01 Jan 2004

Unmanned Vehicle Navigation Using Swarm Intelligence

Sheetal Doctor

Ganesh K. Venayagamoorthy

Missouri University of Science and Technology

Recommended Citation

S. Doctor and G. K. Venayagamoorthy, "Unmanned Vehicle Navigation Using Swarm Intelligence," *Proceedings of International Conference on Intelligent Sensing and Information Processing, 2004*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2004.
The definitive version is available at https://doi.org/10.1109/ICISIP.2004.1287661

Unmanned Vehicle Navigation Using Swarm Intelligence

Sheetal Doctor
Department of Electrical and Computer Engineering
University of Missouri - Rolla, MO 65409, USA
skdkv6@umr.edu

Ganesh K. Venayagamoorthy
Department of Electrical and Computer Engineering
University of Missouri - Rolla, MO 65409, USA
gkumar@ieee.org

Abstract
Unmanned vehicles are used to explore physical areas where humans are unable to go due to different constraints. There have been various algorithms that have been used to perform this task. This paper explores swarm intelligence for searching a given problem space for a particular target(s). The work in this paper has two parts. In the first part, a set of randomized unmanned vehicles are deployed to locate a single target. In the second part, the randomized unmanned vehicles are deployed to locate various targets and are then converged at one of targets of a particular interest. Each of the targets carries transmits some information which draws the attention of the randomized unmanned vehicles to the target of interest. The Particle Swarm Optimization (PSO) has been applied for solving this problem. Results have shown that the PSO algorithm converges the unmanned vehicles to the target of particular interest successfully and quickly.

Keywords
Unmanned Vehicles, Navigation, Optimization, Swarm Intelligence.

INTRODUCTION

Autonomous unmanned vehicles have generated much interest in recent years due to their ability to perform relatively difficult tasks in hazardous or remote environments. Different stochastic iterative search methods have been investigated for optimization of continuous non-linear functions. Various algorithms like evolutionary computations [7], genetic algorithms, adaptive cultural models etc. have been used to perform this task.

Swarm intelligence [5] links artificial intelligence to the concept of fish schooling or swarming theory. It is based on the social behavior of flocks of birds/schools of fish and the success of the group is because of the communication established between them. Particle Swarm Optimization (PSO) reported by James Kennedy and Russell Eberhart [6], in 1995 is based on this simple concept and the paradigms can be easily implemented. It is relatively a new concept and is used for target tracing by autonomous communicating bodies. Other applications using PSO include calibration of instruments [10], generator maintenance scheduling [1], etc.

This paper presents the application of PSO technique for searching targets in a given problem space using a number of unmanned vehicles referred to as *particles* in the rest of the paper. *The* work in this paper has two parts. In the first part, a set of randomized unmanned vehicles are deployed to locate a single target. In the second part, the randomized unmanned vehicles are deployed to locate various targets and are then converged at one of targets of a particular interest. Each of the targets carries transmits some information which draws the attention of the randomized unmanned vehicles to the target of interest. A study has been done on the effect of the number of particles in the swarm and the number of iterations required for converging them at the target(s). The effects of changing some of the PSO parameters on the results have been also studied.

PARTICLE SWARM OPTIMIZATION

Particle swarm optimization is a new concept which broadly falls under evolutionary computation techniques. A problem space is initialized with a population of random solutions in which it searches for the optimum over a number of generations and reproduction is based on prior generations. The concept of PSO is that each particle randomly searches through the problem space by updating itself with its own memory and the social information gathered from other particles.

Within a defined problem space, the system has a population of particles. Each particle is randomized with a velocity and 'flown' in the problem space. The particles have a memory and they keep track of the previous best position (P_{best}) with respect to the target. Thus each 'P_{best}' is related to a particular particle. The best value of all these '$P_{best's}$' is defined as the global best position 'G_{best}' with respect to the target. Therefore each particle has its own 'P_{best}' and the whole swarm has one 'G_{best}'. The velocities and positions of the particles are constantly updated until they have all converged at the target. The basic PSO velocity and position update equations are given by (1) and (2).

