Exploring and Evolving Process-oriented Control for Real and Virtual Fire Fighting Robots

Kathryn Hardey
Centenary College
Shreveport, LA 71104
khardey@my.centenary.edu

Eren Corapcioglu
Centenary College
Shreveport, LA 71104
ecorapci@my.centenary.edu

Molly Mattis
Allegheny College
Meadville, PA 16335
mattism@allegheny.edu

Mark Goadrich
Centenary College
Shreveport, LA 71104
mgoadric@centenary.edu

Matthew Jadud
Allegheny College
Meadville, PA 16335
matthew.c@jadud.com

ABSTRACT

Current research in evolutionary robotics is largely focused on creating controllers by either evolving neural networks or refining genetic programs based on grammar trees. We propose the use of the dataflow languages for the construction of effective robotic controllers and the evolution of new controllers using genetic programming techniques. These languages have the advantages of being built on concurrent execution frameworks that lend themselves to formal verification along with being visualized as a dataflow graph. In this paper, we compare and contrast the development and subsequent evolution of one such process-oriented control algorithm. Our control software was built from composable, communicating processes executing in parallel, and we tested our solution in an annual fire-fighting robotics competition. Subsequently, we evolved new controllers in a virtual simulation of this parallel dataflow domain, and in doing so discovered and quantified more efficient solutions. This research demonstrates the effectiveness of using process networks as the basis for evolutionary robotics.

Categories and Subject Descriptors

I.2.9 [**Robotics**]: Autonomous vehicles; D.1.3 [**Concurrent Programming**]: Parallel Programming

General Terms

Experimentation, Algorithms

Keywords

Evolutionary Robotics, Concurreny, CSP, occam-pi

1. INTRODUCTION

Evolutionary robotics explores the use of genetic algorithms to learn controllers for autonomous robots. Recent surveys show that neural networks account for the basis of approximately 40% of such controllers, while genetic programming is used in another 30% [10]. Progress has been made in the learning of simple behaviors such as locomotion and obstacle avoidance [12], phototaxis [16], and searching and foraging [13]. As research has moved to attempting tasks involving multiple behaviors, tension has developed between learning controllers for these complex tasks and limiting the amount of *a priori* background knowledge used in the learning frameworks.

We show here how process networks written in dataflow languages can be used as a basis for evolutionary robotics, where the level of controller complexity can be tuned. Our work examines the creation of one such set of processes and their subsequent evolution to improve the performance of a robot within a task involving coordination of multiple simultaneous behaviors. We first describe process networks, and then contrast them with neural networks. We next demonstrate the use of process networks by detailing our creation of a real world controller for a robotic fire fighting competition. This task is then abstracted into a virtual simulation, where we employed an evolutionary algorithm using the basic control processes to improve our performance on the locomotion and object avoidance aspects of the fire fighting task. Finally, we discuss the results of this experiment, related work in evolutionary robotics, and directions for future work.

1.1 Dataflow Languages

A robot is continuously and simultaneously engaged in three activities: it reads inputs, computes over that data, and controls motors and other actuators. By using a parallel instead of a sequential language, these can be written as separate processes that execute concurrently and flow data from one process to the next. Recent research has explored the use of inherently parallel programming languages for writing robotic control systems [2, 7, 17].

Parallel, process-oriented dataflow languages such as occam-pi, Erlang, and Google's Go are based on the Communicating Sequential Processes (CSP) formalism described by Hoare [6]. Programs are written using two main structures, namely processes and channels. The processes involved are self-contained sequences of computation, typically looping forever. These processes only share data (or state) via communication over well-defined channels. This allows the compiler and/or runtime environment of the language to guarantee properties like freedom from deadlock or the absence of race hazards, thus greatly simplifying the process of writing concurrent programs. Another convenient aspect of these languages is that programs can be completely represented as a human-readable network that describes the flow of data from one independent process to another.

1.2 From Neurons to Brains

In comparison to evolutionary systems based on neural networks, dataflow networks can be composed of higher-order processes that have two distinct benefits. First, the complexity of individual nodes can be controlled and augmented beyond a simple activation threshold, and second, all stages of the evolution are understandable by humans for inspection and refinement.

For example, success in a complex locomotion domain could involve developing one set of behaviors to react in an XOR fashion to two input sensors. A neural network would need to refine the connections between individual activation nodes across multiple layers. In a process network, the connections in the neural network can be recreated as simple threshold processes connected with channels of communication, or (alternatively) the entire XOR function can be created as one process, to be connected as a building block in a larger process network of other programmed functions.

Because processes and channels are the primary abstraction mechanisms in dataflow languages, processes become the genetic material upon which evolution can act. In this manner, we can incorporate varying amounts of prior knowledge into the genome, from processes as simple as neurons, to more complicated circuit functions, and even complete programs for specific behaviors arranged in a complex process hierarchy.

2. THE PHYSICAL PROVING GROUND

Each year Trinity College hosts a fire fighting home robot contest (TCFFHRC) [1]. The purpose of this contest is to navigate an arena, locate a candle, and extinguish it. A sample arena is shown in Figure 1.

