The Use of Neural Networks and Reinforcement Learning to Train Obstacle-Avoidance Behavior in a Simulated Robot.

Michael Schuresko
Machine Learning (cmps 242)
Winter 2005

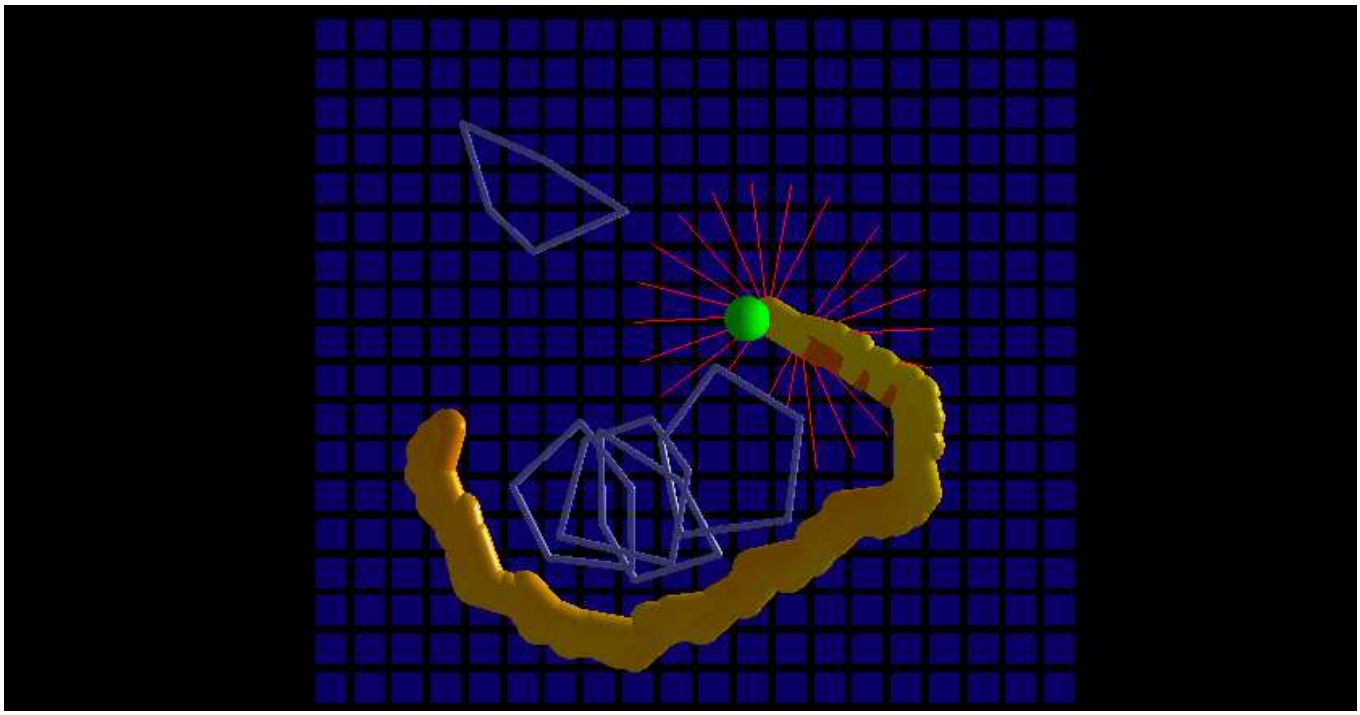

Fig 1. Path of the robot after learning, showing obstacles, goal, past robot positions, and range finders

Abstract

I programmed a simulation of a robot to use neural networks and reinforcement learning to reach a goal position while avoiding obstacles. The preceding figure shows a visualization of the path taken by the robot during such a simulation, displaying the robot in yellow/orange, the obstacles in light blue, and the goal in green.

Introduction / Problem Description

Obstacle avoidance is a common problem in the field of mobile robotics. In this program, the task was to reach a specified "goal" location, given bearing to the goal, some measure of distance to the goal, and a simulation of laser rangefinders to detect the presence of obstacles. The "obstacles" were pseudo-randomly-generated polygons of a given expected radius. This particular problem is

important to nearly every aspect of mobile robotics, as a robot cannot accomplish very many tasks in an open environment without navigating and avoiding obstacles.

Related work

Much work has been done on robotic path-planning. In general, algorithmic path-planning methods are preferable to reactive policies when such things can be used. Drawbacks to algorithmic path-planning include both the required computational time, and the necessity for a fairly accurate world model with which to plan. However, some notable approaches to robot obstacle avoidance have used either reactive or learning-based methods. (see glossary for "reactive obstacle avoidance")

In general reactive obstacle avoidance methods have trouble with getting robots stuck in local minima. However, they still have their place in mobile robotics. They tend to be more robust to changes in the environment then path-planning methods which require a map of the obstacles. They also tend to execute quite quickly, which is a big advantage when dealing with a robot that has to interact with the environment in real time. A reactive obstacle avoidance method can also be used as a lower-level behavior for a path planner, either by having the path planner generate "waypoints", which are locations that the robot can drive between using a reactive algorithm, or by using the reactive algorithm as a low-level behavior for emergency obstacle avoidance. The last use of reactive obstacle avoidance fits very well into a robot control framework known as a "subsumption architecture" (see Jones and Flynn). A subsumption network is a network of interacting robotic behaviors (or agents), each of which can listen to sensor inputs and choose to "fire". When multiple behaviors fire, there is a precedence for which behaviors get control of the robot at a given time. Frequently behaviors with the highest precedence are agents like "back up if I'm about to bump into a wall" or "stop if a moving object wanders into my immediate path".

