Repeatability of Real World Training Experiments: a Case Study

Dean Hougen, Paul Rybski, and Maria Gini
Department of Computer Science and Engineering,
University of Minnesota

Abstract

We present a case study of reinforcement learning on a real robot, and we show how the need to perform a large number of experimental learning runs on a real robot and in a systematic way has required to redesign some of the robot hardware. We describe in detail the design of the robot and present results of the learning algorithm.

Introduction

The project we describe in this case study uses a neural network to learn to solve a control problem. The specific control problem is learning how to back a car and trailer rig to a target location by steering the front wheels of the car. The inputs into the robot consist of the hitch angle as well as the angle of the rear trailer to the target. The output is what direction to turn the steering mechanism. This system uses laterally connected artificial neural networks to improve the efficiency of a basic reinforcement learning mechanism. This system is capable of rapid learning of output responses in temporal domains through the use of eligibility traces and data sharing within topologically defined neighborhoods.

The method has been tested extensively in simulation and on a real autonomous mini-robot that we have constructed for this task. Doing multiple runs in simulation was easy, and we collected large sets of data using different values for the initial distance from the target, angle to the target, and hitch angle. For each trial new initial conditions were chosen randomly at the start of the trial with a uniform distribution over the entire range of values. We experimented with different ranges and with slightly different learning modalities, and obtained a very high performance after a very small number of trails (40 to 50 in most cases). Results have been reported in (Hougen *et al.* 1996).

Once we started using our mini-robot extensively for multiple experiments, a new unexpected problem appeared. Because the robot was used to collect data on a series of runs, each of 100 trials, it became extremely tedious for a person to have to pick the robot up each time and place it in the new starting position. To solve the problem, we decided to have the robot moving forward autonomously by a random distance to a new starting position after each trial. We built a ring for the robot to stay in, so the robot would never get farther away from the target than we desired.

Unfortunately, as we started collecting data a new problem appeared. Our learning algorithm was not learning as well as in simulation. To understand what was happening, we collected the initial values for the hitch angle and angle to the target for each trial. We discovered that despite using a random movement to position the robot to start a new trial, we were not generating a uniform distribution of initial starting values. This was sufficient to make the learning algorithm to learn less and not to reach the same performance as if we had given it a sequence of uniformly random situations. To solve this problem, we modified our robot and we built a robot that could pick itself up and place itself in the required starting position (fed to it from the workstation used to record the results.) The mechanical design of the lifting mechanism is quite interesting, considering the limited size of the robot.

In solving this problem, we have made a contribution to the understanding of what it takes to do learning on a real robot.

The Learning Method

Control-learning for autonomous robotic systems is challenging because

1. the learning system must be robust enough to overcome the problems of noisy input data and uncertain interactions between motor commands and effects in the world,
2. be compact enough to fit into available on-board memory, and
3. be able to give responses in real time.

Connectionist control-learning systems have recently received much attention, but most of the work has concentrated on simulated systems. Approaches such as the Cerebellar Model Articulated Controller (CMAC) (Shelton & Peterson 1992), adaptive fuzzy systems (Kong & Kosko 1992), backpropagation through time (Nguyen & Widrow 1990), and "fuzzy BOXES" (Woodcock, Hallam, & D.Picton 1991) have all been applied to versions of the trailer-backing problem.

Our method combines the self-organizing capabilities of Kohonen Maps (Kohonen 1989) with the temporal sensitivity of eligibility traces. More details are given in (Hougen *et al.* 1996) and (Hougen, Gini, & Slagle 1997). Learning requires constructing a mapping from the input space to output space. The input space is two-dimensional. The inputs are (1) the angle from the spine of the trailer to the goal and (2) the angle of the hitch.

The output space is one-dimensional but thresholded at zero, giving the system bang-bang steering (i.e. the steering wheels may only be moved to 30° left or right). If the rear of the trailer reaches the goal, success is signaled. If the angle to the target exceeds 90° or the angle of the hitch exceeds 90°, failure is signaled. The system is clocked to operate in discrete time units. No other sensory data or feedback is available.

