Clicker Training: Reinforcement learning for a mobile CotsBot

Nathaniel Ebel, Travis DeVault, Juan Marulanda, Jayandra Pokharel, Joshua Rubini,
Robert B. Heckendorn, Terence Soule

Department of Computer Science, University of Idaho, Moscow, ID 83844 USA

We created two methods for on-line learning through clicker training in autonomous robots built using Commodity-Off-The-Shelf design principles (COTSBots). The first strategy uses a look-up table to map current input information to a set of behavior probabilities while the second method uses a learned grammar to represent the controller as a state machine. Both controllers can be trained to solve problems the same way a dog may be trained. A robot's trainer gives positive or negative reinforcements during a training phase. This feedback modifies available behavior probabilities. Both strategies were tested using a benchmark problem of going to a target object and returning to the starting position. We compare the impact of multiple rounds of training on performance starting from a single starting position and multiple starting positions. Testing showed clicker training was an effective control strategy using both the table and the grammar. Results showed that the number of rounds of training significantly impacted the performance of the grammar method but not the table.

***Index Terms*—Ruled-based grammar, on-line learning, object detection, clicker training, COTSBot**

I. Introduction

Intelligent, autonomous, robotic agents are desirable because of their potential application to a wide variety of fields, such as natural disaster response and exploration of extraterrestrial landscapes [1], [2]. For complex, and dynamic applications such as these, it is difficult to design effective controllers before deployment, and it is therefore desirable for a controller to learn while it is deployed.

Robotic learning can take several forms including: supervised, unsupervised, and reinforcement.

Supervised learning strategies involve telling an agent what it should do given a set of inputs and then using that information to generalize behaviors for an inputs. An example of this type of learning is using back propagation to train a neural network.

Unsupervised learning attempts to cluster unlabeled data without any feedback. Common approaches include hidden Markov models and principle component analysis.

Reinforcement learning strategies include guiding an agent towards desired behaviors and then reinforcing those behaviors in some way to influence their chances of happening in the future.

We present two implementations of clicker training, which is a form of reinforcement learning, as a means of training a mobile, autonomous robot. The first method uses a look-up table, and the second use a learned grammar. Clicker training is the approach commonly used to train pets, but it can be easily expanded to the training of any agent; including robots. In this method, a robot learns a behavior through trial-and-error interactions with a trainer [3]. The results of these interactions are reinforced in an effort to increase or decrease their probability of occurring in the future based on whether a positive or negative reinforcement was received. Training in this fashion leads to robots that fall within the *human acting* category of the Artificial Intelligence description in [4].

Corresponding authors: (email: {nebel, deva0562, maru2593, pokh6200, rubi4714}@vandals.uidaho.edu, {heckendo, tsoule}@uidaho.edu).

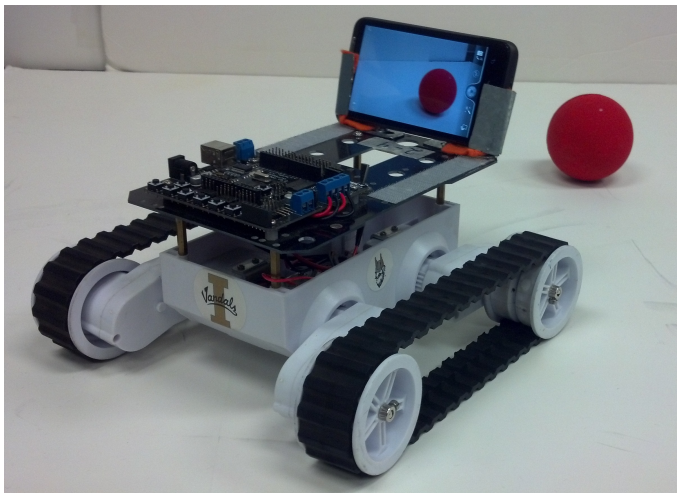

Fig. 1. Robot designed and built at the University of Idaho following Commodity Off The Shelf (COTS) design principles. The robot consists of three basic components: a smartphone, an Arduino microcontrollers, and a mobile robot platform. A full description of the robotic platform is given in Section III

The controllers were implemented on a mobile robot built using Commodity-Off-The-Shelf design principles (COTS-Bots). Reinforcements were given to the robot by clicking either a positive or negative reinforcement button on a connected smartphone. The robot starts by taking random actions and then pausing for one second to allow the trainer to provide reinforcement. If reinforcement is given, the controller increases or decreases the probability of the previous action occurring in the future based on whether positive or negative reinforcement was received. After training, the robots could operate autonomously , where no further reinforcements are given. Testing showed that clicker training, using either control method, was an effective way of training a robot to complete

a task, and training multiple times has a significant impact on the performance of the grammar method.

This research expands on previous applications of reinforcement learning to robotics in an effort to show that a simple training system can be developed that allows a robotics layman to train robots for a particular task within the trainer's own field of expertise. Furthermore, we show that the internal representation of the training system does not matter by testing two different approaches for internal representation.

This article is divided as follows: Section II gives a brief overview of robotic learning and reinforcement training. Section III presents a full description of the experimental procedure including hardware specifications, the problem to be solved, the testing arena, and test configuration. Section IV presents a description of the table and grammar methods for implementing reinforcement learning. Section V presents the testing data and provides a detailed analysis of the results. Finally, section VI presents our conclusions followed by some proposals for future work.

II. Background

Because of the complexity of real world environments, it's crucial that autonomous robots possess the ability to learn new behaviors from information gained during operation. This allows them to perform better than robots whose controllers are developed ahead of time [5], [6]. This learning can take several forms and includes learning through evolution (Evolutionary Robotics) [7], [8], learning by imitation [9], [10], and reinforcement learning [11], [12].