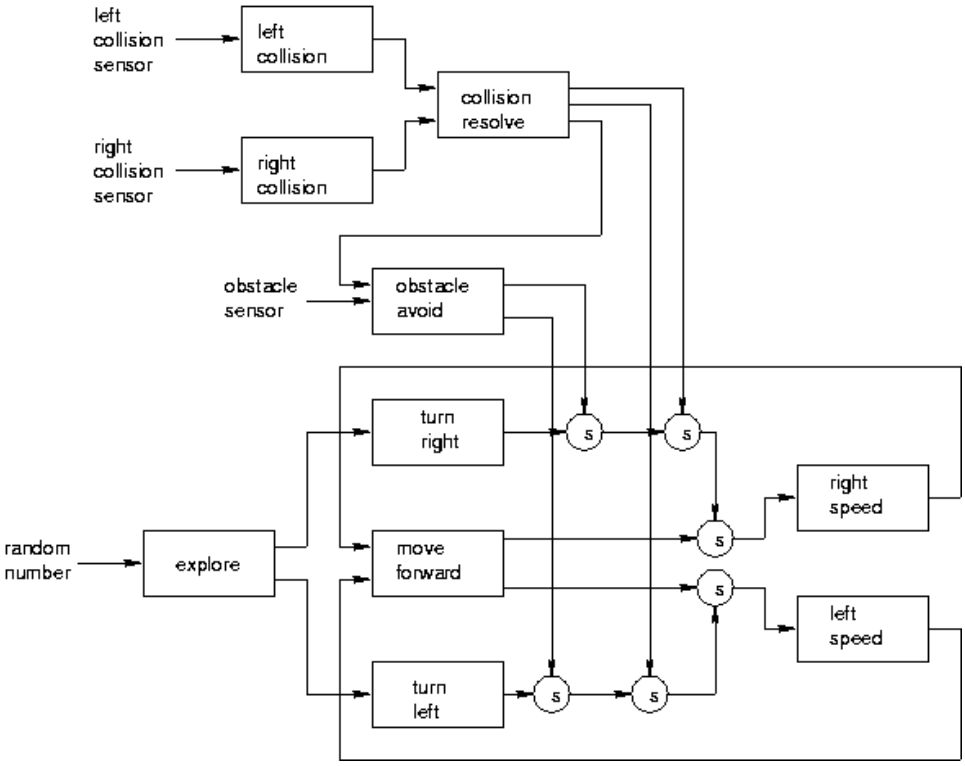

Fig 2: A sample subsumption network, courtesy Chuang-Hue Moh (chmoh@mit.edu). Taken from pmg.lcs.mit.edu/~chmoh/6.836/proposal.doc

Grefenstette and Schultz have used genetic algorithms to learn policies for a variety of robot tasks, including obstacle avoidance. It is somewhat of a tricky issue determining whether Grefenstette's work constitutes "reinforcement learning", as the particular genetic algorithm he used, involved reinforcement learning for a Lamarckian evolution component to the algorithm. Some of his genetic algorithms have produced spectacular results, not only for obstacle avoidance, but also for sheep-and-wolf games and other tasks in the intersection between game theory and control theory. However, the number of iterations required for convergence of this algorithm essentially necessitates that the robot learn in simulation. Unfortunately, I believe this to be the case with my experiments as well.

J Borenstein uses reactive technique called "vector field histograms" for obstacle avoidance. His technique does not use any machine learning at all. It has the advantage of allowing the robot to take advantage of previous sensor readings in the form of an "evidence grid", or probability map of obstacle occupancy.

Su Mu Chun et al have done work using reinforcement learning, and an object very much like a neural network, but with Gaussian operators at nodes instead of the standard logistic function. It looks as if they might have an algorithm that converges quickly to decent behavior, however it is unclear from their paper, as the number of time steps required to perform various learning tasks are described as "after several iterations" and "after a sequence of iterations"

Gaudiano and Chang have a paper on a neural network that can learn to avoid obstacles on a real robot wandering in a cluttered environment. The "real robot" part is actually somewhat important,

as some of the advantages of learning the obstacle avoidance task include the ability of learning algorithms to cope with wear and tear on the physical robot, far better then a pure-controls approach. I believe that their work is on the task of a robot learning to wander while avoiding obstacles, rather then the harder problem of learning to reach a particular location while avoiding obstacles.

Grefenstette and Schultz also have a paper on how a learning-algorithm can adaptively cope with changes to a physical robot. Their genetic-algorithm based technique requires learning in simulation, but is capable of producing extremely creative problem solutions. For example, when they clamped the maximum turn rate of one of their simulated robots, it learned how to execute a "three-point turn", much like human drivers do on a narrow street.

It is worth noting that much of the mobile robotic learning research assumes a different kinematic model for the robot then in my project. Frequently the assumption is that the robot is "non-holonomic". The precise definition is somewhat complicated, but in robotics it usually means that the robot can take actions to change any one of its state variables independently of the others. For instance, a mobile robot that can move in any direction without changing its orientation is non-holonomic. A car, however, cannot move perpendicularly to its current direction. It must instead move forward and turn until it reaches an orientation from which it can drive in a particular direction. Many research robots approximate non-holonomic motion by having wheels which can independently re-orient themselves over a 360-degree range. This distinction is not as relevant in the pure obstacle-avoidance case I treat here as it would be in the case of "parallel parking", but it does have some non-negligible effects.

Methods used

Simulation details

Originally I had intended to teach a robot how to parallel park. For ease of coding reasons, I eventually simplified the problem to teaching a simulation of a robot to reach a goal position from a starting configuration while avoiding obstacles. The robot had a kinematic model similar to that of an automobile (i.e. it had a front wheel, and a rear wheel, the rear wheel maintained constant orientation, while the front wheel could turn within a fixed range). In the simulation the robot had a fairly simple suite of sensors. The goal sensors returned the relative angle between the robot orientation and the goal heading, as well as the inverse square of the distance between the robot's rear axle and the goal center. This is physically reasonable, as a radio transmitter on the goal could easily return a noisy approximation of similar data. Obstacles were sensed using "laser rangefinders", which, in the simulation, were rays traced out from the robot body until they hit an obstacle. Conventional research robots traditionally use sonar sensors for detecting obstacle range, but the returns from those are more difficult to simulate efficiently, and laser range-finders are not unheard-of. I used the inverse of the ray-cast distance as the sensor data from each range-finder. This is also physically reasonable for a time-of-flight laser rangefinder or sonar sensor, as time-of-flight is c/distance. The rangefinders had a maximum returned distance of 30 units. If the rays shot out encountered nothing within this distance, the return value would simply be 1/30.