These are called the quality factors.

$$V_{new} = w_i * V_{old} + c_1 * rand() * (P_{best} - P_{old}) + c_2 * rand() * (G_{best} - P_{old}) \quad (1)$$

$$P_{new} = P_{old} + V_{new} \quad (2)$$

Where,

- V_{new} - New velocity calculated for each particle
- V_{old} - Velocity of the particle from the previous iteration
- P_{new} - New position calculated for each particle
- P_{old} - Position of the particle from the previous iteration
- P_{best} - Particle' best position
- G_{best} - The best position a particle attained in the

whole population/swarm

w_i - Inertial weight constant

c_1 & c_2 - Weights for the terms dependent on the particles' position (acceleration constants)

The population responds to the factors 'P_{best}' and 'G_{best}' in order to find a better position. The particles are drawn towards the position of their own previous best performance and the best performance of a particle in the group.

The procedure for the implementation of PSO involves the following basic steps [5], [6].

(i) Define the problem space with its boundaries.

(ii) Initialize an array of particles with random positions and velocities. These random positions are initially assigned to be the P_{best} of the particles. Also initialize the target(s) position(s).

(iii) Evaluate the desired fitness function of the particles in step (ii). In this case, the Euclidean distance from the target. Select the G_{best} from the the P_{best} of the particles.

(iv) Compute the particles' new velocities and positions using (1) and (2) respectively.

(v) Check if the particles are within the problem space. If the particles are not within the problem space, then the velocity is set to the maximum velocity (pre-defined) and the particle's new position is set to its previous best position.

(vi) Calculate the new fitness function for all the particles' new positions. Determine the particles' new P_{best}. Compare with the particles' previous P_{best} and update the value with the new one if necessary.

(vii) Calculate the new global best position G_{best} among all the particles' new P_{best}. Compare with the previous best and update the global best before the next iteration.

(viii) The steps (iv) to (vii) are repeated until all the particles have attained their desired fitness.

The differences between particles' positions with respect to the global best (G_{best}) and the respective particle's best (P_{best}) are weighted by the constants c_1 and c_2 and a random number between 0 and 1. The effect of variations of these constants on the performance of the PSO algorithm has also been presented in this paper.

TARGET SEARCH

This paper investigates two types of search problems. The first type involving a single target location and the second type involving two targets but only one target is of interest. Figures 1 and 2 show the graphical representation of the two cases studied. In Figure 2, the darker target is the one to be located by the particles. The particles have all been shown at random locations.

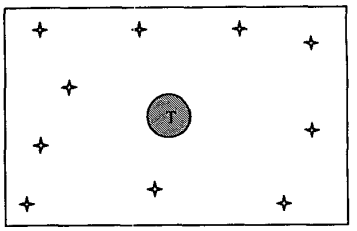

Figure 1. Randomized particles and a target in the search space.

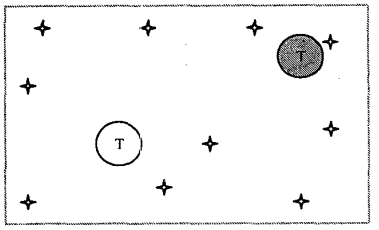

Figure 2. Randomized particles and two targets in the two neighborhoods in the search space. The darker target is the one of interest in the search.

A MATLAB program was developed for implementing the PSO algorithm. The inertial weight w_i was taken to be 0.8 and 0.6 given the dynamic range for w_i to be 0.2 to 1.2. The acceleration constants c_1 and c_2 are taken to be 2, but the study has been carried out for different values of these constants. The other parameters that need to be defined are the domain within which the search is to be carried out and the maximum allowable velocity for the particles needs to be fixed.