Intelligent, autonomous, robotic agents are desirable because of their potential to operate in variety of dangerous environemnts, such as natural disaster areas and extraterrestrial landscapes [1], [2]. These environments can be widely varied, complex, and dynamic. Because of this, creating controllers ahead of time to operate in one of these environments is very difficult and moving a robot to a new environment often requires a new controller to be developed. This development requires both time and money which may not be readily available in situtaions such as disaster response. To avoid the difficulties involved with creating pre-defined controllers, robotic controllers that facilitate learning can be developed. The ability to learn from new environments and situations allows a robot to adapt to the widely varied environments and tasks in which they may be used. This adaptation allows the robot to outperform pre-defined controllers because it can modify its behaviors during operation to perform optimally in the current environment or task.

Reinforcement learning techniques are commonly used to train animals [13], [14], [15] and can be applied to the field of mobile robotics as a means of simplifying the robotic training process. A common approach to animal training is to have the animal achieve the desired behavior first and then associate that behavior with a command. Because a trainer cannot explain to the animal what to do, the animal needs to discover the behavior by itself [12]. If a desired behavior rarely, or never, occurs naturally the trainer must guide the animal into discovering the behavior. Once the animal performs a behavior, the trainer can associate that behavior with a command and give a reinforcement For example, when training a dog to roll over, the trainer may first reward a dog for sitting, then for laying down, and finally for rolling over. This method is known as shaping, and these same principles can be applied to robotic training.

In [12], shaping was used to implement clicker training on the Sony AIBO robot. In clicker training, a device capable of producing positive and/or negative reinforcement signals (the clicker) is used to reinforce actions taken by the trained entity. Behaviors were represented internally using hierarchical trees where their roots represent the general goals and they can tell only if the task was done successfully, and not how it was done.

In this paper, we present two methods for on-line learning using clicker training as a means to train autonomous robots. By combining a simple training method, and cheap, easy to build robots, we've developed a research tool which requires no expert knowledge of robotics to use.

III. Experiment Description

A. Test Problem

To demonstrate that a simple training method can be used to guide a robot towards the completion of a task, the test problem used was 'Search, Go to, and Return,' because it is a fundamental benchmark problem in robotics. The robot was tested on its ability to find a target object, navigate to that object, and then return to its starting location. Both the target object and starting location were represented as colored balls of known size (see Section III-C for details). The effect of multiple rounds of training and multiple training positions on performance was tested by training the robot multiple times from the same starting position and from different starting positions. Testing from each position was done for both controller types so the performance of each controller method could be compared.

It's worth noting that the grammar method's ability to recognize states was tested using a pre-defined grammar to guide a robot out to a target object, have the robot drive around the object, and then return to the starting location. This is a task that is very dependent upon state information and could not be solved without some form of state differentiation of inputs. The pre-defined grammar was able to successfully complete the task, but it was determined that the table controller could not solve this task, so this test problem was not used.

B. Robot Platform

These experiments utilized a physical robot designed and built at the University of Idaho following Commodity Off The Shelf (COTS) design principles. The robot consists of three basic components:

1) An Android smart phone. For these experiments a HTC Desire HD running Android 2.3.3 was used. This phone has a 1GHz Qualcomm Snapdragon processor, with 768 MB of RAM, a Qualcomm Adreno 205 GPU, and a 480 by 800 pixel camera. The camera is the only input sensor used for this experiment.

2) An Arduino based microcontroller. The microcontroller possesses Bluetooth capabilities which handles communication between the smart phone and the robot body. For these experiments the DFRomeo microcontroller from DFRobot was used.[1] The microcontroller is only used to receive commands via Bluetooth from the smart phone and to activate the motors on the robotic platform, therefore, the computational characteristics of the microcontroller are not a concern.

3) A mobile robot platform. For these experiments the Rover 5 Tank chasis was used.[2] This is a small (22.5cm by 24.5cm), tracked, mobile robot platform. It can run continuously for several hours while carrying the microcontroller and smart phone.

The robot's decisions are made by an application running on the phone which receives input from the camera, performs vision processing, chooses an action using one of the two proposed control methods, and then sends the corresponding signal to the microcontroller. Reinforcement is sent from the trainer using a 'clicker' phone and is received by the smartphone on the robot. These reinforcements are then handled by the controlling application.

OpenCV 2.3.1 is used to capture and process sequential images from the phone's camera. The algorithm searches for the closest round object of a specified color, and can find objects of different colors simultaneously. If an object is found, it is assumed to be a colored ball. A determination is made on whether the detected object is near or far from the camera using a threshold value of 0.6 meters and knowledge of the size of the target and start balls. If multiple objects of the same specified color are detected, only the closest one is considered. Each frame is processed, and the information is stored in a four integer array. The first two values store input information corresponding to the start object, and the last two represent inputs corresponding to the target object. This input array is then handled uniquely by each control.

C. Testing Arena

The experiments were performed in a 4.7 by 2.8 meter room. The start and target locations were represented by 3" diameter red and green balls respectively. The balls were placed 1.1 meters away from the north and south walls, and were centered 1.4 meters away from each of the west and east walls (see Fig. 2). Each starting position was 30 cm away from the start ball so the robot would have room to move when placed in positions 2, 3, and 4 (see Figure 3). Figure 2 shows the robot in starting position 1, and Figure 3 shows the position of each starting location.

D. Experimental Design

The experimental method used to test the controllers consisted of two phases: training and testing. Both are described in this section.

