Learning Basic Navigation for Personal Satellite Assistant Using Neuroevolution

Yiu Fai Sit and Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712, U.S.A.
{yfsit, risto}@cs.utexas.edu

ABSTRACT

The Personal Satellite Assistant (PSA) is a small robot proposed by NASA to assist astronauts who are living and working aboard the space shuttle or space station. To help the astronaut, it has to move around safely. Navigation is made difficult by the arrangement of thrusters. Only forward and leftward thrust is available and rotation will introduce translation. This paper shows how stable navigation can be achieved through neuroevolution in three basic navigation tasks: (1) Stopping autorotation, (2) Turning 90 degrees, and (3) Moving forward to a position. The results show that it is possible to learn to control the PSA stably and efficiently through neuroevolution.

Categories and Subject Descriptors

I.2.6-Learning; I.2.8-Problem Solving, Control Methods, and Search; I.2.9-Robotics

General Terms

Algorithms, Experimentation, Performance

Keywords

Neuroevolution, Enforced Sub-Populations, PSA

1. INTRODUCTION

The Personal Satellite Assistant, or PSA [2], is a small robot that is proposed by NASA to aid the astronauts in daily life and in carrying out experiments and maintenance in the space shuttle or space station (Figure 1). In order to be a helpful assistant, the PSA has to be able to navigate independently and in a controlled fashion. Since the PSA operates in a nearly frictionless environment, it can easily be set into motion but it is more difficult to make it stop. Translational and rotational velocities can increase rapidly, which can be dangerous if it hits the astronauts or the equipment on board. Maintaining controlled navigation is thus the most important task for the PSA.

Control is complicated by the way the thrusters are arranged. A 2-dimensional model of the PSA is shown in Figure 2. All the thrusters generate propulsions in one direction only. Thrusters 0 and 2 propel to the right and therefore together can accelerate the PSA to the left. Similarly, thrusters 1 and 3 together move it forward. The PSA therefore can move forward or sideways to the left along a straight path, but it cannot move backward or to the right. As a result, there is no simple mechanism built in that can stop a forward or left motion. Although control can be made easier by adding enough thrusters to the PSA, such minimalist design can make maintenance easier aboard. Even with enough thrusters, using only four thrusters for navigation is also useful when one or more of the thrusters fail. Learning stable navigation with only four thrusters is thus a meaningful and challenging task.

There are problems with rotation too. Although the PSA can rotate both clockwise (using thrusters 2 and 3) and anticlockwise (thrusters 0 and 1), making it possible to stop autorotation, any positive propulsion also results in a translational force. As the PSA initiates or stops a rotation, it also gains velocity and starts to drift from its original position, and the drift must be stopped as well. In other words, whenever the PSA turns, its position also changes. This interdependence makes navigation in a small environment like the space shuttle or space station difficult.

In this paper, these problems are addressed by using Enforced Sub-Populations (ESP) [3, 4] to evolve neural network controllers for three basic navigation tasks in a 2-dimensional world. The networks are evolved in incrementally more challenging tasks to speed up learning.

Previously, evolving neural networks, i.e. neuroevolution, has been shown efficient in many control tasks such as pole-balancing [12], robot arm control [8], food collection [9], and hexapod walking [1]. ESP is one of the more advanced neuroevolution methods, and has been used successfully e.g. to learn to actively guide a finless rocket [5]. The throttles of the rocket's thrusters are controlled so that it will not tumble. The rocket control task is similar to controlling the PSA and hence ESP is used in this paper. However, whereas the rocket only needs to go up, the PSA has to move around and stop. Such variety in behavior makes the PSA task a versatile next challenge for neuroevolution methods.

In the next section, the three basic navigation tasks for the PSA are defined. The learning methods, the simulation model, and the experimental design are then introduced followed by experimental results, discussion, and conclusion.

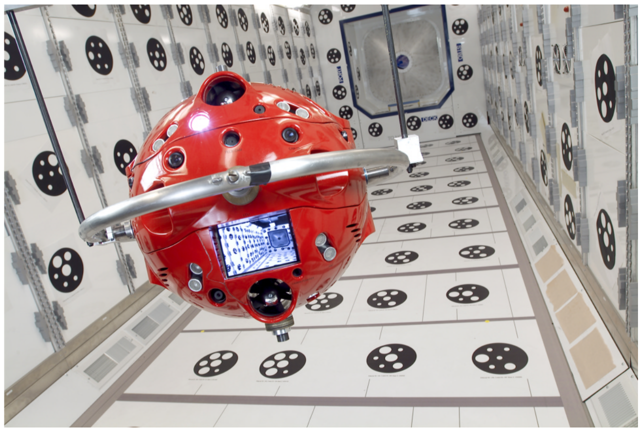

Figure 1: A functional prototype of the PSA in the test facility. The PSA has a video display unit that shows information to help the astronaut on the tasks at hand. There are microphone and speaker that provide two-way communication between the astronaut and the mission control center or other astronauts on board. There is also a camera for remote visual inspection of the instruments and other parts of the space shuttle or space station. There are also range finder, motion sensors and a search light for navigation and other tasks. (Reprinted with permission.)

