Teaching Artificial Intelligence Using Lego

Dr. Jianna Jian Zhang Irgen-Gioro
Jianna.Zhang@wwu.edu

Department of Computer Science, Western Washington University
Bellingham, WA, USA

We present 12-years teaching experience on Artificial Intelligence using Lego RCX, NXT, and EV3 Robotics platforms for both undergraduate and graduate classes at Western Washington University. We present the curriculum design, methods used, and outcomes of this kind of teaching. We discuss the advantages and disadvantages of using Lego. We suggest using a systematic method to raise the quality of teaching computer science, particularly to address how to teach students to code their idea efficiently using our research methodology of flowcharting-to-code. We will address some of the current problems in our education system for computer science and present our future plan.

Keywords — University Education, Robotics, AI, Lego

I. Introduction

Lego has a long and impressive history as an effective educational tool for all ages. Many educators adopt Lego to teach math, computer science, and engineering in different levels [5,6,8] including educational therapy for children with autism [1]. Lego Robot Kits such as RCX, NXT, and EV3 can be used to make prototypes for a real world robots using Artificial Intelligence (AI). In this paper, we present and discuss methods used to teach AI using Lego Mindstorms Robotics Kits in the classroom for undergraduate and graduate students at Western Washington University (WWU) since 2004.

We will first briefly introduce the history of RCX, NXT, and EV3, and the schematic for each with comparison discussion. In Section III, and IV, we will present the class curriculum design, practice, and the changes over the last 12 years. We will present and discuss the outcome of the classes, conclude our current research, and present future teaching plan and strategy in Section V.

II. History of RCX, NXT, and EV3

In 1939, with 10 employees [3], Lego started as a toy company. By 1986, Lego created software that enables a PC to control light, and sound sensors. After 12 years, in 1998, Lego released the Robotic Command Explorer (RCX), a plug and ready to program microcontroller brick [3,4]. RCX is a bench mark which made a significant leap for Lego education. At WWU, we created both introductory and AI application robotics classes using RCX. Due to the highly reusable Lego electronical and mechanical parts, students can create several different robots and apply different AI techniques during one academic quarter. With only 32kb of RAM, students must learn how to create memory-efficient code which is a permanent important skill of computer programming. Not long after the creation of RCX, Lego released its NEXT (NXT) generation of RCX in 2006, and Evolution (EV3) microcontroller in 2013 (See Figure 1). With NXT and EV3 bricks, we started to teach more AI algorithms including Artificial Neuron Network (ANN), Reinforcement Learning (RL), and Genetic Algorithms (GA) to tackle more difficult AI problems such as indoor navigation, and color shade recognition.

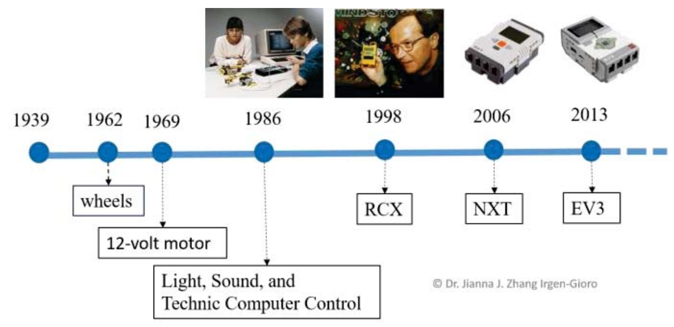

1939
1962
1969
1986
1998
2006
2013
wheels
12-volt motor
Light, Sound, and Technic Computer Control
RCX
NXT
EV3
© Dr. Jianna J. Zhang Irgen-Gioro

Figure 1. Summary of Lego Development Timeline [3,4]

A. *RCX: Robotic Command Explorer [5,6,8]*

Inside of the RCX, is an 8-bit Hitachi H8/3292 microcomputer with 32 kb RAM and 16 kb flash memory. The 16 kb flash is divided into 5 program slots. Instructions are executed with a speed of 16 MHz. The numeric display of the RCX is very useful for real time debugging. The numeric LCD has 43 segments that can be used to display sensor or internal values at run time. Instead of USB, RCX uses infrared signals to communicate with a PC or another RCX. The popular languages for RCX includes Forth, C, Pascal, and Java using BricxCC [16] (famous free ware), and Mindstorms SDK. A very useful feature, that disappeared in later versions of Lego microcontrollers, is a power adapter jack to allow continuous operation without consuming batteries.

B. *NXT: The Next Generation [5,7,9]*

Inside of NXT is a 32-bit AT91SAM7S256 microcontroller with 64 KB RAM and 256 KB flash memory which equivalent to a typical PC in 20th Century. On top of the main processor, NXT uses a co-processor, an Atmel 8-Bit ATmega48, to control motors. The communications between main processor and the co-processor is through a I2C bus. The execution speed is 48 MHz which is three time as fast as the RCX. Two other significant advantages of the NXT over the RCX are Bluetooth communication and the 100 × 64 LCD matrix display. On top of the original supporting languages, NXT also support Ada. However, it has no power adapter and consumes batteries much faster than both RCX and EV3.

