Biogeography-Based Optimization for Robot Controller Tuning

Paul Lozovyy, George Thomas, and Dan Simon
Cleveland State University, Cleveland, Ohio, USA

ABSTRACT

This research involves the development of an engineering test for a newly-developed evolutionary algorithm called biogeography-based optimization (BBO), and also involves the development of a distributed implementation of BBO. The BBO algorithm is based on mathematical models of biogeography, which describe the migration of species between habitats. BBO is the adaptation of the theory of biogeography for the purpose of solving general optimization problems. In this research, BBO is used to tune a proportional-derivative control system for real-world mobile robots. We have shown that BBO can successfully tune the control algorithm of the robots, reducing their tracking error cost function by 65% from nominal values. This chapter focuses on describing the hardware, software and the results that have been obtained by various implementations of BBO.

1. INTRODUCTION

The purpose of engineering is to find feasible or optimal solutions to problems in the world. As engineering challenges have become larger, more complex, and more multidisciplinary, evolutionary algorithms (EAs) have become attractive optimization methods for problems that are not amenable to traditional, analytic algorithms (Rao, 2009). For example, EA research is being used for smart highway systems to solve traffic problems (Baluja, 2001), and for medical diagnosis (Brameier & Banzhaf, 2001). The prevalence of optimization problems has created an explosion in the development of EAs, especially in the last decade.

The idea and the origin of EAs can be traced to the 1950's and the works of Bremermann, Friedberg, Box, and others. The scientific community was mostly unaware of the field of EAs for its first three decades due to lack of computer power. Since then, many EAs have been devised, each being named according to its specific emphasis or the model from which it was derived. These works slowly started to be published in the 1970's. The fundamental works of Holland, Rechenberg, Fogel, and Schwefel began in that decade. The basis for most EAs can be traced to one of three separate and independent approaches: genetic algorithms, evolutionary programming, and evolution or mutation strategies (Bäck, Hammel, & Schwefel, 1997).

Powerful computers became more accessible in the early 1980's, which facilitated more research in EAs. Since 1985, worldwide conferences have been established with workshops focusing specifically on EAs (Bäck, Hammel, & Schwefel, 1997). The advantages of these algorithms relative to traditional optimization approaches are their flexibility and their adaptability. They are designed for solving complex optimization problems. EAs explore a large search space to find optimal solutions using specific evolutionary techniques.

There are several main topics in the forefront of evolutionary robotics. They can be broken down into two groups: off-line tuning and on-line tuning. Off-line tuning means that the EA optimizes the system from

the outside, whereas on-line tuning means that the EA is built into the system. The simplest and most common off-line application of EAs to robotics is the use of EAs to tune robot controller parameters. EAs have been applied this way many times in the past. Robot control tuning can be used as a real-world benchmark test for EAs (Iruthayarajan & Baskar, 2009).

There are several on-line applications of EAs to robotics: on-line controller tuning; automatic controller building (Fleming & Purshouse, 2002); and evolutionary robotics, which is composed of training phase evolution and lifelong adaptation by evolution (Walker, Garrett, & Wilson, 2006). On-line control parameter tuning is useful for situations in which a human cannot tune a controller or compensate for changes in the system. With EA-based tuning software on-line, a robot is able to improve its actions over time without human intervention. Control tuning can occur in real time, or it can be performed before the controller starts, during a less risky training phase (Fleming & Purshouse, 2002). However, the controller may fail while the EA explores the search space. For instance, if an EA is tuning the control parameters in an industrial robot, the EA may choose a poor solution that causes the robot to fail at its task, which can be expensive or dangerous. This is a significant problem with on-line controller tuning.

Automatic controller building is often used in applications where the physical system to be controlled is complicated, hard to model, or hard to define. In these cases an EA can be used to build a control system out of predefined parts. For instance, a robot that performs functions analogous to a human will have many parts, and without a control system optimized for the problem it may not have the ability to complete its tasks (Fleming & Purshouse, 2002).

In evolutionary robotics, training phase evolution and lifelong adaptation by evolution are distinct from each other. Training phase evolution is an initial step in which the EA searches through a wide range of safe conditions before robotic operation begins. Lifelong adaptation by evolution is a method in which an EA continuously optimizes the controller in order to adapt it to changing conditions. Both of these approaches may use on-line parameter tuning or automatic controller building, and both may be performed and applied to the real world, a simulation, or a combination of the two (Fleming & Purshouse, 2002). The training phase approach helps reduce controller failure by narrowing the search space before taking any risky actions, and lifelong adaptation helps the robot adapt to changing conditions throughout its life.

Previous simulation-based research has shown that a new EA called biogeography-based optimization (BBO) works well in a noisy environment. However, simulations cannot fully match real-world complexity, and sometimes they can actually be misleading (Mataric & Cliff, 1996). Simulations are good for preliminary studies but they are not conclusive. This is why we have made the focus of this research the application of BBO to optimize the control parameters of physical robots. The hardware that we use is based on previous work (Churavy, et al., 2008). In our research, these robots were reconstructed to make them suitable for the use of BBO to tune their control parameters. This work is the first implementation of BBO in a physical system.

This work focuses on the application of BBO to robot controller tuning, and on distributed implementations of BBO. In distributed optimization, a central processor is not in control of the optimization process, but rather individual candidate solutions to the optimization problem communicate directly with each other. As the optimization generations proceed, the subsets of communicating candidate solutions change. This obviates the need for a centralized communication structure at the expense of performance.

In Section 2, an overview of standard centralized BBO, and distributed BBO (DBBO) is presented. The robot hardware is described in Section 3. In Section 4 the operation and structure of the robots' controller is explained. Section 5 discusses the robot simulations and experiments, and presents results.

2. BIOGEOGRAPHY-BASED OPTIMIZATION (BBO)

2.1 Centralized BBO

BBO is a new EA based on the migration of species across habitats, a study which is referred to in the biology literature as *biogeography* (Lomolino, Riddle, & Brown, 2009), (Whittaker, 1998). Mathematical models of biogeography were first published in the 1960s, and were first applied to optimization with the introduction of BBO in 2008 (Simon, 2008). BBO terminology uses the following terms from the theory of biogeography: *habitats*, which are analogous to problem solutions; *habitat suitability index* (HSI), which is analogous to fitness in other EAs; and *suitability index variables* (SIVs), which are the independent variables (the domain) of the optimization problem.

Migration is reciprocal—there is both immigration and emigration between habitats. Their respective rates, λ and μ, are determined by the HSI of a habitat; a habitat with a high HSI will have a low immigration rate and a high emigration rate. This is based on natural biogeography in which a habitat with a high HSI already hosts a large number of species and thus cannot accept many immigrants. Conversely, a habitat with a low HSI has a high immigration rate and a low emigration rate. Again, this is based on biogeography in which a habitat with a low HSI has a low number of species and thus is a more likely destination for the immigration of newcomers. Algorithm 1 gives an outline of the BBO algorithm.