The obstacles were randomly generated polygons, chosen so as not to intersect the initial robot or goal positions, and so as to have a given expected radius (I think I used something in between 15 and 20). The units used in this simulation should make more sense in the context of the picture. The robot is 20 units long and 10 units wide. The red lines indicating the range-finder beams are each at most 30 units long.

Below is a picture of approximately what I am describing

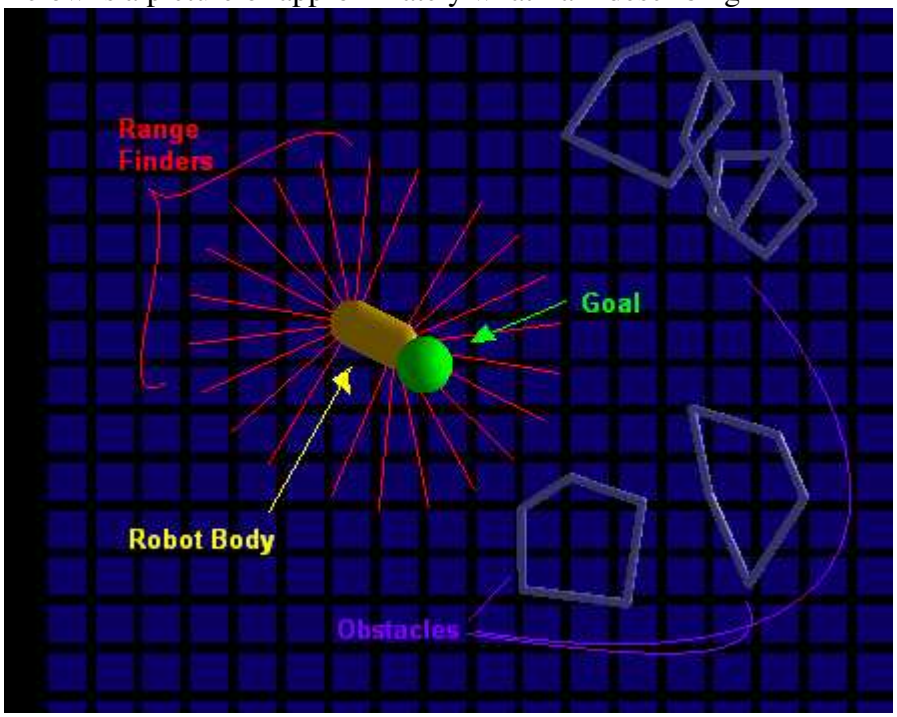

Figure 3: A more detailed picture, showing components of the simulation

The robot "car" was a pillbox, or a rectangular segment with hemispherical end-caps. Because I was training the robot to perform "obstacle avoidance" it became necessary to have a somewhat fast collision-detection system, and pillboxes are among the easiest shapes to perform collision-detection on. The robot was moving slowly enough compared to its own radius and the obstacle sizes that I had the luxury of only performing surface collisions between the robot and the obstacles, rather then "is the robot inside the polygon" tests or the even-harder "did the robot move through the obstacle in the last time-step" tests.

Controller details

Rather then explicitly learning a control policy, I instead learned a mapping between state/action pairs and reward values. At every time step, the robot evaluated its sensors, then searched through the space of possible actions, generating an expected reward for each one. It then chose an action probabilistically based on the learned evaluation of that action in the given state.

$$Pr(Action=Ai) = Reward(A_i)^k / \Sigma_{Aj \in Actions} Reward(A_j)^k$$

For evaluation purposes, I chose k to be 20, having the effect of placing an emphasis on the actions of highest expected reward. In addition to the advantage of random choices between two actions with identical outcomes, this also allowed me to easily anneal the "exploration" tendencies of my robot over the set of training runs.

I ended up using a total of 22 rangefinders, 11 for the front, and 11 for the back. It is worth noting that there were no rangefinders along the sides of the robot (see picture), which had interesting behavioral results I will discuss later.

I ended up restricting the space of possible "actions" to 3 choices for turning rate (left, right, and go straight) and 2 choices for the speed control (forwards, and reverse). I would like to try an experiment in which the robot controls acceleration over a range of speeds, rather then setting the speed directly.

Learning details

Neural network

I experimented with several different topologies, but the one I ended up using was a single feed-forward neural net with two hidden layers. The first (i.e. closest to the inputs) hidden layer had 15 nodes, while the second had 5. Both "having fewer layers" and "having fewer nodes in the first hidden layer" made the robot converge faster in the case of no obstacles, but degraded the performance in the presence of obstacles. Having fewer range-finder sensors (i.e. fewer inputs) also caused the obstacle-avoidance performance to degrade (for obvious reasons, in retrospect). Each hidden node had sigmoid functions applied to the outputs and an extra constant input. I used the standard back-propogation algorithm for the neural network.

Reinforcement learning

I ran a series of training episodes, during which I would go through the cycle of

1) Randomly placing the robot, goal and obstacles within a 150 by 150 square
2) Running the robot and storing previous state-action choices
3) Assigning rewards.

For each training episode, I ran the robot for a set number of iterations, or until it collided with an obstacle or the goal. I got the best qualitative results by setting the number of iterations per episode to 400. At the end of the episode, I assigned a reward to the final state-action pair in the following way.

Case Hit Goal: Reward = 1

Case Hit Obstacle: Reward = 0

Case "went outside of bounds" (defined to be 150 units away from the goal): Reward=0

All other cases: Reward = Reward predicted by the Neural network for the last action taken.