C. *EV3: The Evolution (3rd Generation) [5,9]*

EV3 has a 32-bit ARM9 processor while NXT has an ARM7 processor. The size of the EV3 RAM is increased from 64 KB to 64 MB and the size of the flash memory is increased to 16 MB. The system clock speed is increased to 200MHz which is 3 times faster than that of NXT and 12 times faster than that of RCX. EV3 uses the same pixel matrix display with larger screen, 178 × 128. New features of EV3 over NXT and RCX is Wi-Fi connection to a network. It requires a wireless adapter which is also called a "dongle".

III. Curriculum Design and Practice

The CS robotics classes at WWU were created in 2003. We have used Lego RCX, NXT, and EV3 as the platforms for both first-year, third-year undergraduate and MSc. graduate robotics courses over the past 12 years. These courses provide an introduction to robotics, AI algorithms, and how to program AI. Students start by learning to build a variety of robots then programming motors and sensors. Applications using AI algorithms depends on different levels of the courses. The general goal is to provide students a first-hand experience in quantitative and symbolic reasoning. It is not uncommon for students to be curious about AI, but fearful that it is too difficult to learn. We try to provide a positive learning environment for students who do not have AI background knowledge, and change serious and fearful learning process into a natural and interesting learning process. The following guideline defines the curriculums.

Students will be able to:

- Clearly define target problems
- Design strategies to solve these problems
- Design and build robots that are suitable to solve these target problems
- Analyze both the fundamental and complex logical relationship between input and output
- Make intelligent decisions using AI techniques to deal with physical environments
- Use flowcharts method to present decisions to the problems
- Translate ideas using flowcharts-to-code method
- Document program code
- Test and record results
- Discuss and analyze the test results
- Make future plans

With this guideline, students are required to creatively build their own robots using classroom knowledge for the final project. Throughout the course of hands-on learning, students gain a deeper understanding of AI algorithms and the applications of these algorithms in different physical environments. We found that students' motivation and efforts in learning AI theory and concepts increase as they observe intelligent behaviors from their own robots.

Due to the hardware changes and cost increases over the last 12 years, our curriculums have changed. For example, we originally used three RCX light sensors in an ANN line following robot because the cost of RCX light sensors are much cheaper ($1.76) than that of NXT ($24.95) and EV3 ($39.95). The learning strategy or the application algorithms evolved into more complexed ones that must be work with less sensor input, for example learning color recognition or balancing.

IV. Teaching AI Using Lego Bricks

In this section, we highlight a few cases of our approach for teaching AI with Logo Bricks. We have three levels of robotics classes, and currently using RobotC with NXT and EV3. The first year class emphasizes teaching plain RobotC programming with simple AI techniques such as left-hand rule for line followers and maze solvers. While the third-year and graduate classes, we apply reinforcement learning, artificial neuron networks, and genetic algorithms.

A. *Teaching a First-Year GUR Robotics Class*

Line following and maze solving are two most commonly used projects to teach robotics classes at the beginner's level. Most of the students who signed up for this class did not have prior robotics or programming experiences. They are working towards different majors from different departments at WWU. Line following and maze solving are two of most commonly used projects to teach introductory robotics classes. We strive to teach the following:

- Understand how Lego motor and sensor work
- Design intelligent strategies for the robots to follow
- Present these strategies in flowcharts
- Translate these flowcharts to program code

Currently, the "Introduction to Robotics" class uses the NXT platform with RobotC. The Lego Mindstorms kit includes touch, light, sound, and ultrasonic sensors. Although most of the students in this class are not in computer science major. we want them to gain an appreciation for how the hardware works in addition to learn how to program. For example, the NXT touch sensor returns default value 1 or 0 (touched or not touched). We do not only show the electronic circuit, but also the internal structure of the sensors, and how to program them. We use the light sensor combined with the ultrasonic sensor to teach a simple, yet powerful, AI technique called "left-hand" or "right-hand" rule. The environment is a loop of black line on a

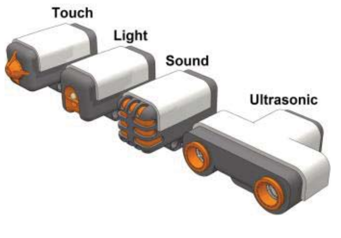


Figure 2. NXT Sensors [12]

white surface, or walls of a maze. As a light sensor reads darker color, it directs the motors turn towards the white surface and vice versa. For the maze example, when the ultrasonic sensor reads the walls are “too far away”, the robot should turn itself closer to the wall such that the robot continues to follow either left or right-hand side of the walls as designed by the students. This simple AI application makes students truly understand what AI is about, and aware that AI is everywhere in our life.

