UNDERGRADUATE REPORT

The Simulation of the Movement of Fish Schools

by Neha Bhooshan
Advisor: P.S. Krishnaprasad

UG 2001-4

The Simulation of the Movement of Fish Schools

Neha Bhooshan
Massachusetts Institute of Technology
Advisor: Dr. P.S. Krishnaprasad

Research Experience for Undergraduates
Institute of Systems Research
University of Maryland, College Park

August 3, 2000

The Simulation of the Movement of Fish Schools

Neha Bhooshan

Abstract:

In this paper, I explain a school behavior model, which was constructed by Aoki, Huth, and Wissel, used to describe the motion of schools of fish. Schools of fish are characterized by strong cohesion and high parallel orientation without using a leader. In this model, each fish can exhibit one of four basic behavior patterns – repulsion, parallel orientation, attraction, and search – based on its proximity to a neighbor fish. I modified the model in how the fish mixed the influence of its neighbors; the fish takes a weighted average of the influences of its neighbors. I constructed a computer simulation model using robots to test this model, and my data has shown that the model is quite successful in simulating the characteristics of a school of fish. The ultimate goal of this research is to apply the school behavior model to algorithms for robot formations.

Introduction:

Researchers, in hoping to manipulate robots into formations, have looked at how animal flocks maintain their shape and direction while turning, moving, or displaying other types of motion. Animals, of which birds and fish form the most visibly organized groups, form such congregations for the reasons of reproduction, energy savings, migration, and survival (search for food and defense against predators). They are able to accomplish such coordination through the sensory integration system in which each individual receives and processes information from internal and external stimuli. The signals then propagate throughout the group, and the resulting decision is blended into a unified group response. (Schilt 1991) More members in a group lead to more information gathered which results in a more informed group decision.

Discovering the rules that govern the movement of animal formations can have broad applications in the field of mobile robotics. The development of autonomous mobile robots has applications in such tasks as space missions, operations in hazardous environments, and military operations. (Sugihara et al 1990) One aspect of this research is the efficient coordination of many mobile robots in achieving a goal, and the algorithms used in animal movement models can be extended to be algorithms for robot formations.

School Behavior Model:

Many fish schools, in contrast to bird flocks, do not have a “hierarchical structure” (Huth et al 1991), meaning there is not a designated leader. Research has shown that such self-coordination results from effective sensory input in which each fish uses primarily vision and the lateral line (an organ sensitive to the displacement of water) to know the speed and bearing of its nearest neighbors. Fish schools’ behavior is typified by a high degree of both cohesion and polarization; the fish are able to stay together in a parallel orientation, which dictates a common moving direction for the entire group. Thus the group is able to cover a large distance quickly.

There are a number of published papers, describing different behavior models to explain how such animal formations occur. One popular underlying theory proposes the interplay of attractive and repulsive forces (Warburton 1991), which directs the movements of the individuals. I chose to base my behavior model on a more biologic model, which was based on this theory and which was first proposed by Aoki (Aoki 1982) and later revised by Huth and Wissel (Huth et al). There are several fundamental assumptions of the model:

1. Every member of the group moves according to the same behavior model. This guarantees that the group will move without a leader.
2. The school moves independently of external stimuli.
3. The movement of each fish is influenced only by its nearest neighbors. Research has shown that only a few neighbors (the nearest ones) have to be observed to produce school behavior.

Basic Behavior Patterns:

Aoki described that each fish has four distinct zones surrounding itself as a measurement of its proximity to its nearest neighbor (please refer to Appendix 1 for an illustration of these zones). Based on the possible positions of the neighbor, the individual can behave in four different ways: repulsion, parallel orientation, attraction, and searching, which gives the influence angle or preferred heading with respect to the neighbor.

If a neighbor is located less than R1 distance away from the fish, the fish will show repulsion behavior in which it will turn to swim perpendicular to the neighbor in order to avoid a collision. So the influence angle is $\pm 90°$.

If a neighbor is located between R1 and R2, the fish will swim in the same direction of the neighbor. The neighbor is located in the preferred distance range or parallel orientation area; this is the boundary between the attraction and repulsion zones to maintain an equilibrium distance. So the influence angle is the heading of the neighbor.

If a neighbor is located between R2 and R3, the fish will display biosocial attraction and wants to approach its neighbor. So the fish will turn to swim in the direction of its neighbor.

Huth and Wissel added the last behavior pattern. If a neighbor is located greater than R3, the fish cannot perceive its neighbor since its senses are limited. So the fish will show searching behavior, turning by chance.

Please refer to Appendix 1 for a detailed illustration of each of these behavior patterns.

Both Aoki and Huth and Wissel used probability distributions to determine the turning angle by which the fish will turn to account for the small degree of uncertainty in detecting the neighbor's position and orientation. They employed different methods with respect to the distribution, which will be discussed later. Both papers also constructed the velocity as a stochastic variable to take into consideration other possible random influences, meaning that the velocity is chosen independent of the neighbor. The velocity was calculated using a typical distribution, which was constructed from data from experiments and fitted by a Gamma distribution.

Mixture Model:

The last element of the model is how the fish responds to the influences of several neighbors. If a fish only responds to one neighbor, then the model will result in a confused group. Aoki used a decision model concept in which the fish will decide which neighbor to follow. Each neighbor is given a weight factor based on the fish's front priority, and the probability distribution for the turning angle consists of two or more differently weighted normal distribution. If neighbor #1 is located more in the front of the fish than neighbor #2, the weight of its distribution will be double the weight of the second fish and therefore the fish will follow neighbor #1. Huth and Wissel proposed a different concept in which the fish mixes the influences of its neighbors by taking the average of each neighbor's influence angle, and the probability distribution consists of only one normal distribution around the mean angle.

