShark threshold, VicTron spends 30 energy units to move fast and avoid being consumed by the shark. The angle of movement is calculated as (prevAngle + SharkClosestAngle) mod 360. "prevAngle" is a static variable that remembers the previous movement angle. This angle is incremented by $30°$ every execution. This allows VicTron to escape the shark in a circular motion, rather than just randomly moving away. This also allows VicTron to target any food behind the shark. The rotation is bounded from $90°$ to $270°$. This is to prevent VicTron from going in a loop and landing right into the sharks mouth.

The second priority is to look for food. VicTron again expends 30 energy units to move in the direction of food.

The third priority is to avoid collision with other robots. So if any other robot enters VicTron's vicinity defined by the variable "PersonalBubble", VicTron moves in a direction perpendicular to that robot by expending 5 units of energy to avoid collision.

The final priority is executed when nothing is close by. Here VicTron moves slowly towards food.

2.3 Survival Mode

VicTron enters survival mode when the its energy drops below MinEnergy. Here the first priority is to look for food. Since it is low on energy, it moves very slowly towards food. The energy expenditure is calculated as (5 + FoodClosestDistance / 10). This makes the robot gradually speed up as the food gets closer. This helps is better expenditure of remaining energy.

The second priority is to run away sharks. Only 14 units of energy are spent as VicTron is already low on energy.

Final priority is to go to sleep. If there is no shark or food nearby, then the robot goes to sleep. Here no energy is spent thus conserving the remaining energy to hunt food as it comes close.

2.4 Implementation Notes

The above implementation was done in C and the implementation code is attached in the Appendix section of the report.
Case-sensitivity is ignored while using variable names. This is done with the intention to help better understand the implementation code. Thus same variable names were used in the report with the same cases as it appears in the implementation code.

3 Results

On average VicTron survived longer than other robots when the total amount of food was set to 30 and number of sharks were set to 3. Survival was difficult when the total amount of food was reduced to 5.

Overall performance of VicTron was better than other competitor robots. The survival of the robots depends upon the initial spawning position, spawning position of sharks and food and the simulation parameters such as total amount of food, number of sharks and number of competitor robots.

4 Conclusion

The report describes the behavior modeling of robot, VicTron, to achieve longest survival time in the simulated world. Importance of prioritizing the behaviors has also been addressed in this report. The co-relation of survival time and simulation parameters were studied and different behavioral strategies were implemented to handle varying simulation parameters.

References

[1] `https://en.wikipedia.org/wiki/Behavior-based_robotics`

[2] `http://cecas.clemson.edu/~ahoover/ece854/`

Appendix

Name of the Robot

VicTron was derived by combining words Victory and Tron. This is to say that the robot will be victorious and Tron just makes it sound like a cool robot. Also Vic is my nickname.

Implementation code

```
void VicTron(int FoodClosestDistance,    /* input - closest food in pixels */
    int FoodClosestAngle,        /* input - angle in degrees towards closest food */
    int RobotClosestDistance,    /* input - closest other robot, in pixels */
    int RobotClosestAngle,        /* input - angle in degrees towards closest robot */
    int SharkClosestDistance,    /* input - closest shark in pixels */
    int SharkClosestAngle,        /* input - angle in degrees towards closest shark */
    int CurrentRobotEnergy,        /* input - this robot's current energy (50 - 255) */
    int *RobotMoveAngle,        /* output - angle in degrees to move */
    int *RobotExpendEnergy)        /* output - energy to expend in motion (cannot exceed Cu

{
    // Define the Trigger Points
    //static int MoveAngle = 2;
    int RunFromShark = 80; //Closest a shark can get
    int MinEnergy = 20; //Energy below which hunger strikes
    int RunForFood = 100; //Run when food gets this close
    int PersonalBubble = 5; //When other bots come too close, run away!
    static int prevAngle = 180; //Static to store previous value for next cycle
    int angleInc = 30; //Angle Increment step

    /*Priorities:
    When energy is high:
    1: Move away from the shark!!! scary shark
    2: Go for food! Yumm
    3: Do not get blocked by other robots!
    4: Just slowly move around looking for food
    When energy is low:
    1: Go to the closest food! Hungry!!
    2: Run away from the shark if it gets too close!
    3: Just stay idle and do not spend any energy if food and shark is far away!
    */

    //Change angles at an increment of 30 degrees every time a shark gets close. Go from 90
    if (prevAngle > 270) {
        prevAngle = 90;
    }
```

```
//Do good stuff if we have sufficient energy
if (CurrentRobotEnergy > MinEnergy)
{
    if (SharkClosestDistance < RunFromShark)
    {
        //Shark too close, just run away!
        (*RobotExpendEnergy) = 30;
        (*RobotMoveAngle) = (prevAngle + SharkClosestAngle) % 360;
        prevAngle = angleInc + prevAngle;
    }
    else if (FoodClosestDistance < RunForFood)
    {
        //Food! Yay
        (*RobotExpendEnergy) = 30;
        (*RobotMoveAngle) = FoodClosestAngle;
    }
    else if (RobotClosestDistance < PersonalBubble)
    {
        //Other bot is too close, move away from it
        (*RobotExpendEnergy) = 5;
        (*RobotMoveAngle) = (90 + RobotClosestAngle) % 360;
    }
    else
    {
        //Nothing is close.. Move randomly to catch food by luck
        (*RobotExpendEnergy) = 10;
        (*RobotMoveAngle) = FoodClosestAngle;
    }
}
//Survival Mode: Now we are low on energy. Food is Priority.
else
{
    //Go for food first! Need food to run away
    if (FoodClosestDistance < RunForFood)
    {
        (*RobotExpendEnergy) = 5 + FoodClosestDistance/10;
        (*RobotMoveAngle) = FoodClosestAngle;
    }

    //If shark gets too close, run. The dead cant eat!
    else if (SharkClosestDistance < RunFromShark)
    {
        (*RobotExpendEnergy) = 14;
```

```
            (*RobotMoveAngle) = (prevAngle + SharkClosestAngle) % 360;
            prevAngle = angleInc + prevAngle;
        }

        //Just stay idle until something comes close.
        else
        {
            (*RobotMoveAngle) = 0;
        }
    }
}
```