The learning system we use consists of a grid of neural elements. These elements each have a sensitivity region defined for them. When an input vector falls closest to the center of an element's sensitivity region, that element is selected for response and is said to fire. In effect, the sensitivity regions partition the input space into a number of discrete regions.

The centers of the sensitivity regions are initially random within the range of expect input values. As new input values are given to the system, the element's change their sensitivity regions in a self-organizing manner, based on Kohonen Maps (Kohonen 1989). Details of this self-organizing process are given in (Hougen, Gini, & Slagle 1997).

The system learns good output responses for the neural elements through reinforcement. Initially, all elements are given random output values. The rig is started from some pose and backs until failure or success occurs. This is known as a trial. At the end of a trial, all neural elements that fired during that trial have their values adjusted. On failure they are punished, which pushes their values towards the negative of what they were during the trial. On success they are rewarded, which pulls their values in the direction of their sign.

The degree to which a neural element has its output values adjusted on trial completion is determined by its eligibility for adaptation. When a neural element fires it becomes eligible for change. This eligibility decreases over time but produces a trace that allows for adaptation. The more frequently an element fires during a trial, and the more recently it fires before the trials ends, the greater its adaptation. The direction of change is determined by the failure or success signal; the magnitude of change is determined by the eligibility.

Because adjacent neural elements in the grid respond to adjacent partition regions in the input space, there is a good chance that similar responses for these elements may be helpful. By having neural elements communicate their output values to one another, elements may learn their behavior in part from the punishments and rewards received by their neighbors. This interneural cooperation allows the system to learn roughly ten times as fast as without this information sharing.

Experimental Results in Simulation

The system was tested extensively in simulation and, to a much smaller extent, on the real robot.

For the simulation testing, the car-trailer rig was started with the front of the cab six feet from the target, with an angle to the target between $-60°$ and $+60°$, and with a hitch angle between $-15°$ and $+15°$. New initial conditions were chosen randomly at the start of each trial with a uniform distribution over the entire range.

Figure 1 shows the distribution over 10,000 trials of the robot. The x-axis represents the allowed light-tracking angles of $\pm 60°$ and the y-axis represents the allowed hitch angles of $\pm 15°$.

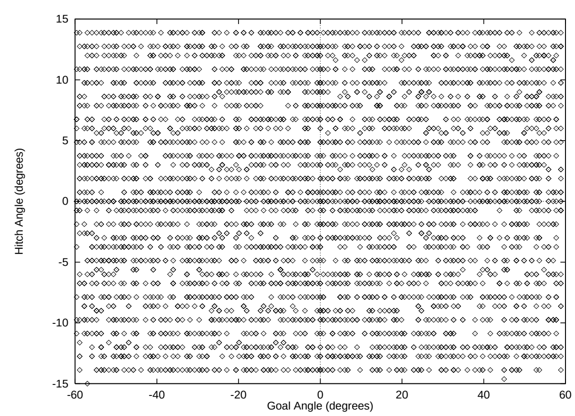

Figure 1: Uniform distribution of initial starting values

Using sixty-four neural elements in both the input and output networks, the system is able to achieve remarkable success. Average results for 100 runs on this task, each starting from random performance and progressing to competence, are shown in Figure 2.

An average success rate of greater than 85% is achieved by the system after only 100 trials and an average success rate of nearly 95% is achieved by the system after roughly 300 trials.

As can be seen from the graph, the system quickly improved its performance, reaching its maximum performance within 300 trials. (While a realistic comparison of systems cannot be made it might be of interest to know that other authors have used as many as 20,000 trials to train their systems to control trailer-backing). In fact, the success rate of completely random control

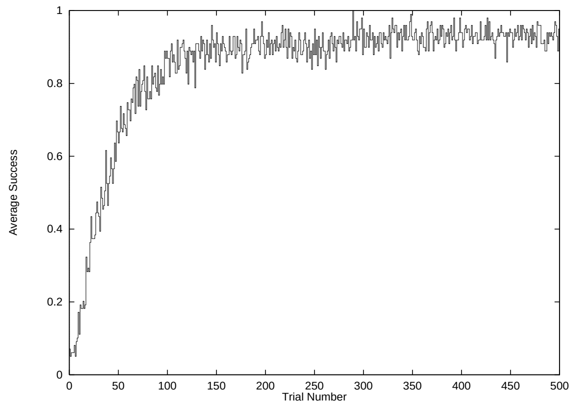

Figure 2: Average performance for 100 runs of 500 trials each using cooperative response learning with a learned eight by eight partitioning on the truck-backing task.

for the given initial conditions was determined experimentally to be zero, and the success rate using control by a random network without learning was determined to be approximately 6%, so it is obvious that significant learning has taken place within the first twenty trials. The steep initial slope of the graph indicates that significant learning occurred on failure as well as success.