Unlike searching in a huge deterministic database, calculate with large quantity of numbers, or processing 3-D graphics, the program that we create for robots must have the ability to deal with ever changing real physical world. This can be seen when considering the simple light sensor example above. When the lighting condition changes at the different time of the day (sun rise, down, sunny, rainy, etc.), immediately students noticed that their robots behave differently. “It does not listen to me”, as some of the students commented. This demonstrates the need for dealing with environmental changes, and we must “give” the robots the ability to do so. We further direct students to think about how would a person deal with this situation, and how could we code the robot to collect the environmental statistics, such as color intensity, and then automatically calculate a threshold. Students now realize that it is not hard to code an intelligent behavior limited to a fixed environmental condition, but it is very difficult to make a robot adapt to the ever changing environments. At this point, we can introduce some more advanced AI: learning behavior.

For this introductory class, we introduce a simple learning behavior: automatically set the light thresholds by collecting environment statistics. That is, read the black and white color under any lighting condition, and then calculate the thresholds on the fly. With this learning behavior, robots can follow the black line under any lighting condition. As result, students suddenly realize that we can code a learning behavior using less than 10 lines of code.

We teach students to use an ultrasonic sensor to detect the walls of a maze. The task is to keep a constant distance between the robot and the walls of the maze when the robot is moving forward to the end of the maze. Note, we eliminate loops of walls in the maze to simplify the learning environment for first year students. Two major difficulties we want to address here. The first one is that the ultrasonic sensor reading for less than 3cm or longer than 180cm are both equal to infinity. Theoretically, the NXT ultrasonic sensor has a reading 0-254 cm, but the reality is quite different. The readings of distance 0 and 255cm are equally set to the value “??????”. This error default reading not only occurs with Lego ultrasonic sensors but also with general ultrasonic sensors, such as Arduino compatibles. The second difficulty is the blind spots. There blind spot is about 12-15° between the transducer to the receiver of the sensor. Students must learn to deal with these problems while programing the left-hand or right-hand rule.

We now discuss our teaching method to transfer these ideas into a computer language such as RobotC. The approach starts from defining the target problem. We require each student describe the target problem using their own language. They must define the general problem, and divide the general problem into sub-problems. Then the students must decide what kind of robot should be used. Based on this information, we worked with the students to define a logic table.

In a logic table, we first lay out the input and output variables and then display the logical relationship between these variables. Table 1 illustrates an example of a logic table that shows the relationship among a light sensor, a sonar sensor, and two motors. The task is to follow a black line and avoid objects on the black line using a right-hand rule (Example 1).

Table 1. Logic Table Example 1

Light	**Sonar**	**Left Motor**	**Right Motor**	**Action**
<= Thr	>= Max	slower	Faster	Turn left
> Thr	>= Max	faster	Slower	Turn right
Don’t care	< Max	stop	Fast	90° left turn
		Master (full speed)	Slave (30%)	Circle around

Each row shows one relationship, and it is the flowchart’s job to show how these individual actions arranged sequentially or parallel in order to fulfill the overall line following and object avoidance task. We must help students understand that for each set of input (situation), there must be a set of output (actions). Note, the logic table does not necessarily show the sequence of actions but the logic between sensors and motors.

After the logic table is generated, we show students how to create a flowchart based on the logic table and task description. We strongly encourage students to draw a flowchart first, examine the flowchart carefully to see if there are any logic errors, problems, redundancies, and or any inefficiencies before they start programming. However, students, particularly most of the computer science students are not used to drawing flowcharts. They tend to start coding without thoroughly understanding the problem and the logistics. This results in inefficient time usage. That is, taking longer time to debug in order to solve unknown logical errors. In the worst case, they have to rewrite the entire code due to “unknown” or “hard to figure out” logical errors. This inefficient style also has a tendency to produce redundant code, waste CPU time, and memory. We should not waste memory just because the size of memory increased dramatically. On contrast, we should keep educating our students to pay attention to using their resources efficiently.

How do we make a flowchart? There are two simple rules:

- Using correct symbols (just like using correct alphabet for English language for communications, Figure 3)
- Directed graph with clockwise direction (mixing both counter clockwise and clockwise arrows make a diagram hard to read)

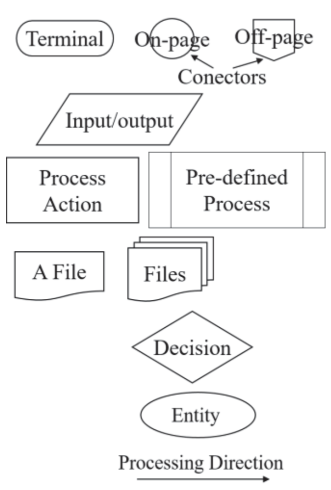


Figure 3 shows the essential flowchart symbols, and Figure 4 shows the flowchart for Example 1. All lines have its own direction, the entire flow is in clockwise direction, and the whole chart should be simple and clean. The arrows should be connected to the next action sequentially, and the loops should be indicated by clockwise arrows. To avoid cross over lines and crowding, we should use either on-page or off-page connectors. If one flowchart becomes crowded, we should break it up.