For each solution H_i
  For each solution feature s
    Select solution H_i for immigration with probability proportional to λ_i
    If solution H_i is selected then
      Select H_j for emigration with probability proportional to μ_j
      If H_j is selected then
        $H_i(s) \leftarrow H_j(s)$
      end
    end
  next solution feature
  Probabilistically mutate H_i
next solution

Algorithm 1. One generation of a centralized BBO algorithm.

In BBO, migration is an evolutionary operator which plays the same role as recombination in a genetic algorithm (GA). The reciprocal nature of BBO's migration is what makes it different from crossover philosophies. In GAs, parents contribute their genetic information to an offspring and die at the end of the generation (Goldberg, 1989), whereas the solutions in BBO share their information with each other at each generation and continue to survive.

BBO also features the mutation operator, which acts to increase the diversity of a population of solutions. Mutation is also featured in many other EAs. In GAs mutation has traditionally been a background operator. However, in other algorithms, like evolutionary programming, mutation has a more significant role (Fleming & Purshouse, 2002). In BBO, a mutation probability parameter is specified with a range of zero to one, which gives a 0% to 100% chance of mutation occurring for each independent variable in each candidate solution. BBO's default mutation scheme replaces the selected solution features with randomly generated ones within the specified domain.

In order to retain the best solutions, elitism can be applied to BBO. Elitism has been used for a long time in other EAs to prevent the loss of desirable information. An elitism parameter which is coded into BBO specifies the number of solutions that the algorithm considers elite for each generation. At the end of each generation, the poorest solutions are replaced by the previous generation's elite solutions.

2.2 Distributed BBO

In this section, we introduce a distributed implementation of BBO. Distributed BBO is in distinction to the standard BBO algorithm discussed in the previous section, which we refer to as centralized BBO. In BBO, GAs, and many other EAs, solutions represent independent candidate solutions to some optimization problem, but the evolution of these candidate solutions is controlled by a centralized processor. In a situation where an EA's individuals correspond to autonomous entities, such as robots that make their own decisions in real time, the EA may need to be implemented in a distributed manner in order to retain the independence of each individual from the group. This means that the EA algorithm must be performed by the EA individuals rather than by a centralized processor. In our case, the candidate solutions are robot control parameters, and in our initial experiments, the robots were used to generate cost function data for a standard, centralized BBO algorithm as depicted in Algorithm 1. However, for situations that are incompatible with a centralized optimization approach, a distributed implementation must be used. The constraints that may force a distributed implementation include the case when the solution entities cannot always be within proximity of a centralized computer.

We propose a distributed BBO (DBBO) implementation which is inspired by peer-to-peer (P2P) networks (Oram, 2001). In DBBO, the solutions are thought of as peers; for each generation, a few randomly-selected solutions are chosen to be peers, which interact with, and only with, each other. Although there is no simple analog to mutation in P2P networking, mutation is an important part of an EA, so we consider it necessary to include in DBBO. For each generation in DBBO, each peer has a probability of mutation. This results in the number of mutation opportunities being equal to the number of function evaluations, which is the same as mutation in centralized BBO. It is difficult to conceptualize an elitism scheme for DBBO, and since elitism is considered a second-order effect in EAs, we do not implement elitism in DBBO. An outline of a single generation of the DBBO algorithm is shown in Algorithm 2.

The DBBO implementation depicted in Algorithm 2 can still be implemented in a centralized processor. Therefore, depending on its implementation, it may not provide autonomous optimization. However, it still simulates the logic of the interactions that each robot would have with each other in a distributed optimization algorithm. This method of implementing DBBO reduces the complicated communication structure that is required in centralized BBO. Distributed BBO will be useful for systems in which each individual must learn from a subset of the entire population, such as industrial robots on an assembly line that communicate only with a subset of their peers.

```
Initialize n as the number of peers
Randomly select a group of n peers {P_i}
For each peer P_i
   Update each peer's estimate of the best and worst cost in the population:
      P_i(max) ← max_i{ max[ P_i(max), Cost(P_i) ] }
      P_i(min) ← min_i{ min[ P_i(min), Cost(P_i) ] }
next peer
For each peer P_i
   Set each peer's relative immigration and emigration probabilities:
      μ_i ← [ P_i(max) − Cost(P_i) ] / [P_i(max) − P_i(min) ]
      λ_i ← 1 − μ_i
next peer
For each peer P_i
   For each solution feature s
      Select solution P_i for immigration with probability proportional to λ_i
      If solution P_i is selected then
         Select P_j (j = 1, …, n) for emigration with probability proportional to μ_i
         If P_j is selected then
            P_i(s) ← P_j(s)
         end
      end
   next solution feature
next peer
Probabilistically mutate each peer
```

Algorithm 2. One generation of a distributed BBO algorithm.

2.3 Performance Evaluation

We used a set of 13 popular benchmark functions (Simon, 2008) to gauge the performance of DBBO versus BBO. We optimized each of the benchmarks with BBO and three versions of DBBO (with the number of peers set to 2, 4, and 6). For each algorithm, we used a population size of 50, a problem dimension of 20, and a mutation probability of 1% per independent variable per generation. We did not use elitism. We used a function evaluation limit of 500,000. Table 1 shows the mean of the best costs from 50 Monte Carlo simulations of each of these benchmarks for each BBO algorithm. We use the geometric mean to represent the overall performance of each algorithm since the geometric mean provides a better indication than the mean of the order of magnitude of a set of data.

One can see from Table 1 that for the same number of function evaluations, DBBO obtains worse results than BBO. However, DBBO can be used to obtain suboptimal performance results in cases where full communication between population members is not possible or not practical. Figure 1 shows typical centralized and distributed BBO results for the Ackley benchmark.

Benchmark Function	Mean of the Best Costs of 50 Monte Carlo Runs			
	BBO	DBBO (2 peers)	DBBO (4 peers)	DBBO (6 peers)
Ackley	8.44E-03	2.25E-01	2.03E-01	2.16E-01
Fletcher	4.39E+03	8.97E+05	8.58E+05	8.45E+05
Griewank	1.00E+00	4.54E+01	3.92E+01	3.45E+01
Penalty #1	7.13E-04	1.64E+07	1.17E+07	1.17E+07
Penalty #2	1.05E-03	3.94E+07	3.02E+07	3.40E+07
Quartic	8.01E-04	4.12E+00	3.33E+00	3.60E+00
Rastrigin	3.01E-04	1.10E+00	1.58E+00	8.35E+00
Rosenbrock	1.48E+01	6.62E+02	5.84E+02	5.92E+02
Schwefel 1.2	5.71E-02	4.51E+02	4.36E+02	4.90E+02
Schwefel 2.21	8.34E+01	1.87E+04	1.76E+04	1.63E+04
Schwefel 2.22	1.91E-04	6.85E+06	8.31E+01	2.65E+04
Schwefel 2.26	4.98E-01	1.48E+01	1.21E+01	1.09E+01
Sphere	5.70E-04	1.13E+01	1.09E+01	1.11E+01
Geometric Mean	**5.29E-02**	**1.49E+03**	**5.66E+02**	**1.01E+03**

