ECE 8540 Analysis of tracking systems

Lab 8 - Simulation behavior-based robot

Rahil Modi
C14109603

December 2, 2020

1 Introduction

In this lab we were expected to develop a C-code to simulate a behavior based robot. Behavior based robots behave based on the behavior we develop and implement in them such as in this lab Greedy or Scary is a behavior of the robot. It will respond based on the sensor readings and the environment it is in, this is why it is called behavior based robot. For this kind of robots the world setup does not matter, you place it in any world situation it will respond to it based on the behavior it knows.
In this report modeling of the simulated robot is done where it has to survive by saving itself from the hungry sharks, also maintain energy by eating food to stay alive and have energy to move.

2 Methods

In the simulated world provided there are hungry sharks which eat food to survive as well as they eat robots as food. Robots can only eat food to stay alive and perform motion. If the robot runs out of the energy it cannot move any further and is stuck where it is an becomes a sitting duck for the shark.
I had implemented two behaviors for my robot Ultron based on the energy level it has it will perform one of the behavior. Each of the behavior had a different priority sequence which it performed.

- If the energy level was above the a given threshold it would perform this behavior. I call this mode as Power Mode:
  1. It will try to run away from shark first.
  2. And then go towards the nearest food.
  3. Avoid collision with other robots
- If the robot's energy level is below the given threshold it will go into this behavior which I have called as Survival Mode:
  1. It will go for the nearest food first for energy.
  2. Then run away from sharks.
  3. If it is in Survival Mode and no one is around and there is no food nearby it will go into sleep and wait for food to come towards it and regain energy.

2.1 Implementation

The robot has 255 as energy when it starts and every movement costs it some energy. The speed of the robot is directly proportional to energy so faster the robot is it will cost it more energy. The direction of the movement is specified by angles in degrees. The goal of the lab is to make the robot survive as long as possible in the environment. The variables used are:

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. EnemySpotted_FallBack : Distance to shark below which the robot starts to run away from the shark
11. NeedFood : Minimum energy below which the robot enters survival mode
12. iSeeFood_Attack : Distance to food below which the robot starts to follow and hunt that food item
13. PersonalBubble : Distance to other robot, below which the robot 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

This mode is activated when the robot has energy more than the given threshold. The priorities for this mode have been discussed above. As the highest priority in this mode is run away from shark and as the robot has energy to spend when the EnemySpotted_FallBack falls below the threshold the robot tries to run away from the shark and spends 30 energy for it. The movement of the robot is determined by (prevAngle + SharkClosetAngle) mod 360, prev Angle is the Static varible which remembers the last angle which was taken and the angle of the movement is increased by 30° every time, due to this robot can perform a circular type motion through which it can evade the shark and can also go to food that might be behind the shark. The motion of the robot is bounded between 90° and 270° so that robot does not perform a circle and go back to where it started and end up as food for shark.
Second priority of the robot is to go towards food so when the food is nearby which is triggered by the iSeeFood variable the robot will spend 30 energy and go towards the food. Third priority is to avoid collision with the competitor robots so a PersonalBubble variable

is created so whenever there is a robot in that bubble the robot will go perpendicular to that robot and consume 5 energy.
The last priority is when no one is there nothing nearby it will move slowly towards the food.

2.3 Survival Mode

The robot enters the Survival Mode when the energy level falls below the NeedFood. In this mode the first priority of the robot to find food for that it moves slowly towards the food. The energy expenditure is calculated as (5 + FoodClosetDistance/10), this way the robot starts slowly and as the distance towards the food decreases the speed of the robot keeps on increasing, the reason for doing is it helps for the better expenditure of the energy.
Once it has regained some energy it will start running from the sharks again but in this mode it will only be spending 14 energy to move away from it as it already is on low energy. Lastly when there is no food and shark nearby it will go into sleep as it does not have the energy to make moves so it will sit there for the food to arrive to it.

3 Simulation Behavior

During the RoboWars in the lecture my robot Ultron won once during the smaller rounds and when all the robots were placed together in the simulation it was always in the top 20% of the robots that lasted till end. The behavior of the robot was not exactly what I expected, it did behave at many places like I how I intended to, when evading the sharks I wanted the robot to perform circular motion and move around the shark but it would run out of energy after some time due to which it could not completely perform its motion but after that I worked and implemented the static variable we were supposed to for the second day, I created a static variable called prevAngle and looked at the results, with help of it the robot was able to perform the motion I wanted it to due to memory it had of previous motion. So the motion was much better than the earlier robot
I also noticed when there were too many robots that sometimes my robot would collide with another robot and stay there as it had lost its energy, to overcome that in the need new code I have implemented the personal bubble which was explained in detail above.
I have attached the two codes, one of the first day which was demonstrated in the lecture and the second one which I modified with incorporation of the static variables.