For the simulation, the search space was taken as 10 units and the maximum velocity was limited to 2 units. The initial position and velocity for the particles were randomly generated. The successive new velocities and positions were calculated using the equations given in (1) and (2) respectively. Initially the particles' best position P_{best} is the same as the initial random positions. The initial global best G_{best} is calculated from the initial P_{best}. This is done by calculating the Euclidean distance of the particles with the target and then searching through this array for the minimum value. The co-ordinates corresponding to this minimum value is the global best. Within a loop the algorithm calculates the new velocity depending on the parameters passed to it from the previous iteration. The new positions of the particles depend on the current velocity of the particle. After updating the position for every particle, the particles' best position and the global best position need to be recalculated.

The various parameters that can be manipulated for different results are the number of particles, the problem space, the weights and the number of iterations. By increasing the problem space, the relative number of iterations required to reach the target reduced.

In the second part of the simulation, two targets have been chosen in the space. Each target has been assigned a

parameter I, which describes its intensity. The particles deployed here need to be divided into the neighborhoods. For simplicity, the particles have been randomized separately within the local neighborhoods. The procedure for the implementation of PSO involves the following basic steps:

(i) Define the problem space and its boundaries. Also define the intensities of the targets.

(ii) Divide the space into local neighborhoods depending on the number of the targets (in this case 2).

(iii) Randomize the particles' position in the individual neighborhoods and their velocities.

(iv) Perform steps (iii) to (viii) of the general PSO described in the previous section above for both the targets individually. In this case the global best 'G_{best}' is replaced by local best 'L_{best}'.

(v) After the respective particles have converged at their targets, the intensities of the targets are read using sensors located on one or more particles and the target of particular interest is identified.

(vi) Now all the particles need to converge at the desired target and the steps (iii) to (viii) of the general PSO case are repeated.

The logic employed in this part is basically the same as the single target case, only executed twice. First the targets are isolated within a neighborhood. Therefore within this domain, it essentially becomes a single target case with half the number of particles. Similarly the same case exists for the other target. After the particles converge at each of the targets, the intensity of each target is read using sensors that will be mounted on the particles in a practical situation. The desired target is decided on the target having the greater sensor outcome/intensity. The particles at the other targets need to move towards the desired target.

RESULTS

The MATLAB code was executed by varying various parameters. These include the inertial weight, c_1 and c_2, the number of particles and the number of iterations.

The graphs presented below show the results when the code was executed for two different values of the inertial weight. The program was also executed a number of times with different number of particles.

The graph in Figure 3 shows that initially increasing the number of particles in the swarm reduces the number of iterations required to reach the target. But, after increasing the number of particles beyond a certain value, the number of iterations required started increasing. This follows the law of diminishing returns. The same is true for the case where $w_i = 0.8$. But it can be seen that the lowest point in the graph for $w_i = 0.8$ is around 35 to 30 particles in a swarm while for $w_i = 0.6$ is 20 to 25.

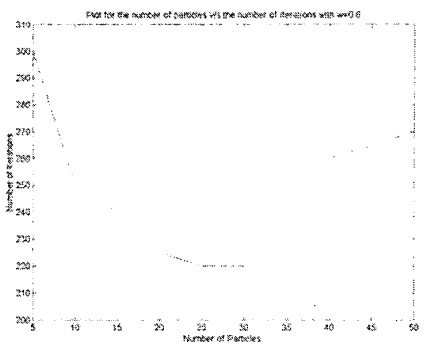

Figure 3. Plot for the number of particles v/s the number of iterations with $w_i = 0.6$.

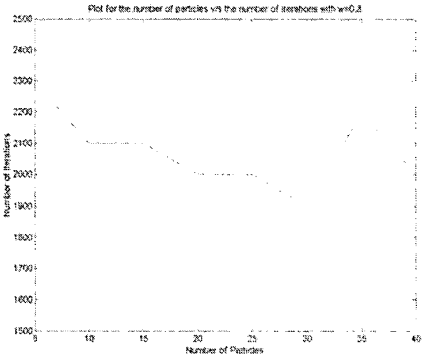

Figure 4. Plot for the number of particles v/s the number of iterations with $w_i = 0.8$.