2. THE THREE BASIC TASKS FOR PSA NAVIGATION

The three basic tasks that are needed for the PSA to maintain stable navigation and remain stationary afterwards are defined as follows:

1. Stopping autorotation, pointing to an assigned direction and remaining approximately at the same position.
2. Turning 90 degrees, stopping and maintaining approximately at the same position.
3. Given some forward velocity, stopping at an assigned position.

To simplify the problem, the PSA is assumed to operate in a 2-dimensional world. Using the solutions found in the first and the third tasks, a moving and rotating PSA can be brought into a halt by first ceasing autorotation and then stopping at a safe position. Simple vertical and horizontal navigation can be accomplished by applying the solutions of the second and the third tasks.

The 2-dimensional model can help us understand and analyze the behavior of the controller more easily and is an important first step for designing the controller. Possible future extension to 3-dimensional space and more interesting behaviors will be discussed in section 6.

3. LEARNING METHODS

3.1 Enforced Sub-Populations (ESP)

Enforced Sub-Populations (ESP) [3, 4] is an advanced neuroevolution method that is able to learn complex control tasks efficiently. Instead of evolving chromosomes that encode complete networks, ESP evolves sub-populations of neurons. For a fully-connected two-layer network with *h* hidden nodes, the population is divided into *h* separate sub-populations. Each chromosome in a particular sub-population is a list of real numbers that represent the weights of the incoming and outgoing connections of the hidden unit with which that sub-population is associated. During evolution, genetic operators only apply to chromosomes in the same sub-population.

To form a complete network for evaluation, one chromosome is chosen from each sub-population to construct the hidden layer. After the network is evaluated, its fitness is passed back to each of the participated chromosomes. In this way, a certain number of networks are formed within each generation to find the fitness of the chromosomes.

The detailed algorithm of ESP is as follows [4]:

1. Initialization. The number of hidden units *h* in the networks that will be formed is specified and a sub-population of neuron chromosomes is created. Each chromosome encodes the input and output connection weights of a neuron with a random string of real numbers.

2. Evaluation. A set of *h* neurons is selected randomly, one neuron from each sub-population, to form the hidden layer of a complete network. The network is submitted to a *trial* in which it is evaluated on the task and awarded a fitness score. The score is added to the *cumulative fitness* of each neuron that participated in the network. This process is repeated until each neuron has participated in an average of e.g. 10 trials.

3. Recombination. The average fitness of each neuron is calculated by dividing its cumulative fitness by the number of trials in which it participated. Neurons are then ranked by average fitness within each sub-population. Each neuron in the top quartile is recombined with a higher-ranking neuron using 1-point crossover at a random boundary between the weights and weight mutation to create the offspring to replace the lowest-ranking half of the sub-population.

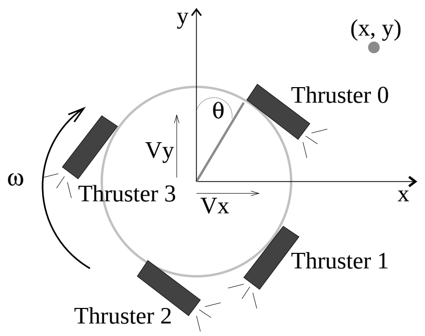

Figure 2: A 2D model for the PSA. Each thruster generates propulsion in one direction only. Thrusters 0 and 2 move the PSA to the left, while thrusters 1 and 3 move it forward. The gray dot at the top-right corner is the destination of the PSA. The inputs to the neural network controller representing the states of the PSA are x, y, V_x, V_y, θ, and ω. The 2D model preserves the essence of the control task and is a meaningful first step to understanding the behavior of the PSA under such configuration of thrusters.

4. The Evaluation-Recombination cycle is repeated until a network that performs sufficiently well in the task is found.

Similar to other genetic algorithms, the diversity within a sub-population in ESP may decline as evolution proceeds. To tackle this problem, a technique called *burst mutation*, which is similar to the idea of Delta-Coding [14], is used. When performance of the whole population has stagnated for a predefined number of generations, burst mutation will be triggered. New sub-populations are created from the best chromosomes by adding noise to them. The noise is generated following Cauchy distribution so that much of the noise is small while there are still some chances for large changes. The resultant sub-populations will contain many chromosomes that are similar to the best chromosomes but also a few that are significantly different. By doing this, the new sub-populations diversify without losing much of the solution found.

3.2 Incremental Evolution

It is often possible to learn faster and solve harder tasks by learning a simpler task first. The solution for the simpler task is then used as a seed to learn a more difficult task. The initial search introduces a bias that makes it more likely to find a solution to the more difficult task. This idea is common in humans and other animals. For example, to learn to run, infants first have to learn to stand and then to walk. In evolutionary computation, this approach is called shaping, or incremental evolution [3, 10, 11, 15].

