Current biological research suggests certain simple neuronal architectures are responsible for complex organism behaviors.

Such architectures lend themselves easily to sophisticated control via genetic algorithms.

Therefore, we decided to implement genetic algorithms on a simple simulated robot with a biomimetic neuronal control system.
Initial Research
Motor neurons are the closest neurons to muscle groups along the neural path.

They directly or indirectly control one or more muscles at a time.

They tend to control muscles in such a way as to individually command basic actions such as “lift foot” (as opposed to controlling “twitch this one muscle”).

They are inherently analog, with the command signal sent to them being amplified by them proportionally.
Multiple motor neuron signals can be superimposed on the same muscle groups

Superimposition leads to complex motions
Command neurons control multiple motor neurons to generate complex actions.
Different amplitudes of control signal to command neurons produce different sequences of motor neuron action.
Command neurons can control other command neurons, leading to extremely complex actions.

- Step Forward
- Foot Forward
- Raise Foot

Biomimetic Research: Command Neurons
Central pattern generators are essentially an organism’s “clock” signal, a sinusoidal wave of a given frequency.

This sinusoidal pattern is selectively fed into command neurons to generate reciprocating actions such as walking.

Command neurons can be superimposed:

- Walk sideways + Walk forwards = Walk diagonally
GAs are algorithms that mimic evolutionary processes to produce improved function performance given a set of parameters to modify and an output metric to compare against. They include biological functions such as selection, breeding and mutation. They’re used for optimization where analytical understanding of a system is difficult. Require extensive computing power, but always produce a “good” result, and always get “better” with more computational power.
Genomes
- A genome is a term for parameters the genetic algorithm modifies, that are inputs to the simulation
- In any one iteration of a genetic algorithm, multiple genomes are run and their performance is compared

Selection
- Simulations provide outputs given genome inputs to compare against a selection metric, for each set of generated parameters
- Selection algorithm then determines which genomes are “good” and which are “bad”, and selects the top performers to continue forward

Breeding
- Good genomes live on to the next generation
- Empty spaces in the iteration docket are filled with “children”, crossed genomes of well-performing parents

Mutations
- Some genome attributes are randomized within a restricted range
The Project
Create a robot model whose movement is dictated by a set of simple motor neurons, and evolve several command neurons that control simple coordinated actions.
Simple simulator
- Written in Matlab due to previous student experience and ease of data display
- Planar world with high viscous drag

Simple robot model
- Robot is modeled as a disc with 6 massless legs
- Each leg can generate a force in any planar direction
- Robot has linear and rotational inertia

Simple genome
- Each robot has 6 leg genes
- Each gene corresponds to a command neuron that responds to the central pattern generator
- Total genome represents a command neuron at the level of “walk forward,” “strafe,” or “turn in place.”

Genetic algorithm
- Selection, breeding and mutation algorithms
Simulation

- Uses a planar rigid body kinematics solver
- Solves for robot position and velocity given genomes, using ode45
- Includes viscous drag intended to stop robot motion within three periods of the Central Pattern generator (CPG)
  - -50 Ns/m linear drag
  - -.0358 Nms/rad rotational drag
- Written in Matlab
We needed a physical model that gave us reasonable physical values to use in the simulator.

We modeled the robot with the following physical parameters:

- 1 kg steel disc
- 87 mm x 22 mm
  - Geometry was a function of desired weight
- 9.66e-4 kg*m^3 rotational inertia
  - Inertia was a function of geometry and density
Robot Model

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density (kg/m³)</td>
<td>7900.008</td>
</tr>
<tr>
<td>Mass (kg)</td>
<td>1.821</td>
</tr>
<tr>
<td>Volume (mm³)</td>
<td>126296.762</td>
</tr>
<tr>
<td>Surface area (m²)</td>
<td>17304.836</td>
</tr>
<tr>
<td>Center of mass (mm)</td>
<td>X: 0.000</td>
</tr>
<tr>
<td></td>
<td>Y: 0.000</td>
</tr>
<tr>
<td></td>
<td>Z: 0.000</td>
</tr>
<tr>
<td>Principal axes of inertia</td>
<td></td>
</tr>
<tr>
<td>Principal moments of inertia</td>
<td></td>
</tr>
<tr>
<td>Taken at the center of mass</td>
<td></td>
</tr>
<tr>
<td>Ixx</td>
<td>523.474</td>
</tr>
<tr>
<td>Iyy</td>
<td>523.474</td>
</tr>
<tr>
<td>Izz</td>
<td>966.814</td>
</tr>
<tr>
<td>Taken at the center of mass and aligned with the output coordinate system</td>
<td></td>
</tr>
<tr>
<td>Lxx</td>
<td>523.474</td>
</tr>
<tr>
<td>Lyy</td>
<td>966.814</td>
</tr>
<tr>
<td>Lzz</td>
<td>523.474</td>
</tr>
</tbody>
</table>
Robot legs have the ability to produce forces in a given direction, whose amplitudes are linear functions of the CPG.

The robot genome controls 5 parameters:
- Leg force amplitude with respect to CPG
- Direction of force with respect to body
- Phase of leg force oscillation with respect to CPG phase
- Phase of leg liftoff with respect to CPG phase
- Duty cycle of leg liftoff with respect to CPG period

Leg liftoff is a binary value; either the leg is producing force with respect to the CPG, or it is “off the ground” and unable to produce force.
- This allows legs to produce net forces in single directions without producing negative forces due to the CPG oscillation.
Our selection process first removes certain unfit genomes from the potential pool
  - Examples include genomes which move backwards when instructed to move forwards

The process then picks the top 20% of the remaining pool and moves them into the next iteration as-is
Our breeding process fills the remaining spaces in the iteration dockets with children of the remaining genomes.

A random half of the parameters of one parent are mixed half of the parameters of another parent.

Mutations are then applied.
Our mutations happen within a restricted range of potential values of the genome.

A point value mutation happens about once per child.
Results
Data: Forward Path
Data: Forward Path

Max Distance Forward as a Function of Iterations

Best Distance Traveled

Iteration #
Data: Strafing

Path Progression of the Strafing

- Path 5
- Path 10
- Path 15
- Path 20
- Path 25
- Path 30
- Path 35
- Path 40

Y (meters)

X (meters)
Data: Strafing

Max Distance Sideways as a Function of Iterations

Best Distance Travelled

Iteration #
Data: Turning In Place

Path Progression of the Turning in Place
Data: Turning In Place

Max Rotation as a Function of Iterations

![Graph showing the maximum rotation as a function of iterations.]
Results are indicative of benefits and pitfalls of genetic algorithms.

Results show that robots are improving, but are getting caught in local optima.

Paths are also not as constrained as we’d like them to be – optimizing for multiple variables (orientation and translation, for instance) is very hard.

Changing the selection, mutation, or breeding algorithms is difficult, and not analytical.
Selection is difficult
- Deciding how many members to keep is hard
  - Keeping few members speeds iterations
  - Keeping more members maintains diversity
- Optimizing for multiple variables is also hard
  - Fewer variables optimize well for those variables, but other aspects are completely uncontrolled (i.e., it walks forward but spins at the same time)
  - More variables increases solution space, leading to relative ineffectiveness of algorithm

Escaping from local optima is difficult to do, especially after several iterations
This was an appropriately sized, interesting project for two people, for four weeks.

We learned a lot about how genetic algorithms work, what they’re good for, and what they’re not good for.