[1]http://www.dfrobot.com/index.php?route=product/product&filter_name=Romeo&product_id=56#.UltL4BaHrzI

[2]http://www.dfrobot.com/index.php?route=product/product&path=37_111&product_id=386#.UltPFBaHrzI

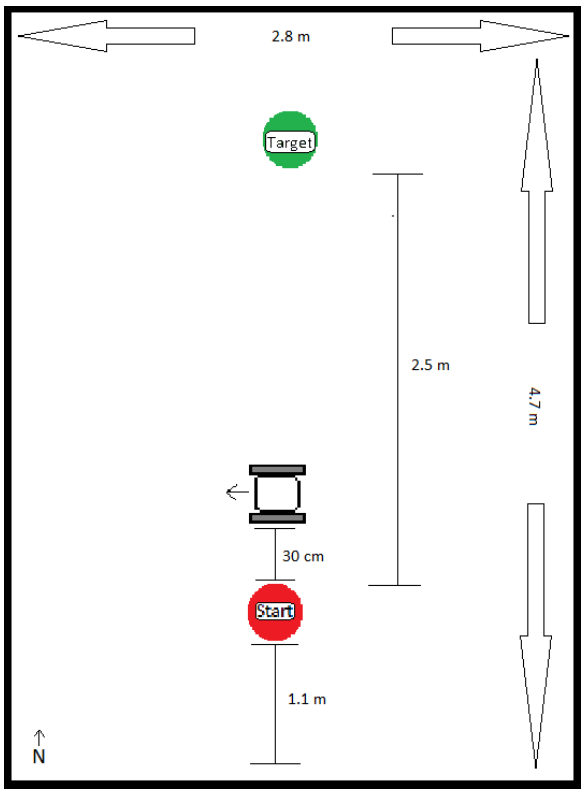

Fig. 2. Experiment arena with a robot positioned near the starting location (red ball) in position 1 facing the west wall which was the standard orientation for each starting location. The robot and balls are not drawn to scale.

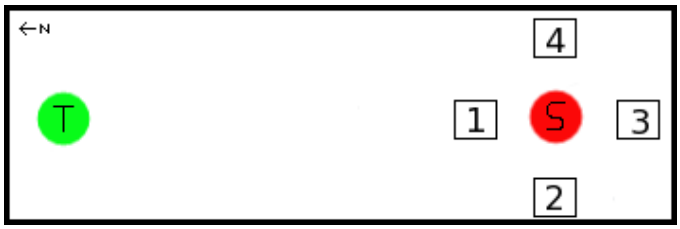

Fig. 3. Positions of experiment starting locations. The robot was trained starting from each of these positions. The robot was always placed at the starting position facing West.

1) Training Phase

During the training phase, the robot was trained by a human operator to solve the test problem. Two different sets of training (A and B) were performed, each consisting of four rounds of training. Set A was designed to test the effect of multiple rounds of training from the same starting location, while Set B was designed to test the effect of training from multiple starting locations. A round of training included the following steps:

1) Place the robot at a starting position facing West
2) If it's not the first round of training for a training set, load the previously generated rules as the starting rules for the current round of training
3) Start the robot in 'Training Mode'
4) The robot receives input, takes an action, and pauses for one second to allow time for accurate reinforcement
5) The robot's actions are reinforced to guide it through

TABLE I
ROUNDS OF TRAINING AND WHICH POSITIONS WERE TRAINED. FOR EXAMPLE, TRAINING 4 WAS TRAINED FOUR TIMES FROM POSITION 1. TRAINING 7 WAS TRAINED ONCE FROM POSITIONS 1, 2, 3, AND 4. THE GENERATED RULES FROM TRAINING ARE CUMULATIVE MEANING THAT THE OUTPUT RULE SET FROM ONE ROUND OF TRAINING IS USED AS THE STARTING RULE SET FOR THE NEXT ROUND OF TRAINING (EX. TRAINING 5 USED THE OUTPUT RULES FROM TRAINING 1 WHICH STARTED WITH AN EMPTY RULE SET).

	Round	Locations Trained
Set A	Training1	1
	Training2	1 1
	Training3	1 1 1
	Training4	1 1 1 1
Set B	Training1	1
	Training5	1 2
	Training6	1 2 3
	Training7	1 2 3 4

the test task

6) Stop the robot after it has completed the task, or five minutes of training have elapsed
7) Save the generated rules to an output file

Table I describes the training data sets including which positions were trained for each round of training.

The reinforcement process is critical for training. If an action is judged as good or bad by the trainer, a reinforcement can be given using the 'clicker' smartphone. When a reinforcement signal is received by the robot, the previous action is reinforced. To ensure the user has a chance to reinforce the correct action, the robot pauses for one second after it makes an action. If the robot operates without any delay, the training process is much less accurate. The decision to pause increases the time to complete an action, but the overall training time is decreased through increased reinforcement accuracy.

For this experiment, three different people trained the robot. Each trainer performed both sets of training using both of the control methods. This was done to avoid bias caused by variations in who was training the robot, and what method they were using.

2) Testing Phase

To test the rules generated during training, saved output files containing the rules generated from each round of training were used. By loading a rule file before autonomous operation started, the robot would operate using that set of rule. Each of the 7 training files were loaded, and run 3 times from each of the 4 starting locations for a total of 12 tests per training. A test consisted of placing the bot at the starting location facing west, and clicking the start button on the clicker phone allowing the robot to run autonomously.

A test was a success if the robot was able to move to within the near threshold (0.6 meters) of the target location and return to within the near threshold of the start location. If during this process the robot ran into a surrounding wall, ran over either the start or target ball, or exceeded 5 minutes of run time the trial was considered a failure. The robot started each test facing West so each trial had the same initial start state. Recorded data included the time from the start to target, time from target to start, the number of times the bot bumped one of the balls, the distance a ball moved if it was bumped, and the overall success or failure of the run. The complete data set consisted of 3 trainers, each using both of the 2 controller methods, training each method 7 times, and testing each round of training 12 times for a total of 504 data points.

- 3 trainers
- 2 controller methods
- 7 rounds of training
- 12 trials per round of training
- 504 total data points

IV. CONTROL METHODS

To show that clicker training is effective regardless of internal representation, we tested two different methods of representing the internal logic of the robot: lookup tables and grammars. The lookup table has the advantage of simplicity in concept and implementation whereas the grammar approach has the advantage of being able to differentiate similar inputs based upon state information, and therefore can chain together complex actions with limited inputs. While both methods allow robotic training through action reinforcements, the methods do operate much differently from one another.