Incremental evolution is particularly useful for evolutionary algorithms in tasks where suboptimal solutions are abundant and very different from the optimal solution. Such algorithms tend to find suboptimal solutions first. The fitness of the chromosomes for these suboptimal solutions is therefore higher than the others in the beginning, even though some chromosomes may be near the optimal solution. After several generations, the population will be dominated by suboptimal solutions. Since they are very different from the optimal solution, diversification methods like burst mutation may not be able to escape from them in the first try. By first learning a simpler task that has an optimal solution similar to the original task, a more useful bias can be established.

4. EXPERIMENTS

4.1 The Simulator

There is no publicly available simulator for the PSA at present time. A simulator that models the dynamics in a 2-dimensional world was therefore developed as part of this project.[1] The simulator is based on several simplifying assumptions. First, the world is frictionless and has no gravity. This is a close approximation of the environment in which the PSA operates. To simplify learning, there is no noise in the sensors and the outputs of the thrusters. The weight distribution of the PSA is assumed to be uniform, implying that the center of mass is at the center. The thrusters and the simulator operate at 10Hz: At 0.1 second intervals, the thrusters receive control commands and output the desired propulsion throughout this period. Following the expected specifications, the PSA weights 5kg with a diameter of 0.3m, and the maximum power of a thruster is 1N.

The environment is a 3m $\times$ 3m space in which the PSA can move freely. There are no obstacles but they can be added in future experiments to model more complex environments.

[1]The source code of the simulator is available under `http://nn.cs.utexas.edu/keyword?psa`

4.2 Neural Controller Architecture

For each of the tasks defined in the previous section, a separate controller was evolved. Each controller is represented by a two-layer feed-forward network with 18-20 hidden units. This size was found to be effective experimentally; generally any network with fewer or more hidden units performs reasonably well.

The controller receives a vector of 6 inputs every 0.1 second. The inputs consist of the x and y coordinates of the destination relative to the center of the PSA, its velocities along the x and y direction (V_x, V_y), its heading θ, and its angular velocity ω (Figure 2). These inputs are reasonable readings that are expected be available to the PSA. Most of the inputs are scaled so that they are in the range of [-1, 1].

The networks have four outputs with each controlling the force (0 to 1N) that the corresponding thruster has to generate.

4.3 Experimental Setup

Three sets of experiments, one for each of the control tasks, were carried out. In all experiments, each sub-population had 40 chromosomes and ESP ran for 2000 generations. All the tasks are episodic.

4.3.1 Task 1: Stopping Autorotation

At the beginning of an episode, the PSA is at the center of the environment and is set with an initial angular velocity of $\frac{\pi}{2}$ rad/s clockwise. There is no translational velocity and x and y are both set to 0. The desired final heading is 0°. If this task is learned, the PSA can be set to an arbitrary final heading by rotating the x- and y- axes in the input.

An episode ends if the controller successfully completes the task, or if the PSA moves out of the 3m $\times$ 3m environment, the translational velocity exceeds 4m/s, or the angular velocity becomes larger than 2π/s. The fitness is 0 in the last three cases. An episode has a maximum length of 1000 time steps (100 seconds). Although the PSA should be able to stop autorotation in a small fraction of this time, a long episode is needed to ensure the PSA remains stationary.

The most straight-forward fitness function to use is the sum of squared error. Since the PSA needs to point to 0°, it is possible to just add the square of the heading at each time step t ($\sum_{t=1}^{1000}\theta_t^2$). Although a network that accomplishes the task efficiently will have a high fitness, the reverse may not be true. The PSA can drift while keeping a small error in heading. A measure on the position of the PSA has to be included in the fitness function. A commonly used fitness function would be the sum of a linear combination of the errors in heading and position at each time step, and it usually requires tuning the weights of the two errors to achieve the best result. Instead of trying to find the best weights, the fitness function used is the number of time steps that satisfy all of the following constraints:

- $|\theta| < 0.05 \quad (\approx 3^\circ)$;
- $|x| < 0.2$;
- $|y| < 0.2$.

4.3.2 Task 2: Turning 90 Degrees

At the beginning of the episode, the PSA is at the center of the environment. It points to the right ($\theta = \pi/2$) and there is no translational and angular velocity.

The goal is to turn the PSA 90 degrees anticlockwise so that it points to 0°, while staying at the same position. The fitness function is defined similarly to the first task, i.e. the number of time steps that satisfy all the following constraints:

- $|\theta| < 0.035 \quad (\approx 2^\circ)$;
- $|x| < 0.15$;
- $|y| < 0.15$.