At the start of a trial, the robot is placed on the home circle. The robot then has a five minute time limit to extinguish the candle placed in one of the four rooms. A successful robot must be able to coordinate and execute the behaviors of locomotion, object avoidance, and phototaxis, along with some method of extinguishing the candle. Points are awarded based on the duration of the trial and whether the contestant chooses to participate in any optional point multiplier challenges. Our team participated in the Senior Division of the April 2011 contest using a system based on the Arduino (for hardware) and occam-pi (for software).

2.1 Hardware

Our hardware was a low-cost, differential-drive platform with a trailing caster. We used three infrared range sensors for obstacle sensing, three (tuned) infrared diodes for flame detection, a fan for extinguishing flame, and the commonly

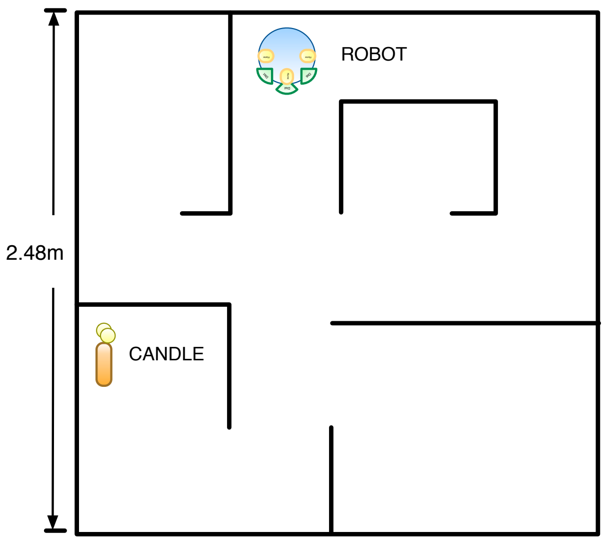


Figure 1: The TCFFHRC Competition Arena

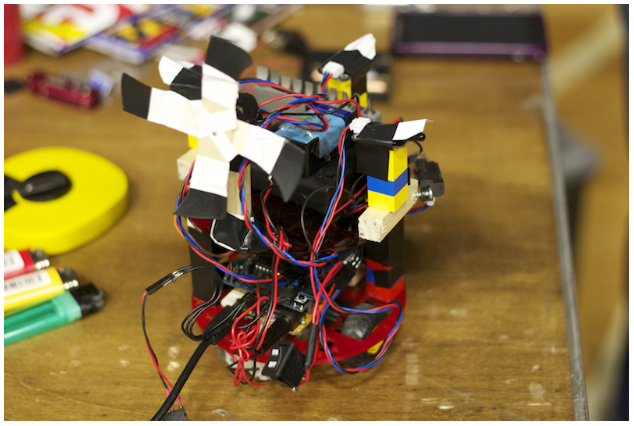


Figure 2: The robot used in competition

available Arduino platform to handle sensor processing and computation. Figure 2 shows our competition robot from the rear, and Figure 3 diagrams the placement of the sensors on the robot.

The microcontroller board we used to control our robot was an Arduino Duemilanove[1] with an Adafruit Motor Shield[2] attached. The Arudino is open hardware, and uses a 16MHz processor with 32K of internal flash for program storage and 2K of RAM. The Arduino can read digital and analog inputs (via a built-in, 10-bit ADC) and produce output through these same pins.

2.2 Software

Our controller was hand-coded in occam-pi. Our process network is shown in Figure 4, and the full source to our controller is open and available for download[3]. We have

[1]http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove
[2]http://www.ladyada.net/make/mshield/index.html
[3]http://code.google.com/p/occam-rescue/