A. Table

The table implementation maintains a table mapping different input states with their corresponding actions to be taken. The training process is started with an empty table, and through the training process it is filled with mappings from input states to actions which are then chosen probabilistically. An entry in the table is represented by a four integer input array and a three integer output array organized in the form:

- Input
  1) START_DIRECTION
  2) START_DISTANCE
  3) TARGET_DIRECTION
  4) TARGET_DISTANCE
- Output
  1) OUTPUT_LEFT
  2) OUTPUT_CENTER
  3) OUTPUT_RIGHT

Direction and distance are represented as integers with the following integer to input mappings:

1) DIRECTION
   - 0 - UNSEEN
   - 1 - LEFT
   - 2 - CENTER
   - 3 - RIGHT
2) DISTANCE
   - 0 - NEAR
   - 1 - FAR

Direction and distance input is received for both the target and start locations. The direction refers to what third of the

screen the object is. If the object in question is not visible to the camera it is considered UNSEEN and an input value of 0 is used. Distance is measured as either NEAR or FAR using a value threshold of 0.6 meters to seperate the two; when the robot sees a vall and determines it's closer than 0.6 meters to that object the input received is NEAR, otherwise it's FAR.

The three output values represent weighted probabilities of an action (left, forward, right) being taken by the robot. These three action values are initialized with a default value of 5 so each has an equal initial chance of occuring. The values are incremented or decremented based upon positive or negative reinforcements during the training process. When positive reinforcement is received, the previous action's value is incremented by one and all other action values are decremented by one with a min value of 0 and a max value of 10. If a negative reinforcement occurs, the last action's value is decremented by 2 and the other values are not changed. To choose which action to take a proportional selection is used where each action's chance of being chosen is its action value divided by the sum of the three action values. If the three values are 2, 5, and 2, forward would have a 55% (5/9) chance of being the chosen action. The following are example rules in the table:

1) $< 0, 0, 0, 0 >< 8, 2, 2 >$
2) $< 2, 1, 0, 0 >< 0, 10, 0 >$

The first entry tells the robot to turn left when neither the target or start objects are seen, but the robot thinks it is close to both of them. The second rule says when the start object is in the center of the screen and far away, and the target object cannot be seen but is close, to drive forward towards the start object.

When running autonomously, the robot matches its current state with the corresponding state in the table and then determines the action to be made using the process described previously. The chosen action is then sent to, and carried out by, the robot.

B. Grammar

The grammar models the robotic controller as a state machine defined by a learned, regular grammar using a predefined set of input tokens. This provides the controller with the ability to perform sequential reasoning, and allows different behaviors to be chosen for identical inputs based upon state information. The grammar defines a state machine that can be modeled as a graph with the non-terminals (NT) mapped to the vertices, and the rules representing particular edges on the graph. Figure 4 shows an example of a state machine generated by the grammar that is able to solve the test problem.

Grammar rules take the form of:

- $\beta \leftarrow \omega, \eta : \rho$

where:

- β: Non-terminal to push onto the stack
- ω: Input string
- η: Current non-terminal
- ρ: Actions probabilities (Left, Forward, Right, Stop)

The input string and current non-terminal are used to match a rule. The matching rule's action probabilities are used to determine the action to take, and the specified non-terminal is pushed onto the stack to represent the current state. A generated rule looks like:

1) NT1 ←EMPTY NT0 : .3 .3 .3 .1

The grammar operates using a predefined set of input tokens (see Table II). The input array generated through image processing is further processed by the grammar to generate an input string using the tokens available to the grammar. This is different from the table which uses the input array directly. The grammar maps the array values to the proper tokens to build valid input strings. Input strings take one of two forms:

1) $<$"$EMPTY$"$>$
2) $< DIRECTION, DISTANCE, OBJECT >$

An EMPTY input means their are no recognizable objects in the screen, and if an object is detected, the input describes that object using the following mapping.

1) DIRECTION
   - LEFT
   - CENTER
   - RIGHT
2) DISTANCE

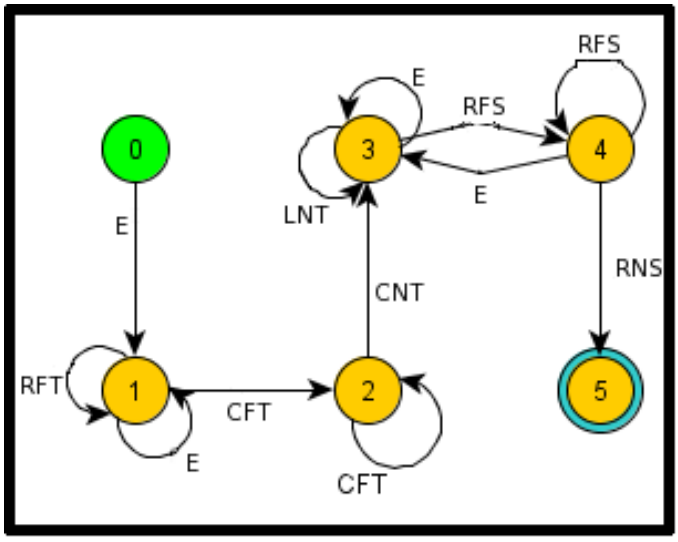

Fig. 4. State machine for example grammar. Vertices/states correspond to non-terminals and edges correspond to inputs. Edge labels use abbreviated forms of the vocabulary defined in Table II. For example, the input CFT refers to "CENTER FAR TARGET" and E is "EMPTY."

TABLE II
GRAMMAR VOCABULARY

Word	Description
LEFT	Object is on the left 3rd of the camera's view
CENTER	Object is on the center 3rd of the camera's view
RIGHT	Object is on the right 3rd of the camera's view
NEAR	Object is less than 0.6 meters away from the robot
FAR	Object is more than 0.6 meters away from the robot
START	The start object (red ball) is detected on screen
TARGET	The target object (green ball) is detected on screen
EMPTY	No object is detected on screen
NT#	Non-terminal # where # is some unique number from 0 to N (ex. NT0)

- NEAR
- FAR

3) OBJECT
   - START
   - TARGET

The direction indicates which third of the screen an object is located in. The distance indicates whether the object is greater than 0.6 meters away, and the OBJECT token indicates whether the detected object is the START or TARGET location. Example inputs include "CENTER FAR TARGET", and "RIGHT NEAR START". If both a TARGET and START object are in the field of view, the TARGET input is used. This does create a bias towards the TARGET object when both are seen. In future work a different way of handling input containing both objects could be used to test the effect of the bias towards the TARGET object.