For reasons discussed in the next section, this task was learned both by using the above fitness function directly and by incremental evolution using an initial fitness function with relaxed constraints:

- $|\theta| < 0.05 \quad (\approx 3^\circ)$;
- $|x| < 0.2$;
- $|y| < 0.2$.

When incremental evolution was used, each episode consisted of 100 time steps and the neurons were evaluated using the relaxed fitness function initially. The network that learned successfully was then used to seed the population in ESP to learn the fully constrained task using the original fitness function. The episodes were also extended to 2000 time steps.

4.3.3 Task 3: Moving Forward to a Position

The PSA is initially set 0.4m below the destination. It has an upward velocity of 0.04m/s which is equivalent to the result of having both the thrusters on the side generate full power for 1 time step. The goal of this task is to stop at the destination.

This task is difficult because there is no obvious braking action in the PSA. The learning algorithm has to find a way to stop using only forward movements. Incremental evolution was used to learn this task.

The fitness function used initially in incremental evolution is the number of time steps that satisfy the constraints:

- $|\theta| < 0.05 \quad (\approx 3^\circ)$;
- $|x| < 0.15$;
- $|y| < 0.15$.

Each episode had 100 time steps. After this task was learned using the above fitness function, the constraints were tightened:

- $|\theta| < 0.05 \quad (\approx 3^\circ)$;
- $|x| < 0.1$;
- $|y| < 0.1$;
- $|\omega| < \pi$.

The extra constraint $|\omega| < \pi$ is to penalize solutions that increase the fitness by rapidly rotating about the destination. The episodes were also extended to 3000 time steps.

5. RESULTS

5.1 Task 1: Stopping Autorotation

There are two subgoals in this task: (1) stop the rotation at 0°, and (2) stay near to the starting position. ESP learns to stop the rotational motion first, but the resulting position and velocities usually do not satisfy the constraints in the fitness function early on, and later in evolution ESP manages to control this variables as well. The strategy, outputs, and states of the resulting best solution found in 2000 generations are shown in Figures 3, 4, and 5, respectively.[2]

In performing this task, thruster 0 is never fully activated. It is therefore likely that the PSA can be forced to remain closer to the starting position if so desired, by further incremental evolution with tighter distance bounds. Alternatively, its position can be corrected through translation, as is done in task 3.

5.2 Task 2: Turning 90 Degrees

At first, this task appears easier than the first because the PSA can control its rotational speed. It is therefore surprising that ESP is not able to learn this task effectively using the fitness function defined. The best network after 2000 generations turns the PSA only very slowly: It takes more than 30 seconds to rotate 90 degrees and the fitness is very low. The reason is that the neurons that make a fast turn in the early generations do not know how to remain stationary afterwards. Their fitness turns out to be lower than the fitness of those neurons that turn the PSA very slowly but are able to remain nearly stationary. As evolution proceeds, the population becomes dominated by the slow-turning neurons that are very different from the optimal one.

To fix this problem, incremental evolution is used. The task is first shortened to 100 time steps and the fitness function with relaxed constraints is used. Shorter episodes force ESP to learn to turn quickly because a slow turn can never finish the turn and will have zero fitness. The best network is then used as a seed to learn the original task. Since the population is now highly concentrated around a fast-turning solution, the evolution of the neurons is more likely to follow this line. With this method, ESP is able to solve the task within 2000 generations.

The strategy, outputs, and states of the best network are shown in Figures 6, 7, and 8, respectively.

5.3 Task 3: Moving Forward to a Position

This task is the most difficult of the basic tasks because there is no obvious braking mechanism. It was actually not clear if a solution for this task even existed when this experiment was started. By using ESP with incremental evolution, a solution was found.

The strategy learned is rather surprising and creative: The PSA overshoots the target first, then moves back in an elliptical path. The turning action provides a way to make vertical velocity zero by a precise coordination of the thrusters. Near the end of the path, only a sideways motion to the right and a small rotation remain. Motion to the right is easy to stop because the PSA can accelerate to the left (but not to the right). The rotation is stopped at the time that the horizontal movement is halted (Figure 9). The outputs of the thrusters, the velocities, and the positions during the task are shown in Figures 10, 11, and 12, respectively. This solution is an interesting example on how an non-intuitive,

[2]Animations showing evolved solutions for the three tasks are also available at `http://nn.cs.utexas.edu/keyword?psa`