I would then train the neural network on all state-action pairs for actions chosen during the episode with an exponentially discounted reward. In order to put extra penalty on the obstacles (and on going outside of bounds) I had the rewards exponentially decay to "0.6" rather then 0. For most experiments, I chose the "discount per iteration" to be 0.98. While this may seem to be a somewhat small discount, after hundreds of iterations it significantly compounds. $.98^{200}$ is about .017. $.98^{400}$ is a very negligible .0003. I decided not to use the standard Q-learning technique of training on each state-action pair with the discounted max value of the state-action pair resulting from that state. My reasoning was that I believed the neural network training rate would cause results to trickle extremely slowly from far future actions. In fact, I believe that the contribution of an update from an action k steps into the future is something like $(\text{learning rate})^k$.

Evaluation Methods

Every so many iterations, I would randomly generated a scenario from which the robot would not learn. I collected the results of these experiments over time, and plotted 200-episode moving averages of "frequency of hitting goal" and "frequency of not colliding with obstacles". The advantage of training on simulated data, is that it is quite easy to simulate extra episodes to hold out as a "test set"

Software Used

C++, OpenGL (for the visualization) and MFC (for the gui)

It actually turned out to be a good thing that I wrote a decent visualization for the robot, as I had initially written bugs into my collision-detection that I don't think I would've found easily otherwise.

Results

Qualitative Results

After training for one million episodes with 5 obstacles of radius about 20, the results looked pretty good. The robot actually appeared to make it to the goal more frequently then my evaluations showed. I believe this to be because my evaluations considered the robot to have "missed" the goal if a certain number of timesteps elapsed.

While I cannot include animations in my report, I did turn on an option for the robot to leave "trails" as it navigated around obstacles. I have included some pictures of this in my report.

I also encourage anyone reading this report to go to http://www.soe.ucsc.edu/~mds/nnet_page/ and download the windows executable for the visualization/simulation/learning setup, as well as a few of the neural networks that I trained.

One of the neat things I did in the visualizations was to set the color of the robot at each time-step proportionally to the "reward" value returned by the neural network for that state and the action it

chose. Red indicates “poor expected reward” and yellow indicates “good expected reward”. Hopefully you are reading a color printout of this report, and can see these differences.

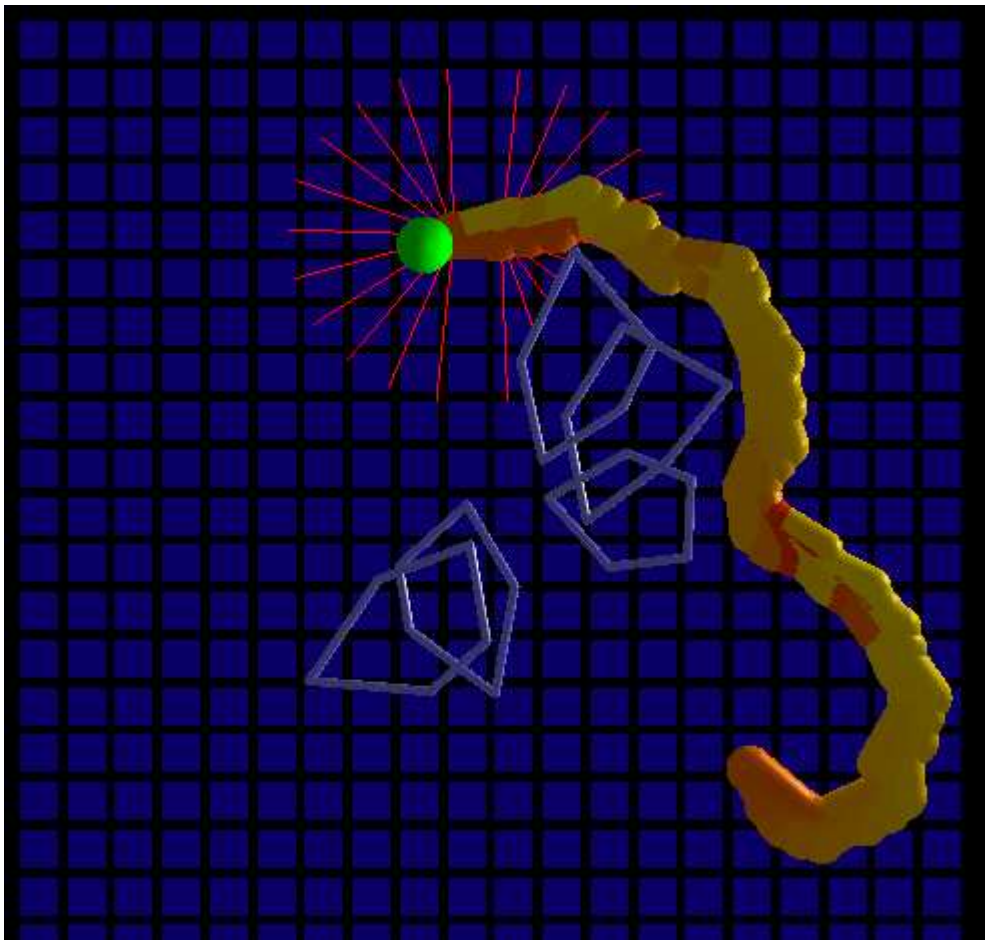

Figure 4: The robot successfully navigating around obstacles.

This was a particularly impressive shot of obstacle avoidance, as the robot quite clearly changed paths when it got within sensor range of the obstacles. Also note that the color is flashing red/orange whenever the robot heads towards an obstacle, indicating poorer expected reward for these states.