Each rule in the grammar has four action probabilities representing each of the four valid actions for the controller to choose: left, forward, right, stop. Each action has its own probability of being chosen, but the four probabilities added together always equal 1.0. Initially the left, forward, and right probabilities are set to 0.3 and the stop probability is 0.1.

A simple grammar may only have a few rules with a small number of non-terminals. This simple grammar is sufficient to turn left until the TARGET is in the center of the screen and then drive towards it.

1) NT0 ←EMPTY NT0 : .5 .2 .2 .1
2) NT1 ←CENTER FAR TARGET NT0 : .1 .7 .1 .1
3) NT1 ←CENTER NEAR TARGET NT0 : .15 .1 .15 .6
4) NT1 ←CENTER NEAR TARGET NT1 : .1 .1 .1 .7
5) NT1 ←CENTER FAR TARGET NT1 : .1 .5 .1 .3
6) NT1 ←EMPTY NT1 : .75 .1 .1 .05

The behaviors generated by these rules can be summarized as follows:

1) When nothing is in the screen, turn left
2) If nothing has previously been seen, and the target is far away and in the center, change state to NT1 and drive forward
3) If nothing has previously been seen, and the ball is near and in the center, change state to NT1 and stop
4) If the target has been seen, and the target is near and in the center, stay in state NT1 and stop
5) If the target has been seen, and the target is far away and in the center, stay in state NT1 and drive forward
6) If the target has been seen, and nothing is on the screen, turn left

This summary assumes the action with the greatest probability is chosen each time, but because actions are chosen probabilistically this isn't always the case. For example, when nothing has been seen (rule 1) the robot only turns left 50% of the time.

Operation during training consists of four main tasks:

1) Input matching
2) Action selection
3) Non-terminal and rule generation
4) Reinforcement

Input matching Input strings are received and pushed onto the stack. The top of the stack is then matched against existing grammar rules. If the input string and the current non-terminal match an existing rule, they are popped from the stack, a new non-terminal is pushed onto the stack, and an action is selected using the action probabilities present in the matching rule. If no matching rule exists, a new rule and possibly a new non-terminal will be generated.

Action selection The actions performed by the robot are chosen probabilistically using a roulette wheel selection. Initially, because all action probabilities start with equal values the chosen behaviors are random. As reinforcement modifies action probabilities certain behaviors will being to occur more often. By using a probabilistic selection instead of just picking the action with the highest probability the robot maintains the ability to break out of an endless loop of repeated behaviors by picking an action that is not the most likely to occur. This ability is desired when an action has been positively reinforced when it shouldn't have been.

Non-terminal and Rule generation If no existing rule matches the current input and non-terminal a new rule must be generated. An important factor in generating a new rule is deciding what non-terminal to push onto the stack for the new rule. This effectively determines how a robot reasons about the order in which parts of the task are completed. When a new rule is generated it can use the current non-terminal, it may use an existing non-terminal different from the current one, or it may generate a new non-terminal. The current non-terminal is not used in the following situations:

- The current non-terminal is the starting non-terminal
- Large change between current and previous input. (ex. FAR to NEAR, or TARGET to START)
- A random number is generated greater than (.25 * (# of matching input tokens between current and previous input))

If the current non-terimanl is not selected, a list of rules with matching input but different non-terminals is generated. If the non-terminals in these rules have a larger ID (ex. $NT3 > NT1$) then they are added to a list of non-terminals that can be chosen for the new rule. If there are no existing non-terminals that fit this criteria a new non-terminal is generated with an ID one greater than the previously created non-terminal, and it is added to the list of possible non-terminals for the new rule. From the list of possible non-termainsl, one is randomly selected to be the non-terminal for the new rule. If the chosen non-terminal is a newly generated one, it is added to the list of non-terminals available for future use. This process allows the robot to expand its known state space through exploration of previously unkown states and environments.

Reinforcement

To train the robot to perform the test task, a reinforcement can be issued after an action is taken. If the previous action was a desired one, a positive reinforcement can be given which increases the probability of that action occuring in the future. Likewise, if the previous action is not desired, a negative input will decrease its chance of occuring. When a positive reinforcement is received for an action the other three probabilities are halved and the probability of the reinforced

action is increased by the sum of the new probabilities of the other actions (see Algorithm 1). A negative reinforcement on an actions causes that probability to be havled, and its remaining half is added evenly between the other three actions.

During testing operation reinforcements are not given, so there is no further reinforcement training. If unfamiliar inputs are received the bot again generates a new rule. If the inputs don't match any in the recent history, the controller attempts to match them to *any* rule in the rule-set in order to once again return to known state space. This approach may not be the best one, and an alternative that was suggested was to simply choose a random non-terminal to push onto the stack if the inputs do not match any rule in the rule set. Further work is needed to test different methods of rule generation during testing.

As the robot operates, the grammar may generate many new rules and non-terminals as it learns how to handle new inputs. This grammar describes the DFA shown in Figure 4.