Modifications:

In my model, I made two distinct changes: the velocity and position are not stochastic variables, and the mixture element is a hybrid of the decision and the average model concepts. The velocity and turning angle were not calculated using probability distributions; instead the velocity was set at a constant value. I did this to simplify the behavior model as well as to take into account the fact that this model is for application in robotics rather than a purely biology interest so the random influences experienced by real fish is insignificant.

The modification I made to the mixture model simply fuses the concept of a weighted factor from the decision model with the average model to produce a weighted average of the influence angles based on distance priority. So the fish will place more consideration on a neighbor in its repulsion range than on a neighbor in its parallel range. A neighbor in the parallel range will have more weight than one in the attraction range, and so on. Then the average is taken of the weighted influence angles to produce the turning angle for the fish. I had originally used the averaging model, but I did not feel that a fish would place equal emphasis on a neighbor in its repulsion zone and on a neighbor in its search zone. Avoiding collisions with the other members in the group is a primary concern so it seemed more natural to use a distance priority for the weighted average. Please refer to Appendix 1 for an example on the difference between the weighted average model and the averaging model.

The Simulation:

I used MDLe (Motion Description Language extended) script to program the computer simulation model. The behavior model was coded as a quark, and each robot ran the quark. Currently I have a working simulation in which three to five robots are placed in arbitrary positions with arbitrary headings, and then as the program starts to run, each robot inputs the ˝other robots' positions and headings ˝using ˝the state ˝ server/client and calculates its two nearest neighbors. It evaluates ˝the influence angle of each neighbor using the basic behavior patterns ˝and then calculates the weighted average of the angles. Then each robot ˝will rotate to its turning ˝angle, and ˝move ˝forward a specified length. The quark is then reset, and the each robot ˝will continue to move in this cycle ˝until a school behavior is produced. It ˝must be noted that due to the sequential processing of the script, the robots cannot rotate and move synchronously. The source code for the program can be found in Appendix 2.

Standard Runs:

I ran simulations using three and four robots, and for the parameter values, I chose R1 = 250, R2 = 400, R3 = 650, and the velocity ˝was set at a constant value of 50.

Figure 1a shows the paths of the robots in which three robots comprise the group, and Figure 1b shows the orientation of the three robots.