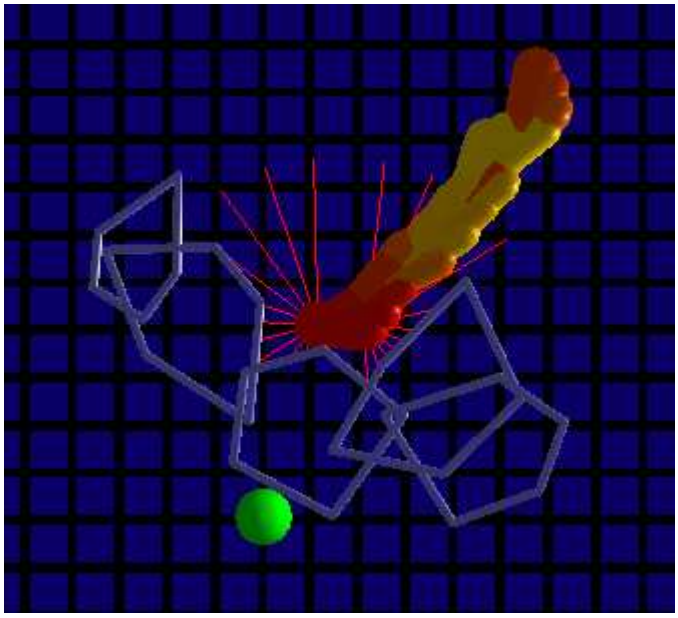

Figure 5: The robot failing to avoid obstacles

This is a path taken by the robot when it failed to reach the goal. Note that it appears to have attempted to turn away from the obstacle. Also note that the "reward color" turned to "poor reward red" before the collision. I believe that in this case the collision was aided by a peculiarity of the car-like kinematics of this simulation. An odd thing that happened, is that it appears that the robot learned to drive almost entirely in "reverse". As anyone who has tried to parallel park knows, you get far more turning maneuverability when driving in reverse, in exchange for this slight problem where when you turn to the right, the front of the car actually swings to the left for a brief period of time. In most of the training runs I visualized the robot explicitly avoided doing this when near an obstacle, but in this case I believe it tried to swing to one direction, and ended up colliding into the obstacle from the side in the process.

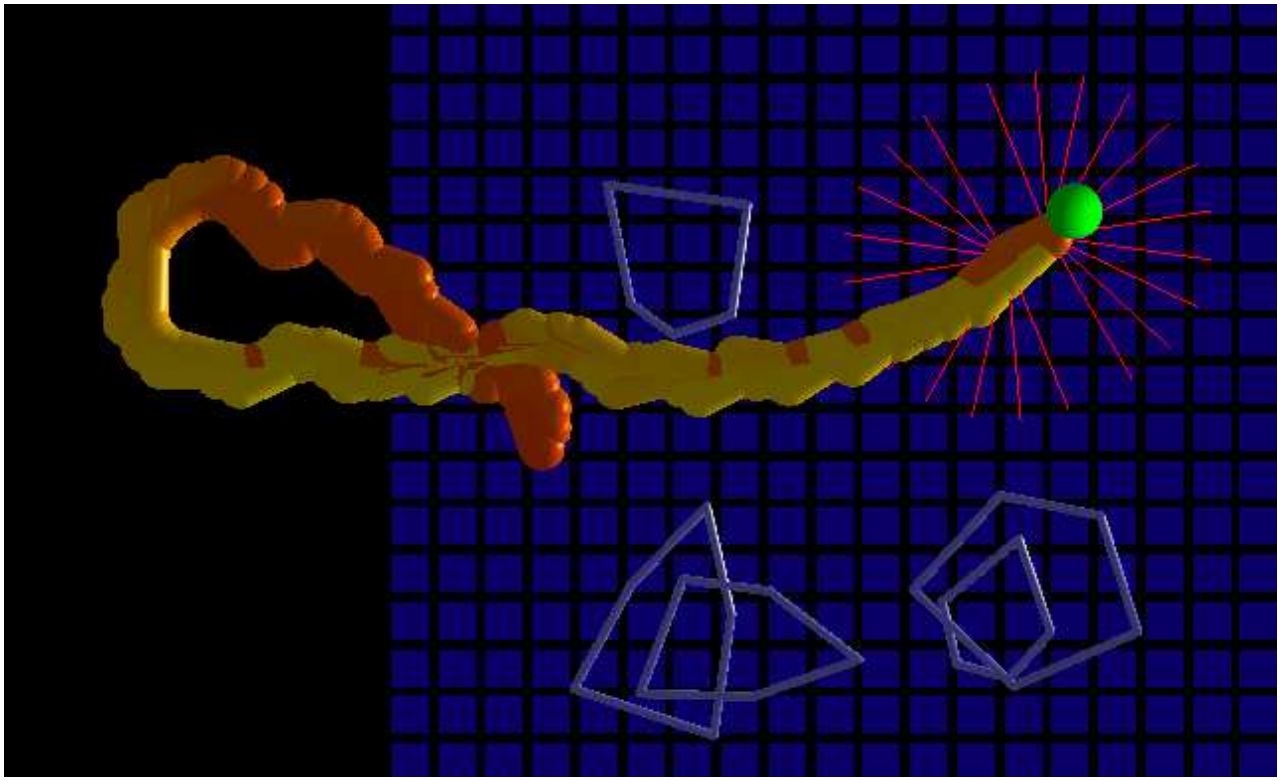

Figure 6: The robot turning around in order to drive backwards.

In fact the robot learned such a strong preference for driving in reverse, that it would frequently turn around so that it could approach the goal backwards, as seen in figure 6.