- NT1 ← EMPTY NT0 : .1 .1 .7 .1
- NT1 ← EMPTY NT1 : .1 .1 .7 .1
- NT1 ← RIGHT FAR TARGET NT1 : .1 .1 .7 .1
- NT2 ← CENTER FAR TARGET NT1 : .3 .3 .3 .1
- NT2 ← CENTER FAR TARGET NT2 : .1 .7 .1 .1
- NT3 ← CENTER NEAR TARGET NT2 : .1 .1 .7 .1
- NT3 ← LEFT NEAR TARGET NT3 : .1 .1 .7 .1
- NT3 ← EMPTY NT3 : .1 .1 .7 .1
- NT4 ← RIGHT FAR START NT3 : .3 .3 .3 .1
- NT4 ← RIGHT FAR START NT4 : .1 .7 .1 .1
- NT3 ← EMPTY NT4 : .3 .3 .3 .1
- NT5 ← RIGHT NEAR START NT4 : .3 .3 .3 .1

V. Results

Tables III and IV, and Figures 5, 6, 9, and 10 show the successes of each round of training for both the table and the grammar respectively. The data shows that after one round of training for each trainer the table performed better than the grammar (see Figures 7 and 8). We used a χ^2 test to determine the dependency of success on training. The test used the number of successes after being trained one time vs. being trained four times. The results from Table IV showed that increased training iterations were significant to the success of the grammar($\chi^2 = 8.41$, $P \leq 0.01$). The results from Table III showed that increased training was not significant for the table($\chi^2 = 0.96$, $P \geq 0.35$).

The effect of multiple rounds of training on performance can be seen in Figures 7 and 8. They show that multiple rounds of training for the grammar have a significant impact on performance while multiple rounds of training for the table don't significantly increase performance above what is achieved by a single training. From this information we deduce that the data shows that the grammar is more dependent on training than the table.

Figures 5, 6, 9, and 10 show that most successful tests for either method came from positions 1 and 2. Position 3 was the most difficult testing position, and had the least number of successful trials.

We performed the same tests to determine if training from multiple positions leads to a greater probability of success.

Algorithm 1 Grammar Controller Description

```
P_r = Probability of a reinforced behavior
P_n = Probability of a non-reinforced behavior
Push NT0 onto the stack
if Top of stack matches rule then
  Pop matching tokens off stack
  Push NT from matching rule onto stack
  Choose action based on probabilities
  if Reinforcement was received then
    if Reinforcement was positive then
      val = 0
      for Each non-reinforced behavior do
        P_n = P_n/2
        val += P_n
      end for
      P_b = P_b + val
    else
      P_b = P_b/2
      for Each non-reinforced behavior do
        P_n = P_n + P_b/3
      end for
    end if
  end if
else
  if Top of stack is a NT then
    Get new visual inputs
  else
    useNewNT = useNewNonTerminal()
    if useNewNT = True then
      if Rule in history similar to current visual input then
        Generate rule that pushes NT of the similar rule
      else
        if In Training Mode then
          Generate new NT
          Generate rule pushing the new NT
        else if In Test Mode then
          if Existing rule similar to current visual input
          then
            Generate rule that pushes NT of the matching
            rule
          else
            Pick random NT
          end if
        end if
      end if
    else
      Generate rule to match inputs
      Push same NT onto stack
    end if
    Add new rule to rule set
  end if
end if
```

TABLE III
TABLE TRAINING RESULTS

		Position1	Position2	Position3	Position4	Total out of 36
Set A	Training1	9	9	0	3	21
	Training2	9	9	0	2	20
	Training3	8	9	0	5	22
	Training4	8	9	2	6	25
Set B	Training1	9	9	0	3	21
	Training5	9	9	0	2	20
	Training6	8	6	1	6	21
	Training7	9	7	0	9	25

TABLE IV
GRAMMAR TRAINING RESULTS

		Position1	Position2	Position3	Position4	Total out of 36
Set A	Training1	3	4	0	0	7
	Training2	8	6	1	2	17
	Training3	8	6	0	1	15
	Training4	9	8	1	2	20
Set B	Training1	3	4	0	0	7
	Training5	4	4	0	1	9
	Training6	6	6	2	1	15
	Training7	6	6	1	3	16

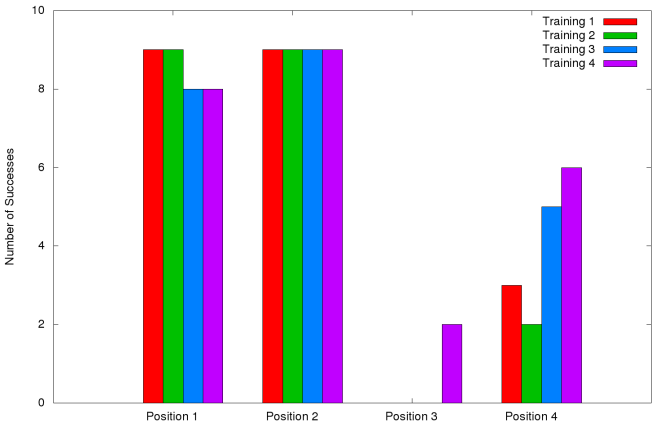

Fig. 5. Results of testing data set A from all positions using the table method. This graph shows the fewest number of successes came from starting location 3. Successes from locations 1 and 2 were consistent regardless of the number of rounds of training.

To do this we performed the χ^2 test again, but this time we compared the success of Training4 to the success of Training7. This compared the performance of training 4 times from a single position to training 1 time from 4 different positions. The test for the table method showed that change in position was not significant($\chi^2 = 0.06$, $P \geq 0.8$) (see Figure 11). The results for the grammar method showed that change in position was also not significant($\chi^2 = 0.88$, $P \geq 0.35$) (see Figure 12), although it still received more gain from repeated training than the table.

To determine which controller method performed best, another χ^2 test was used, this time comparing whether success was dependent on the method used. The results showed that the type of method used was not significant ($\chi^2 = 1.46$, $0.25 < P < 0.1$).