4 Conclusion

In this lab I learned to implement various behaviors in behavior based robot and how small changes can make the robot perform differently. I learned about static variables and how having some memory can help the robot perform better. During the lecture I learned about why the robot was having some jerky behavior, I understood that it was caused as there was no filtering a sudden change in behavior caused it to behave so abruptly, if a filtering system would be implemented the motion could be smoothed out and there would be no jerky motion.

5 Appendix

First Day Code:

```
void Ultron(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 Current-50

{
/* Define the Trigger Points*/
int EnemySpotted_FallBack = 125;
int NeedFood = 48;
int iSeeFood_Attack = 90;

/*Priority
1. I am strong and have energy. I will tackle the shark first.
2. Now I will eat, as I am always hungry.
3. Won't mess with my competitior robots.
4. Browsing around for food slowly.
*/

if (CurrentRobotEnergy > NeedFood)
{
if (SharkClosestDistance < EnemySpotted_FallBack)
{
/* Shark ahead fallback and save your life*/
(*RobotExpendEnergy) = 20;
(*RobotMoveAngle) = (130 + SharkClosestAngle) % 360;
}
else if (FoodClosestDistance < iSeeFood_Attack)
{
/* I am on a seafood diet, I see food and eat it */
(*RobotExpendEnergy) = 25;
(*RobotMoveAngle) = FoodClosestAngle;
}
else
{
/* Area is clear will just browse and check for food slowly*/
(*RobotExpendEnergy) = 10;
(*RobotMoveAngle) = FoodClosestAngle;
}
}
/* Energy Low I need food.
1. I am weak, survival mode on, I will find food first.
2. I am recharged I will again run from shark.
3. Won't mess with my competitior robots.
4. Browsing around for food slowly.*/
else
{
if (FoodClosestDistance < iSeeFood_Attack)
{
(*RobotExpendEnergy) = 10 + FoodClosestDistance/10;
(*RobotMoveAngle) = FoodClosestAngle;
}
else if (SharkClosestDistance < EnemySpotted_FallBack)
{
(*RobotExpendEnergy) = 18;
(*RobotMoveAngle) = (130 + SharkClosestAngle) % 360;
}
else
{
(*RobotMoveAngle) = 0;
}
}
}
```

Updated code with static variable:

```
void Ultron(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 Current-50

{
/* Define the Trigger Points*/
int EnemySpotted_FallBack = 125;
int NeedFood = 48;
int iSeeFood_Attack = 90;
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*/

/*Priority
1. I am strong and have energy. I will tackle the shark first.
2. Now I will eat, as I am always hungry.
3. Won't mess with my competitior robots.
4. Browsing around for food slowly.
*/
/* Change angles in an increment of 30 degerees*/
if (prevAngle > 270) {
prevAngle = 90;
}
if (CurrentRobotEnergy > NeedFood)
{
if (SharkClosestDistance < EnemySpotted_FallBack)
{
/* Shark ahead fallback and save your life*/
(*RobotExpendEnergy) = 20;
(*RobotMoveAngle) = (prevAngle + SharkClosestAngle) % 360;
prevAngle = angleInc + prevAngle;
}
else if (FoodClosestDistance < iSeeFood_Attack)
{
/* I am on a seafood diet, I see food and eat it */
(*RobotExpendEnergy) = 25;
(*RobotMoveAngle) = FoodClosestAngle;
}
else if (RobotClosestDistance < PersonalBubble)
{
/*Other bot is too close, move away from it*/
(*RobotExpendEnergy) = 5;
(*RobotMoveAngle) = (90 + RobotClosestAngle) % 360;
}
else
{
/* Area is clear will just browse and check for food slowly*/
(*RobotExpendEnergy) = 10;
(*RobotMoveAngle) = FoodClosestAngle;
}
}
/* Energy Low I need food.
1. I am weak, survival mode on, I will find food first.
2. I am recharged I will again run from shark.
3. Won't mess with my competitior robots.
```

```
4. Browsing around for food slowly.*/
else
{
if (FoodClosestDistance < iSeeFood_Attack)
{
(*RobotExpendEnergy) = 10 + FoodClosestDistance/10;
(*RobotMoveAngle) = FoodClosestAngle;
}
else if (SharkClosestDistance < EnemySpotted_FallBack)
{
(*RobotExpendEnergy) = 18;
(*RobotMoveAngle) = (prevAngle + SharkClosestAngle) % 360;
prevAngle = angleInc + prevAngle;
}
else
{
(*RobotMoveAngle) = 0;
}
}
}
```