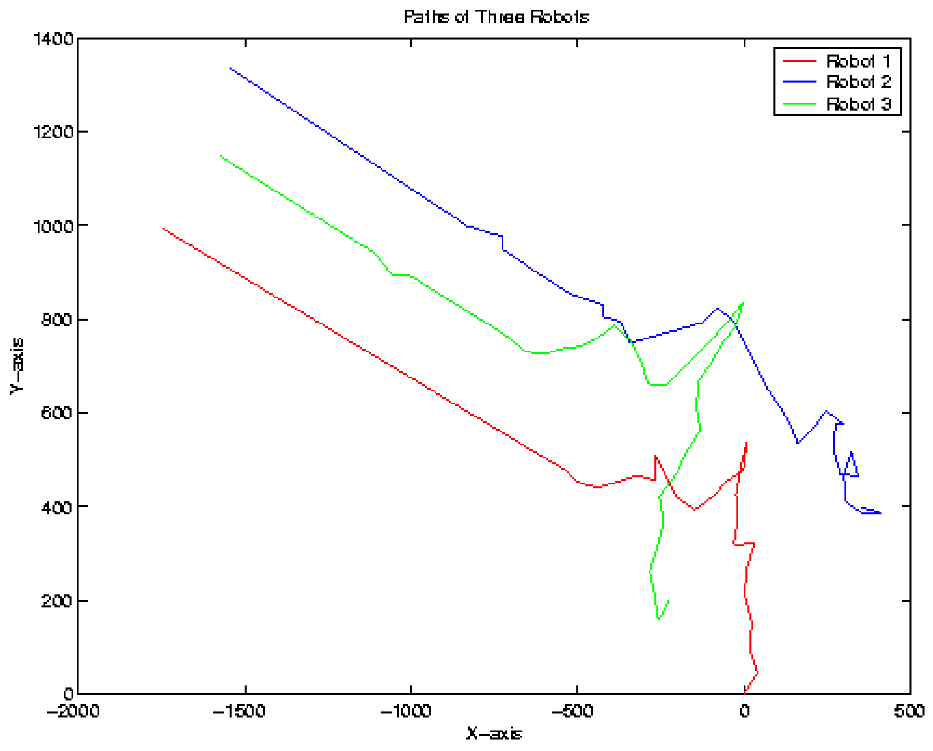


Figure 1a

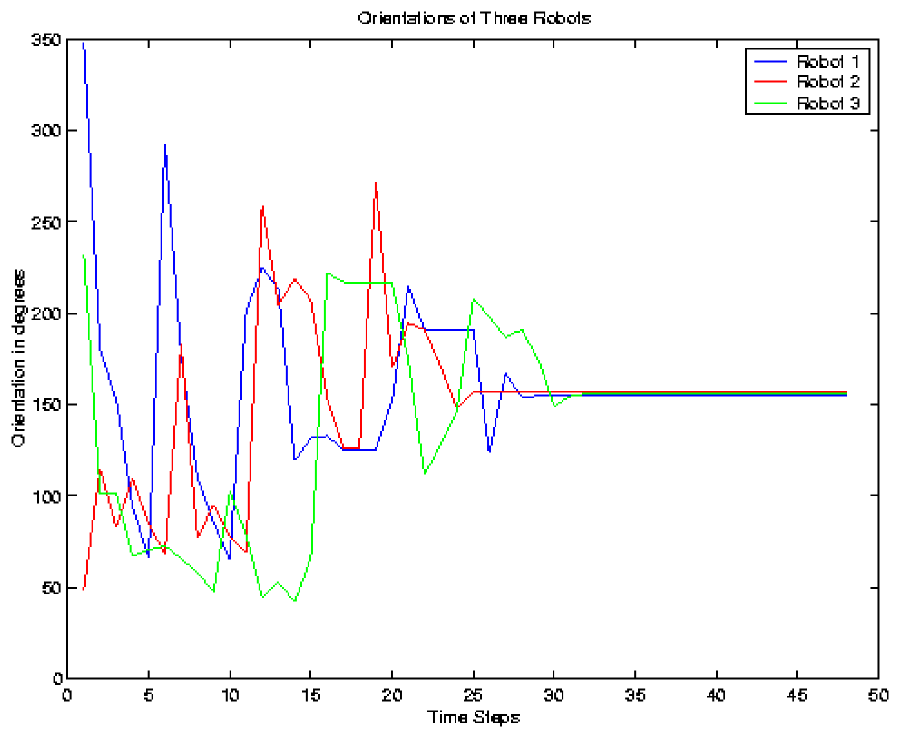


Figure 1b

As seen from the Figure 1, the three robots start in the arbitrary positions of (0,0), (450,400), and (-240,200), and when the program first starts to run, their paths have quite a few bends as each robot reacts to the its neighbors. After about 20 time steps, they have aligned themselves so that they are an equilibrium distance apart and are moving in the same direction. Unfortunately the graph does not show the actual formation that the three robots moved into, but from watching the simulation, I could tell that the formation was a triangle. They are able to keep the direction for many time steps and cover a lot of distance once in formation. Figure 1b, which shows the orientation of each robot, corroborates this analysis. The three robots started off with arbitrary headings of 50°, 235°, and 350°. As the program ran and each robot was calculated its new heading using the weighted average of the influence angles from each neighbor, one can see that all three robots converged to the same heading of around 150°, and therefore were able to move in a common direction.

Similar data was taken using four robots. Figure 2a shows the paths of the four robots and Figure 2b shows the orientation of the four robots.