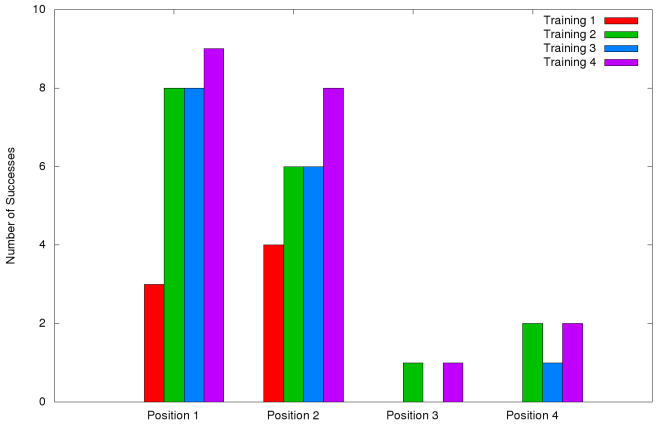

Fig. 6. Results of testing data set A from all positions using the grammar method. This graph shows few successes came from starting locations 3 and 4. Successes from all locations increased between the first and last rounds of training.

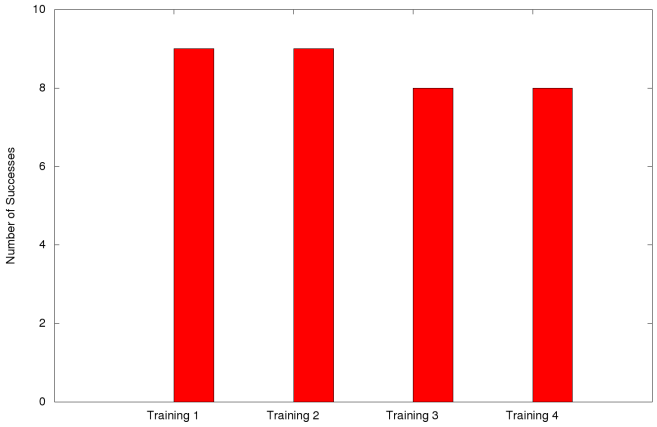

Fig. 7. Results of testing data set A from position 1 using the table method. This graph shows the number of rounds of training does not have a significant effect on the performance of the table controller.

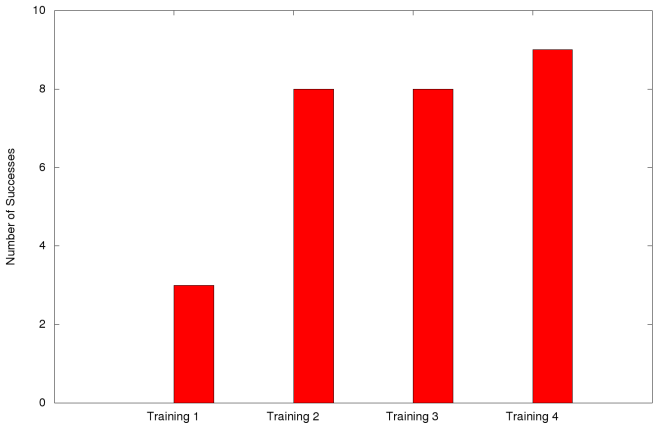

Fig. 8. Results of testing data set A from position 1 using the grammar method. This graph shows the number of rounds of training does have a significant effect on the performance of the grammar controller.

VI. CONCLUSION

Using a clicker training reinforcement learning strategy, as used on animals, it is possible to create a learning system for

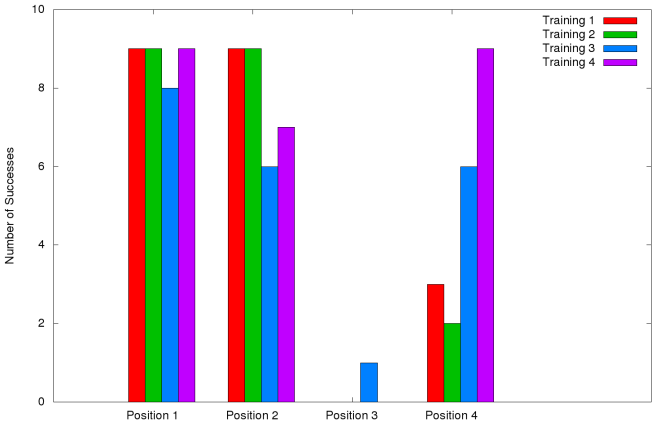

Fig. 9. Results of testing data set B from all positions using the table method. Training from multiple starting locations didn't have any significant effect on the performance of the table. The data shows position 3 was the most difficult starting position for the table as it only succeeded from that postion one time.

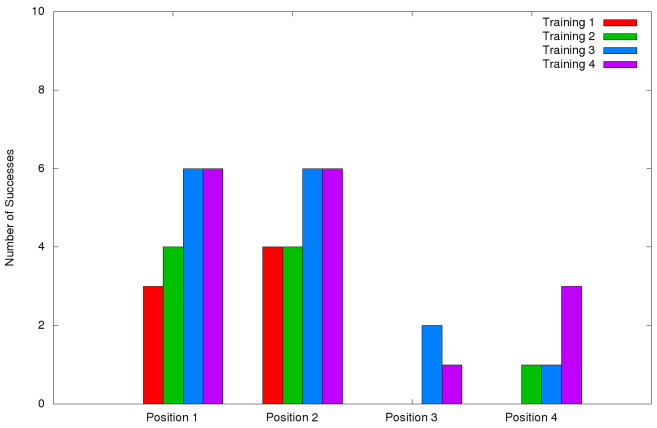

Fig. 10. Results of testing data set B from all positions using the grammar method. The data shows performance generally increases by training from multilpe starting locations. The grammar controller showed greater success from position 3 than did the table method. This could suggest the grammar could potentially be more effective in training the robot to handle more complex tasks.

