Rodney Junior: Smarter than the Average Robot

Part 1 of 3

By Camp L. Peavy, Jr.

My original 1987 “Rodney” from “How to Build Your Own Self-Programming Robot”... note the address and data toggle switches on top and relays on bottom in front.

I graduated from Florida State University in 1982 with a degree in marketing; I had to sell something what else does one do with a degree in marketing? At the time Apple was going public and coming out with the Apple III. I thought to myself, “Oh, this looks interesting I think I’d like to get involved with this (whatever “this” is)”. Otherwise I would have gone into advertising or real estate sales… the proverbial “Road Not Taken”. Regardless, I started selling Trash 80’s (as they were affectionately known) at the local “Radio Shack Computer Center”. At the time thought to myself… I’d like to take this (whatever “this” is) to the nth degree… which is what? Mobile robots! Mobile robots are nothing more than the microcomputer of the 80’s with eyes, arms, and legs. Because it was grassy and wanted wear I picked up a book by “David L. Heiserman” entitled "How to Build Your Own Self-Programming Robot" (1979 by TAB BOOKS). Wherein he describes an 8085 microprocessor controlled device with an 8-bit data bus, 12-bit address bus… you’d set the data… set the address and punch the “Load” button… programming the robot one byte at a time in binary. Heiserman also had a little theory on “machine intelligence” which involved learning through experience. The robot’s name was “Rodney” and that has made all the difference.

Gladiator Rodney: Yes, that’s an electric chainsaw on an autonomous robot (be afraid). Note the 38KHz Infrared Pringles can beacon on top (homebrewed) and 1000 Watt inverter on back.

After wrestling with this graduate-level electronics course for over five years; building the front-panel, half the mainboard, auxiliary power and robot body I finally managed to re-create “**Alpha**” level machine intelligence as described by Heiserman with a store-bought 386SX motherboard, relay I/O card and GWBasic. This was truly one of those “It’s ALIVE!” moments every robot-builder craves. In fact I used a variant of this algorithm in my Robot Wars 1996 champion robot “Gladiator Rodney” (named for Heiserman’s robot) and my Burning Man Robot (1999-2005) “Springy Thingy”.

“Springy Thingy” guards camp at Burning Man 2003 (I think). She survived 5 “burns” the goal being to leave the event with a functioning robot.

This is an article about building a miniature version of Heiserman's "Rodney". Since he's a small version of the original let's call him "Rodney Jr." But what does "Junior" do? Well, he moves… yep, that's it he moves… the deal with Rodney and his protégé "Junior" is he will try different things until he successfully achieves this goal (that is, moving) and then he becomes more and more efficient at achieving this goal (that is, he learns). Primitive, yes, but like an amoeba bouncing around in a drop of water from random motions it learns the responses that moves it the farthest sooner… not a bad strategy for a little machine exploring its world. Besides, it's not **what** he does but **how** he does it that's important. In other words "Junior" isn't programmed just to go to-and-fro but programmed to monitor the mobility sensor (more about that later) and try random stuff until conditions are met that constitute his "goal" (whatever that might be). Then when he learns a particular successful response for a particular situation or condition… when in that situation again he tries the previously successful response; if the response is again successful "Junior" increments the confidence level of that response; if not the confidence level is decremented. This robot will think for itself and learn from its experience… you might say he's smarter than the average robot!

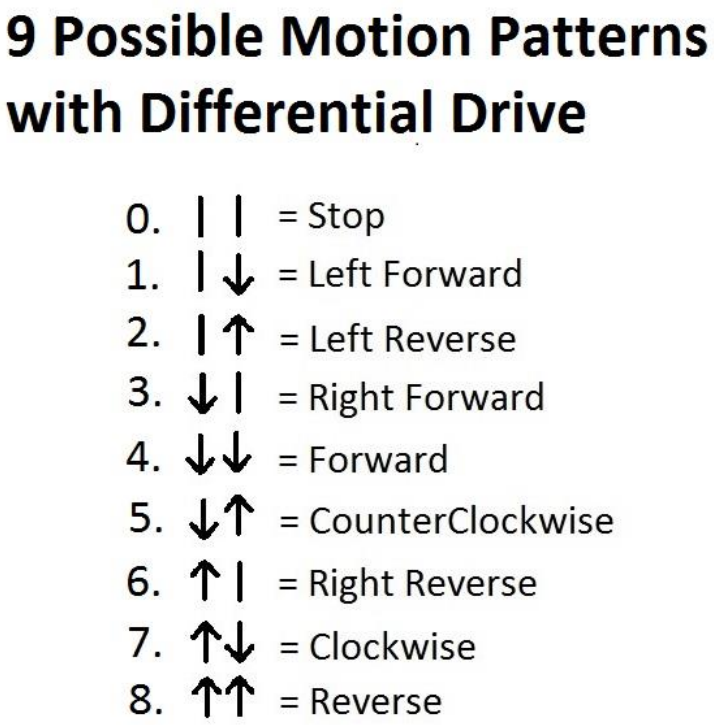
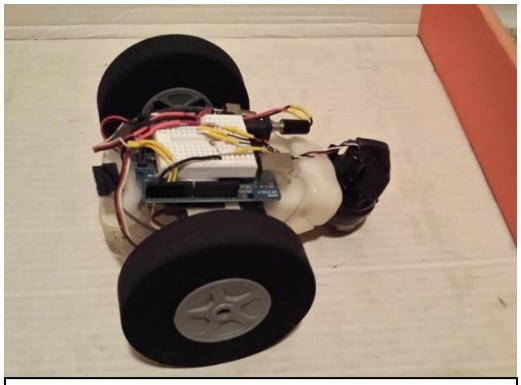