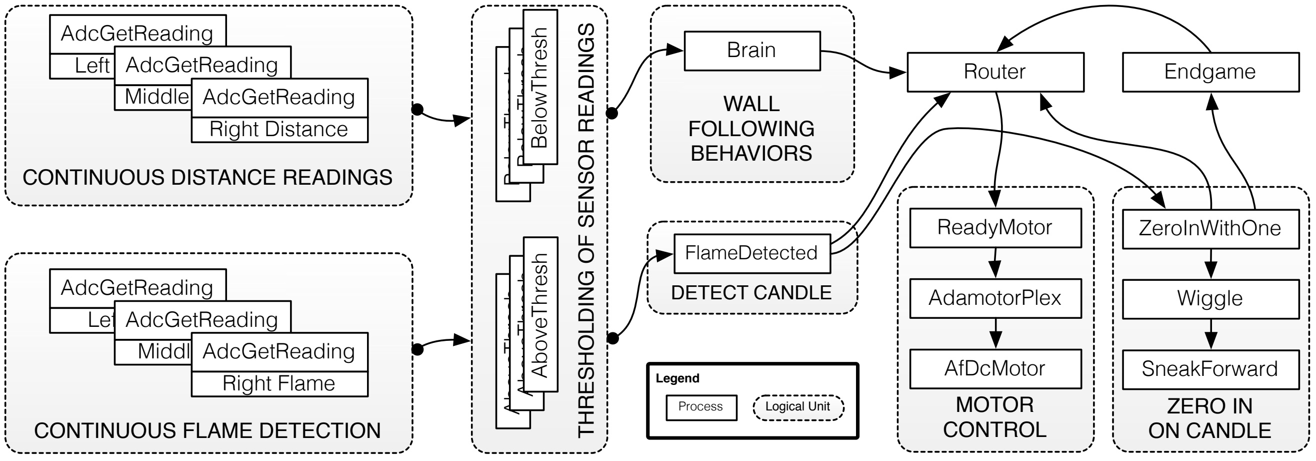


Figure 4: occam-pi process network for physical robot

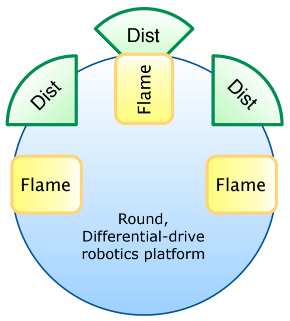


Figure 3: Placement of sensors on the robot

two main modes: a movement and wall-avoidance mode for wandering the arena, and a flame-detected mode for extinguishing the candle.

The Brain of the robot initializes the motor shield and begins giving the motors commands based on signals from multiple AboveThresh and BelowThresh processes. The threshold processes continuously take readings from the sensors using the AdcGetReading process and activate the correct motors such that the robot moves through the arena and avoids walls.

When one of the flame sensors detects a high reading, AboveThresh tells FlameDetected that it should send out signals, which lets other processes know the robot is switching into phototaxis mode. The Router manages this switch and begins listening to the ZeroInWithOne process, which closes in on the candle using the front flame sensor by moving toward the direction where the sensor detects the most light.

Once ZeroInWithOne gets a high enough reading, it signals the EndGame process which turns the robot 180 degrees and starts a fan on the top of the robot to hopefully extinguish the flame.

2.3 Evaluation

In the TCFFHRC, each contestant is given two chances to find and extinguish the candle. If the robot is able to accomplish this in at least one trial, a third trial is awarded. For our first two trials, our robot was able to find the candle. In the second trial, our robot extinguished the candle successfully, granting us a third trial.

Our robot visited most of the arena by following the right wall. However, by using this technique the upper right island room was inaccessible. Thus, if a candle was placed in that room, the robot could not locate and extinguish it. At the contest, the candle was placed in the island room for our third trial. Our flame sensors were unable to see the candle and our robot bypassed the room. Overall, our robot placed 15th out of 41 contestants in the Senior Division, a respectable showing for first-time contestants.

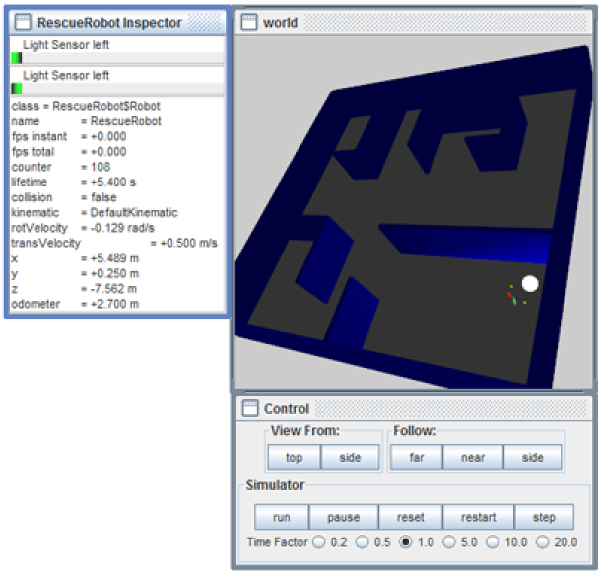
3. THE VIRTUAL PROVING GROUND

To demonstrate the utility of using process networks as a foundation for evolutionary robotics, we implemented each of the four possible TCFFHRC arenas in Simbad, an open-source and easily customizable Java 3D simulator [5]. Figure 5 is an example of an arena with one possible candle placement. Our simulations randomly chose one arena along with a random candle location for each run.

Our virtual robot in Simbad closely followed the specifications of our physical robot used in competition. The motors in Simbad simulated a two-wheeled robot, with the ability to control the speed of each motor independently, and the light and distance sensors in Simbad were altered to closely match the expected range of input from the physical sensors.

3.1 Communicating Processes in Java

As Simbad is written in Java, we chose to express our process network controller using Communicating Sequential Processes for Java (JCSP) [18]. Using this library, we created parallel processes for our virtual robots using message passing as opposed to the usual threads and shared memory. This enabled a near one-to-one mapping of our original control code into the simulator. Figure 6 shows the *wallbrain*



```
class = RescueRobot$Robot
name          = RescueRobot
fps instant   = +0.000
fps total     = +0.000
counter       = 108
lifetime      = +5.400 s
collision     = false
kinematic     = DefaultKinematic
rotVelocity   = -0.129 rad/s
transVelocity            = +0.500 m/s
x             = +5.489 m
y             = +0.250 m
z             = -7.562 m
odometer      = +2.700 m
```

Time Factor ○ 0.2 ○ 0.5 [✓ ● 1.0] ○ 5.0 ○ 10.0 ○ 20.0

Figure 5: A sample arena in Simbad

portion of our original occam-pi robot, now programmed using a JCSP process network to move and avoid the walls within the arena.