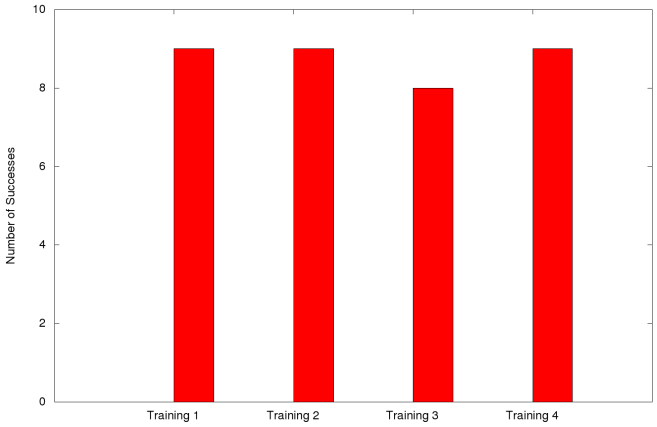

Fig. 11. Results of testing data set B from position 1 using the table method. This graph shows that training from multiple positions does not have a significant effect on the performance of the table controller.

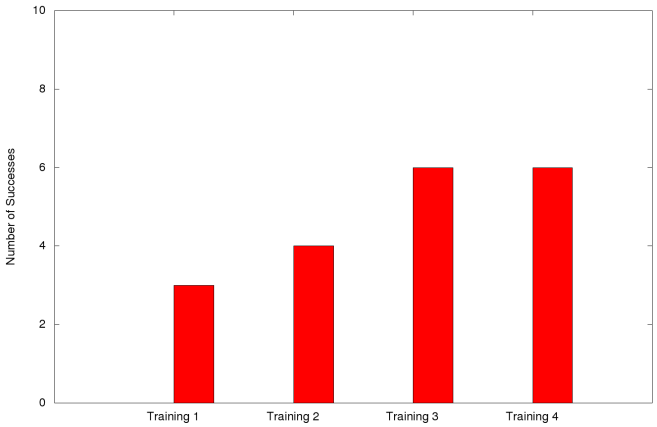

Fig. 12. Results of testing data set B from position 1 using the grammar method. This graph shows that training from multiple positions improved performance during this experiment, but the increases weren't significant.

mobile robotics which enables the training of a robot by a person with little knowledge of the robotic software or hardware. To show the effectiveness of clicker training, regardless of internal representation, we presented two different approaches to internal control of a robot, a classical table and a novel grammar.

Though testing the robot without any training was not done, we feel confident that neither the table nor grammar would have had any realistic chance of succeeding in the task without a single round of training. Though the χ^2 analysis showed only the grammar was dependent on training, both methods needed at least 1 training to be successful at all.

Starting from position 3 (see Figure 3), neither approach was able to consistently succeed in the task. Our data showed that the least amount of successful trials came from this position. We suspect this is due to several reasons. The table lacked state memory, and thus would have difficulty moving around the start object to get to the target object. The grammar has the ability to sequence tasks, so could potentially learn to move around an object in a particular state. We believe this would take many rounds of training and it was never tested.

The table benefited the most from the first round of training, and thus had small gains from continued training. The grammar controller showed more significant gains from continued training than that of the table method. In addition to performance gains from additional training, the grammar was more successful during testing from some of the more challenging starting locations suggesting it may be a better option for more complicated tasks; further experimentation will be required to show this. On the chosen task, training from different starting positions had little effect on the overall success of the robot, and may have even been detrimental due to the chance of negatively modifying an established rule.

We have demonstrated that clicker training shows promise as a robotic training technique that allows specialists in a particular field to train a robot without needing expert robotic knowledge. Clicker training can be implemented using differ-

ent internal representations which can allow on-line learning on low cost, but powerful, mobile robots. This could lead to further adoption of robots as tools for research and as workers in a wide variety of applications.

REFERENCES

[1] A. Birk and S. Carpin, "Rescue roboticsa crucial milestone on the road to autonomous systems," *Advanced Robotics*, vol. 20, no. 5, pp. 595–605, 2006.

[2] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans, "Modular reconfigurable robots in space applications," *Autonomous Robots*, vol. 14, no. 2-3, pp. 225–237, 2003.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[4] S. Russell, *Artificial intelligence: A modern approach, 3/E*. Pearson Education India, 3rd ed., 2010.

[5] R. A. Brooks, "Artifical life and real robots," in *Toward a practice of autonomous systems: Proc. of the 1st Europ. Conf. on Artificial Life*, p. 3, 1992.

[6] S. Koos, J.-B. Mouret, and S. Doncieux, "Crossing the reality gap in evolutionary robotics by promoting transferable controllers," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 119–126, ACM, 2010.

[7] A. Eiben, E. Haasdijk, N. Bredeche, *et al.*, "Embodied, on-line, on-board evolution for autonomous robotics," *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution.*, vol. 7, pp. 361–382, 2010.

[8] G. Capi and H. Toda, "Evolution of neural controllers for robot navigation in human environments," *Journal of Computer Science*, vol. 6, no. 8, p. 837, 2010.

[9] S. Schaal, "Is imitation learning the route to humanoid robots?," *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[10] E. Guizzo and E. Ackerman, "The rise of the robot worker," *Spectrum, IEEE*, vol. 49, no. 10, pp. 34–41, 2012.

[11] F. Kaplan, P.-Y. Oudeyer, E. Kubinyi, and A. Miklosi, "Taming robots with clicker training: a solution for teaching complex behaviors," in *Proceedings of the European Workshop on Learning Robots*, Citeseer, 2001.

[12] F. Kaplan, P.-Y. Oudeyer, E. Kubinyi, and A. Miklósi, "Robotic clicker training," *Robotics and Autonomous Systems*, vol. 38, no. 3, pp. 197–206, 2002.

[13] T. Desmond and G. Laule, "Use of positive reinforcement training in the management of species for reproduction," *Zoo Biology*, vol. 13, no. 5, pp. 471–477, 1994.

[14] H. Colahan and C. Breder, "Primate training at disney's animal kingdom," *Journal of Applied Animal Welfare Science*, vol. 6, no. 3, pp. 235–246, 2003.

[15] G. E. Laule, M. A. Bloomsmith, and S. J. Schapiro, "The use of positive reinforcement training techniques to enhance the care, management, and welfare of primates in the laboratory," *Journal of Applied Animal Welfare Science*, vol. 6, no. 3, pp. 163–173, 2003.