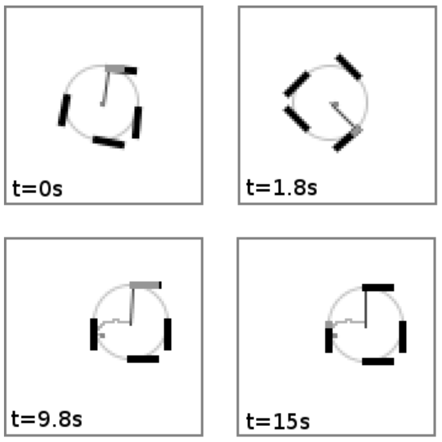

Figure 3: A sequence of snapshots showing how the learned network solves task 1, stopping clockwise autorotation. The thick gray bars represent the force output by the thrusters. The dot at the center is the origin at which the PSA tries to stay. The gray line is the trajectory of the PSA's center. At 0s, the PSA is rotating clockwise; thruster 0 exerts about half of its full power to decrease the angular velocity. The power is then rapidly decreased but maintained at low level to slow down the rotation. When the PSA points downward (at t=1.8s), thruster 0 again increases the output and reduces it gradually. This causes rightward movement but also minimizes vertical displacement. At 9.8s, the heading is about 0° and thruster 0 gives a quick burst to stop the rotation completely. There are some very small adjustments (order of 10^{-4}) afterwards that keep it at the same position (t=15s).

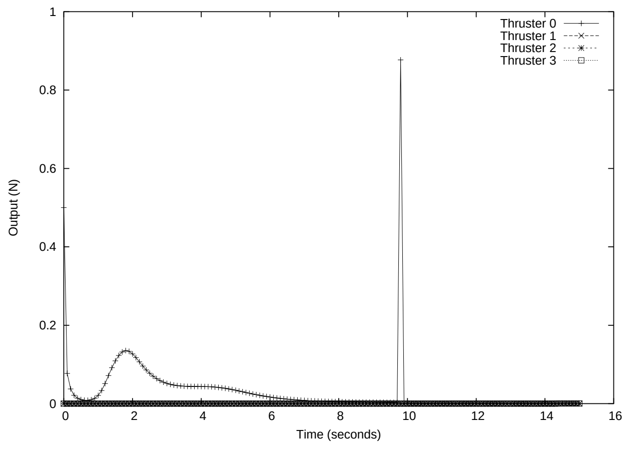

Figure 4: The outputs of the thrusters versus time in task 1. In this task, thruster 0 plays a major role while others only generate insignificant forces. At 0s, thruster 0 is at half of the maximum power which is reduced quickly at the next time step. This action slows down the rotation a little and also produce a movement to the left. The propulsion increases again when the PSA points downward, and is then reduced gradually. The resulting force reduces vertical motion but causes the PSA to accelerate to the right. Since this force is applied for a longer duration, the initial leftward movement is canceled and the PSA starts moving to the right. When the PSA is pointing at 0° again with a slow rotation at 9.8s, thruster 0 gives a sudden shot to stop the rotation and translation.

challenging control problem can be solved using the discovering mechanisms of evolutionary computation.

6. DISCUSSION AND FUTURE WORK

The PSA control tasks in this paper demonstrate the possibility and power of applying neuroevolution to control problems. Even rather complex solutions can be discovered in this way, yet it is still possible for human observers to understand why most actions are taken and what their consequences are. This understanding can give the designers or engineers insight into the control problems in general. In addition to evolving a network that solves the whole task, one may also use neuroevolution methods to obtain partial solutions and ideas on how to extend them to the whole problem, thus speeding up the design process.

The most immediate direction of future work is to make the simulation more realistic by including noise in the sensors and the outputs. Extending the model to a 3-dimensional environment is also a necessary future step. A 3-dimensional model will need, in addition to the *x*- and *y*-axes (Figure 2), an extra *z*-axis that points out of the paper. There will be totally six thrusters that moves and rotate the PSA about the three axes.

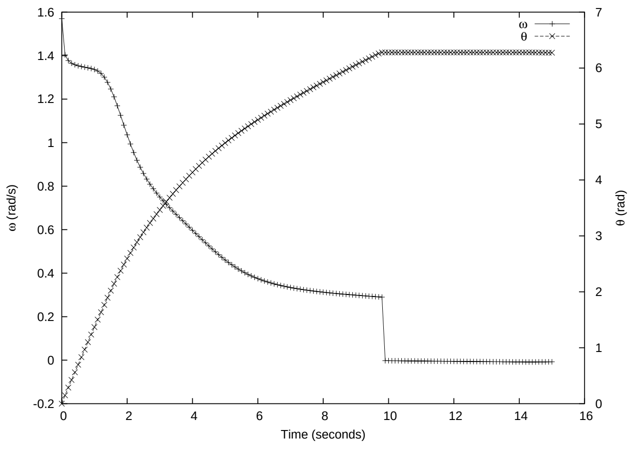

Figure 5: The angular velocity ω and the heading θ versus time in task 1. At the beginning, the PSA is rotating at $\pi/2$ rad/s. This rate was decreased rapidly by the first burst of the thruster 0. It is then gradually reduced as the force from thruster 0 is adjusted. Finally, the rotation is brought to a stop by the sudden force at 9.8s. The heading is displayed using the *y*-axis on the right. The PSA points to $2\pi = 0$ after the rotation is stopped.