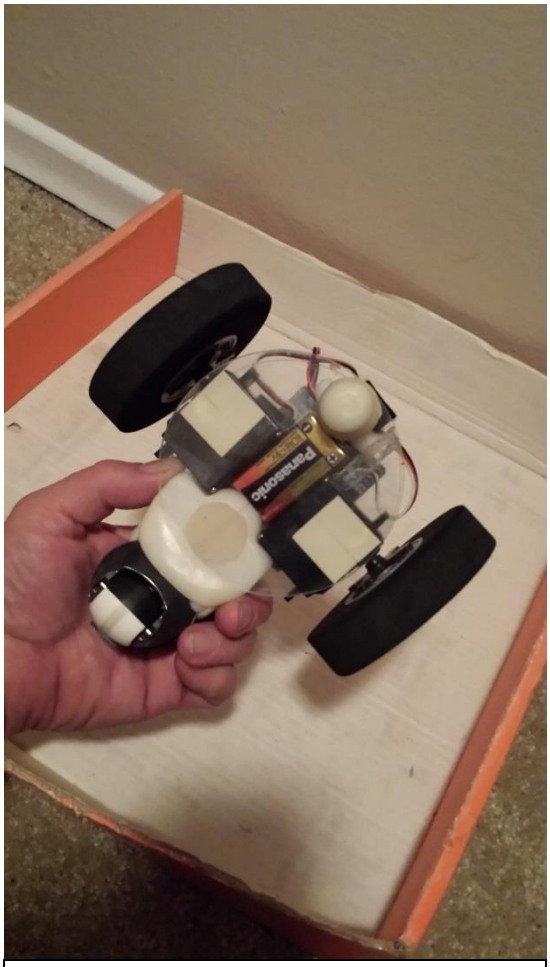
Figure 1: The only two patterns with any significant magnitude or displacement are "forward" and "reverse" (codes 4 and 8). All other motion patterns rotate around a

According to Heiserman **Alpha** level machine intelligence can be thought of as a basic reflex… like the aforementioned amoeba bouncing around in a drop of water. In Heiserman's Rodney the robot's goal was simply to move. In a differential drive system there are 9 possible motion patterns… including "stop" however, the only two patterns with any significant magnitude or displacement are "forward" and "reverse" (codes 4 and 8). All other motion patterns rotate around a point (Figure 1).

I quickly realized this when running the store-bought PC version of Rodney where every pattern or code (with the exception of "stop") would detect motion so the robot would go clockwise (code 7) or right wheel forward (code 3) or whatever code was randomly selected forever because the mobility detector would always detect motion… not very interesting and not very useful for a fighting robot. What I did with "Gladiator Rodney" was let the robot run for a limited period of time and then randomly change patterns regardless of whether or not the robot had stalled… overridden when the robot detected the opponent's beacon (In the autonomous division of "Robot Wars" basically both robots have modulated 38KHz beacons which are targeted by the opponent). This way the robot would basically "look" and roam randomly around the arena until it "saw" the opponent's beacon. At which point he would target the opponent with its IR eyes (TV remote control receivers) and charge towards him (more like a meander but you get the idea).



This is "Rodney Jr" based on two servos sandwiched between two acrylic disks, driven by an Arduino. The key to Junior's intelligence is the Roomba caster wheel in the rear. 1000 times simpler than the original "Rodney" which had to be programmed in binary.