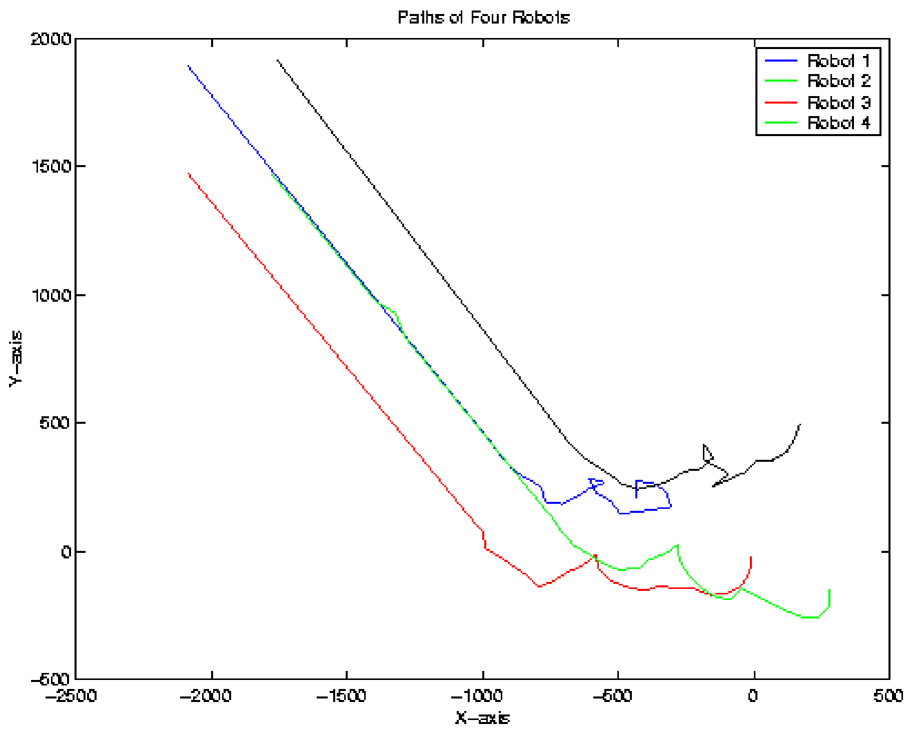


Figure 2a

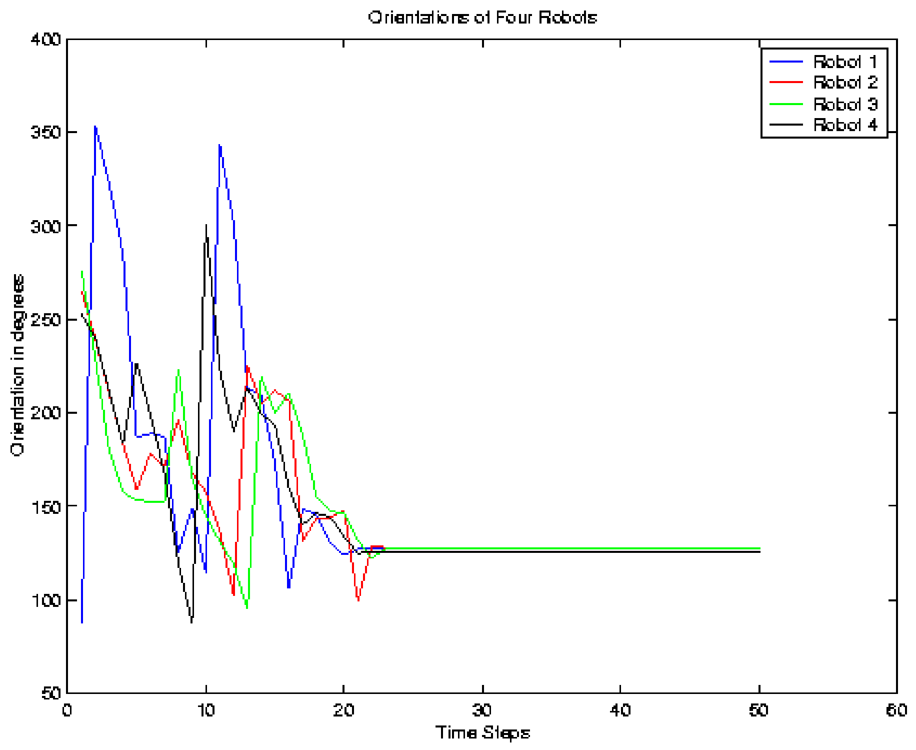


Figure 2b

Looking at the graphs of the paths and orientation of the four robots, the model also works for groups of four robots. The robots start in arbitrary positions and with arbitrary headings, and, after a certain period of time, do show school behavior in maintaining an equilibrium distance from each other and in having the same heading. This strong cohesion and parallel orientation leads the robots to move efficiently in a group. With the group of four members, the robots also moved in a formation – a diamond. I am currently working with groups of five robots, and it will be interesting to see what kind of shape a group of five members will form.

I used two measurements, which Huth and Wissel developed, to evaluate and quantify the two important characteristics of a fish school: cohesion and parallel orientation. The polarization, p, was calculated by taking the average of the angle deviation of each robot to the mean direction of the group, which is calculated by averaging the headings of the robots. Highly parallel orientation would result in a low polarization, and a 'confused' group would have a high polarization. To measure the compactness of the group, the nearest neighbor distance (NND) for each robot was used.

	Robot 1	Robot 2	Robot 3	Mean
Deviation (degrees)	1.1	1.0	.1	(p =) .73
NND	364.918	352.785	363.886	360.530

Table 1

	Robot 1	Robot 2	Robot 3	Robot 4	Mean
Deviation (degrees)	0.1	0.6	0.4	1.3	(p =) 0.6
NND	343.286	311.058	295.332	323.262	318.235

Table 2

As seen from the tables, both three-robot and four-robot groups have small polarization values, which result in the high parallel orientation and a rather uniform NND among the robots in the group. The range of NND values fall within the parallel orientation area, which has the range of 200 to 450 units. It is interesting to note that the four-robot group not only has a smaller polarization with a value of 0.1° less than that of the three-robot group, but also has a smaller mean NND with a value of 42.3 units less than the mean NND of the three-robot group. The values of the polarization and NND verify the analysis of the robots' school behavior from Figures 1 and 2.

Experimentation with the parameters including R1, R2, R3, velocity, and rate of rotation can greatly influence the school behavior as noted by Huth and Wissel. According to their research, an increase in the parallel orientation area can lead to a higher parallel orientation, but lower compactness. In contrast, a larger repulsion area will lead to lower parallel orientation and have an insignificant effect on the cohesion.

Conclusion:

In this paper, I have discussed a school behavior model and have shown that the modified model was able to reproduce the characteristics a moving school of fish quite well. Both groups of three robots and four robots exhibited strong cohesion and high parallel orientation without using a leader. Instead the robots relied on each other to maintain the correct distance and orientation.

Future Work:

For further investigation, I would like each robot to use its three instead of two nearest neighbors to calculate its turning angle and increase the group size to eight in order to compare results with Huth and Wissel who used those conditions for their averaging model simulation. I also would like to run the program on real robots rather than just on the simulator to see if it is feasible for robots to behave in the same manner as fish.

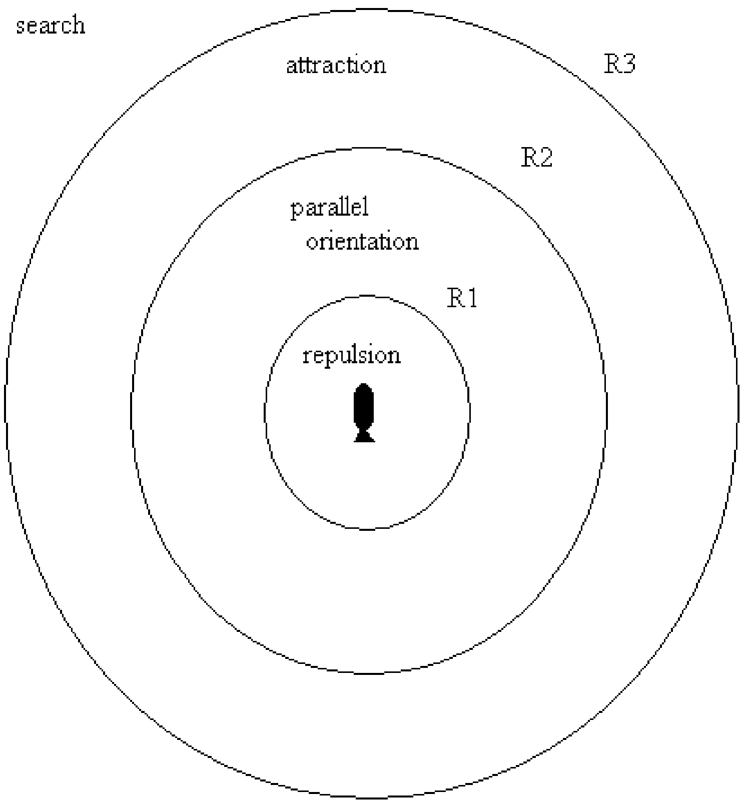
Acknowledgments:

I would like to thank Dr. P.S. Krishnaprasad for his guidance in the direction of this project. I am also grateful to Fumin Zhang and Sean Andersson, his graduate students, for their help in developing this project and in answering my numerous questions

References:

Huth A. and C. Wissel. 1992. The simulation of the movement in fish schools. Journal of Theoretical Biology 156:365-385.
Aoki, I. 1982. A simulation study on the schooling mechanism in fish. Bulletin of the Japanese Society of Scientific Fisheries 48: 1081-1088.
Warburton K. and J. Lazarus. 1991. Tendency-distance models of social cohesion in animal groups. Journal of Theoretical Biology 150: 473-488.
Schilt C. 1991. Fish schools, impulse sounds, and sensory integration or the Ballad of J-Dock. University of California, Santa Cruz, MSc thesis in Marine Sciences.
Sugihara K. and Suzuki I. 1990. Distributed Motion Coordination of Multiple Mobile Robots. IEEE.

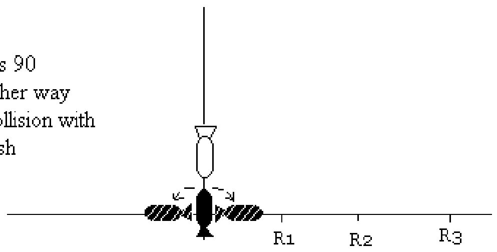
Appendix 1



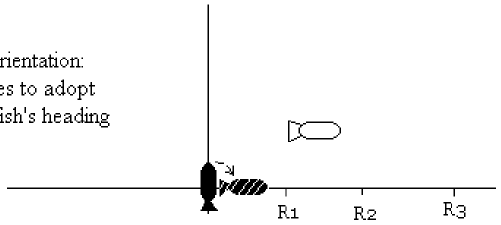
As illustrated, the fish has 4 concentric zones surrounding itself which divides the area surrounding itself into the repulsion zone, parallel orientation zone, attraction zone, and searching zone. These zones are defined by R1, R2, and R3.

Basic Behavior Patterns with One Neighbor

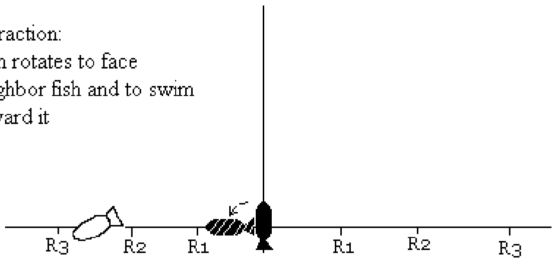
Repulsion:
Fish rotates 90 degrees either way to avoid collision with neighbor fish



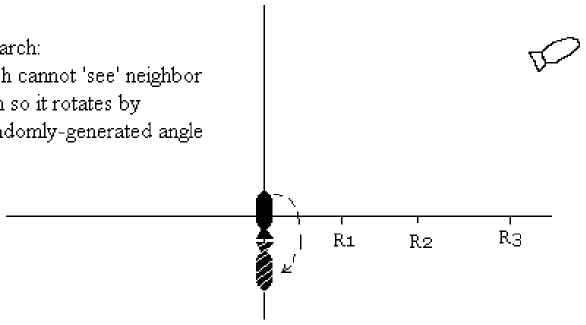
Parallel Orientation:
Fish rotates to adopt neighbor fish's heading



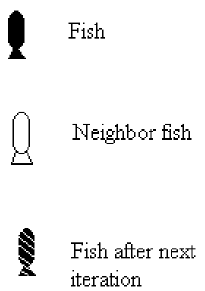
Attraction:
Fish rotates to face neighbor fish and to swim toward it



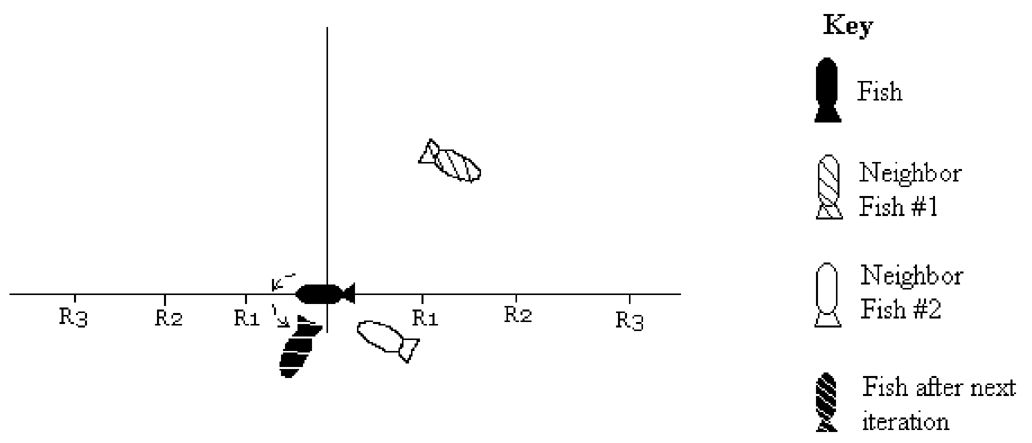
Search:
Fish cannot 'see' neighbor fish so it rotates by randomly-generated angle



