TinyGA

Posted on Jun 13, 2012
1 comment (https://graycat.io/projects/tinyga/#comments)

From time to time I come up with potentially cool ways I could implement different algorithms on robots, especially the fun ones like genetic algorithms (http://en.wikipedia.org/wiki/Genetic_algorithm), neural networks (http://en.wikipedia.org/wiki/Neural_network) and particle filters (http://en.wikipedia.org/wiki/Particle_filter) for localization, but I am lacking the tools to do so quickly and efficiently. Somewhere along the way I started toying with the idea of how I could get a fully functional genetic algorithm to fit comfortably on a microcontroller and have it run quickly. In the spirit of wanting to test ideas as quickly and painlessly as possible, the Arduino seemed like the best platform to try this out on. The idea soon turned into the very ambitious plan of creating a collection of Arduino libraries that covers a number of robotics related algorithms implemented as small as possible; for the time being, though, here's TinyGA, my tiny genetic algorithm for the Arduino.

Before I could begin writing TinyGA, I had to decide where I could shrink the algorithm. I chose to favor population size over diversity, so I decided on using unsigned chars for the individuals. While this limits a population to only 256 possible individuals, it allows for the storage of much larger populations than a larger individual would, both in program space as well as in the EEPROM if a population is to be retained when the Arduino shuts off. Once I had that settled, it was mostly just a matter of plugging in the pieces.

The algorithm requires a user-defined fitness function that returns an unsigned char from 1-255 as a fitness rating, or 0 to signal that the correct answer has been reached and the algorithm should stop running. Selection is done using a simple best-half approach, where the individuals with fitness scores above the population's average survive to reproduce. The surviving individuals are then randomly chosen and combined using a single random crossover point. There is also a probability of random mutations of offspring, which is defined in TinyGA.h, along with some other configuration.

The API is quite simple, requiring only the init() and run() functions to run. Here's an example sketch, showing the basic usage of the API, that prompts the user for a number over the serial port then tries to guess it. This sketch compiles to 4208 bytes. (this example is also included in the library)

```
/*
TinyGA_Test.pde - Alexander Hiam - 1/2012
```

```
Uses the TinyGA genetic algorithm library to guess numbers.
*/

#include <TinyGA.h>

// Create TinyGA instance, giving it a fitness function:
TinyGA ga = TinyGA(fitness);

uint8_t number_to_guess, char_num;

void setup() {
  Serial.begin(9600);
  // Must seed random number generator before initializing TinyGA:
  randomSeed(analogRead(A4));
  start();
}

void start() {
  uint8_t i;
  char buffer[10]; // Buffer for serial receiver
  // Initialize with 10 individuals:
  ga.init(10);
  char_num = 0;
  Serial.println("TinyGA TestnEnter a number between 0-255 and TinyGA will try to guess it");
  Serial.flush();
  while (!Serial.available()); // Wait for input
  i = 0;
  while(Serial.available()) {
    buffer[i++] = Serial.read();
    delay(10); // Give plenty of time for next character to arrive
  }
  buffer[i] = ''; // End array
  number_to_guess = atoi(buffer); // Convert to int
  Serial.print("Looking for number: ");
  Serial.println(number_to_guess, DEC);
}

uint8_t fitness(uint8_t individual) {
  /* The fitenss function; takes a guess and returns a score that is larger
     the closer to the correct number it is, and 255 if correct. */
  return 255 - abs(number_to_guess-individual);
}

void loop() {
  uint8_t result;
  if (char_num++ >= 60) {
    // Just keep track of where the status character is and move to a new line if >=60
    Serial.println();
```

```
    char_num = 1;
  }
  Serial.print(".");
  // Run for 100 generations:
  result = ga.run(100);
  if(result) {
    // Run returns 0 unless it got the correct answer, so we know we have it here
    Serial.println();
    Serial.print(result, DEC);
    Serial.println(" - Got it!");
    ga.print(); // Have TinyGA print its current population
    Serial.println();
    start(); // Back to the beginning
  }
}
```

Here's an example of it running and converging to the correct answer relatively quickly:

```
TinyGA Test
Enter a number between 0-255 and TinyGA will try to guess it
Looking for number: 54
.
54 - Got it!
Generation 82:
Population size: 10
51, 51, 51, 51, 51, 54, 51, 51, 51, 51,
```

And here's an example of why genetic algorithms make terrible number guessers:

```
TinyGA Test
Enter a number between 0-255 and TinyGA will try to guess it
Looking for number: 128
.........................................
128 - Got it!
Generation 4306:
Population size: 10
127, 127, 127, 127, 127, 127, 127, 128, 127, 127,
```

4306 evolutions to guess one of 256 possible numbers!