Real World Experiments

The Trailer-Backing Mini-Robot (TBMin, seen in Figure 3) was built to extend the learning algorithm out of simulation and into the real world. TBMin consists of two parts, a cab unit and a trailer unit. The cab is an ackerman-steering 4-wheeled vehicle with rear-wheel differential drive. The front of the cab has a binary bumper switch that lets the robot know when it has contacted something from the front. The trailer is attached to the cab by a hitch constructed from a single-turn potentiometer. A servo with a set of photoresistors (CdS cells) is mounted on the back of the trailer so that the robot can keep track of the target (a light bulb) that it backs up towards. The trailer also has a binary bumper switch on its rear to tell the robot when it is in contact with the target.

TBMin uses LEGO for its construction base and the Motorola 68HC11-based Handyboard microcontroller (Martin 1998) as its on-board computer. LEGOs were chosen as the construction platform because of their ease of use and rapid prototyping facility. The Handyboard and Interactive-C (Wright, Sargent, & Witty 1996) were chosen as the hardware and software components of the development environment because of their ease of programming and hardware interfacing capabilities.

The potentiometer on the hitch measures the angle that the trailer makes with the cab. Valid angles that the trailer can make are within a 180° arc due to the physical constraints of the system. The light tracking servo on the trailer is also limited to a 180° arc.

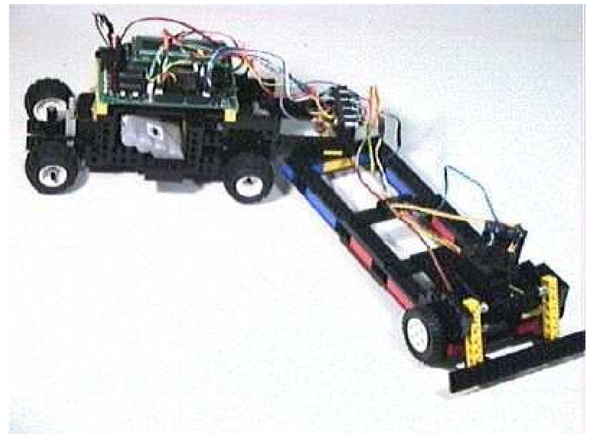

Figure 3: TBMin, the trailer backing mini-robot

Training the Robot

The environment in which the robot conducts its experiments consists of a circular arena 6' in radius with a light bulb placed at the center (Figure 4). The robot is connected to a workstation via its serial port for the purpose of data collection only. The robot can be untethered from the workstation for stand-alone operation. The entire neural network architecture (data structures and supporting source code) fits into the on-board RAM of the Handyboard.

At the beginning of a trial, the robot starts out touching the perimeter of the arena with its front bumper. The light-tracker and the trailer hitch are given some initial starting angles (within a valid range) and the robot backs up, attempting to touch the target bulb with its rear bumper, until it either generates a failure or success signal.