From the results it can also be seen that by reducing the value of w_i, the number of iterations required to achieve the goal reduces by a large number.

It can also be seen that the two graphs look different. When the value of w_i is 0.8 the number of iterations required to reach the goal does not follow the same path as that for $w_i = 0.6$.

Table 1 below compares the number of iterations obtained by changing the value of the weight w_i in (1). Also increasing the value of w_i, it was seen that the number of iterations required increased. But when w_i was taken greater than 1, the particles reached the extreme ends of the problem space and the velocities saturated at the maximum velocity defined. This happens because for a higher w_i, the velocity is higher. Therefore, the distance covered by the particle between iterations is greater. This results in the particle overshooting the target location and hence would need more number of iterations to get pulled back towards the target.

Also it was seen that by increasing the number of iterations for a fixed number of particles, sometimes some of the particles got randomly saturated. The particles reached the maximum velocity and the extreme edges of the space.

Table 1. Number of Iterations

Number of Particles	Inertial weight w_i	Number of Iterations
5	0.8	2300
	0.6	300
10	0.8	2100
	0.6	250
15	0.8	2100
	0.6	240
20	0.8	2000
	0.6	225
25	0.8	2000
	0.6	220
30	0.8	1900
	0.6	220
35	0.8	2200
	0.6	240
40	0.8	2000
	0.6	260

The system can be made adaptive by varying the weights according to the positions of the particles. A higher value of w_i means that the dependence of the new velocity on the previous velocity is greater. Therefore, to make the program adaptive the value of w_i can be defined greater than one initially. As the particles start approaching the target the velocity needs to decrease. The two terms that depend on the difference of the particles position with the 'P_{best}' and 'G_{best}' positions will become smaller. Therefore, defining a new w_i which is smaller than one, starts reducing the velocity which implies less exploration and more exploitation.

Table 2. Effect of c1 and c2 on the number of iterations with swarm size= 10 and w_i=0.6

c_1	c_2	# of iterations
2	2	250
0.5	2	50
2	0.5	saturates
0.5	0.5	175

The acceleration constants c_1 and c_2 also have an effect on the velocity of the particles. Constant c_1 corresponds to the 'P_{best}' of the particle and c_2 corresponds to the 'G_{best}' term of the velocity equation. The simulation was tested for different values of c_1 and c_2 other than 2. Table 1 shows the results with values c_1 and c_2 taken to be 2. Table 2 shows the effect on the number of iterations for different values of c_1 and c_2. When these values were taken greater than 2 the particles saturated. But it was observed that by reducing these values faster convergence was achieved for the same number of particles and value of w_i. This result shows the importance of the 'P_{best}' and 'G_{best}' terms on the speed of the system. Also another interesting observation was that when the values taken were c_1 =0.5 and c_2 =2, the convergence was faster for a given w_i and number of particles. Since c_2 corresponds to the 'G_{best}' term, it can be seen that the 'G_{best}' term plays a greater role in the convergence of the particles.

The results of the two target case are very similar to the single target case. Here there are two loops in the program. Therefore the total number of iterations is the sum total of the iterations required to identify the target and the number of iterations required for converging at the desired target.

Table 3. Effect of *c1* and *c2* on the number of iterations with swarm size= 10

c_1	c_2	# of iterations w=0.6	# of iterations w=0.8
2	2	340	700
0.5	2	110	270
2	0.5	---	115
0.5	0.5	100	145

The values of the number of iterations vary slightly over the number of times the program is run. This depends on where in the space the particles get initialized. Since the initialization is random, this number varies. Therefore the values above are approximates. The number of iterations required to achieve this task is relatively less than the single target case. This is because once the target interest has been identified; a few particles are already at that particular location hence only the others need to be moved. This can be done by either simply supplying the new location information or forcing the particles to move there or by implementing particle swarm algorithm again. The results shown above are obtained by employing particle swarm again. The number of iterations will decrease further if instead of the PSO algorithm the particles are moved directly to the new location.