Figure 3. Flowchart Symbols

Students must think very carefully. Which sensor value has higher priority? How do we make a loop of sequential processes to fulfill the task? Where are the loops, and where to put the condition for each loop? Because the logic table only shows logically related segments of information for coding, we must teach students to make the sequential connections. We begin with asking students to determine which condition the program must check first? For our example, it is the sonar sensor because when the robot is too close to an object, the robot must go around the object no matter what color the light sensor reads. Students noticed that when the sonar sensor was used as the first condition in the loop, then the rest of the parts of the flowchart are merged together naturally. Translating the flowchart to code becomes an easy task if the students are familiar with RobotC syntax, which we have taught through many examples prior to the lectures on creating flowcharts.

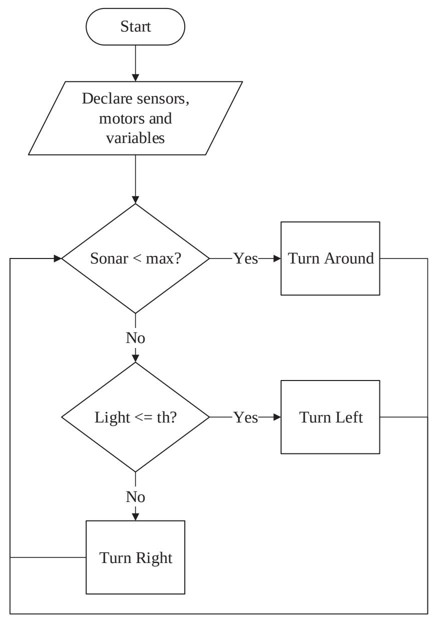
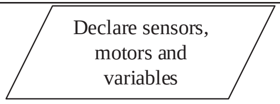


Figure 4. Example 1 Flow Chart

Here are the translation steps. First we show students how to start a RobotC program:

```
task main( ) {
}
```

Then using the built-in method of RobotC to declare sensors and motors:

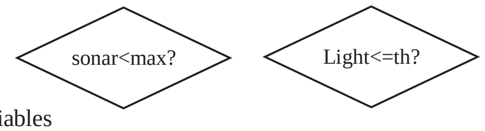


```
#pragma config (Sensor, S1, light, sensorLightActive)
#pragma config (Sensor, S2, sonar, sensorSONAR)
#pragma config (Motor, motorA, leftWheel,
                    tmotorNXT, PIDControl, encoder)
#pragma config (Motor, motorB, rightWheel,
                    tmotorNXT, PIDControl, encoder)
    //*!!Code automatically generated by 'ROBOTC' … !!*//
```

Next the variables are declared:

```
task main()  {
  //declare variables
  int max = 25;  //max distance between robot and object
  int th = 40;      //threshold for distinguish dark color
  //check sonar sensor
  //check light sensor
}
```

Now we show students how to use the input sensor values as control conditions using the "if" statement:



```
task main()   {
   //declare variables
   int max = 25;  //max distance between robot and object
   int th = 40;      //threshold for distinguish dark color
   //check sonar sensor
   if (SensorValue[sonar] < max)   {
      //turn around
   }
   //check light sensor
   else if (SensorValue[light] <= th)   {
      //turn left
   }
   else //turn right
}
```

The code is almost complete except the loop which repeats the sequence of processes. We guide students in the completion of this task by pointing out the flowchart again. Immediately students realized that the loop can be created with the addition of a "while" statement before the "if" statement. The resulting program produced with students in a class is shown below:

```
task main()   {
    //declare variables
    while(true)  {
            //check sonar sensor
            if (SensorValue[sonar] < max)   {
             //turn left
            }
            //check light sensor
            else if (SensorValue[light] <= th)  {
               //turn left
            }
            else   {  //turn right   }
      }
}
```

We have presented a systematic and logical method called “flowchart-to-code” to develop computer programs. This approach is especially useful for larger and more complex software.

B. *Teching Atificial Neuron Networks with Lego Robots*

We have been teaching ANN using RCX, NXT, and/or EV3 since 2003. The basic technique we use in higher level undergraduate and graduate classes are the same as we have shown in the previous section. We first explain the theory of ANN, show samples of the topology of forward networks, explain how to compute the gradient, and then the computation of the delta value for the back propagation. It is particularly effective when we connect the algorithm with a real problem such as learning to distinguish colors or learning to follow a black line instead of hard coding the behavior.