In JCSP, we recreated the AboveThresh and BelowThresh processes, which listened to a distance or light sensor and compared the sensor readings to a fixed threshold level. All processes send their signals to an AltControl process, which listens to and prioritizes the signals. This process takes the place of and abstracts our Brain and FlameDetected processes in the previous network so that our networks can be more flexible. Each AltControl listens for the threshold process signals and passes signals along to the next process stage.

In our setup, AltControl passes an incoming signal to the respective MotorControl process. The MotorControl process stores the direction and speed for the left and right motor along with the duration for the movement. All MotorControl processes feed into a MotorFinal process that guarantees motors are not controlled simultaneously from separate sources. If the MotorFinal has not received any signals, it will perform a default behavior of moving forward for a brief time interval.

4. EVOLVING PROCESS NETWORKS

To test the feasibility of evolving process networks, we focused on evolving networks capable of performing the locomotion and obstacle avoidance elements of the TCFFHRC task.

4.1 Initialization

Our population was initialized to twelve randomly generated robots. Each robot in the population was given one AltControl process. The AltControl is then augmented with a random number of *information pathways*. An information pathway can be seen as the flow of data starting from a sensor, and passing through a threshold process to a particular MotorControl (gray arrows, Figures 6, 7, and 8).

For each of these pathways, a threshold process was ran-

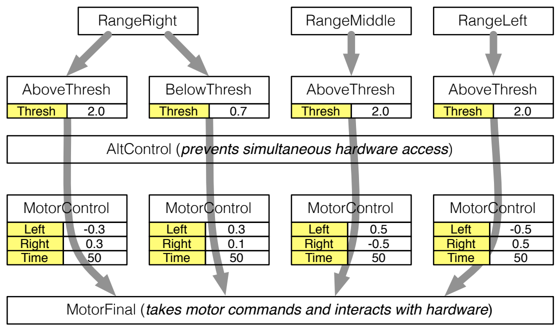


Figure 6: The wallbrain portion of our competition controller, reformulated for execution in Simbad

domly chosen by creating an AboveThresh or BelowThresh process with a valid, random threshold level. Each threshold was then connected to a particular sensor from which to receive readings; in these experiments the possible sensors were the range sensors. In this way, sensors are allowed to send information to more than one threshold, which can allow for complex behaviors to emerge.

Next, MotorControl processes were created for each threshold. Random values are chosen for the left and right motors; negative values turn the wheels backwards, while positive values spin the wheels forward. The time interval for movement was selected from between ten and 99 milliseconds. This range was chosen based on our experience with the physical robot. Finally, a MotorFinal process is created and the MotorControl channels are connected to complete the flow of data.

4.2 Evaluation

The contest specifies that a robot has five minutes to complete the task of extinguishing the candle, with the score based on the time used in the arena. We found this to be an incomplete evaluation metric, where many robots would succeed in identification of the candle but fail to extinguish it properly, yet receive the same score as a robot that failed to move.

To increase the learnability of the task, we wanted to track the effectiveness of the robot scanning the whole arena. To calculate this, we overlaid a 20x20 grid on the arena, and recorded how many squares were left unseen at the end of the run as a percentage of the total number of squares. This forms the basis for our fitness function, and refer to it as the unseen grid.

Our uniform, unseen grid was improved by adding additional emphasis when the robot visited a room as opposed to a hallway, as candles will always be placed in rooms. The unconnected room in the upper-right corner of the arena is weighted even higher, since this room could not be easily found with our hand-crafted solution. Therefore, squares in a hallway are assigned one point, squares in the three wall-connected rooms earn two points, and squares in the unconnected room earn three points. Our weighted score is then the sum of the weights on unseen squares divided by the total of all the weighted squares, with the goal being to minimize this score.

4.3 Selection

We used tournament selection for all of our experiments. This incorporated the absolute ranking of the robots, as the *unseen* metrics of the robots were tightly clustered. We performed selection twelve times to keep our population size constant, discarding the original population completely.

4.4 Mutation

Our mutation methods were focused on altering individual processes or the flow of information between processes. For a given run of the evolutionary algorithm, we chose a mutation rate, and each mutation method in a process was performed with that probability. This mutation rate is applied to each individual in the selected population.

Above and Below Thresholds

Alter the level for the threshold

Listen to a different sensor

Toggle between Above and Below threshold types

AltControl

Add information pathway . Create a new Threshold, MotorControl, and necessary channels to connect to a sensor, the AltControl, and MotorFinal processes.

Delete information pathway . If there is currently more than one pathway, delete a selected channel and pathway connected to it.

Switch pathways . Select two Thresholds and switch which MotorControl process they activate by switching their channels.

MotorControl

Re-selected direction values for left and/or right motor.

Reset time interval for movement to another number between ten and 99.

Figure 7 provides an example of a brain being mutated using several of the above methods. In this example, each of the Threshold mutation methods were executed. The leftmost BelowThresh **altered** its value from 1.5 to 0.3. The rightmost threshold was mutated to **listen** to the RangeMiddle sensor rather than the original RangeLeft sensor. The leftmost threshold originally was a BelowThresh and was **toggled** to AboveThresh.