This is the underside of the robot. Note the 9-Volt battery visible through the clear acrylic disk. Two servos are sandwiched between two disks and adhered with foam double-stick tape also visible though the disk. The Roomba caster is mounted with Shapelock in the rear… it only spins when going forward or reverse.

Same thing with "Springy Thingy" (the dancing robot)… I would basically let her go for a limited number of clicks and then change patterns (code 0-8). For Gladiator Rodney and Springy Thingy the mobility detector was a spring-loaded gate-wheel from ACE Hardware with rare earth magnets glued (E6000) all around the perimeter. I also glued a reed switch to the caster frame so that when the wheel spun (i.e. the robot was moving) the reed switch would switch off-and-on. It's funny the mind connects the random motion to music and it appears the robot is reacting to the music (i.e. dancing). Springy Thingy is alive and well (15 years young) and generally shows at Maker Faire and HomeBrew Robotics Club Challenge Meetings.

With "Rodney Jr." the Alpha level machine intelligence tries random motion patterns until the robot discovers either forward or reverse (again the only two motion patterns with any potential magnitude) like a basic

The heart and mind of the “Rodney Jr” system is the Roomba mobility sensor… all the robot wants to know is whether it is moving or not. He eventually learns the motion patterns which provide the most magnitude or displacement.

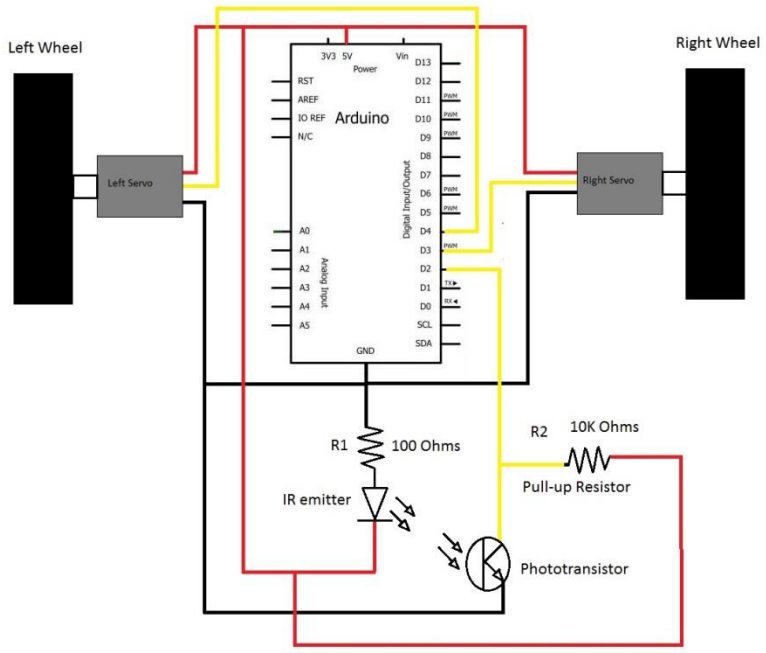
reflex trying random things to see what it can do to reach its goal or make it “happy”. This “goal” can be thought of as the robot trying to move the maximum displacement for a given area… again, not a bad plan for a little creature exploring its big world.

"**Beta**" class machine intelligence features memory; that is for a given motion pattern when the robot discovers either forward or reverse it remembers that successful response and uses it the next time when in that motion pattern or situation it encounters a stall condition. The program also increments a confidence level if the response continues to be successful and decrements it if it fails. Beta intelligence remembers successful responses and uses them when in the same circumstance.

"**Gamma**" class machine intelligence generalizes this information for heretofore unknown circumstances. That is when the robot encounters a stall condition in a motion pattern for which it has no memory (that is no Beta response) it tries a high-confidence Beta level response from other motion patterns before reverting to Alpha level behavior (random/reflex responses). With each sequential class of machine intelligence, Alpha, Beta and Gamma, the robot learns the motion patterns which maximize magnitude sooner and sooner.

To build “Rodney Jr” I’m using an old tabletop robot that was part of a HomeBrew Robotics Club build known as “The Club Bug”. The basic design is two continuous rotation servos sandwiched between two 4” plastic disks from TAP Plastics (you know… the fantastic plastic place) controlled by an Arduino Uno and powered by a 9-Volt battery; pretty common setup as far as tabletop robots go. The wheels are from a radio-controlled plane (Lite Flite) but could easily be one of those Servo Wheels from Parallax or Pololu… you could even screw the servo horns onto jar lids with rubber-band treads or any kind of wheel just make sure it’s not too slippery so you can get traction! The back-half of the robot is made from Shapelock. If you haven’t used this stuff yet you’re in for a treat. Shapelock (or InstaMorph or Friendly Plastic or any one of a number of name brands) is Low Temperature Thermoplastic … it comes in little beads that you put it into water and microwave for a minute or two… it comes out moldable (Hot! but moldable)… after it cools… it becomes a hard plastic… which can then be reheated and remolded if necessary… anyway I used this stuff to glom the Roomba stall sensor onto the rear. It’s best to fish the Shapelock out with chop-sticks… it sticks (bonds) to plastic and you don’t want to use the good silverware!