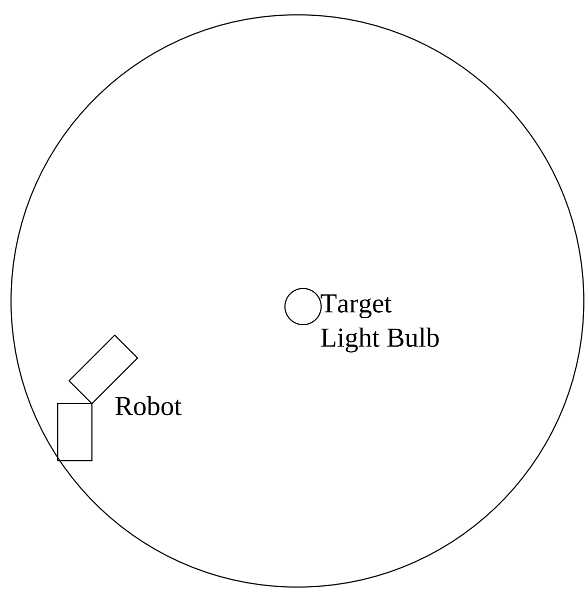

Figure 4: The arena used for the training experiments

One failure signal is produced when the trailer jackknifes. This occurs when the measured hitch angle comes close to its physical limits of ±90° with respect to the cab. The robot is physically unable to recover from this condition without pulling forward to straighten itself out, and so the system halts. The other failure signal is generated when the light-tracker reaches its physical limits of ±90° with respect to the trailer. This failure condition represents the case when the robot has missed the light entirely and has no hope of hitting it without pulling forward or turning around completely. Unfortunately, neither of these are guaranteed to succeed because the tracker has lost track of the light and may not necessarily re-acquire it. A success signal is generated when the trailer's bumper touches the light bulb, since the light bulb is the only obstacle in the environment. The robot cannot contact the arena with its rear bumper because the light tracker would have lost the bulb. When either a success or failure signal is produced, the robot stops its motion and then applies the learning algorithm to the neural network. After the network completes a training iteration, the robot is moved back to its starting position for another run.

Initial Positioning Problems

Re-initializing the simulated trailer-backer is just a matter of generating a new set of initial angles for the hitch and light tracker and re-drawing the robot on the screen. Achieving the same random variance on the physical robot involves having to manually reset its position and orientation between each trial. However, in a single experimental run, the robot may need to be re-initialized several hundred times. Having to move the robot each time becomes prohibitively expensive in terms of manpower.

To help automate this process, a solution was devised in which TBMin would attempt to re-position itself after each trial. This simply involved having the robot pull forward from where it had stopped in the trial until its front bumper made contact with the outer perimeter of the arena. In order to induce a random set of initial hitch and light tracking angles, the robot continually changed the position of its steering mechanism in a random fashion until the bumper made contact with the boundary.

Initially, this solution seemed to provide the necessary mechanism for generating new random starting positions. Only after completing several runs did a problem become apparent. Apparently, the neural network was unable to converge to a solution as well as it had in simulation.

Analyzing the initial starting positions that were being produced, it was discovered that the "pseudo-random wiggle" approach for automatically setting the initial position was producing a strange distribution of initial configurations. In simulation, we used a uniformly random distribution to generate the initial hitch and light-tracking angles. However, the distribution produced by the robotic hardware was not uniform.

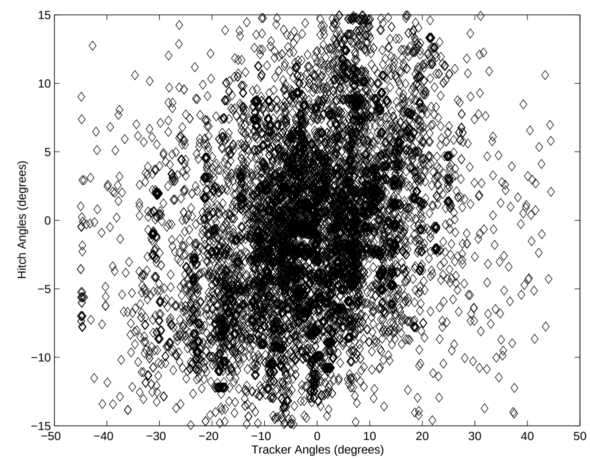

Figure 5: Non-uniform distribution of initial starting values

Back to the Simulation

Modifying the simulation to mimic the robot's automatic random positioning mechanism, a series of experiments was run. The results we obtained from these simulations also showed a non-uniform distribution.

Figure 5 shows the distribution over 10,000 trials of the robot. The x-axis represents the allowed light-tracking angles of ±60° and the y-axis represents the allowed hitch angles of ±15°. As can be seen, this distribution has a severe bias towards points near the origin.

After averaging the convergence results, it was found that the simulation now behaved just as badly as the real robot. The training cases being presented to the learning system were not allowing the neural network to generalize the problem properly. Thus, the neural network was converging to a sub-optimal solution. The results are shown in Figure 6.