Key:



Weight Model versus Averaging Model



As can be seen from the graph, the fish has two neighbors. Neighbor Fish #1, with a heading of 330 degrees, is located in the parallel orientation zone between R1 and R2 of the fish so its influence angle is 330 degrees. Neighbor Fish #2, with a heading of 155 degrees, is located in the repulsion zone of the fish. So the fish wants to be swimming perpendicularly to Neighbor Fish #2 so it can turn either to 245 degrees or 65 degrees. Since turning to 245 degrees will take shorter time, the influence angle 245 degrees.

Using the averaging model, the average of the two influence angles is taken:

$$(245 + 330) / 2 = 287 \text{ degrees}$$

Using the weight averaging model, Neighbor Fish #2 is given double the weight of Neighbor Fish #1 since it is located closer to the fish. So the weighted average is taken:

$$(2/3)(245) + (1/3)(330) = 273 \text{ degrees}$$

There is a difference of 15 degrees between the two calculated turning angles, and the one calculated using the averaging model has the fish turning more towards the incoming neighbor fish. By using the weight averaging model, the fish has a better chance of not hitting Neighbor Fish #2. Clearly, the weight averaging model is more effective in avoiding collisions which is crucial for school behavior.

Appendix 2

```
// qSchool.hpp
//
// header file for School quark
//
// created by Neha Bhooshan
// July 19, 2001
// modified last: July 27, 2001

#ifndef QSCHOOL_HPP
#define QSCHOOL_HPP

//random number generator
#include "rmrandom.h"
//position header
#include "posQuark.hpp"
//rotation header
#include "qRotate.hpp"
//both headers are used to network each robot to each other
#include "mStateServer.hpp"
#include "mStateClient.hpp"

struct position {
  double x, y, steer;
};

class qSchool: public posQuark {
protected:
  char NeighborRobot1[256], NeighborRobot2[256];
  char NeighborRobot3[256], NeighborRobot4[256];
  char currentRobot[256];
  double distance1, distance2, distance3, distance4;
  double *dist_array;
  int robotId, numberOfRobots; //number of neighbor robots
  position my, neighbor, initial, desired;
  double dx, dy;
  double D1, D2, D3;
  double theta;
  double distance, reference_angle;
  int velocity, k;
  double length;
  ofstream fout;
  double  beta;
  double expanse, pol_angle, polarization, deviation;
  //for weighted average
  int n;
  int b1, b2, b3;
  double w3, w1, w2;
  double ta1, ta2, ta3;
  int total_angle;

public:
  qSchool(istream&);
  virtual ~qSchool();
  virtual const char* isA() const {return "qSchool"; } ;
  virtual int update();
  virtual void reset();

protected:
  virtual int getdth(long, long);
  virtual void repulsion(double, double);
  virtual void parallel(double);
  virtual void attraction(double, double);
  virtual void search();
  virtual double weight(int, double, int, double);
  virtual void orient(int, int);
  virtual double goLength(position,position);
  virtual void reinitialization();
  virtual void behavior();
  virtual double testdistance(int);
  virtual void sort(double*, int );
};

#endif
```

```
// qSchool.cpp
//
// This quark simulates the movement of a school of fish,
// following the model developed by I.Aoki and later by A.Huth
// and C.Wissel.
// Each robot has 4 distinct ranges surrounding
// itself and depending on its proximity to a designated neighbor
// robot, will execute 4 different behaviors which will determine its
// its heading for the next iteration.  The robot will read in
// the positions and headings of the other robots in the group
// and determine its two nearest neighbors.  It will then calculate
// desired turning angle using weighted average of
// influence angles from each neighbor.  The robot will then rotate
// to that new heading and move at a constant velocity for a certain
// length to its new position. The quark will then reinitialize the values
// and start the process over again.
//
// Three to five robots can currently be used for the group.
//
// call: (school number_of_robots address_of_neighbor1
//           address_of_neighbor2 address_of_neighbor3 robotId)
//
// created by Neha Bhooshan
// July 19, 2001
// modified last: July 27, 2001

#include "qSchool.hpp"
#define _ROTATE_MIN_VELOCITY_ 1
//#define COUNTER 1

qSchool::qSchool(istream &in): posQuark() {
  in >> ws;
  in >> numberOfRobots;
  in >> ws;

  //reading in each neighbor
  in.getline(NeighborRobot1, 255, ' ');
  if (numberOfRobots >= 2) {
    in >> ws;
    in.getline(NeighborRobot2, 255, ' ');}
  if (numberOfRobots >= 3) {
    in >> ws;
    in.getline(NeighborRobot3, 255, ' ');}
  if (numberOfRobots >= 4) {
    in >> ws;
    in.getline(NeighborRobot4, 255, ' ');}
  in >> ws;
  in >> robotId;

  //opening text file to write data to