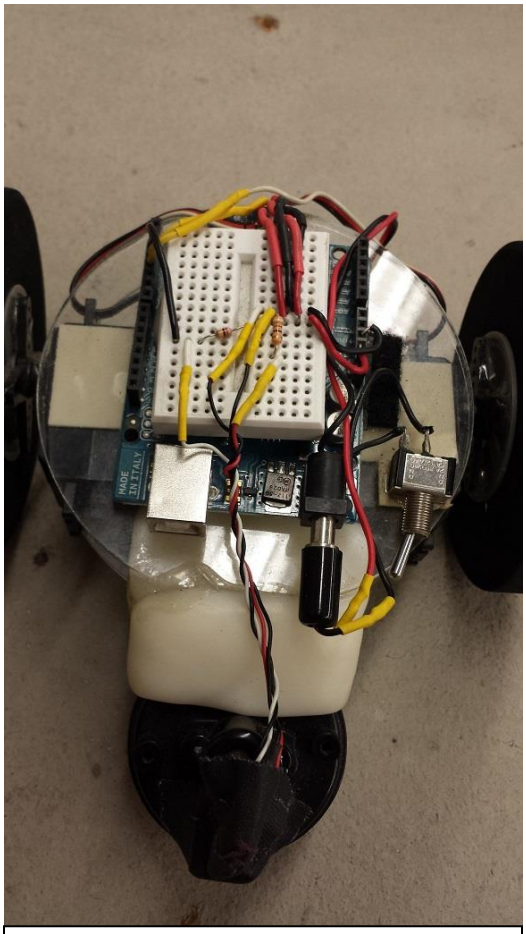
The beauty of this project is it can be used with most any tabletop robot… HomeBrew, BoeBot, LEGO, Arduino, etc… you're basically just adding the Roomba stall detector (although in this application I prefer to call it a **mobility sensor**) and monitoring to see whether the robot is traveling forward or backwards. The mobility sensor (if you will) in my Rodney Jr is connected to Pin 2. The servo signals lines are connected to Pins 3 and 4. The robot looks at the status of the mobility sensor (Pin 2) either high or low (I seed it with a "0" (stop)), determine whether or not the mobility detector is oscillating (did it change?), if it **is** oscillating continue the current motion pattern, if **not** try another random motion code (0-8), look at the status of the mobility detector (Pin 2) again (either high or low), determine whether or not the mobility detector is oscillating (that is going up and down), if it **is** oscillating continue the current motion pattern, if **not** try another pattern (0-8) randomly, over and over and over. This can be thought of as the little creature considering change over time. Where it looks at its current state, applies that to memory, now looks at its current state again, compares that to its past "current state" and does something based on whether or not these two are the same. In the **Beta** and **Gamma** stages Junior remembers his successful responses and uses them when appropriate. With **Alpha** he continuously uses random actions. For other ideas on how to scratch-build tabletop robots there are plenty of pages on the web or take a look at my "ProtoBot" and "Ameoba" articles from years ago in the "Resources" section. As you can sense this is not a 1,2,3,4… article on how to build a "machine intelligent" robot… it's more like guidelines for anyone who would like to investigate an accessible example of "artificial" intelligence and play around with a genuine robotic creature.



Typical IR led phototransistor pair circuit. If the original iRobot sensor is not available you can homebrew one. Most any emitter/detector pair will do (Radio Shack 276-142).

You'll want to get the Roomba caster wheel **with sensor** if only so it comes with the bracket to mount your own emitter/detector pair. They're generally available through eBay for about $25. Don't worry if you have the caster with no sensor… most any infrared emitter/phototransistor pair will do. Basically arrange the emitter and detector so they detect the rotation of the caster/encoder wheel as it changes from black-to-white. You can see this with "serial monitor" in the Arduino IDE. Mine is mounted

HomeBrew style (that is with gaffer's tape). R1 (the resistor for the emitter) should be adequate to limit the current so you don't melt the LED. Generally the formula is subtract your source voltage from the forward voltage drop in this case 5V-1.5V=3.5V (the amount of voltage the resistor needs to drop) and divide that by the continuous current (which in this case is 150mA (a lot of current for a little diode; generally LEDs are ~20mA). If you crunch the numbers the resistor value comes out to 23.33 Ohms… I tried a 100 Ohm resistor and could see the IR light… also it triggered the detector when I rotated the wheel so what the heck… save a little energy… 30 instead of 150mA.