Table 1: Results of 50 Monte Carlo benchmark simulations

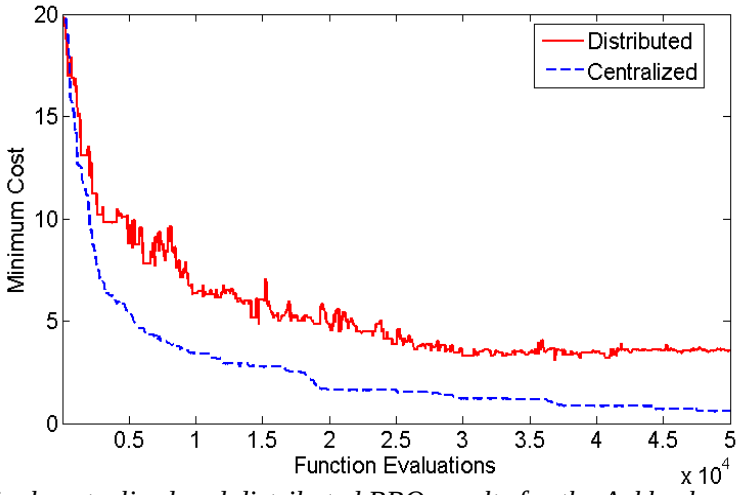

Figure 1. Typical centralized and distributed BBO results for the Ackley benchmark function. The distributed BBO simulation results shown here uses two peers.

In order to further examine the relative performance of BBO and DBBO, we use a *t*-test. The *t*-test was developed by William Sealy Gosset and published under the pseudonym *Student* (Zabell, 2007). The *t*-test can be used to determine the probability that two sample means come from the same distribution. This allows one to determine how similar two sets of data are (Fisher, 1925). We use the *t*-test to compare BBO and DBBO, and also to compare DBBO with different numbers of peers (parameter *n* in Algorithm 2).

With the cost data from the 50 Monte Carlo simulations, we computed *t*-values and their respective probabilities using Microsoft Office Excel®. Table 2 contains the geometric mean of the *t*-test probabilities that each algorithm results in a performance difference that is statistically significant from the other algorithms.

t-test results	Geometric mean of probabilities			
	BBO	DBBO (2 peers)	DBBO (4 peers)	DBBO (6 peers)
BBO	N/A	8.7E-18	3.7E-21	5.2E-18
DBBO (2 peers)	8.7E-18	N/A	1.2E-1	4.8E-2
DBBO (4 peers)	3.7E-21	1.2E-1	N/A	2.8E-1
DBBO (6 peers)	5.2E-18	4.8E-2	2.8E-1	N/A

Table 2: Geometric means of the t-test probabilities of each algorithm versus the others for all 13 benchmarks.

As before, we used the geometric mean to represent the probabilities because it provides an appropriate average of the order of magnitude of a set of data. When using the *t*-test to compare BBO with DBBO, regardless of the number of DBBO peers, the probability value is very low, as seen in the first row and first column of Table 2. This indicates that these data definitely did not come from the same distribution, and thus, BBO provides optimization results that are significantly better than DBBO. This provides a statistical corroboration of our earlier observations on the cost data; our DBBO implementation performs very differently from the original BBO. The probability values from the *t*-tests between DBBO with different numbers of peers are much greater, and range from 5% to 29%. These values are large enough that the differences between the groups of DBBO data may not be statistically significant (Eberhart, Kennedy, & Shi, 2001).

3. HARDWARE

3.1 Assembly

Figure 2 is a photo of one of the robots. Each robot is built for easy access to its components. Electronics such as a microcontroller, voltage regulators, and H-bridges, are mounted and soldered on a two-layer printed circuit board (PCB). The robot electronics are mounted on thin plastic boards separated by aluminum standoffs. Two geared DC motors are attached with brass brackets. Two AA battery packs, which each hold eight rechargeable batteries, supply power to the motors and PCB separately.

A Microchip PIC18F4520 microcontroller is used in this work. Of its peripherals, the pulse width modulation (PWM) modules and hardware universal asynchronous receiver/transmitter (UART) are used for motor control and communication with the base station (via wireless radio) respectively. The base station is a personal computer running MATLAB® which controls the BBO algorithm to optimize the control parameters in each robot.

Two voltage regulators are used to provide a steady 5 V power supply--one for the microcontroller and one for the motors. We use two separate voltage regulators to ensure that inductive loads such as motors cannot draw all the current from the microcontroller and cause it to reset. The SN754410NE quad H-bridge is used to amplify the signal from the microcontroller to drive the motors since the PIC outputs can only provide 25 mA.

Each robot is equipped with a MaxStream 9Xtend radio, which is used to communicate with the base station computer. The base station can transmit commands to each robot, update robot control parameters, and receive robot tracking data. The radio can transmit with an output power of between 1 mW and 1 W as configured by the user, and has a maximum outside range of 40 km. It uses a serial interface that can operate at several baud rates. The radio ($200 US) is the most expensive device on each robot.

Each robot has two digital ultrasonic range finders. The robots use these as inputs to the motion controller. The range finders are mounted on the left side of the robots as shown in Figure 2. Since they are mounted above one of the motors, the ultrasonic noise from the motors interferes with the sensor readings. At first, the cause of the ultrasonic sensor noise was unclear. We tried several things to reduce this noise, such as using styrofoam as sound insulation, and soldering capacitors onto the motors to reduce back EMF, but these methods did not solve the problem. Finally, we raised the range finders further away from the motors, which reduced the noise to a level at which the robots could function consistently.

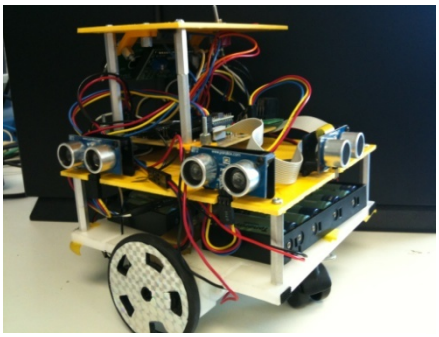

Figure 2: One of the robots.

3.2 Operation

Once the robot is ready to follow the wall, an acknowledgment ping is requested by the base station PC. The ping returns from the robot to the base station to indicate that the robot's radio and hardware is intact and ready for the next command. The next step is to transmit randomly generated K_p and K_d controller parameters wirelessly from the BBO algorithm (which executes in the base station) to the robots and store them in microcontroller EEPROM. Once that step is complete, the command is sent from the base station telling the robot to drive forward using PD control and to collect tracking data with range finders. As the robot runs, it saves one data point (tracking error) every 0.1 seconds to flash program memory. The robot stops after it records 200 data points, which takes 20 seconds. When the robot completes its run, it stops and waits for the next radio command from the base station. The base station then sends a message to the robot requesting tracking data, and the robot responds by sending its tracking data to the base station. We used six robots for our initial experiments, which means that the BBO population was comprised of six individuals. Once the base station has received the tracking data for all six robots, it runs one generation of the BBO algorithm to generate new K_p and K_d parameters for each robot in the next generation. The process then repeats for the next BBO generation.