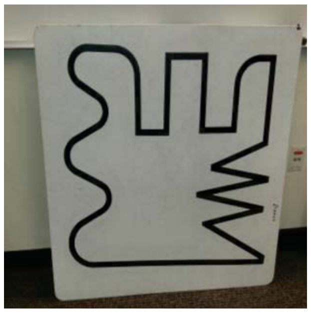


Figure 5. Line Track Used by ANN Robots

Students are quick to understand the concept of ANN, but tend to have difficulties applying the theory and concept to real world projects. In the past we have used the RCX with three light sensors to learn line following behavior around a complex course (see Figure 5). The general idea is to train a robot to take correct actions when it sees a black line.

This may seem straight forward, but it is not as simple as it sounds because the robot is not in an ideal virtual world and make sharp turns accordingly. The test results were not ideal compared with hard coded robots, however all the ANN RCX robots can correctly follow the most part of the black line shown in Figure 5. With faster speed, say > 75% power level, they all have difficulties following the zigzag sharp corners closely. The typical topology for the RCX line flowing ANN robot is 3×5×2. Three input light sensor value is fed into the hidden layer, and the output are two actions: left turn or right turn.

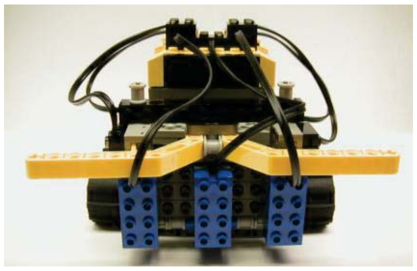


Fig. 6. RCX Line Follow ANN Robot

We applied ANN and general AI algorithms to many other projects such as learning to walk, self-balancing, Tic Tac Toe game engine, a Hanoi Tower problem solver, searching for objects in an open area, learning to type on a keyboard, learning to generate a piece of acceptable music, and more.

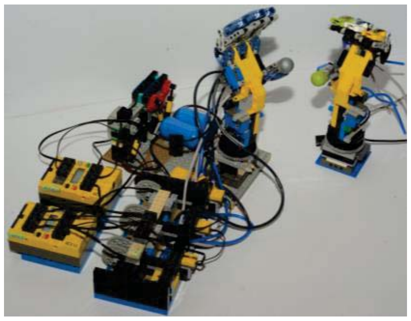


Fig. 7. Pnuematic Hand Learn to Grap

Figure 7 shows another robot that we use for research and teaching ANN. These robot hands are based on the pictures posted on the Brickshelf [10]. It is not only a good platform for ANN learning but also good platform for reinforcement learning. We use two RCXs, six motors, and two air tanks pneumatic system. The details of this project will be published in a separate paper in the near future.

A NXT Turing Machine shown in Figure 8, is another classroom example which implements an abstract Turing Machine. This Lego NXT Turing Machine is built based on [13] with many modifications. We use 16 lift arms to represent 16 bits of I/O (2-symbol: 0 or 1), and 3-states (L, M, R). The initial head position is on the top of the 4th bit of the “tape”. The initial instruction is one byte located from the 4th to the 11th bits. There are two kinds of operations for current application: do math or display text. We use right hand side 8 bits (0-7) to represent the first operand, and the left hand side 8 bits (8-15) to represent the second operand, where both can be any of the 128 ASCII code. The input data can be entered by users or randomly generated by the NXT brick. The tape (with the lift arms on it) moves back and forth. We use an infinite number of finite states to represent the infinitely long tape while the head of the Turing Machine stays in place. If a lift arm is away from the head reading point, the input for that particular bit is a “0”, and “1” otherwise. Let the input string from the Turing Machine be: $\{0,1\} \mapsto \mathit{Ascii\ code}$ such that the following finite states hold:

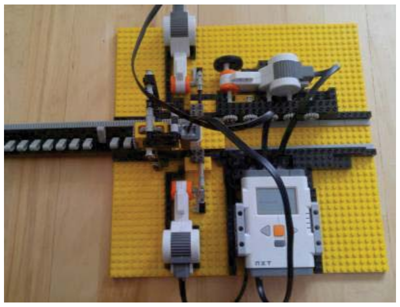
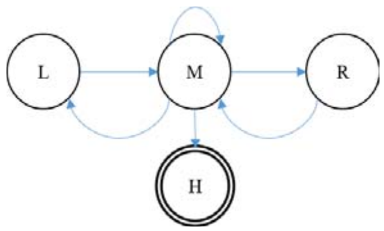


Fig. 8. NXT Turing Machine



The corresponding state table for the NXT Turing Machine is defined as the follows:

Previous State	Current State	Scanned Symbols	Print Symbol	Move Tape	Next State
Start	M	O: Ascii	0/p	fwd	L
M	M	0	0/p	stop	H
M	M	1	1/p	idle	M
M	L	R1: Ascii	0/p	rev	M
L	M	R2: Ascii	0/p	rev	R
M	R	No scan	0/p	fwd	M
R	M	No scan	0/p	fwd	L

Table 1. NXT Turing machine State Table

We move the “tape” forwards or backwards based on the M state value scanned at the beginning of operation. Then the tape will move to the beginning position which is R. Then the machine will scan two values, R1 and R2. For example, if the operator is Ascii code “+” or “/”, the machine will perform the addition or division of R1 and R2 respectively. While if the operator is “T”, then the Turing Machine reads a sequence of Ascii code, and translate them into text. The text is displayed on the LCD panel. In the future, we want to make it into text-to-speech Turing Machine.

Although the RCX and NXT can be used for many AI algorithms, currently both RCX and NXT are no longer supported by Lego company. We have now switched to EV3 bricks. As discussed in previous sections, EV3 has improved tremendously in comparison with the RCX and NXT. Figure 9 shows a multi-purpose EV3 robot which we currently use to teach both introductory and higher level robotics classes. We made this robot as simple as possible to save material and building time. It can be used for line following, navigation, obstacle avoidance, detecting color, climbing stairs, and as a platform for AI applications.

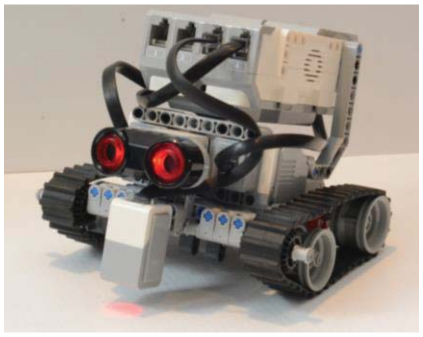


Fig. 9. EV3 Color Recognition ANN Robot

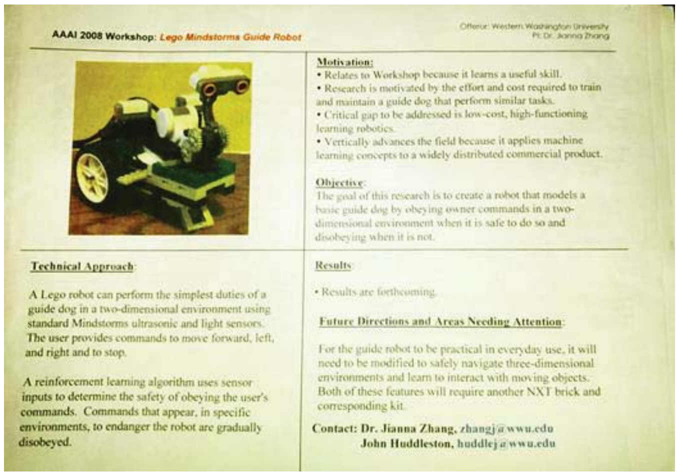
For example, we use this robot to teach classical Backpropagation ANN. The learning task is to make the robot recognize 5 different colors: red, green, blue, yellow, and brown while it is traveling on a black ring on a white surface. Based on our classroom experiments, EV3 has less accurate reading for “blue” than that of “red” and “green”. RobotC language provides a convenient way to obtain the standard Lego color: 0 = colorNone (not color object detected); 1 = colorBlack; 2 = colorBlue: 3 = colorGreen: 4 = colorYellow: 5 = colorRed: 6 = colorWhite; 7 = colorBrown. However, if any color that does not belong to the standard Lego color, this mode would fail to classify them. To learn colors that are different from the standard Lego color, we have to use the raw EV3 RGB value. Theoretically, each of the R, G, and B channels have a value from 0 to 1024. That is over one million (1,070,598,144) distinct RGB color sequences. However, after many tests, we found that the maximum RGB value never exceed 800. Thus in reality, the input RGB color sequence is less than 510,081,600. As we can see, the input space is still huge. To make this computation possible, we use the hierarchical ANN techniques as shown in our previous research [11].

We first roughly classify these half million input RGB sequences into three classes using a generalization. That is, the first class of the RGB input sequences are those causing the R-output neuron to fire, the second class of the RGB input sequences are those causing the G-output neuron to fire, and the third class of the RGB input sequences causing the B-output neuron to fire. Then we use these three classified RGB sequences to train three identical but separated sub ANNs. The final output of the entire ANN system can recognize 125 colors in the RGB gamut cube. The general approach for this project is the classical Backpropagation using Gradient Decent. The topology of the ANN is: 3×3×3 for the first ANN, and 3×40×125 for the three identical sub ANNs. We will report our testing results for this more complex example in a separate paper in the near future.