I usually place a solderless breadboard on top of the Arduino so that I can plug into the sockets. Here the power on the servos is going to the built-in 5V supply as are the emitter and detector.

R2 (the resistor for the detector) should be the maximum resistance of the detector (that is when it is in-the-dark)... usually ~10K Ohms. The one from Radio Shack measured 13K Ohms… so I tried a 10K and as mentioned above it worked so good enough is good enough! Also note that the phototransistor (i.e. the detector) is reverse biased… that is the cathode is connected to "+" and the anode "-". Be sure and check your specifications as your mileage may vary! And of course you won't be able to "see" the IR LED (because it's **below** (infra) red on the spectrum). However you view can view it through a digital camera (i.e. your phone).

As far as the software is concerned the first thing you do is load the Servo library… then initialize your variables (standard Arduino stuff). Set the "mobility_sensor" to digital Pin 2. Initialize the "mobility_read" variable. Initialize "mobility_read_past". Initialize "stall_ticks. Initialize "MotionCode". Initialize "PastMotion". Initialize "Confidence" and finally initialize the "MemArray". If you look at the sample code and walk through it you can literally see what the robot thinks and how it learns.

So the first thing your program should do is read the current state of the mobility sensor… either a "1" or a "0"… then run the wheels a random pattern or "code"… look at the mobility sensor again and see if it's changed… if the sensor did not change increment "stall_ticks" one unit until it reaches some threshold (mine is set at "100"). Eventually it will discover forwards or backwards (the only two motion codes with magnitude) and hopefully have some room to maneuver so the mobility sensor oscillates between "0" and "1" (i.e. it's moving). As long as the mobility sensor is spinning the robot's "happy". You can test this by picking up the robot and spinning the caster wheel… whatever motion pattern the wheels are spinning will stay in that pattern until you stop spinning the wheel. You can also view the status of the mobility sensor's pin with "serial monitor" in the Arduino IDE. It should switch between "0" and "1" when you spin the wheel.

The current state of Rodney Jr. is that of **Beta** intelligence… it has some issues with sometimes the mobility sensor detecting motion when either left or right wheel goes reverse (codes 2 and 6)… this is actually very interesting because it still accomplished the goal of maximizing distance around the pen. He would go forward and turn slightly in reverse which allowed him to extradite himself by then going forward (code 4) the motion pattern that he **remembered** as a successful response. You can tweak with the "stall ticks" to make Rodney Jr more or less sensitive. Also I had to remove the decrement confidence level component as when Junior learned forward or reverse it got removed once he eventually ran into a barrier. What needs to happen is travelling beyond a longer distance needs to define success rather than just a short distance. Finally we need to get to the G**amma** level where he will draw reactions from other successful reactions rather than reverting to **Alpha** behavior (random). So like most robot projects this one is still not finished… that's okay though… because when the robots take our jobs… our work will be done. In the meantime I'll produce another article that take Rodney to **Gamma** and incorporates the distance level. If you want to play along the code is online and there's an online forum from SERVO where we can discuss this article.

Other features I'd like to integrate into Rodney Junior are battery sensing and automatic charging, light sensor, microphone, speaker (audio output)… maybe make a new goal like follow a ball, a line or a wall incorporate these things into the learning situation where the robot is keeping track of more variables than just the mobility detector… and more outputs than just the drive motors… it's not so much programing it but interacting with this intelligent little creature on his own terms.

Resources:

http://tinyurl.com/JuniorPlayPen1
http://tinyurl.com/JuniorPlayPen2
http://tinyurl.com/GladiatorRodney
http://www.camppeavy.com/articles/protobot.pdf
http://www.camppeavy.com/articles/ultimate.pdf
http://www.camppeavy.com/articles/amoeba.pdf

Parts List:
Arduino Uno - Digikey #A000066 - $28.28
Tap Plastics 4" diameter Acrylic Circles - $2.05ea x2
Dave Brown Lite Wheels 4-1/2" - (2) $9.95
Shapelock, 250 Grams - $14.95
Roomba 500 700 Series Front Wheel Caster with sensor - ~$25 eBay