4. ROBOT CONTROLLER

4.1 PID

PID (proportional, integral, derivative) control is a feedback algorithm that has been in use for almost 100 years and has been applied in virtually every area of control engineering (Araki, 1996). In spite of many advances in control theory, PID is still one of the most widely used controllers today because of its ease of use. The proportional gain is referred to as the *P* gain, the integral gain as the *I* gain, and the derivative gain as the *D* gain. An investigation performed in 1989 in Japan indicated that more than 90% of the controllers used in process industries are PID controllers and their variants (Araki, 1996).

The basic structure of conventional feedback control systems is shown in Figure 3. In this figure, the process *G(s)* is the system being controlled by the controller *F(s)*. The purpose of control is to make the process output variable *y* follow the set-point value *r*. To achieve this purpose, the manipulated control variable *u* is determined by the controller. As an example of a process that needs to be controlled, consider a mobile robot following a wall and trying to maintain a desired distance from the wall by adjusting the speed of its two direct-drive wheels. The distance from the wall is measured by range finders that are mounted on the side of the robot. The process output variable *y* is the distance of the robot from the wall, and the manipulated control variable *u* is the signal determined by the controller. The

disturbance is any factor other than the manipulated control variable that influences the process. Figure 3 assumes that only one disturbance is added to the manipulated variable. However, this does not mean that only one disturbance exists. Most simulations can be adjusted to add external and unpredictable noise to the simulation of a control system. The error *e* is defined by $e = r - y$. The controller *F(s)* is the computational rule (PID in our case) that determines the manipulated control variable *u* based on its input data, which is the error *e*.

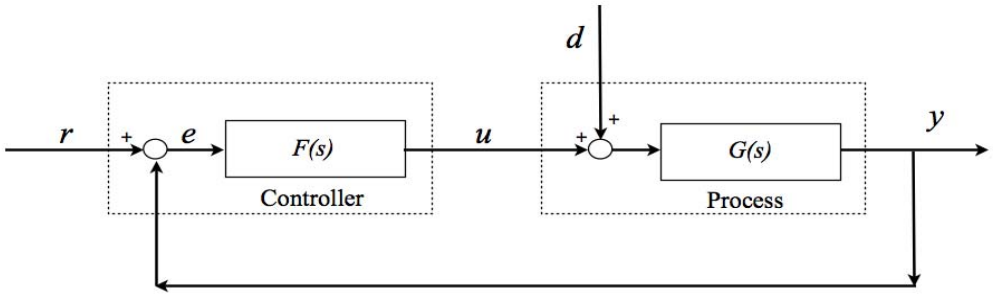

Figure 3: Conventional feedback control system, adapted from (Araki, 1996).

A PID controller has the following transfer function from the tracking error (controller input) to the process input (controller output):

$$F(s) = K_p + \frac{K_i}{s} + K_d s \tag{1}$$

where *s* is the Laplace transform variable. The proportional term K_p determines the amount of control signal which is proportional to the error. A proportional gain that is high results in a large controller output. If the proportional gain is too high, the system becomes unstable. If the proportional gain is too low, it results in a small output response to a large input error, which results in a sluggish controller. A larger K_p value typically gives a faster response.

The integral term K_i accelerates the movement of the process output towards the desired reference value and eliminates the steady-state error that may result from a proportional-only controller. A large K_i parameter eliminates steady state errors more quickly but the tradeoff is larger overshoot, and possibly instability.

The derivative term K_d decelerates the rate of change of the controller output. Therefore, derivative control is used to reduce the amount of overshoot created by the proportional and integral terms and improve controller stability. A large K_d value will decrease overshoot but slow down the response time. One potential danger of derivative control is that measurement noise may be amplified in the differentiation of the error, and this can lead to instability.

4.2 Robot Control Implementation

The function of each of our robots is to follow a wall. The purpose of the robot controller is to maintain a fixed, specific distance from the wall. The angle between the robot orientation and the wall orientation is calculated as follows. First, the difference between the distances read by the two ultrasonic range finders is obtained. This provides one minor leg of a right triangle ($d_1 - d_2$). Second, the distance between the two

sensors themselves is used as another leg of the right triangle (d_b). Finally, the arctangent of the first leg over the second leg provides the desired angle:

$$\theta = \arctan\left(\frac{d_1 - d_2}{d_b}\right) \tag{2}$$

This gives an angle which tells the robot which way it is moving. The tracking error is calculated as

$$e(t) = y_{ref} - \frac{d_1 + d_2}{2}\cos(\theta) \tag{3}$$

where y_{ref} is the desired tracking distance between the robot and the wall. The controller output

$$\Delta u(s) = F(s)e(s) \tag{4}$$

is the change in the square wave PWM duty cycle which is applied to the motors. In this case we use the duty cycle of the PWM waveform to approximate an analog voltage. The PWM value that we apply to each motor is an unsigned 8-bit integer which is translated by the H-bridge driver to ±5 V for application to the motors. Therefore, motor voltage resolution is (10/255) V. The PWM value 128 corresponds to 0 V; values below 128 correspond to negative voltages. The controller output is used to change the motor voltages as follows:

$$u_r = u_{ref} - \frac{1}{2}\Delta u(s) \qquad u_l = u_{ref} + \frac{1}{2}\Delta u(s) \tag{5}$$

where u_r is right wheel PWM value, u_l is left wheel PWM value, and u_{ref} is the nominal duty cycle, which is equal to 192 (3/4 of the maximum duty cycle). The robot's maximum wheel velocity is 0.22 m/s at 60 RPM under normal robotic load, with a wheel diameter of 0.07 m.

Figure 4 depicts a robot tracking toward the reference line, $y = y_{ref}$. The dotted line represents the oscillatory path that the robot takes. This is a qualitatively typical response generated by our robots' PD controllers. This figure also provides a geometric representation of many of the quantities that we discussed in the previous paragraphs.

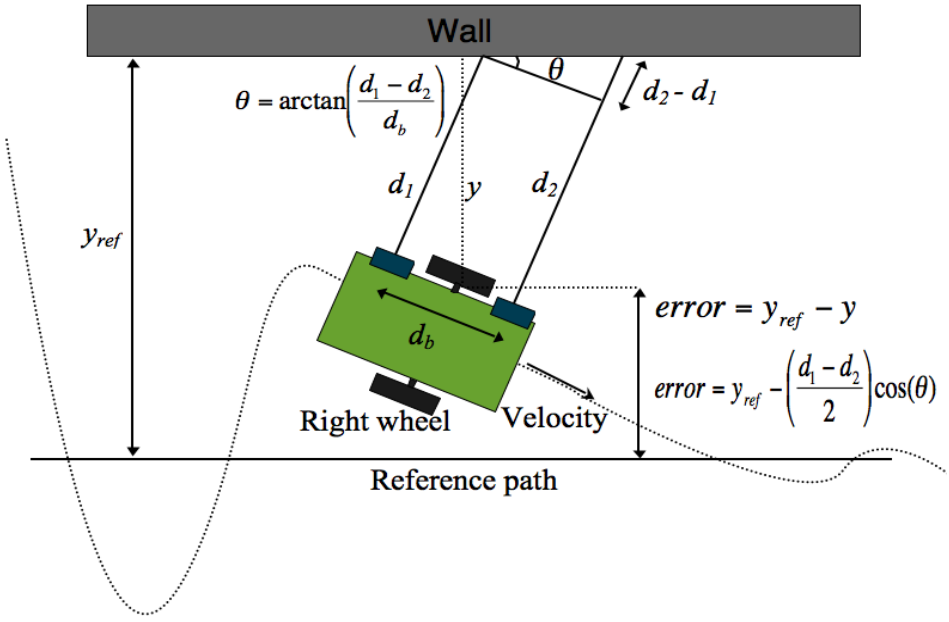

Figure 4: Depiction of the robot's path and related quantities.