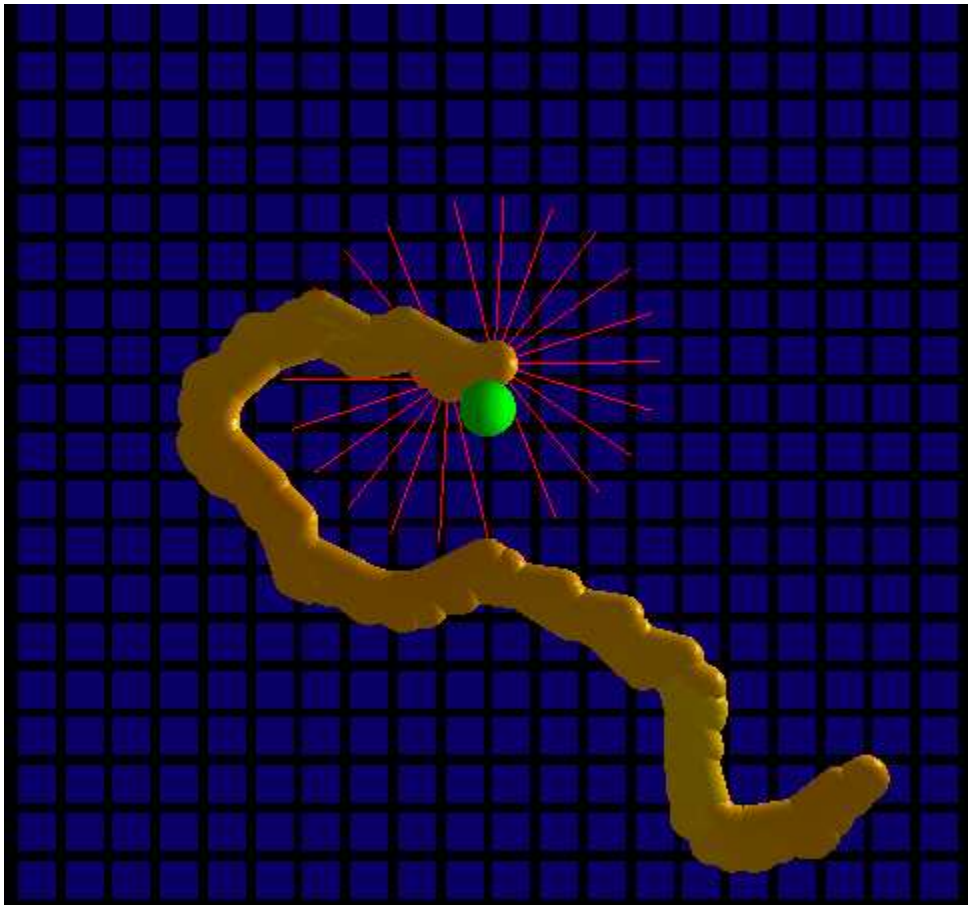
Figure 7 Idiosyncrasies in paths taken when learning in the absence of obstacles

When learning in the absence of obstacles the robot took far fewer training runs to converge then it did with obstacles. However, it sometimes produced somewhat strange paths as seen here.

Quantitative Results

The robot appeared to learn the task of reaching the goal in the absence of obstacles quite quickly.

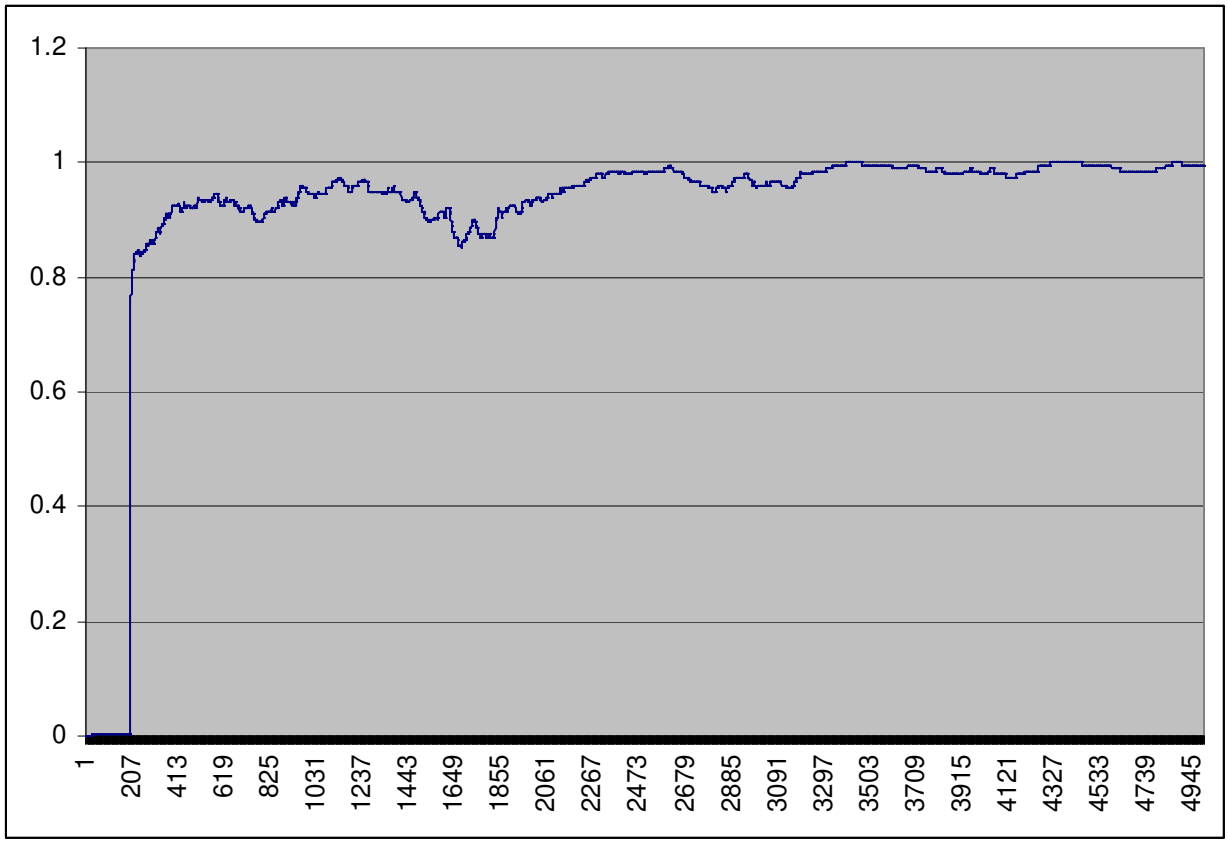

Figure 8: Average probability of reaching goal in the absence of obstacles

Figure 8 shows the average probability of reaching the goal in the absence of obstacles from the evaluation runs of a 100000 episode learning experiment. Evaluation was done once every 20 episodes, so to get the number of training episodes, simply multiple the numbers on the bottom of the graph by 20.