Currently, we request students apply a simpler version of the ANN color recognition exercises. We use Lego predefined 8 color as input training data on the classical Backpropagation ANN. We request graduate level students apply the more advanced Hierarchical ANN approach. No matter how much more complex the programming task may be compared to those for the first year students, we found that the teaching method explained in Section IV is very effective. Students are taught to use a systematic method to program AI, and help them reduce confusion when implementing the details.

C. *Teaching Reinforcement Learning with Lego Robots*

Reinforcement Learning is also taught in our classroom using Lego robots. Figure 10 shows a NXT Lego guide “dog” robot that uses a Q-Learning algorithm to train itself not just listen to the owner’s command, but also learn to disobey when it encountered a dangerous situation [14] [15]. The dangerous situation including, but not limited to, objects blocking the way, a hole or a ditch nearby when the owner gives command to move to that direction.


AAAI 2008 Workshop: *Lego Mindstorms Guide Robot*

Offeror: Western Washington University
PI: Dr. Jianna Zhang

Motivation:
• Relates to Workshop because it learns a useful skill.
• Research is motivated by the effort and cost required to train and maintain a guide dog that perform similar tasks.
• Critical gap to be addressed is low-cost, high-functioning learning robotics.
• Vertically advances the field because it applies machine learning concepts to a widely distributed commercial product.

Objective:
The goal of this research is to create a robot that models a basic guide dog by obeying owner commands in a two-dimensional environment when it is safe to do so and disobeying when it is not.

Technical Approach:

A Lego robot can perform the simplest duties of a guide dog in a two-dimensional environment using standard Mindstorms ultrasonic and light sensors. The user provides commands to move forward, left, and right and to stop.

A reinforcement learning algorithm uses sensor inputs to determine the safety of obeying the user's commands. Commands that appear, in specific environments, to endanger the robot are gradually disobeyed.

Results:

• Results are forthcoming.

Future Directions and Areas Needing Attention:

For the guide robot to be practical in everyday use, it will need to be modified to safely navigate three-dimensional environments and learn to interact with moving objects. Both of these features will require another NXT brick and corresponding kit.

Contact: Dr. Jianna Zhang, zhangj@wwu.edu
John Huddleston, huddlej@wwu.edu

Fig. 10. NXT Guide Dog Robot, 2008 [14] [15]

This NXT guide dog robot operates in the 2-dimentional *n*×*m* grades-world using an ultrasonic and a light sensor. A policy must set to enable the initial learning. We must first set the reasonable policy for initial learning. We set this initial policy as follows: the robot can move one step forward, left, right, and reverse within the closed environment. For example, given the environment is a 3×5 grades, and the current position of the robot is $p = (1, 1)$, then the robot cannot move to (1, 2), (2, 1) but not outside the limit. The goal of the learning is to obtain an optimal policy to avoid obstacles automatically, and disobey the owner’s command in a dangerous situation. The general Q-Learning function given this initial policy on action "*a*" given a state "*s*" can be written as

$$Q(s, a) = r(s, \mathrm{a}) + \gamma \operatorname{argmax}_{a'} Q(\delta(s, a), a') \quad (1)$$

where "*r*" is the reward value for action *a* in state *s*, γ is an arbitrary learning speed between 0 and 1, and *a′* is one of the next actions that is allowed by the current policy. We want to maximize the Q-value for each given state and generate two things: an optimal action and a new Q-value. For example, if the next action "*a′* = left" gives the highest Q-value based on the current state value $\delta(s, a)$, then the robot would turn left. After performing this action, the robot would update its Q-table with the optimal Q-value. The robot repeats this learning process from different starting positions, and eventually it can continually improve its performance by looking up the learned Q-table: the state-action pair. This practice helps students gain an understanding of the fundamental machine learning theory.

D. Teaching Genetic Algorithm with Lego Robots

The students are asked to reuse their Backpropagation ANN programs to learn the general Genetic Algorithm (GA). Instead of using the Backpropagation, we show students how to use GA to generate candidate ANNs, evaluate these candidates, select parent pool of ANNs from these candidates, and then use GA operators to produce child pool of ANNs. For example, we start with random generated ANNs (sets of weights), then test them to see how many colors that each ANN classified correctly. These test results show how "good" each ANN is. Several selection methods are introduced, such as Ranking, Rolette Selection, Tournament, Steady State, and Elitism Selections. Students can use one of these selection methods or combine them to generate a pool of parent ANNs. After that we explain how to use GA operators, such as Cross Over and Mutation, to produce a pool of child ANNs. This process of measurement, selection, and mutation are repeated over and over until several satisfactory ANNs are generated. We found that by modifying previous ANN programs to learn GA, not only helps students learn a new machine learning algorithm, but also reinforce their knowledge on ANN.