The derivative contribution to the control signal is the rate of change of error multiplied by the derivative gain. The rate of change of error is calculated by subtracting the error at the previous time step from the error at the current time step, and then dividing the result by the sampling time. This provides information about how fast the error is changing with respect to time. The rate of change of error depends on the heading angle of the robot. If the robot is moving parallel to the wall then there will be no derivative error, but if the robot is at an angle with respect to the wall, then there will be a nonzero derivative error. The larger the angle, the larger the magnitude of the derivative error, which will result in a larger contribution to the control signal.

In a PID controller there is also an integral term which is used to remove steady-state error by generating a contribution to the control signal which is proportional to the integral of the error. The robots used in this work exhibited little steady-state error, so the added complexity of the integral component was not needed.

5. BBO ROBOTICS EXPERIMENTS

5.1 Experimental Setup

This section details the infrastructure of the proof-of-concept BBO robotics experiment. The base station is a PC connected to a radio module via a serial port and a level-shifting integrated circuit. The base station's software was written in MATLAB. We chose MATLAB for the base station software because the generic BBO code was originally written in MATLAB, which made it easy to adapt to this research. We took advantage of MATLAB's built-in graphical-user-interface development environment (GUIDE) to create a GUI to simplify experimentation, and its serial port capabilities for communication with the robots through a radio module. Figure 5 provides an illustration of the procedural flow of the experiments, which is controlled by the GUI on the PC.

When the user presses the "Run BBO" button on the GUI for the first time, initialization code will execute and the first BBO generation will start; on subsequent presses of the "Run BBO" button, a new generation will start. When the user presses the "Begin Run" button on the GUI, the PC transmits the appropriate command to the robots, and they start tracking the wall and taking distance measurements. For a BBO fitness evaluation of the robot control algorithm in each robot, the robot takes 200 distance measurements, with each measurement taken about 1/10 second apart. After the robot finishes taking 200 distance measurements, execution on the robot's microcontroller enters an idle loop, during which the robot waits until it receives another command from the base station. The user then presses the "Get Data" button on the GUI, which sends an appropriate radio command to the robots that tells them to transmit their tracking distance array to the base station. Finally, when the user presses the "Compute Costs" button on the GUI, the BBO algorithm on the PC computes a cost value for each array of tracking data (i.e., for each robot), concludes the current generation, and displays results for that generation in the GUI.

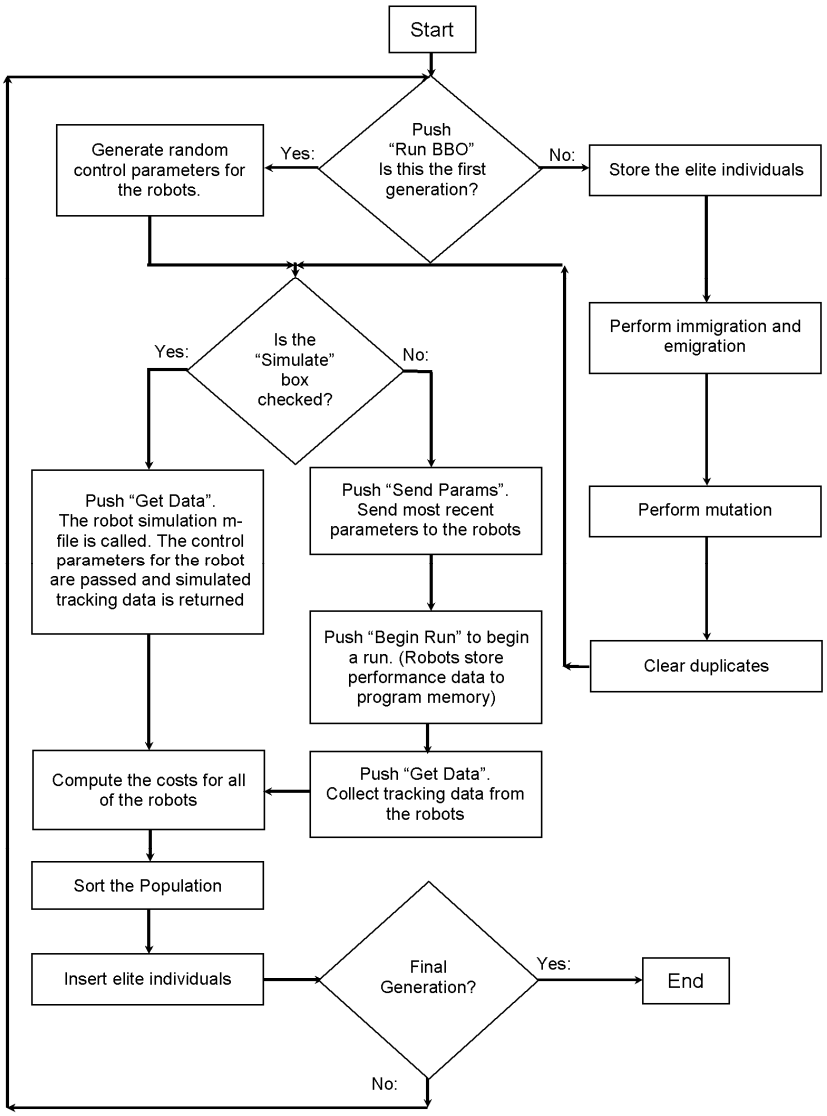

Figure 5: A flow chart of the experimental procedure.

We established a simple communication protocol to transmit commands and data between the robots and the base station. We programmed the robots to transmit only in response to a command from the base station. If the base station communicates with several robots, it will do so sequentially in order to prevent radio interface and in order to allow for the use of a single radio channel. Our protocol is inspired by machine code, so we consider a packet to be a command, which is composed of an 8-bit opcode, a context-specific variable-length operand, and a 16-bit termination word (0x00FF). The variable-length operand with a terminator was chosen over a fixed-length operand for flexibility. Each of the robots' radios has its own unique 16-bit radio address. The base station must switch its radio address (which is done by holding the radio's command pin high and then sending commands to its transmission pin) to that of one of the robots to communicate with it. Radio addresses enable the base station to communicate with one robot at a time, and make sure that the robots do not communicate with each other.