This mutation session did not **Add** nor **Delete** a pathway from the AltControl process. However, the middlemost and rightmost thresholds were **switched** to different MotorControl processes. The rightmost MotorControl's right **direction** was updated from -0.8 in the original brain to 0.7 in the mutated brain. The same MotorControl also mutated its **time interval** from 56 milliseconds to 23 milliseconds.

4.5 Crossover

To perform crossover, all individuals in our selected population were randomly paired, and then children were created by swapping all of the information pathways connected to one randomly chosen sensor. If one parent did not have a pathway attached to the chosen sensor, the sensor in the other parent now has no pathways. In cases when this leads to an empty robot, it was replaced with a new randomly generated robot.

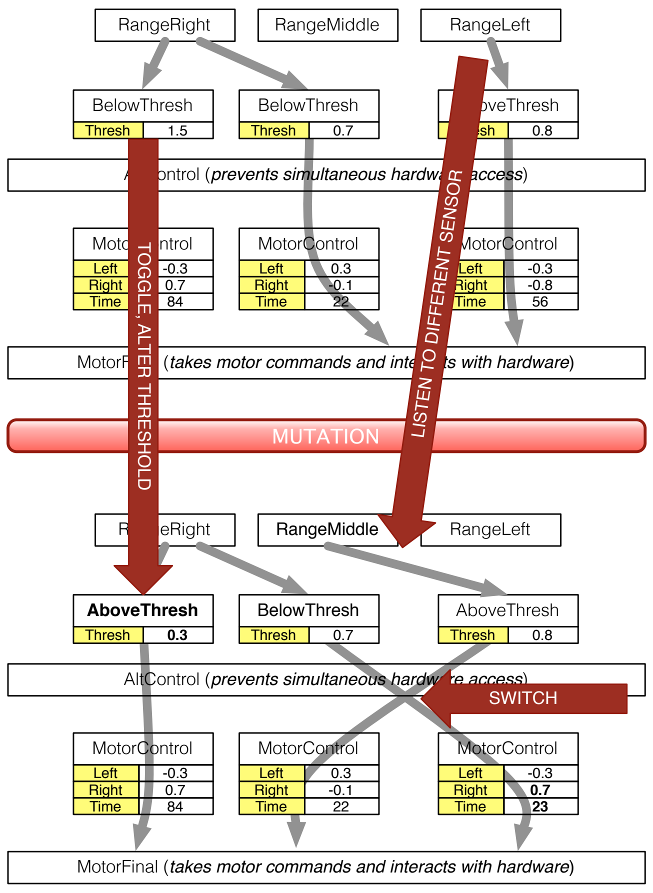


Figure 7: Example of mutation methods on sample brain

Figure 8 demonstrates our crossover method applied to two sample parent brains. In this example, the RangeMiddle sensor has been selected for crossover. The first parent lacks a pathway attached to this sensor while the second parent has two pathways attached to RangeMiddle. This leads to two children, the child on the left having four pathways and the child on the right remaining with one.

5. EXPERIMENTS

Our experiments explored the effect of different mutation rates on our process networks, and compared the best results to our original process network written for the TCFFHRC. We investigated two different ways to specify the amount of mutation applied to each robot in the population:

1. We first set a mutation rate, and then applied the mutation methods with this probability for each process. We chose to vary the mutation rate from five to 45 percent, increasing in increments of five. A robot could potentially be mutated in multiple ways over one generation.