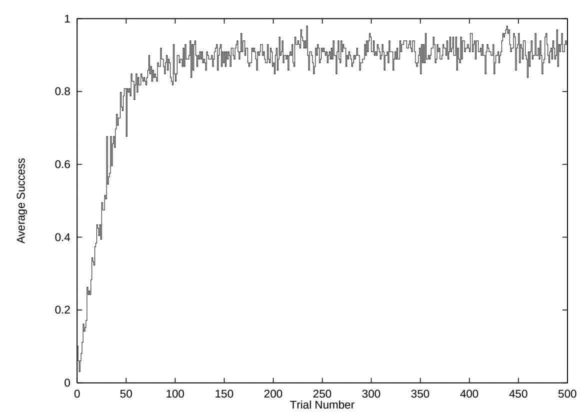

Figure 6: Average performance for 100 runs of 500 trials each using cooperative response learning with a learned eight by eight partitioning on the truck-backing task and a no-uniform distribution of the input data.

The graph of the two learning curves seems to show that both methods are learning roughly equally well. What it doesn't show, however, is that the pull-forward method is generating "easier" starting positions. By separating learning from testing, we can see that the uniform distribution allows the network to learn a better solution than the non-uniform distribution.

We ran each of the trained networks (one trained on a uniform distribution, one on the non-uniform distribution used in the pull-forward method) over a series of runs with uniform and skewed angles. We ran both neural networks on exactly the same input data and we compared the performance of the system when trained with a uniform distribution versus the non uniform distribution produced by the pull-forward method. The table shows the performance when tested on the same distribution used for learning and on a different distribution.

Robot placement	Uniform distribution	Non-uniform distribution
Uniform positioning	0.936500	0.981900
Non-uniform positioning	0.792900	0.951900

As we can see, the uniform positioning does much better than the non uniform distribution used with the pull-forward method when given uniform data, and it even does slightly better when given non-uniform data. This shows that the network trained using the pull-forward method was not able to generalize as well.

Figure 7: The Self-Positioning Trailer-Backing Mini-robot

Robust Experimental Procedure

In order to correct this problem, a method had to be devised to control the distribution of initial starting positions on the real robot. The current method of automatically setting the position of the robot was not controllable enough to be useful. What was required was a new mechanical system that would automatically reposition the robot accurately and repeatably. The solution to this problem was the design and construction of a new trailer-backer robot, shown in Figure 7. This robot, called the Self-Positioning Trailer-Backing Mini-robot (SPTBMin), has the same physical drive characteristics of the original TBMin (size, length and turn radius), but is fitted with a novel automatic re-positioning mechanism.

Hardware

Like TBMin, SPTBMin uses a Handyboard to run the neural network software and communicate with the controlling workstation via an RS232 serial port. A Miniboard (Martin 1995) is connected to the Handyboard via its SPI (fast synchronous serial) port and serves as a slave device controller. The Miniboard manages all of the lifting mechanisms and drive motors on the robot. An additional light-tracker servo is placed on the front of the cab to assist with the self-positioning process. Top-view diagrams of the layouts of the segments are seen in Figures 10 and 11.

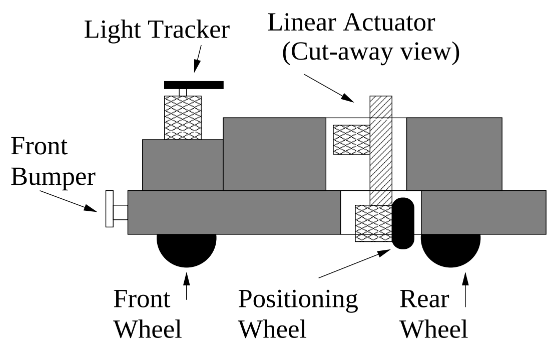

Figure 8: Side view of the cab segment

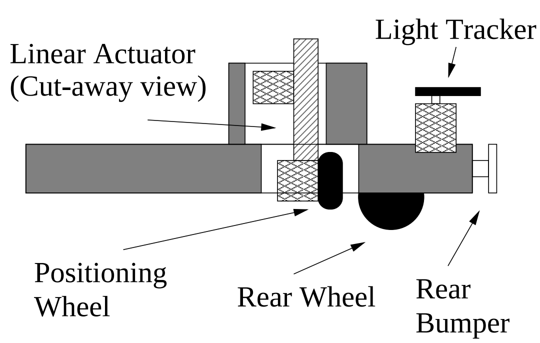

Figure 9: Side view of the trailer segment

Each segment of the robot is outfitted with a lifting mechanism consisting of a linear actuator which lifts that segment's rear wheels off of the ground, and a motor which rotates the segment laterally (Figures 8 and 9.