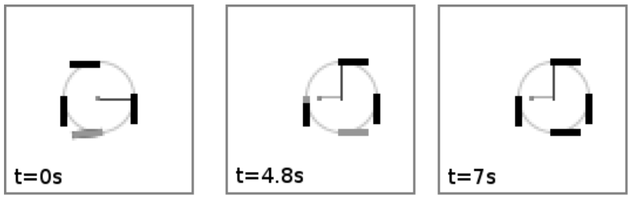

Figure 6: A sequence of snapshots showing how a learned network solves task 2, rotating 90 degrees. At time 0, the PSA is stationary at the origin, pointing at 90°. Thruster 1 gives out full power for one time step and then stops. This action gives the PSA some angular velocity to turn to 0°. Since there is no friction, the rotation continues. The force from thruster 1 also produces a small translational velocity that moves the PSA to the right. At 4.8s, the desired heading is reached and thruster 2 outputs full thrust to counter the rotation and translation. Thruster 3 also outputs a very small force of about 0.001N to help stop the rotation. The PSA then remains stationary afterwards.

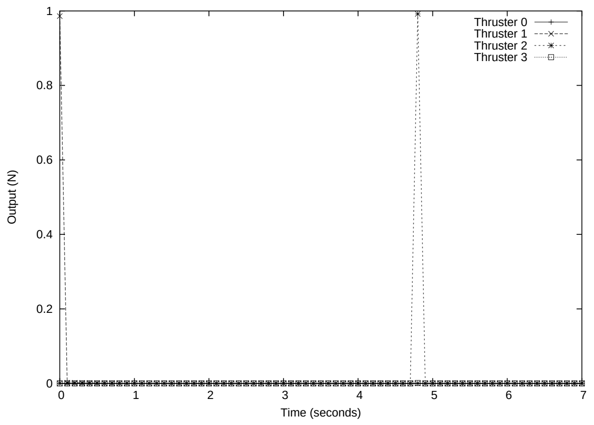

Figure 7: The outputs of the thrusters during task 2. Thruster 1 starts the rotation at time 0 and then its output remains near zero. When the turn is complete, thruster 2 gives a maximum output to stop the motion. The output of thruster 3 at this time is too small to be seen in this figure.

One possible way to approach the 3-dimensional tasks is to use the 2-dimensional task as an incremental step. Interestingly, the solutions found in tasks 2 and 3 in the 2-dimensional space can be extended to the same tasks in the 3-dimensional space. Moving and rotating in 3-dimensional space can be carried out in two steps: (1) move or rotate in the *x*-*y* plane, and (2) move or rotate in the *x*-*z* (or *y*-*z*) plane. Rotating about an arbitrary axis requires the PSA to be able to turn an arbitrary angle in 2-dimensional space, which can be seen as an extension of task 2. However, solutions of task 1 cannot be used directly in 3-dimensional space because the magnitude of the torque in the *x*-*y* plane now depends on the orientation of the PSA in addition to the force exerted by the thrusters. Although the 3-dimensional control tasks are more difficult, the principles of finding a solution as discussed in the 2-dimensional simulation should still apply.

Another interesting direction is to construct more complex behaviors by combining the learned networks of the basic tasks, possibly with further training. Such an approach has been successful e.g. in the robotic soccer domain: Given an appropriate decomposition that reduces the original task into a combination of several independent and simpler tasks, complex behaviors like keepaway can be learned [13]. Since the three PSA control tasks described in this paper are necessary for all stable navigation maneuvers, it might be possible to apply the same approach to learn more interesting behaviors like following the astronaut closely without colliding with him or her.

Incremental evolution was utilized and found powerful in learning tasks 2 and 3. Similar observations have been reported in other difficult domains [5, 15]. Incremental evolution biases the search around the space where good solutions are more likely to be found. It can therefore be understood more generally as a way to inject human knowledge of the task, or advice, to a learning algorithm.

There has been some prior work on taking advice in reinforcement learning [6, 7]. Hand-coded sets of rules or action sequences are added to the policy of the learner. The advisor thus needs to know exactly what has to be done. A more general method would be modeling an advice as an incremental subtask. The advisor only needs to define the subtask instead of knowing how to solve it. One possible way to do this is by incremental evolution. If

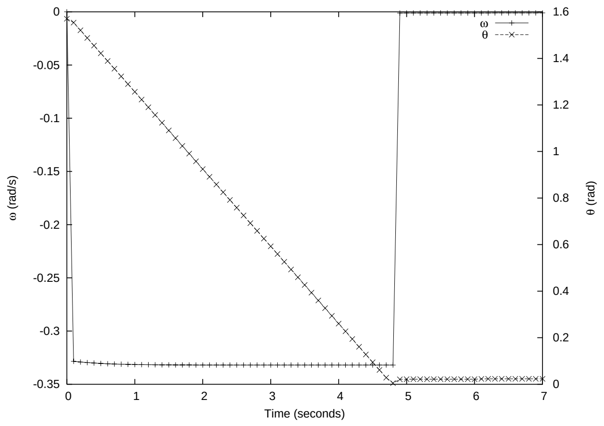

Figure 8: The angular velocity ω and the heading θ versus time during task 2. After the rotation is started, the angular velocity stays about the same until the heading reaches 0. The outputs from thruster 2 and 3 stop the rotation and the PSA stays still, pointing at about 0.4° .

the advice is available at the beginning, a neuroevolution method can learn this subtask first. The resulting network can then be used to seed the population that learns the complete task. This lets the learning algorithm figure out what the best solution for the subtask is instead of having someone to hand code it.