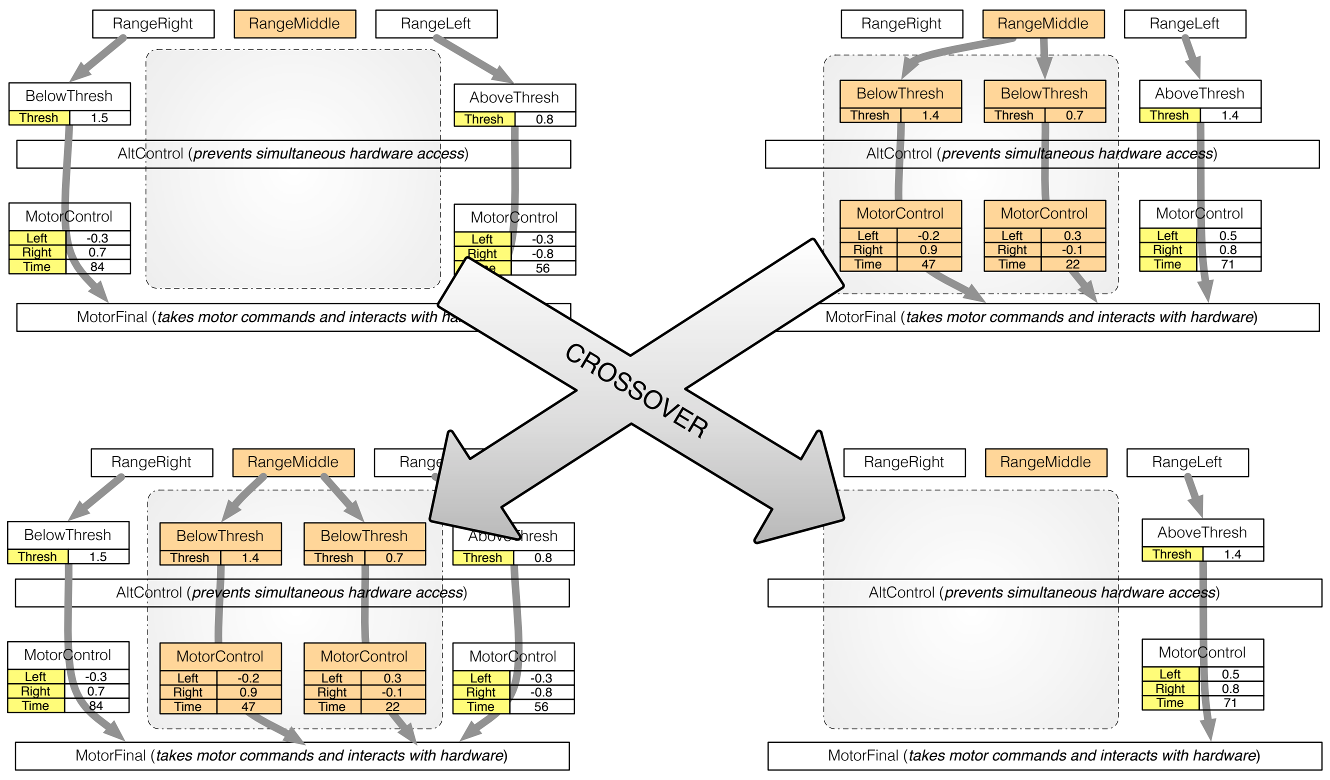


Figure 8: Example of crossover method on two brains, where the chosen swap involves the RangeMiddle sensor

2. To test the effects of this multiple mutation, we also ran experiments allowing only one randomly chosen mutation per robot.

Throughout all our experiments, we used a population of twelve robots and ran for twenty generations, repeated and averaged over five runs. The effectiveness of a given robot brain was evaluated using the the weighted *unseen* evaluation metric (described in Section 4.2). Due to limitations in Simbad, we were forced to run our experiments in real-time. Therefore, our experiment here used a time limit of 60 seconds, rather than the five minute limit for a trial at TCFFHRC. We also declared a trial complete when the robot collided with an arena wall, because our physical robot lacked bump sensors—a real-world crash would have resulted in failure in the contest.

6. RESULTS

Our experiments resulted in process networks that were more effective at covering more of the competition space than the original human-devised solution. The most effective solutions (covering the largest portion of the competition arena) evolved both obstacle avoidance (wall following) and locomotion (wiggling). Both behaviors are consistent with our original solution for the TCFFHRC task.

6.1 Locomotion and Obstacle Avoidance

We found that lower mutation rates produced robots that developed effective locomotion strategies faster. Figure 9 shows the average scores of all the mutation rates with tournament selection. By inspection, low mutation rates ($<$ 15% mutation rate per generation) appear to be different from high mutation rates ($>$ 15% mutation rate per generation). A one-way analysis of variance between results based on mutation rate per generation indicated a significant difference (p-value of 0.0000013), and pairwise t-tests between each experimental condition confirmed a significant difference between a robot evolved using 5% mutation rate per generation and all other mutation rates.

Notably, the *one mutation per brain* method was initially similar to the 5% mutation rate performance, but eventually degraded over time and did not produce statistically significant results: some mutation (but not too much) was good. With higher mutation, any progress toward finding a successful network is destroyed each generation.

6.2 Improved Arena Coverage

As part of this work, we translated our occam-pi contest solution into JCSP (Figure 6). Our contest solution achieved a score of 0.85 in Simbad using our weighted unseen evaluation metric. Guided by our previous results, we compared the best score (lowest weighted unseen metric) from each of the five runs of the brains evolved with five mutations/generation to our hand-crafted solution.

Figure 10 shows that, after twenty generations, the best (evolved) solutions do as much as 10% better than our hand-crafted solution. Controllers that achieve these scores either (1) moved at a faster pace (spending less time wiggling and