Once raised, each segment is free to pivot about a point at its front. In the case of the trailer, this point

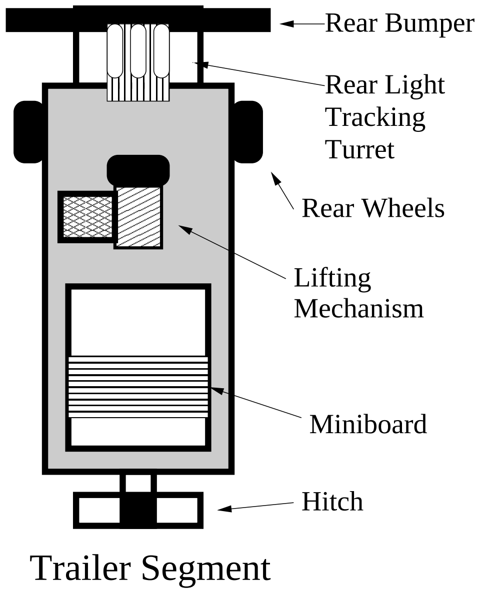

Figure 10: Layout of the trailer segment

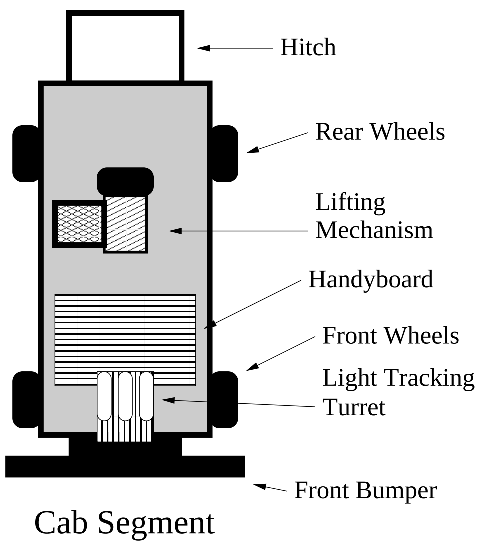

Figure 11: Layout of the cab segment

is the hitch (Figure 14) and in the case of the cab, this point is between the front wheels (Figure 15). Figures 12 and 13 show the actual robot lifting its trailer and cab segments.

Auto-Positioning

After each trial attempt at backing up to the light, the robot pulls forward until its front bumper collides with the circular rim of the arena, just as TBMin did. When the robot reaches the rim of the arena, the connected workstation generates a set of initial hitch and tracker angles from the same uniformly distributed random number generator that was originally used for the