The control parameters, K_p and K_d, are stored in each robot's microcontroller as single precision floating-point values. Since the control parameters are generated in MATLAB on the base station PC, they are inherently represented on the PC in the IEEE 754 floating-point format (IEEE, 1985). We wrote a simple

routine for the robots to convert these floating-point values to Microchip's format, which involves changing the location of the sign bit from to the left of the exponent to between the exponent and the mantissa. Also, the d_b variable (see Figure 4) is different for each of the robots, since each of the robots is slightly different. The d_b, K_p, and K_d values are stored to EEPROM in each robot's microcontroller so that these values can be retained even if the power is interrupted.

All of the trigonometric functions in the robot code are implemented as lookup tables, which use less than 5% of the microcontroller's flash program memory, while the required ANSI C library functions use 10% of flash. Although lookup tables usually take up more space than computational methods, lookup tables are generally much faster than real-time computation (Parhami, 2000). Since we do not need to evaluate any of the trigonometric functions on large domains, we were able to decrease the size of the tables considerably. For example, we only needed to evaluate the inverse tangent of a function in the range [-35°, 35°], since the robots' sensors become very inaccurate at angle magnitudes greater than 35°. The sign of the argument for the inverse tangent function determines only the sign of the final value, not the magnitude. Because of this, we only need to store values for inverse tangent in the range [0°, 35°]. We also needed to compute the cosine of this angle. Since cosine is an even function, we only need to store the values on the domain [0°, 35°].

5.2 Simulation Results

5.2.1 BBO, DBBO, and GA Simulations

For both our simulation and hardware experiments we used a population size of six robots with two control parameters each (proportional and derivative gain). The probability of random mutation was set to the unusually high value of 20% to help increase the diversity of solutions because of the small population size of six robots. No elitism was used in the simulation experiment since no elitism scheme has been developed for DBBO. The ranges of K_p and K_d values for this experiment were [0, 2] and [0, 10] respectively. These ranges were chosen because they are broad and are useful for preliminary study. The optimization cost function was a weighted sum of the controller's rise time and the integral of the magnitude of error:

$$Cost = k_1 \int |e(t)| dt + k_2 r \tag{6}$$

where k_1 and k_2 are weighting constants, $e(t)$ is the tracking error, and r is the controller's rise time, which is the length of time that it takes the robot to reach 95% of the reference tracking distance. For our experiments, k_1 was set to 1 and k_2 was set to 5. These values were chosen to make the orders of magnitude of typical rise time and integral of absolute error values the same, so that they make approximately equal contributions to the costs. The values for k_1 and k_2 are not important, it is the ratio of k_1 to k_2 that is meaningful, since cost is unitless and the absolute range of costs is arbitrary. This weighted sum cost function was chosen over a multiobjective method (Fleming & Purshouse, 2002) to reduce the complexity of the optimization problem.

Before implementing the PD control in our robots, we first wrote a simulation to do some preliminary examination. By approximating the conditions that our real robots would undergo, we were able to find a suitable range of K_p and K_d values that we could use for subsequent work. We also used the robot simulation to compare the performance of centralized BBO, a standard GA, and distributed BBO. Figure 6 depicts the costs versus number of generations for BBO, DBBO with the number of peers set to two and four, and a GA.

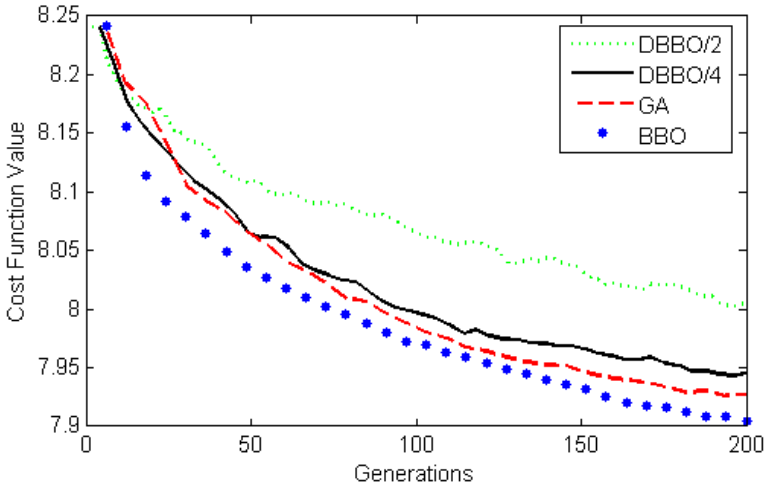

Figure 6. Cost values versus number of generations. The plots show the minimum cost at each generation, averaged over 1000 Monte Carlo simulations.

As seen from Figure 6, BBO outperforms the GA. In addition, centralized BBO performs better than distributed BBO. As the number of communicating peers in DBBO decreases, performance becomes worse. We also collected results for distributed BBO with 6 peers; however, these were almost identical to the results with standard BBO. We suspect that DBBO with the number of peers equal to the population size is operationally identical to standard BBO.

5.2.2 Manual PD Tuning

The first methods developed for tuning PID controllers were manual methods. The simplest of these are ad-hoc methods, in which the parameters are varied without any clear systematic approach. These methods are the most time consuming and rarely produce optimal results, but can be used by people with little to no familiarity with control systems.

One of the most common manual tuning methods for PID is the Ziegler-Nichols method.
In this method, all of the parameters are set to zero, then K_p is increased until the system becomes unstable. This point of instability is called K_{MAX} , and the frequency of oscillation at the point of instability is called f_0. The method then sets K_p to a predetermined fraction of K_{MAX} and sets K_i and K_d as a function of f_0 (Ellis, 2000).This method is useful because it is easy to perform and does not require any extra hardware or software, other than a way of changing the parameters.

We tuned our PD robot controller using a method similar to the Ziegler-Nichols method (Ellis, 2000). K_d is set to zero and K_p is slowly increased to the point of instability. However, instead of setting the other parameters to constants, we held K_p constant at the point of instability and raised K_d until the overshoot was minimized. Performing this method took about 20 minutes using MATLAB. We ran the robot simulation with different K_p values in increments of 0.0005 and different K_d value in increments of 0.05. The parameters we selected were $K_p = 0.039$ and $K_d = 1.4$, and the cost (using the same cost function as the previous simulations) produced by this combination of parameters was 12.18. This cost is 4.28 units higher than the lowest cost produced by BBO (see Figure 6), so BBO produces more optimal results.

Considering that 1000 Monte Carlo simulations with four optimization algorithms depicted in Figure 6 took about an hour of computer time, one run of BBO took about one second (3600 s / 4 / 1000). Therefore, BBO optimized our robot controller about 1000 times faster than manual tuning. These are

rough estimates, but they reflect the fact that automated optimization algorithms can perform much faster than a human operator.