As an example, the advice in learning task 2 is ”turn quickly and try to stay at the same position” and the neuroevolution method is able to learn how to do it without intervention. Such incremental evolution can be an easier and more natural way of providing advice than supplying actual action sequences in tasks like the PSA control. The challenges of such approach would be learning new subtasks without forgetting the learned ones and taking advice during the learning of the whole task. If these challenges can be met, incremental evolution from advice may make it possible to evolve controllers for highly complex devices in the future.

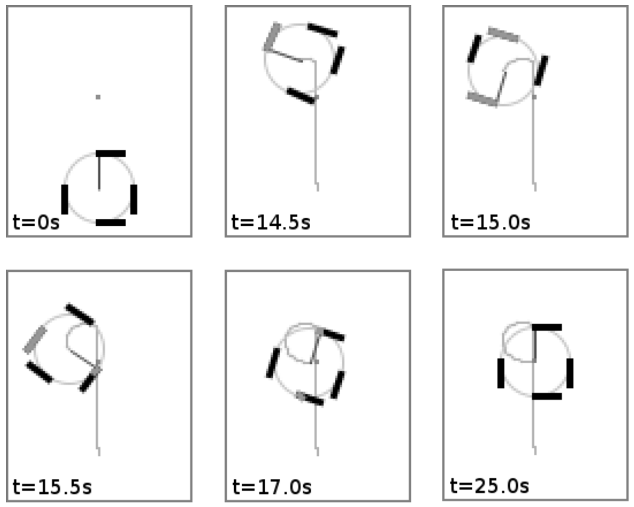

Figure 9: A sequence of snapshots showing how the learned network solves task 3, moving forward and then stopping at an assigned position. At the beginning, the PSA has an upward velocity of 0.04m/s. It keeps going forward until it overshoots (t=14.5s). Then it turns left and moves in an elliptical fashion by activating thruster 0. After a short while (t=15.0s), thruster 2 kicks in, accelerating the PSA towards the destination. Thruster 0 then decreases its output while thruster 2 keeps a full throttle (15.5s). This action slows down the rotation. Both of the thrusters then generate a small amount of thrust to counteract the elliptical and downward motions (17.0s), until the PSA ends up at the destination with a heading of $0°$. There is no vertical velocity at that point and only some horizontal movement which is stopped by thrusters 0 and 2. The PSA then stays at the destination (25.0s). Such complex control sequence is necessary because the arrangement of the thrusters does not allow a simple braking action. Discovering it automatically is an important demonstration of how complex and counter-intuitive problems can be solved through evolutionary computation.

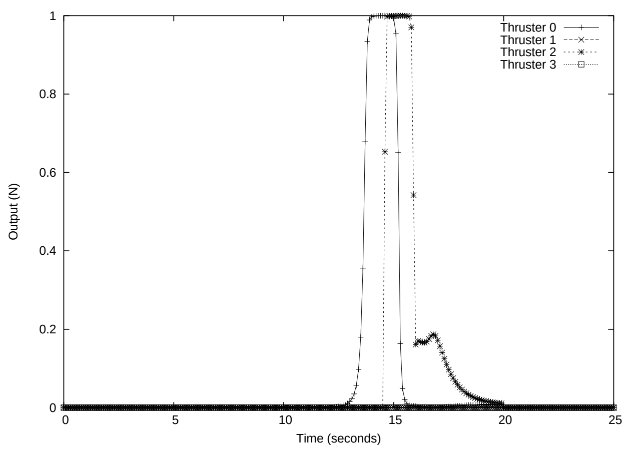

Figure 10: The outputs of the thrusters during task 3. After the PSA overshoots, thruster 0 increases its output sharply towards full power. After the PSA is set in an elliptical path, thruster 2 is activated to accelerate the PSA towards the destination. About 0.5 seconds later, thruster 0 reduces its output rapidly. Thruster 2 maintains low-level throttle, slowing down the rotation set up by thruster 0. This action also reduces the downward velocity. When the PSA is near the destination, there is no vertical motion. Thruster 0 increases its output by a little and together with thruster 2, they stop the horizontal movement of the PSA.