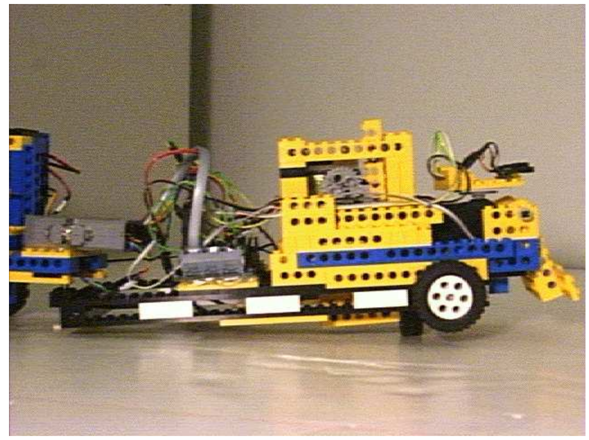

Figure 12: The self-positioning mechanism for the trailer

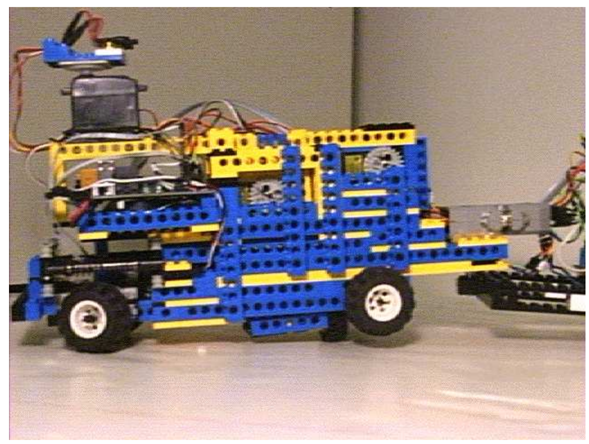

Figure 13: The self-positioning mechanism for the cab

simulator and downloads them into the robot.

Once the angles have been received, the robot must determine how to position itself such that the hitch and light tracker angles match those sent by the workstation. Due to the physical construction of the robot, moving the cab segment causes the trailer segment to move with it, but moving the trailer segment does not affect the cab. Therefore, when the robot re-positions itself for the next trial, it must first place the cab segment into its final position before placing the trailer segment. However, this procedure is complicated by the fact that the angle that the cab makes with a tangent line to the arena is not known and must be derived from the other two given angles.

Modeling the Robot

The physical configuration of the system is modeled in order to solve for the appropriate angles. Figure 16 shows a diagram of the that represent the configuration of the robot with respect to the light:

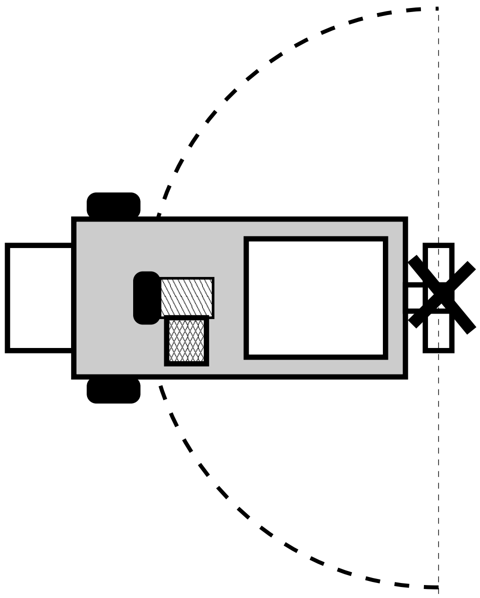

Figure 14: Pivot point of the trailer segment

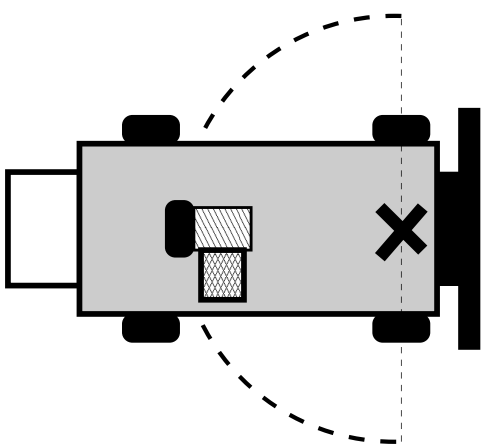

Figure 15: Pivot point of the cab segment

- θ_2 is the angle of the light tracker with respect to the light.
- θ_1 is the angle of the trailer with respect to the cab.
- θ_0 is the angle of the cab with respect to tangent line to the arena. This is the value that is needed to place the cab into its correct position.
- Dc is the distance of the front of the cab to the light.
- Dt is the distance from the back of the trailer to the light.
- Lc is the length of the cab segment.
- Lt is the length of the trailer segment.
- L is the distance between the front of the cab and the rear of the trailer.