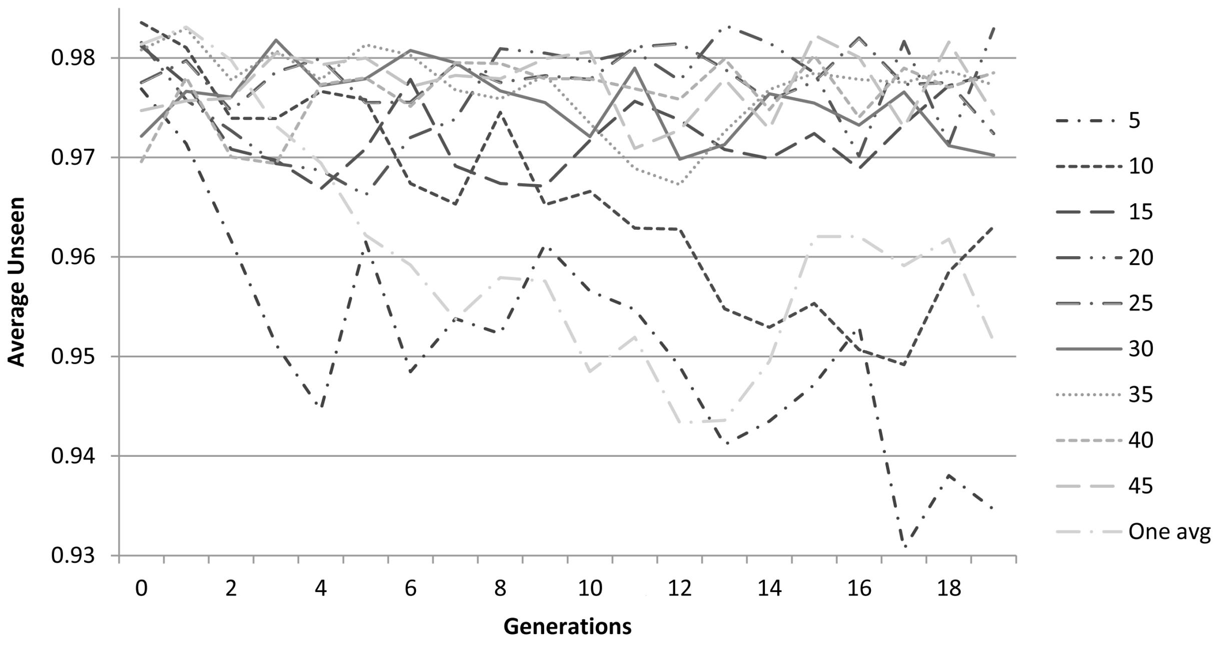


Figure 9: Average scores across five runs for various mutation rates

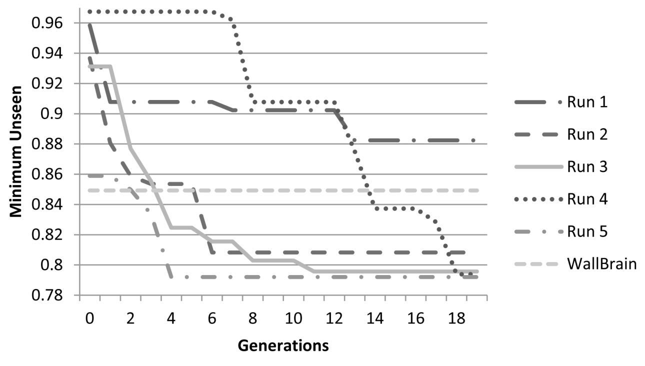


Figure 10: Minimum scores across five runs for 5% mutation rate

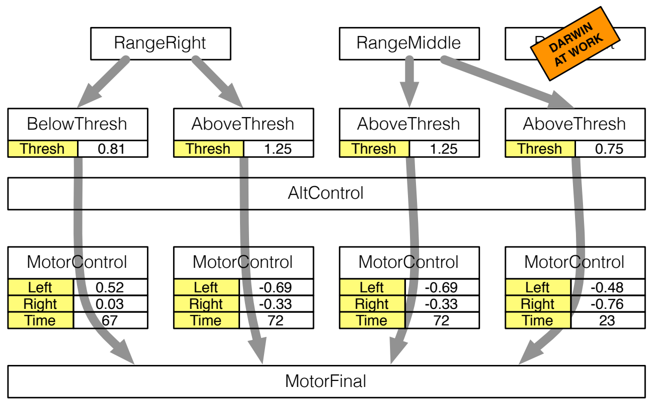


Figure 11: Successful evolved brain for wall navigation

more time moving forward), (2) followed paths that let them cover more ground in rooms (which are worth more in the weighted unseen metric), or (3) moved faster and covered ground in rooms.

6.3 A Successful Strategy: Wiggle and Avoid

We found there were two distinct movements among the best scoring evolved robots: obstacle avoidance was achieved through wall following and locomotion through wiggling. Figure 11 shows one such successful evolved network. The robots learned to use their distance sensors to follow either the right wall or the left wall. Also, for each robot there was some version of a wiggle motion (repeatedly moving slowly left then right while moving forward) with the use of the middle distance sensor. These behaviors align well with our successful implementation for the physical TCFFHRC task.

7. RELATED WORK

Research involving complex domains requiring multiple behaviors is currently on the rise in evolutionary robotics. Nelson and Grant [11] investigate a complex *Capture the Flag* game, and evolve feedforward and recurrent neural networks in a competitive environment for this task in both virtual and real robots. Capi [3] also uses recurrent neural networks to learn a parallel foraging and protection task, and find successful controllers through the use of a multi-objective function.

Ross [15, 14] used genetic programming to evolve concurrent programs capable of solving simple mathematical functions. Ross discusses the CCS algebra implementation, where the language primitives are much more suitable for mathematical functions than the complete programming languages discussed here.

Early work by Koza using a genetic algorithm to evolve a robotic controller [8] created virtual robots which followed the wall of an irregular shaped room. Liu and Iba [9] incorporated the subsumption architecture into evolutionary robotics. They divided a cooperative task into four levels, hand-coding some and allowing a genetic program to evolve the rest in a hierarchical procedure. Our approach differs from these by evolving the values of and connections between high-level parallel processes.