  switch (robotId) {
  case 1:
    fout.open("robot1_data.txt");
    break;
  case 2:
    fout.open("robot2_data.txt");
    break;
  case 3:
    fout.open("robot3_data.txt");
    break;
  case 4:
    fout.open("robot4_data.txt");
    break;
  case 5:
    fout.open("robot5_data.txt");
    break;
  }
}

qSchool::~qSchool() {
}

int qSchool::update() {

//parameters
  D1 = 250;
  D2 = 400;
  D3 = 650;
  velocity = 50;

  posQuark::refresh();

//robot’s current position
  my.x = MyRobot->getStateById(STATE_CONF_X);
  my.y = MyRobot->getStateById(STATE_CONF_Y);
  my.steer = MyRobot->getStateById(STATE_CONF_STEER);

  if (firstTimeFlag) {
        posQuark::orientation = true;
        posQuark::direction = false;
        desired.steer = 0;
```

```
       posQuark::firstTimeFlag = false;

       pol_angle = my.steer;

       //initializing arrays to store distance between robot and each neighbor
       dist_array  = new double[numberOfRobots];
       for (int a = 1; a <= numberOfRobots; a++) {
             dist_array[a-1]=testdistance(a); }

            //calculating data and writing to file
       polarization = pol_angle / (numberOfRobots+1);
       int deviation_angle = my.steer - polarization;
       deviation = abs(deviation_angle);
       fout << "pol_angle:" << pol_angle << " polarization:" << polarization << "
deviation: " << deviation << endl;
       pol_angle = 0;

       distance1 = testdistance(1);
       distance2 = testdistance(2);
       distance3 = testdistance(3);
       distance4 = testdistance(4);

       //sorts array from least to greatest values
       sort(dist_array, numberOfRobots);

       //determines which robot is one of the two nearest neighbors
       for (int b = 0; b <=1; b++) {
             if (dist_array[b] == distance1)
               NeighborRobot1[100]=1;
             else if (dist_array[b] == distance2)
               NeighborRobot2[100]=1;
             else if (dist_array[b] == distance3)
               NeighborRobot3[100]=1;
             else if (dist_array[b] == distance4)
               NeighborRobot4[100]=1;
       }

       //storing the nearest neighbor distance to file
       expanse = dist_array[0];
       fout << "NN: " << expanse << endl;
       cout << "NN: " << expanse << endl;
       n=1;

  //if a neighbor is a nearest neighbor, calls behavior function for that robot
  for (int i = 1; i <= numberOfRobots; i++) {
    if (i==1) {
```

```
    if (NeighborRobot1[100] == 1) {
          currentRobot = NeighborRobot1;
          behavior();
          n++;
          NeighborRobot1[100] = 0;}
   }
else if (i==2) {
  if (NeighborRobot2[100] == 1){
          currentRobot = NeighborRobot2;
          behavior();
          n++;
          NeighborRobot2[100] = 0;}
else if (i==3) {
  if (NeighborRobot3[100] == 1) {
          currentRobot = NeighborRobot3;
          behavior();
          n++;
          NeighborRobot3[100] = 0;}
else if (i==4) {
  if (NeighborRobot4[100] == 1) {
          currentRobot = NeighborRobot4;
          cout << "Robot4 is a neighbor" << endl;
          behavior();
          n++;
          NeighborRobot4[100] = 0;}
}

// weight function returns the desired turning angle
desired.steer = weight(b1, ta1, b2, ta2);

}

//robot is rotating to new turning angle
 if (orientation)
   orient(my.steer, desired.steer);

//once the robot has stopped rotating, it will move a certain distance to its
//new position
if (direction) {
  length = goLength(initial, my);
  fout << "my:" << my.x << ", " << my.y << "; " << my.steer << endl;
  if (length <= 50)
    MyRobot->action.right=MyRobot->action.left = velocity;
  else if (length > 50) {
    reinitialization();}
}
```

```
  delete [] dist_array;
  return 0;
}

void qSchool::reset() {
  posQuark::reset();
  desired.steer = 0;
}

//calculates distance between robot and neighbor robot
double qSchool::testdistance(int robot) {
  switch(robot) {
  case 1:
    currentRobot = NeighborRobot1;
    NeighborRobot1[100] = 0;
    break;
  case 2:
    currentRobot = NeighborRobot2;
    NeighborRobot2[100] = 0;
    break;
  case 3:
    currentRobot = NeighborRobot3;
    NeighborRobot3[100] = 0;
    break;
  case 4:
    currentRobot = NeighborRobot4;
    NeighborRobot4[100] = 0;
    break;
  }

    neighbor.steer = MyRobot->getRemoteStateById(currentRobot,
STATE_CONF_STEER);
    pol_angle += neighbor.steer;

    neighbor.x = MyRobot->getRemoteStateById(currentRobot, STATE_CONF_X);
    neighbor.y = MyRobot->getRemoteStateById(currentRobot, STATE_CONF_Y);

    long tempx = long(neighbor.x - my.x);
    long tempy = long(neighbor.y - my.y);
    double d = (double)sqrt(tempx*tempx + tempy*tempy);

    return(d);
}

//sorts array into least to greatest values
```

```