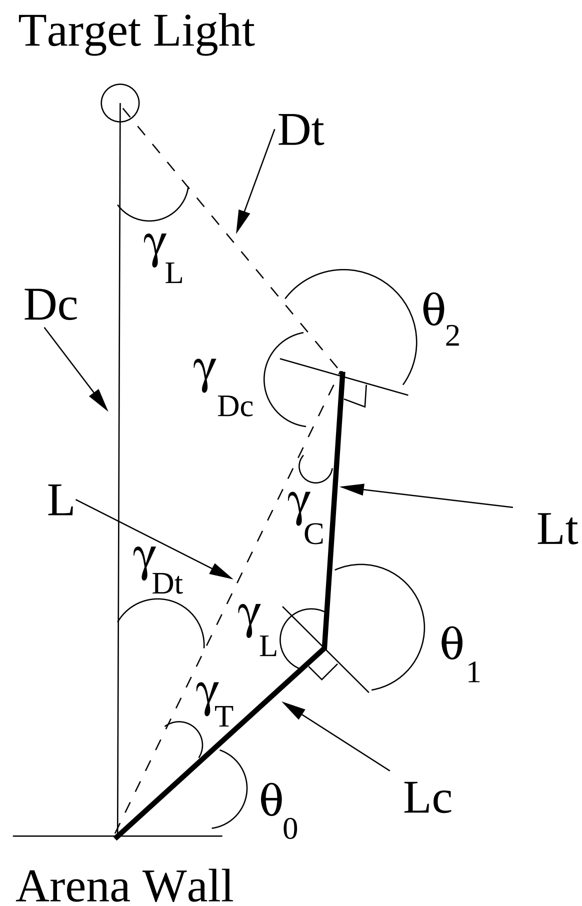

Figure 16: Determining angles for self-positioning

- γ_L, γ_{Dt} and γ_{Dc} are the angles formed by the triangle $\triangle$DcDtL.
- γ_T, γ_C and γ_L are the angles formed by the triangle $\triangle$LcLtL.

θ_2 and θ_1 are measured quantities which are determined when the robot stops at the rim of the arena. Dc is simply the radius of the arena (6' for most of the experiments). Lc and Lt are known before hand and do not change. From these known values, the remaining values can be solved for through repeated use of geometric formula like the law of cosines. Finally, the value of θ_0 can be generated.

This value represents the angle that the cab must move to. Using its own light tracker, it re-positions itself appropriately. Finally, the trailer segment can use its own self-positioning mechanism to line up the hitch and rear light-tracker angles to the appropriate values of θ_1 and θ_2.

Conclusion

We have presented a case study of learning with a real robot and shown how doing experiments with the real robot has exposed a problem in the way the initial positioning of the robot was done and how that positioning was affecting the ability of the learning algorithm to learn. The discovery of the problem has led us to redesign the experimental setup.

References

Hougen, D. F.; Fischer, J.; Gini, M.; and Slagle, J. 1996. Fast connectionist learning for trailer backing using a real robot. In *IEEE Int'l Conf. on Robotics and Automation*, 1917–1922.

Hougen, D. F.; Gini, M.; and Slagle, J. 1997. Partitioning input space for reinforcement learning for control. In *Proc. of the IEEE Int'l Conf. on Neural Networks*, 755–760.

Kohonen, T. K. 1989. *Self-organizing and associative memory.* Berlin: Springer-Verlag, 3rd edition.

Kong, S.-G., and Kosko, B. 1992. Adaptive fuzzy systems for backing up a truck-and-trailer. *IEEE Trans. on Neural Networks* 3(2):211–223.

Martin, F. G. 1995. *The Mini Board Technical Reference.* MIT Media Laboratory, Cambridge, MA.

Martin, F. G. 1998. *The Handy Board Technical Reference.* MIT Media Laboratory, Cambridge, MA.

Nguyen, D., and Widrow, B. 1990. Neural networks for self-learning control systems. *IEEE Control Systems Magazine* 10(3):18–23.

Shelton, R. O., and Peterson, J. K. 1992. Controlling a truck with an adaptive critic CMAC design. *Simulation* 58(5):319–326.

Woodcock, N.; Hallam, N. J.; and D.Picton, P. 1991. Fuzzy BOXES as an alternative to neural networks for difficult control problems. *Artificial Inteligence in Engineering* 903–919.

Wright, A.; Sargent, R.; and Witty, C. 1996. *Interactive C User's Guide.* Newton Research Labs, Cambridge, MA.