V. Future Teaching Plan and Strategy

Our 12-year classroom experience shows that using Lego robots to teach AI is economical and effective. Students can extend their creative ability by making different kinds of robots more easily, spend less time on the mechanical construction, have more time on learning AI algorithms, and programming to deal with real world complexities. We regard this first-hand programming experience as a very important preparation for students to be successful in their future careers. Using Lego is economically desirable compared with other equivalent commercial robots. However, using Lego as the classroom equipment is labor intensive for maintenance, such as tracking and sorting Lego parts as well as the sensors, wires, and batteries.

In the future, we want to create more demo examples on color sensor because there is still much to discover. Currently, the raw RGB value do not show the full range, but we do not fully understand why. More investigation must be done to test the electronics, mechanics, firmware, and RobotC language on EV3 color sensor.

Another interesting sensor is the EV3 Gyro. This sensor is not stable. The well-known problems are its drifting and lagging. To make this sensor useful with RobotC programming language, we have to hard code a counter drifting every mSec There must be better solutions. As for the lag, the most popular solution is to use a third party Gyro sensor. There is some problem located in RobotC language or the firmware. We may create more sophisticated RobotC library as a solution.

We plan to connect cameras to Lego EV3 brick, because we believe computer vision is an important key to indoor navigation robots. We want to further develop our teaching methods to help students learn better and faster. We feel that the current education system of teaching computer science needs more focus on quality instead of quantity. Using robots as real world programming platform helps students raise their comprehension level, and increase the ability to problem solve.

References

[1] Ferrandez, Jose Manuel ; Paz, Felix ; Barakova, Emilia I. ; Bajracharya, Prina ; Willemsen, Marije ; Lourens, Tino ; Huskens, Bibi; "Long-term LEGO therapy with humanoid robot for children with ASD", Expert Systems, 2015, Vol.32(6), p.698(12) [Peer Reviewed Journal].

[2] Gomez-de-Gabriel, Jesus Manuel; Mandow, Anthony; Fernandez-Lozano, Jesus; Garcia-Cerezo, Alfonso, "Mobile Robot Lab Project to Introduce Engineering Students to Fault Diagnosis in Mechatronic Systems", IEEE Transactions on Education, Aug. 2015, Vol.58(3), pp.187-193 [Peer Reviewed Journal]

[3] Lego Group, "Lego History Timeline", http://www.lego.com/en-us/aboutus/lego-group/the_lego_history, 2016.

[4] Mindstorms, "History of LEGO Robotics", http://www.lego.com/en-us/mindstorms/history, 2016.

[5] Official Lego Vikipidia, "Lego Mindstorms", https://en.wikipedia.org /wiki/Lego_Mindstorms#RCX, 2016.

[6] Hitachi, "Hitachi Single-Chip Microcomputer", http://www.legolab .daimi.au.dk/CSaEA/RCX/Manual.dir/H8Hardware.pdf, 3rd Edition, accessed 2016.

[7] ATMEL, "AT91SAM ARM-based Flash MCU", http://www.atmel.com /Images/6175s.pdf, 2012.

[8] University of Aarhus, Department of Computer Science, RCX Manual,

[9] Lego Mindstorms EV3, https://en.wikipedia.org/wiki/Lego_Mindstorms _EV3, accessed 2016.

[10] Brickshelf folder: MOC Bionicle Technic Mecha hand legotic, Folder created: 2003/11/30 13:10:26, Folder modified: 2006/01/26 21:13:35, accessed 2016.

[11] Zhang, J., Blachford, D., and Tien, J., "Chinese Handwriting Recognition System Using Artificial Neural Network" in Chinese Language Instruction and Computer Technology, in Traditional Chinese Language, US Department of Education, Printed in Taiwan, June 2005, pp. 259-273.

[12] LEGO® Education, https://education.lego.com/en-us/, accessed 2016.

[13] Mario Ferrari, Giulio Ferrari, Kevin Clague, J. P. Brown, Ralph Hempel, "LEGO Mindstorms Masterpieces: Building Advanced Robots", 1st Edition, Syngress, 2003.

[14] Jianna Zhang and John Huddlston, "Guid Robot: a Q-Learning Lego Robot", The AAAI 2008 Workshop on Mobility and Manipulation (held during the 23rd AAAI Conference on Artificial Intelligence), 2008, Chicago, USA.

[15] Anderson, Monica; Jenkins, Odest Chadwicke; Oh, Paul, "The 17th Annual AAAI Robot Exhibition and Manipulation and Mobility Workshop", AI Magazine. Volume: 30. Issue: 1, 2009, pp. 95-102.

[16] Bricx Command Center, http://bricxcc.sourceforge.net/, accessed 2016.