void qSchool::sort(double array[], int n) {
  //sorts array into ascending order: least-->greatest

  int curr, next;
  double temp;

  for (curr=0; curr<n; curr++) {
    for (next=curr+1; next<n; next++) {
      if (array[curr] > array[next]) {
            temp = array[curr];
            array[curr] = array[next];
            array[next] = temp;
      }
    }
  }
}

//gets distance between robot and neighbor and decides which behavior based
//on the distance
void qSchool::behavior() {

  neighbor.x = MyRobot->getRemoteStateById(currentRobot, STATE_CONF_X);
  neighbor.y = MyRobot->getRemoteStateById(currentRobot, STATE_CONF_Y);
  neighbor.steer = MyRobot->getRemoteStateById(currentRobot,
STATE_CONF_STEER);

        this->initial.x = my.x;
        this->initial.y = my.y;

  dx = neighbor.x - my.x;
  dy = neighbor.y - my.y;
  distance = (double)sqrt(dx*dx + dy*dy);
  reference_angle = getdth(dx,dy);

  if (distance < D1) {
    repulsion(my.steer, reference_angle); }
  else if ((distance >= D1) && (distance < D2)) {
    parallel(neighbor.steer); }
  else if ((distance >= D2) && (distance < D3)) {
    attraction(my.steer, reference_angle); }
  else if (distance > D3) {
    search(); }

  if (beta < 0 )
     beta += 3600;
}
```

```
//calculates difference in angles of two robots
int qSchool::getdth(long dx, long dy) {

  double tempth;
  double diffth;

  tempth=degToRad(my.steer/10);
  if(dx==0 && dy ==0)
    theta=tempth;
  else
    theta = atan2(dy, dx);

  diffth=10*radToDeg(fmod(theta-tempth,2*pi));
  theta=10*radToDeg(theta);

  if (diffth < -1800)
    diffth = diffth + 3600;
  else if (diffth > 1800)
    diffth = diffth - 3600;

  return (int)diffth;

}

//repulsion behavior -- robot turns so that it is facing perpendicularly
//to neighbor robot
void qSchool::repulsion(double curr_heading, double reference_angle) {
  beta = (-1)*(theta + 900);
  if (n==1){
    ta1 = beta;
    b1 = 999;}
  else if (n==2){
    ta2 = beta;
    b2=999;}
}

//parallel orientation behavior -- robot turns to have same heading as
//neighbor robot
void qSchool::parallel( double neighbor_heading) {
  beta = neighbor_heading;
  if (n==1){
    ta1 = beta;
    b1 = 888;}
  else if (n==2){
    ta2 = beta;
    b2=888;}
}

//attraction behavior -- robot turns to face neighbor robot
void qSchool::attraction(double curr_heading, double reference_angle) {
  if (reference_angle >= 0)
    beta = curr_heading - reference_angle;
  else if (reference_angle < 0)
    beta = curr_heading + reference_angle;

  if (n==1){
    ta1 = beta;
    b1 = 777;}
  else if (n==2){
    ta2 = beta;
    b2=777;}
}

//search behavior -- produces randomly generated angle
void qSchool::search() {
  CRand rand;
  beta = rand.GetRandom(0,3600);

  if (n==1){
    ta1 = beta;
    b1 = 666;}
  else if (n==2){
    ta2 = beta;
    b2=666;}
}

//calculates desired turning angle using weighted average of the two
//influence angles
double qSchool::weight(int b1, double ta1, int b2, double ta2) {
  double angle;

  if (b1==b2) {
    w1 = .5;
    w2=w1; }
  else if (b1 > b2) {
    w1 = .6666;
    w2 = .3333; }
  else if (b1 < b2) {
    w1 = .3333;
    w2 = .6666;}
  angle = (w1*ta1) + (w2*ta2);

  return (angle);
}

//robot rotates from curr_heading to desired_heading
void qSchool::orient(int curr_heading, int desired_heading) {

  double ut;

	if (desired_heading < 0 )
		desired_heading += 3600;
	if (curr_heading != desired_heading) {
		ut = (curr_heading - desired_heading) % 3600;
		if(ut > 1800)
			ut = -(3600 - ut);
		else if(ut < -1800)
			ut += 3600;
		ut *= 2 * M_PI / 1800;
		MyRobot->action.left = (long)(RADIUS * 5 * ut / 2);
		MyRobot->action.right = (long)(- (RADIUS * 5 * ut / 2));

		//Make sure the robot turns a small amount even if it is close to its target
		if(ut < 0) {
			MyRobot->action.left -= _ROTATE_MIN_VELOCITY_;
			MyRobot->action.right += _ROTATE_MIN_VELOCITY_;
		}
		else {
			MyRobot->action.left += _ROTATE_MIN_VELOCITY_;
			MyRobot->action.right -= _ROTATE_MIN_VELOCITY_;
		}
	}
	if ((curr_heading <= (desired_heading+20)) && (curr_heading >=
(desired_heading - 20))) {
		MyRobot->action.right = MyRobot->action.left = 0;
	       posQuark::orientation = false;
	       posQuark::direction = true;

	}

}

//determines distance between two positions
double qSchool::goLength(position startpoint, position endpoint) {
   long dx1 = long(endpoint.x - startpoint.x);
   long dy1 = long(endpoint.y - startpoint.y);
   double  a = (double)sqrt(dx1*dx1 + dy1*dy1);
   return (a);
 }

//resets values so that robot can run through entire process again
void qSchool::reinitialization() {
  direction=false;
  orientation=true;
  firstTimeFlag=true;
  desired.steer = 0;
  cout << "k: " << (k+1) << endl;
  fout << (k+1) << endl << endl;;
  k += 1;

}
```