8. FUTURE WORK

Our current research has focused on learning process networks for locomotion and obstacle avoidance to successfully complete a complex fire-fighting task. The final portion of the task involves phototaxis (finding and extinguishing the candle), and we are currently investigating similar evolutionary methods for learning and refining networks for this behavior. We next plan to evolve the higher-level control structure, which will coordinate the actions of these separate behavior networks.

With our individual wall-following brain abstracted to read from sensors and control motors, we plan to port our learned JCSP process networks back to the occam-pi language for the physical Arduino framework. This will give us insight into the alignment between our Simbad simulator and the actual TCFFHRC arena.

We also plan to investigate the use of JCSP and our process network framework to learn in new and more realistic environments beyond the TCFFHRC. USARSim is a robotics simulation environment based on the Unreal game engine, and is currently used for the annual Robocup Rescue Virtual Simulation competition [4]. By developing an interface layer to make the information to and from the network sockets in USARSim look like our Arduino interface, our evolutionary framework should be easily transferred to this domain.

9. ACKNOWLEDGEMENTS

This research was supported through funding from the NSF CREU program. We would like to thank Centenary and Allegheny Colleges for the use of their facilities in preparation for the TCFFHRC.

10. REFERENCES

[1] D. J. Ahlgren and I. M. Verner. Fire-Fighting Robot International Competitions: Education Through Interdisciplinary Design. In *I*nternational Conference on Engineering Education, 2001.

[2] I. Armstrong, M. Pirrone-Brusse, A. Smith, and M. C. Jadud. The Flying Gator: Towards Aerial Robotics in occam-&amp;pi;. In P. H. Welch, A. T. Sampson, J. B. Pedersen, J. Kerridge, F. R. M. Barnes, and J. F. Broenink, editors, *C*ommunicating Process Architectures 2011, pages 329–340, jun 2011.

[3] G. Capi. Evolution of efficient neural controllers for robot multiple task performance - a multiobjective approach. In *I*CRA, pages 2195–2200, 2008.

[4] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. USARSIm: A Robot Simulator for Research and Education. In *I*EEE International Conference on Robotics and Automation, April 2007.

[5] C. Hartland and N. Bredeche. Evolutionary Robotics: From Simulation to the Real World using Anticipation. In *U*niversite de Paris-Sud, 2000.

[6] C. A. R. Hoare. Communicating sequential processes. *C*ommun. ACM, 21(8):666–677, Aug. 1978.

[7] M. Jadud, C. L. Jacobsen, J. Simpson, and C. G. Ritson. Safe parallelism for behavioral control. In *2008 IEEE Conference on Technologies for Practical Robot Applications*, pages 137–142. IEEE, November 2008.

[8] J. R. Koza. Evolution of subsumption using genetic programming. In F. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 110–119. The MIT Press, 1993.

[9] H. Liu and H. Iba. Multi-agent learning of heterogeneous robots by evolutionary subsumption. In *P*roceedings of the 2003 international conference on Genetic and evolutionary computation: PartII, GECCO'03, pages 1715–1728, Berlin, Heidelberg, 2003. Springer-Verlag.

[10] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *R*obotics and Autonomous Systems, 57(4):345–370, April 2009.

[11] A. L. Nelson and E. Grant. Using direct competition to select for competent controllers in evolutionary robotics. *R*obotics and Autonomous Systems, 54(10):840–857, 2006.

[12] A. L. Nelson, E. Grant, J. Galeotti, and S. Rhody. Maze exploration behaviors using an integrated evolutionary robotics environment. In *J*ournal of Robotics and Autonomous Systems, 2003, pages 159–173, 2004.

[13] S. Nolfi and D. Parisi. Evolving non-trivial behaviors on real robots: an autonomous robot that picks up objects. In *R*obotics and Autonomous Systems, pages 187–198. Springer Verlag, 1995.

[14] B. J. Ross. Pairwise Sequence Comparison and the Genetic Programming of Iterative Concurrent Programs. In *P*roc. Genetic Programming, pages 338–343. Morgan Kaufmann, 1998.

[15] B. J. Ross. The Evolution of Concurrent Programs. *A*pplied Intelligence, 8:21–32, 1998.

[16] H.-S. Seok, K.-J. Lee, and B.-T. Zhang. An on-line learning method for object-locating robots using genetic programming on evolvable hardware. In M. Sugisaka and H. Tanaka, editors, *P*roceedings of the Fifth International Symposium on Artificial Life and Robotics, volume 1, pages 321–324, Oita, Japan, 26-28 January 2000.

[17] J. Simpson, C. L. Jacobsen, and M. C. Jadud. Mobile Robot Control - The Subsumption Architecture and occam-pi. In P. Welch, J. Kerridge, and F. Barnes, editors, *Communicating Process Architectures 2006*, volume 64 of *C*oncurrent Systems Engineering, pages 225–236, Amsterdam, September 2006. IOS Press.

[18] P. Welch, N. Brown, J. Moores, K. Chalmers, and B. Sputh. Integrating and Extending JCSP. *C*ommunicating Process Architechtures, 2007.