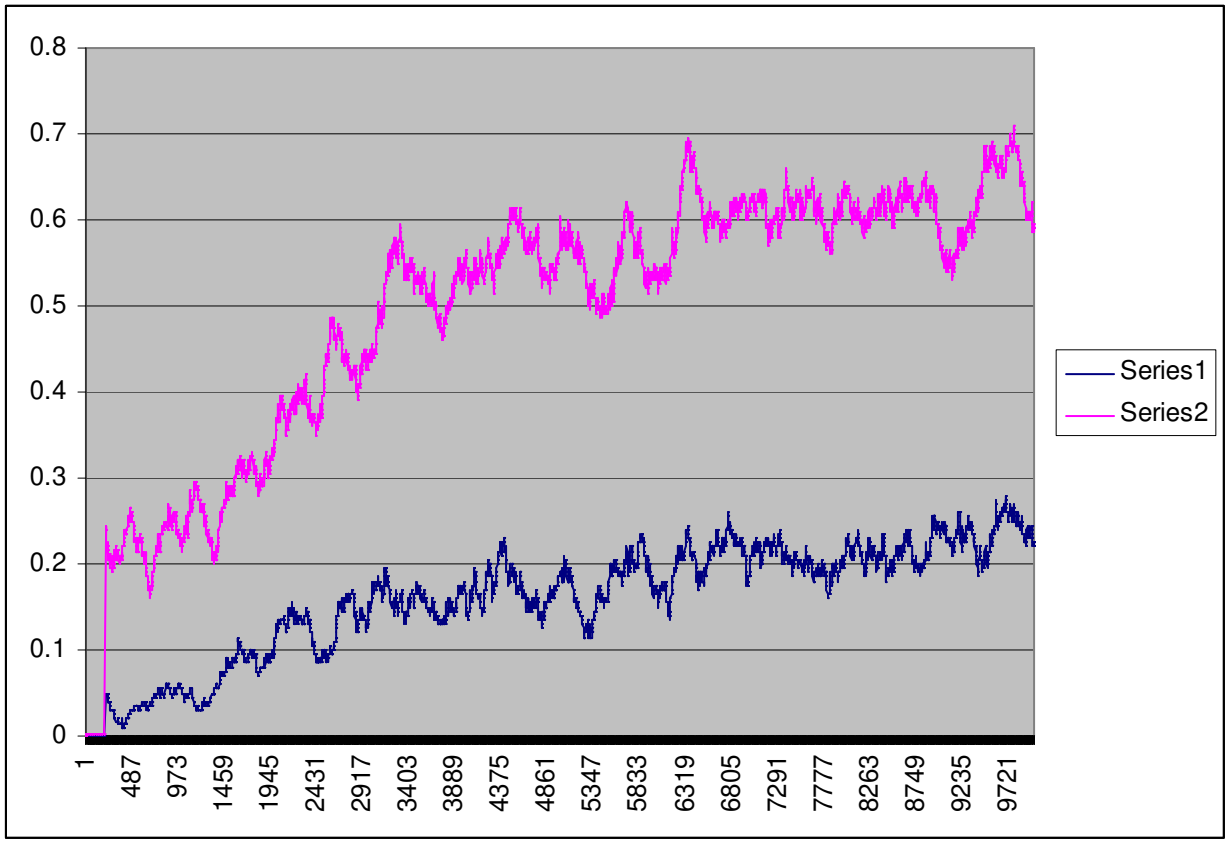

Figure 9: Obstacle avoidance and goal-reaching results for 5 obstacles of radius 20

Figure 9 shows the results of "obstacle avoidance" (Series 2, in pink) vs. "frequency of reaching the goal within the allotted time" (Series 1, in blue) over a 1-million episode training experiment.

When I manually ran the results of the learning in the visualizer, the robot appeared to reach the goal far more often then the 1/5th of the time indicated by this graph. However, as noted before, the evaluation runs tracked the probability of hitting the goal within a fixed time allocation.

In order to see if the robot was actually better at reaching the goal then the graph indicated, I plotted statistics for the same neural network, allowing the robot more time steps to reach the goal.

In this experiment I simply used the network from the million-episode training run and turned learning off. The flat rate of the resulting graph should not, therefore, be taken as an indication of "lack of learning"