There is also the option of storing a population in the EEPROM, which I have yet to write an example sketch for. The way it works is that the load() function looks for a 4 byte flag that indicates that there is a saved population, loads it and returns 1 if the flag is found, or returns 0 if there is no population. This way initialisation and saving/loading is trivial to include in the sketch by simply writing:

```
TinyGA ga = TinyGA(test);

void setup() {
  if (!ga.load()) ga.init(10);
  ...
```

Then at some point:

```
...
ga.save();
...
```

Then the algorithm may be reset either by calling reset() with no arguments to use the same population size, or reset(new_population_size) may be called to change the size. There's a potential bug here if, for example, a sketch using one population size is loaded onto an Arduino where a previous sketch had saved a different size population. This can be resolved by adding one more line:

```
TinyGA ga = TinyGA(test);

void setup() {
  if (!ga.load()) ga.init(10);
  if (ga.pop_size != 10) ga.reset(10);
  ...
```

This way the old population would be loaded, then its size would be compared to the desired size, and if not equal the population will be reset with the correct size.

- Get TinyGA - click here (https://github.com/alexanderhiam/TinyGA)

And just to be thorough here's the complete API documentation (copied straight from the README file):

```
-TinyGA(uint8_t (*fitness_function)(uint8_t))
  Creates and returns an instance of the TinyGA class. Must be passed
  a user-defined function which takes an unsigned char individual and
  returns an unsigned char fitness rating. Fitnesses should range from
  0-255, where 0 is the worst rating and 255 indicates that the correct
  answer has been found and the algorithm should stop evolving.

-TinyGA.init(uint8_t population_size)
  Initialize a randomly generated population of the given size.

-TinyGA.load(void)
  Attempts to load a population from the EEPROM. Returns 1 if successful,
  0 if there is no population in memory.

-TinyGA.save(void)
  Attempts to save the current population. Returns 1 if successful,
  0 if the population size + EEPROM_OFFSET is greater than EEPROM_SIZE.

-TinyGA.print(void)
  Prints the population size, current generation and the current
  population to Serial. Serial.begin() must be called before hand.

-TinyGA.reset(void)
  Resets to random population with current population size.
  The difference between reset() and init() for resetting the algorithm
  is that reset() will write over TINYGA_KEY in the EEPROM and init()
  won't effect the EEPROM.

-TinyGA.reset(uint8_t new_population_size)
  Resets to random population with new population size.
  Also resets EEPROM.

-TinyGA.run(uint8_t n_generations)
  Runs the algorithm the given ammount of generations. Returns 0
  unless the fitness function has returned 255 indicating a correct
  answer, in which case the individual that scored 255 is returned.
```

(http://reddit.com/submit?

url=https://graycat.io/projects/tinyga/&title=TinyGA) 

(http://www.linkedin.com/shareArticle?

mini=true&url=https://graycat.io/projects/tinyga/) 

(http://www.facebook.com/sharer.php?

u=https://graycat.io/projects/tinyga/)  **(http://twitter.com/share?**

url=https://graycat.io/projects/tinyga/&text=TinyGA%20) 

(https://plus.google.com/share?url=https://graycat.io/projects/tinyga/)

AI ARDUINO MACHINE LEARNING

One response to “TinyGA”

1. buzzz says:
   March 7, 2017 at 1:46 am (https://graycat.io/projects/tinyga/#comment-24299)

   can this be used to optimize pid values of a quadrotor in real time? or while the copter is in flight?

   Reply (https://graycat.io/projects/tinyga/?replytocom=24299#respond)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

AI (https://graycat.io/tag/ai/) Arduino (https://graycat.io/tag/arduino/) bash (https://graycat.io/tag/bash/) Battery (https://graycat.io/tag/battery/) BeagleBone (https://graycat.io/tag/beaglebone/) Breakout (https://graycat.io/tag/breakout/) C (https://graycat.io/tag/c/) device tree (https://graycat.io/tag/device-tree/) GPIO (https://graycat.io/tag/gpio/) I2C (https://graycat.io/tag/i2c/) Image processing (https://graycat.io/tag/image-processing/) LCD (https://graycat.io/tag/lcd/) Linux (https://graycat.io/tag/linux/) LiPoly (https://graycat.io/tag/lipoly/) Machine learning (https://graycat.io/tag/machine-learning/) PCB (https://graycat.io/tag/pcb/) Power management (https://graycat.io/tag/power-management/) PyBBIO (https://graycat.io/tag/pybbio/)

Python (https://graycat.io/tag/python/) serbus (https://graycat.io/tag/serbus/) SPI (https://graycat.io/tag/spi/) thermal imaging (https://graycat.io/tag/thermal-imaging/) ThingSpeak (https://graycat.io/tag/thingspeak/) Web server (https://graycat.io/tag/web-server/)


(https://creativecommons.org/licenses/by-sa/4.0/)