5.3 Experimental Hardware Results

The conditions for our hardware experiments are essentially the same as the simulations (Section 5.2) but with the following exceptions. One elite value was stored between generations, which guaranteed that the best solution was retained in the BBO population. For the physical robots the ranges of K_p and K_d values were [0.01, 0.10] and [4.0, 6.0] respectively. These ranges were chosen with previous simulation and hardware experimentation in mind; the robots were shown to function poorly outside of these ranges.

Figure 7a is an illustration of the robot's improvement over 10 BBO generations, represented by the best and mean costs for each generation. Figure 7b illustrates the decrease in tracking error from the first generation of BBO to the 10th generation of BBO on a single robot. The integral of absolute error decreased from 28,907 in the first generation to 10,204 in the 10th generation, a decrease of 65%.

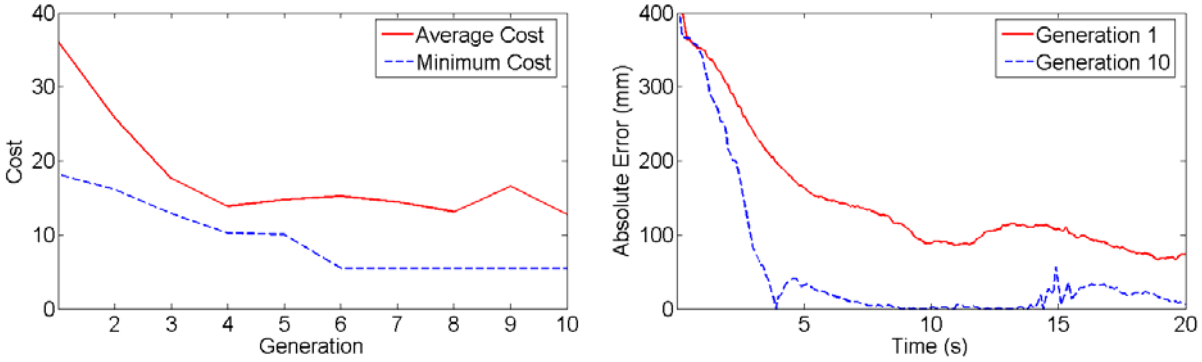

Figure 7. (a) BBO's performance in the proof-of-concept run. (b) Reduction in tracking error after 10 generations of BBO.

In order to be sure that BBO resulted in a statistically significant improvement in the robots' tracking ability as measured by the cost function, we take the standard deviation of the costs from a sample of several runs and compare it to the total improvement in cost from the first generation to the last generation. The total improvement in the best cost is 12.7 (as shown in Figure 7a), the highest standard deviation of the cost out of all of the robots for a single (K_p, K_d) combination is 5.2, and the average of all of the robots' standard deviations of cost for a single (K_p, K_d)combination is 2.6. Because the cost improvement due to BBO is significantly greater than even the highest cost standard deviation for a single robot, we can say with confidence that this improvement is not caused by random variation. Thus, BBO has resulted in a statistically significant improvement in tracking performance.

6. CONCLUSION

Our results have shown that BBO is capable of optimizing robot control parameters in a real experimental system. Further work will include a quantitative analysis of the performance of BBO. This will include running BBO with different settings (e.g., different mutation probabilities and elitism options), or comparing BBO to other EAs for this problem.

Future work will also include reducing the noise in the robot's tracking data. We are pursuing noise reduction by testing analog infrared (IR) range finders for suitability in our research. Since IR range

finders use light instead of sound for range finding, they will not be susceptible to sonic noise created by the motors; however, analog components are inherently more susceptible to electrical noise than digital components. Error due to noise in a digital signal will corrupt the information encoded into it to some extent, but as long as the noise in an analog signal does not cause a change the corresponding digital state, the digital signal's encoded information will remain intact. By experimenting with IR range finders, we will either reduce the tracking noise in the robots, which would mean that the noise in the ultrasonic tracking data has been propagating sonically, or we will narrow down the source of the tracking noise by verifying that the noise is not sonic in nature. It may also be useful to try hardware or software filters for noise reduction.

Also, we want to design new printed circuit boards (PCBs) for our robots. New PCB design is important because the current robot PCBs were designed for ultrasonic range finders and thus cannot be used with the IR range finders that we are proposing for future work. When designing these new PCBs, we will also be able to add hardware filters and other capabilities. The current robot PCBs have headers connected to general purpose input-output pins on the microcontroller, but these are designed for adding specific peripherals like switches and sensors.

The distributed BBO implementation that we discuss in this chapter is a preliminary one. It may be possible to create different implementations of DBBO that produce results as quickly as and more optimally than the distributed BBO presented in this chapter. Also, we have yet to explore different mutation schemes and elitism possibilities for DBBO. Should we mutate every solution or just the selected peers? Should we examine each individual for elitism, or only the peers?

Several modifications to BBO are being examined in other work; these modifications include immigration refusal (Du, Simon, & Ergezer, 2009) and opposition-based learning (Ergezer, Simon, & Du, 2009). It may be fruitful to apply those modifications to the robot control optimization problem. Also, we plan to code an on-line, distributed BBO that executes on the robots themselves instead of on a centralized computer. If this is done, then no centralized computer will be needed.

Some auxiliary material related to this work, including supplemental reading, raw data, and MATLAB and C source code, can be found at http://embeddedlab.csuohio.edu/BBORobotics.

REFERENCES

Araki, M. (1996). Stability Concepts - PID Control. In H. Unbehausen, *Control Systems, Robotics, and Automation.* Kyoto, Japan: Encyclopedia of Life Support Systems (EOLSS).

Bäck, T., Hammel, U., & Schwefel, H.-P. (1997). Evolutionary Computation: Comments on the History and Current State. *IEEE Transactions on Evolutionary Computation , 1* (1), 3-16.

Baluja, S., Sukthankar, R., & Hancock, J. (2001). Prototyping Intelligent Vehicle Modules Using Evolutionary Algorithms. In D. &. Dasgupta, *Evolutionary Algorithms in Engineering Applications* (pp. 241-258). New York: Springer.

Brameier, M., & Banzhaf, W. (2001). A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation , 5*, 17-26.

Churavy, C., Baker, M., Mehta, S., Pradhan, I., Scheidegger, N., Shanfelt, S., et al. (2008). Effective implementation of a mapping swarm of robots. *IEEE Potentials* , pp. 28-33.

Du, D., Simon, D., & Ergezer, M. (2009). Oppositional Biogeography-Based Optimization. *IEEE Conference on Systems, Man, and Cybernetics* (pp. 1035-1040). San Antonio, TX: IEEE.

Eberhart, R. C., Kennedy, J., & Shi, Y. (2001). *Swarm Intelligence.* San Mateo: Morgan Kauffman.

Ellis, G. (2000). *Control System Design Guide*. San Diego: Academic Press.

Ergezer, M., Simon, D., & Du, D. (2009). Oppositional Biogeography-Based Optimization. *IEEE Conference on Systems, Man, and Cybernetics* (pp. 1035-1040). San Antonio, TX: IEEE.