CONCLUSION AND FUTURE WORK

It has been shown that PSO can be successfully implemented for a single target with a known location. The paper has also shown results for a two target case. The same algorithm can be extended to multiple targets with known locations and then further to multiple targets with unknown locations. The parameter that describes the target can be for example, the intensity of a light source, the radiation of a source etc. This parameter is important because it helps in identifying the target of particular interest. The results obtained have shown that PSO has potential for application in unmanned vehicles to be used in hazardous and dangerous environments.

Future work involves extending this work to the multiple target case with known and unknown locations, where the problem space needs to be divided into various neighborhoods. The particles will be divided into groups and each group is made to explore their space to find a target. This part has also been shown by taking a two target case and can be extended further to a number of targets. It is important to find a way to divide the particles

into the separate groups especially in the multiple target case. There are different methods that could be employed to define the groups and the neighborhoods. The groups for the neighborhoods can be defined by successively assigning to each target a particle which is closest to it. When one particle from a group has reached its target, all other particles in that group become immobile. The particles read the targets' parameter and then because of the communication between all the particles the desired target location is identified and then all the particles converge at the particular target. The procedure for this is the same as in the case of the single target or two targets.

REFERENCES

[1] Chin Aik Koay; Srinivisan, D., "Particle swarm optimization-based approach for generator maintenance scheduling", *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, April 24-26, 2003, Page(s): 167 -173.

[2] El-Gallad, A.; El-Hawary, M.; Sallam, A.; Kalas, A., "Enhancing the particle swarm optimizer via proper parameters selection", *IEEE CCECE 2002 Canadian Conference on Electrical and Computer Engineering,* Volume 2, 12-15 May 2002, Page(s): 792 -797.

[3] Gudise, V.G.; Venayagamoorthy, G.K., "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks", *Proceedings of the 2003 IEEE Swarm Intelligence Symposium,. SIS '03,* April 24-26, 2003, Page(s): 110 -117.

[4] Xiaohui Hu; Eberhart, R.C.; Yuhui Shi., "Particle swarm with extended memory for multiobjective optimization", *Swarm Intelligence Symposium, 2003. SIS '03, Proceedings of the 2003 IEEE Swarm Intelligence Symposium,. SIS '03,* April 24-26, 2003, Page(s): 193 -197

[5] Kennedy, J.; Eberhart, R., "Particle Swarm Optimization", Proceedings IEEE International Conference on Neural Networks, Volume: 4, 27 Nov.- 1 Dec. 1995, Vol. 4, Page(s): 1942 -1948.

[6] James Kennedy and Russell Eberhart., *Swarm Intelligence* with Yuhui Shi, Morgan Kauffman publishers, San Francisco, CA. ISBN 1-55860-595-9

[7] Michalewicz, Z.; Michalewicz, M., "Evolutionary Computation Techniques and their applications", *IEEE International Conference on Intelligent Processing Systems, ICIPS,* Volume: 1, 28-31 Oct. 1997 Page(s): 14 -25.

[8] Mostaghim, S.; Teich, J., "Strategies for finding good local guides in multi-objective particle swarm optimization (MOPSO)", *IEEE Swarm Intelligence Symposium,* April 24-26, 2003, Page(s): 26 -33.

[9] Peer, E.S.; van den Bergh, F.; Digelbrecht, A.P., "Using neighbourhoods with the guaranteed convergence pso", *IEEE Swarm Intelligence Symposium,* April 24-26, 2003, Page(s): 235 -242.

[10] Peng Yu; Peng Xiyuan; Meng Shengwei., "Virtual instrument parameter calibration with particle swarm optimization", *IEEE Swarm Intelligence Symposium,* April 24-26, 2003, Page(s): 42 -45

[11] Ding Yuigying; He Yan; Jiang Jingping., "Multi-robot cooperation method based on the ant algorithm", *IEEE Swarm Intelligence Symposium,* April 24-26, 2003, Page(s): 14-18.