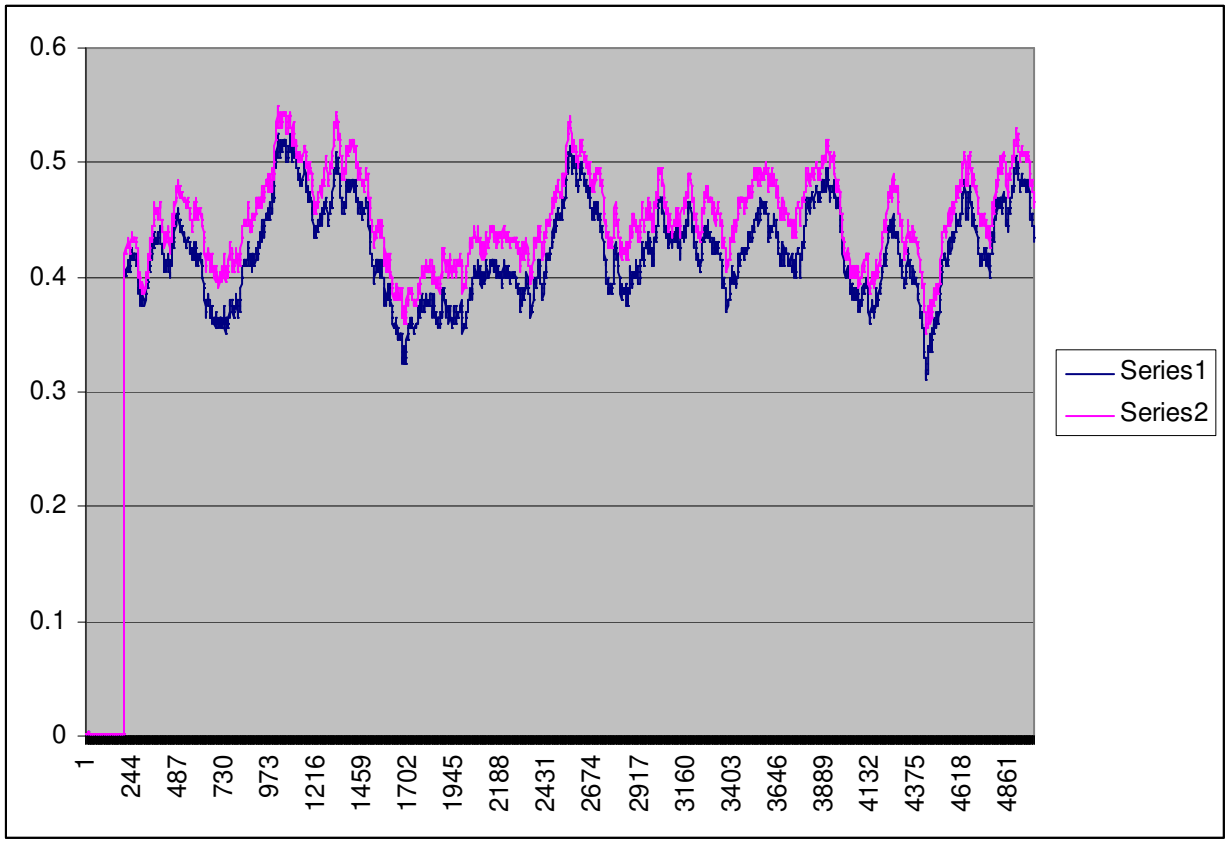

Figure 10: Obstacle avoidance and goal-reaching are much more closely related if the robot is given more time-steps

What does disturb me about figure 10 is that the frequency of avoiding obstacles drops to about 50%

Conclusions

One of the goals of robot learning is to develop algorithms from which a robot can adapt based on physical interactions with its environment. Such a task requires that the robot be able to learn in a reasonable amount of time, despite having to interact with the physical world in order to get feedback. The number of training runs required for convergence of the particular learning setup I used far exceeds what would be acceptable on a physical robot.

It would be interesting to study what factors could bring this "number of episodes required for convergence" factor down. One possibility might be to split up the robots tasks into separate agents, each of which learns a very small task. Another possibility might be to supply the robot with a basic kinematic model of itself, and let the learning simply supply some of the parameters. This has the drawback of being unable to account for changes in parameters that weren't anticipated. Yet another possibility is to turn a very small number of training episodes into a large amount of training data, simply by sub-dividing the time-scale over which the robot repeats the sense-act-train cycle. Repeatedly training on existing data to achieve faster convergence is also a possibility.

It could also be that I chose a bad neural network architecture, or a bad training setup. Perhaps randomly resetting the robot and obstacle positions at the end of each episode is a bad idea. After all, a real robot would be unlikely to encounter such a scenario in a lab setting. It is more likely that a learning robot that collided with an obstacle would stay collided with the obstacle until it chose an action that moved it away from the obstacle.

Glossary

Evidence Grid: A (spatial) probability map of obstacle occupancy.

Lamarckian Evolution: A (debunked) theory of evolution that held that traits learned during an organism's lifetime could be passed down onto its offspring. While agricultural policies based off of this theory have proven to be disastrous, it turns out that introducing forms of Lamarckian evolution into genetic algorithms can provide useful results.

Laser Rangefinder: A device, frequently using time of flight, or phase shift in a modulated laser beam, to determine the distance along a ray to the nearest obstacle (or at least the nearest non-transparent obstacle).

Reactive Obstacle Avoidance: In general, these are obstacle avoidance techniques that involve looking at the state of a robot and its sensors at the current time, or in some window around the current time. Most machine-learning based obstacle avoidance methods fall into this category, as they learn policies for obstacle avoidance, rather then heuristics for path planning.

Bibliography

Borenstein, J. and Koren, Y., 1989,
"Real-time Obstacle Avoidance for Fast Mobile Robots."
IEEE Transactions on Systems, Man, and Cybernetics, Vol. 19, No. 5, Sept./Oct., pp. 1179-1187.
(see also http://www-personal.engin.umich.edu/~johannb/paper10.htm)

Gaudiano, Paolo and Chang, Carolina. "p. 13 Adaptive Obstacle Avoidance with a Neural Network for Operant Conditioning: Experiments with Real Robots" in *1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)* June 10 - 11, 1997, Monterey, CA

Grefenstette, J. J. (1994). Evolutionary algorithms in robotics. In *Robotics and Manufacturing: Recent Trends in Research, Education and Applications, v5.* Proc. Fifth Intl. Symposium on Robotics and Manufacturing, ISRAM 94, M. Jamshedi and C. Nguyen (Eds.), 65-72, ASME Press: New York

Jones, Flynn, Seiger. 1999 "Mobile Robots: Inspiration to Implementation (second edition)" page 288

M. C. Su, D. Y. Huang, C. H. Chou, and C. C. Hsieh, March 21-23, 2004, ¡§A Reinforcement-Learning Approach to Robot Navigation,¡¨ 2004 IEEE International Conference on Networking, Sensing, and Control, pp. 665-669, Taiwan.