Fisher, R. A. (1925). *Statistical Methods for Research Workers.* Edinburgh: Oliver and Boyd.

Fleming, P. J., & Purshouse, R. C. (2002). Evolutionary algorithms in control systems engineering: a survey. *Control Engineering Practice* , 1223–1241.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

Hedar, A.-R., & Fukushima, M. (2003). Minimizing multimodal functions by simplex coding genetic algorithm. *Optimization Methods & Software , 18* (3), 265-282.

IEEE. (1985). IEEE Standard 754-1985 for Binary Floating-point Arithmetic. *Reprinted in SIGPLAN , 22* (2), 9-25.

Iruthayarajan, M. W., & Baskar, S. (2009). Evolutionary algorithms based design of multivariable PID controller. *Expert Systems with Applications* , 9159–9167.

Lomolino, M. V., Riddle, B. R., & Brown, J. H. (2009). *Biogeography.* Sunderland, MA: Sinauer Associates.

Mataric, M., & Cliff, D. (1996). Challenges for evolving controllers for physical robots. *Robotics and Autonomous Systems* , 67–83.

Oram, A. (2001). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* Sebastopol, CA: O'Reilly Media.

Parhami, B. (2000). *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford, NY: Oxford University Press.

Rao, S. (2009). *Engineering Optimization: Theory and Practice.* New York: Wiley.

Simon, D. J. (2008). Evolutionary Biogeography-Based Optimization. *IEEE Transactions on Evolutionary Computation , 12* (6), 702-713.

Walker, J. H., Garrett, S. M., & Wilson, M. S. (2006). The Balance Between Initial Training and Lifelong Adaptation in Evolving Robot Controllers. *IEEE Transactions on Systems, Man, and Cybernetics , 36* (2), 423-432.

Whittaker, R. (1998). *Island Biogeography.* New York: Oxford University Press.

Zabell, S. L. (2007). On Student's 1908 Article 'The Probable Error of a Mean'. *Journal of the American Statistical Association , 103* (481), 1-7.

ADDITIONAL READING SECTION

Burbidge, R., Walker, J. H., & Wilson, M. S. (2009). Grammatical evolution of a robot controller. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 357-362). St. Louis, MO: IEEE.

Camponogara, E., & de Oliveira, L. B. (2009). Distributed optimization for model predictive control of linear-dynamic networks. *IEEE Transactions on Systems, Man and Cybernetics , 39* (6), 1331-1338.

Camponogara, E., Scherer, H. F., & Moura, L. V. (2009). Distributed optimization for predictive control with input and state constraints: preliminary theory and application to urban traffic control. *Proceedings*

of the 2009 IEEE International Conference on Systems, Man and Cybernetics, 11-14. Piscataway, NJ: IEEE.

Cantu-Paz, E. (2003). *Genetic and Evolutionary Computation - GECCO 2003 Part II* (1st Edition ed.). New York, NY: Springer.

Coello, C. A., Christiansen, A. D., & Aguirre, A. H. (1998). Using a new GA-based multiobjective optimization technique for the design of robot arms. *Robotica , 16*, 401-414.

Craig, J. J. (2004). *Introduction to Robotics: Mechanics and Control* (3rd Edition ed.). Upper Saddle River, NJ: Prentice Hall.

Guivant, J., & Nebot, E. (2001). Optimization of the Simultaneous Localization and Map Building Algorithm for Real Time Implementation. *IEEE Transactions on Robotics and Automation , 17* (3), 242.

Choset, H. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations.* Cambridge, MA: The MIT Press.

Jazar, R. N. (2010). *Theory of Applied Robotics: Kinematics, Dynamics, and Control* (2nd Edition ed.). New York, NY: Springer.

Jian, Z., & Ai-Ping, L. (2009). Genetic algorithm for robot workcell layout problem. *Proceedings of the 2009 WRI World Congress on Software Engineering* (pp. 19-21). Xiamen, China: IEEE.

Buford, J., Yu, H., Lua, E. K. (2008). *P2P Networking and Applications.* San Fransisco, CA: Morgan Kaufmann.

Kuntschke, R. (2008). *Network-Aware Optimization in Distributed Data Stream Systems.* Saarbrücken, Germany: VDM Verlag.

Malanowski K.. (1996). *Modelling and Optimization of Disributed Paramenter Systems* (1st Edition ed.). New York, NY: Springer.

Nolfi, S., Floreano, D. (2004). *Evolutionary Robotics: the Biology, Intelligence, and Technology of Self-Organizing Machines.* Cambridge, MA: The MIT Press.

Roy, P., Ghoshal, S., & Thakur, S. (2010). Biogeography-based optimization for economic load dispatch problems. *Electric Power Components and Systems , 38* (2), 166-181.

Thrun, S., Burgard, W. (2005). *Probabilistic Robotics.* Cambridge, MA: The MIT Press.

Siegwart, R. (2004). *Introduction to Autonomous Mobile Robots* (Illustrated Edition ed.). Cambridge, MA: The MIT Press.

Sundaram, R. K. (1996). *A First Course in Optimization Theory.* New York, NY: Cambridge University Press.

Brueckner, S. A., Parunak, H. V. D. (2005). *Engineering Self-Organizing Systems: Methodologies and Applications*. New York, NY: Springer.

Takashi, E. G. (2001). *Evolutionary Robotics: From Intelligent Robotics to Artificial Life.* New York, NY: Springer.

Tan, L., & Guo, L. (2009). Quantum and biogeography based optimization for a class of combinatorial optimization. *ACM/SIGEVO Summit on Genetic and Evolutionary Computation* (pp. 969-972). New York, NY: ACM.

Wan, Y., Wang, G., Ji, S., & Liu, J. (2008). A survey on the parallel robot optimization. 2008 Second International Symposium on Intelligent Information Technology Application (pp. 655-659). Shanghai, China: IEEE.

Wang, L, (2006). *Evolutionary Robotics: From Algorithms to Implementations.* Hackensack, NJ: World Scientific Publishing Company.

Watanabe, K., Hashem, M. M. A. (2004) *Evolutionary Computations*. New York, NY: Springer.

Yu, X., Gen, M. (2010). *Introduction to Evolutionary Algorithms*. New York, NY: Springer.

KEY TERMS & DEFINITIONS

EA: An evolutionary algorithm (EA) is a population-based optimization approach that includes candidate solutions which evolve over time to produce an optimal solution to some problem.

BBO: Biogeography-based optimization (BBO) is an evolutionary algorithm based on the migration of species between islands.

T-test: A statistical test that provides information on the relationship between the means of two sets of data.

DBBO: Distributed BBO is an algorithm that is distributed among the individuals. A centralized processor is not used in DBBO.

Centralized BBO: A BBO implementation in which a single computer controls the algorithm.

Peers: A randomly chosen subset of the DBBO population which interacts during a given generation for the purpose of migration.

Migration: The sharing of solution information among BBO individuals.

PID: Proportional/integral/derivative (PID) control is a widely used feedback control method in which the control signal is proportional to the tracking error, its integral, and its derivative.