7. CONCLUSIONS

In this paper, the Enforced Sub-Populations neuroevolution method was applied to solving three basic PSA navigation tasks. The results show that neuroevolution is a promising approach to solving even complex and non-intuitive control problems. Incremental evolution was found to be crucial in guiding the evolution search towards spaces where good solutions are more easy to find. In the future, it might be possible to devise a principle approach to controller design based on combining human knowledge with the incremental evolution methods.

Acknowledgements

We would like to thank Dr. Gregory A. Dorais of NASA Ames Research Center for providing detailed information about the PSA as well as the image in Figure 1. This research was supported in part by the National Science Foundation under grant EIA-0303609.

8. REFERENCES

[1] R. Beer and J. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122, 1992.

[2] Y. Gawdiak, J. Bradshaw, B. Williams, and H. Thomas. R2d2 in a softball: The portable satellite assistant. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 125–128, 2000.

[3] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[4] F. J. Gomez. *Robust Non-Linear Control through Neuroevolution*. PhD thesis, University of Texas at Austin, Austin, TX, 2003. Department of Computer Sciences Technical Report AI-TR-03-303.

[5] F. J. Gomez and R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, 2003.

[6] G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer. In *The AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems*, 2004.

[7] R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–281, 1996.

[8] D. E. Moriarty. *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, 1997. Technical Report UT-AI97-257.

[9] S. Nolfi, J. L. Elman, and D. Parisi. Learning and evolution in neural networks. *Adaptive Behavior*, 2:5–28, 1994.

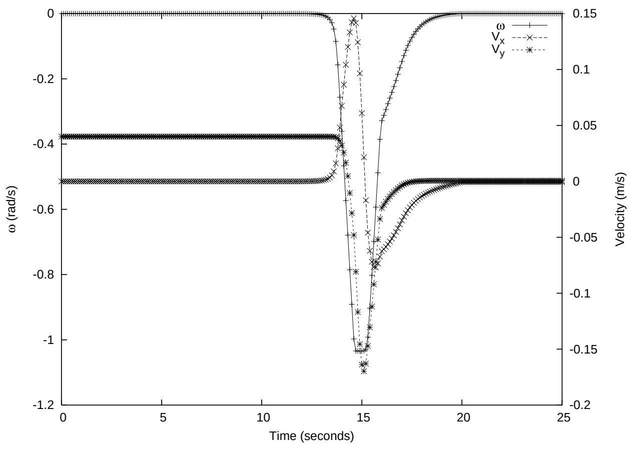

Figure 11: The angular velocity ω and the translational velocities, V_x and V_y along the x and y directions in task 3. The activation of thruster 0 sets the PSA into rotation and at the same time introduces translational motion along both directions. The angular velocity stops increasing when thruster 2 kicks in and then decreases when the output of thruster 0 is reduced. This series of motions stops the vertical movement ($V_y = 0$). Both the remaining horizontal and angular motions are halted by thruster 0 and 2 at about the same time (20s).

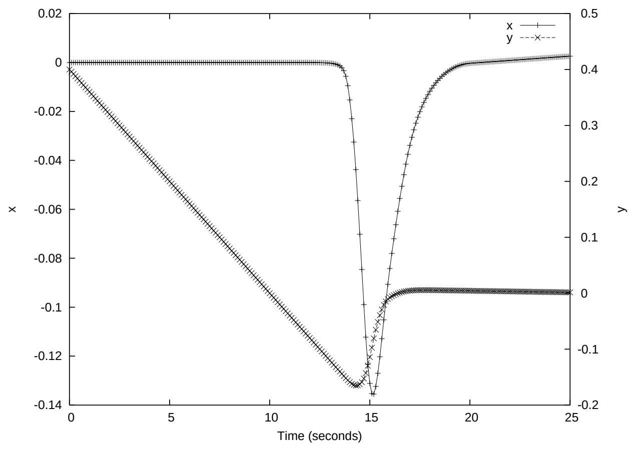

Figure 12: The x and y position of the destination relative to the PSA in task 3. After the overshoot ($y < 0$), the rotation first brings the PSA level to the destination ($y = 0$). Then the PSA moves horizontally until the motion is stopped by thrusters 0 and 2. There are some very small residue velocities (in the order of 10^{-4}) that move the PSA.

[10] S. P. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339, 1992.

[11] B. F. Skinner. *The Behavior of Organisms*. B. F. Skinner Foundation, Morgantown, WV, 1938. Reprinted in 1999.

[12] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, 2002.

[13] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone. Evolving keepaway soccer players through task decomposition. *Machine Learning*, 2005. To appear.

[14] D. Whitley, K. Mathias, and P. Fitzhorn. Delta-Coding: An iterative search strategy for genetic algorithms. In *Proceedings of the Fourth Internation Conference on Genetic Algorithms*, pages 77–84, 1991.

[15] A. P. Wieland. Evolving controls for unstable systems. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Connectionist Models: Proceedings of the 1990 Summer School*, pages 91–102. 1990.