Evolutionary robotics

Don’t code, evolve!

What is evolution?

"Definition:

[1] The process by which different kinds of living organism are believed to have developed from earlier forms during the history of the earth.

[2] The gradual development of something"

- Based on Darwin's evolution theory
  - Reproduction is the key to life
  - Better fitted (adapted) individuals have better chance to reproduce (i.e. survive)
  - Successful phenotype* traits are reproduced, modified, combined

** phenotype - The set of observable characteristics of an individual resulting from the interaction of its genotype with the environment.*

How to represent an individual?

- Each individual has its own genotype → representation of the individual
    - Same as genes in people
- In programming this can be a lot of things
- The simplest problems might have individuals represented as a binary number, hard problems might have complex neural nets

- One of the simplest evolutionary problems is Max1 → evolve an individual with all 1s, starting with a random population
- Simple robot problems can be modeled by using neural nets which for example map the robot's input sensors to signals for its two wheels

Glossary

Fitness
"Score" of an individual. How good the individual is amongst the population. Survival of the fittest -> the higher fitness one has, the more likely their genes are to be passed on.

Fitness function
Function determining fitness of each individual in the population.

Recombination / Cross over
Altering the genes of multiple individuals by combining parts of their genotypes.

Mutation
Pseudo random altering of genes of a single individual.

Selection
Choosing which individuals continue to the next generation, which recombine etc.

Evolution

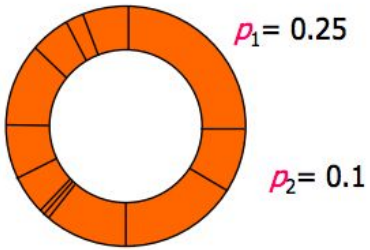
- Evolutionary algorithms are de facto population based stochastic search algorithms
  - We are non-deterministically looking for a sufficient individual in a given population
- Recombination and mutation create variability
  - Also prevent the evolution from getting stuck
- Selection leads the search in the right direction
  - Only the better traits are preserved

Simple genetic algorithm

- In time $t = 0$ generate a random initial population P(0) of n m-bit genes (individuals)
- From P(t) to P(t+1)
    - Compute fitness for each individual from the population
    - Repeat n/2
        - Select a pair x, y from the population
        - Cross over x, y with probability p_c
        - Mutate every bit of x and y with probability p_m
        - Insert x, y to P(t+1)
- Different settings and probabilities of course yield different results
- Multiple crossover, mutation and selection options

Selection

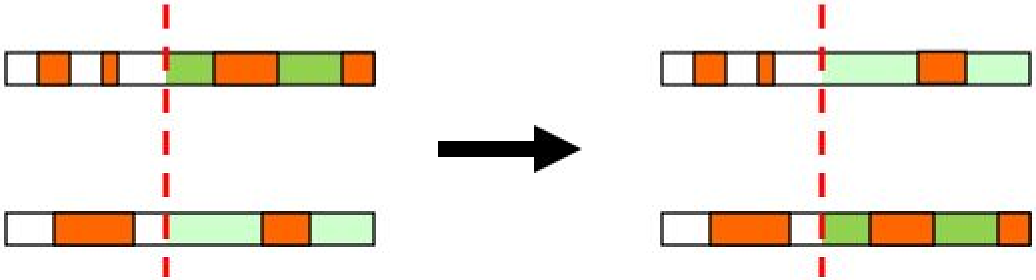
- Based on the individual's fitness
- Basic: Roulette wheel
  - Each individual occupies a slice of the roulette based on its fitness with regard to the sum of all the fitnesses
  - The roulette is spun n-times



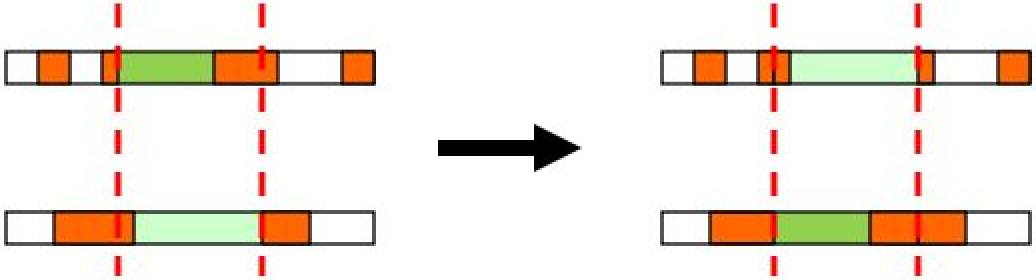
- Other selection types: Tournament, rank based
- Possible elitism (the *m* best individuals are preserved)

Cross over

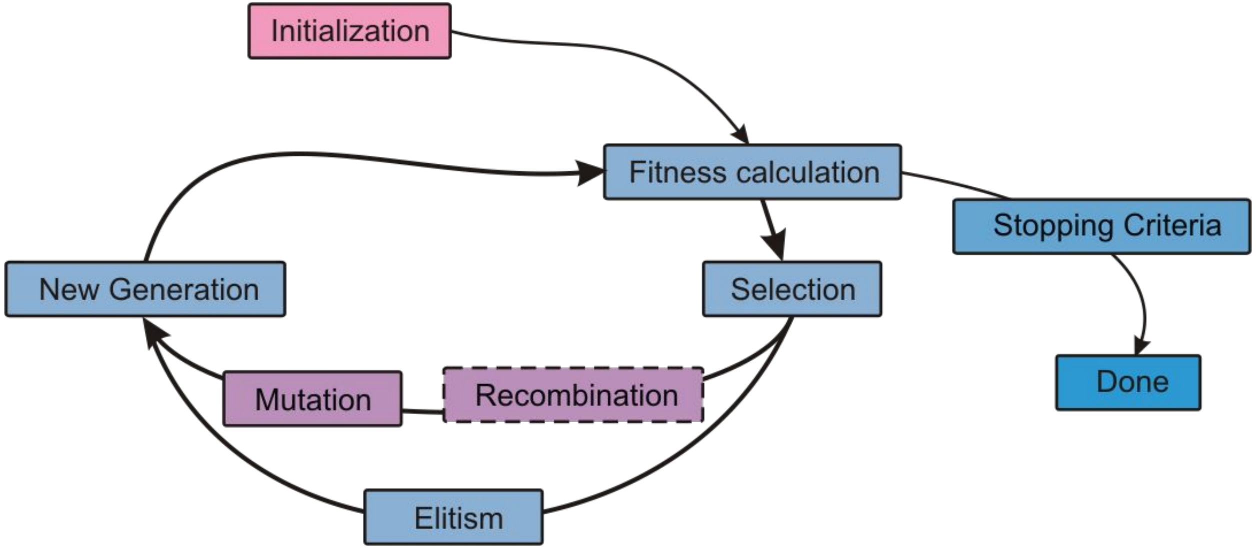
- Single point



- Multi point



Evolution in a picture



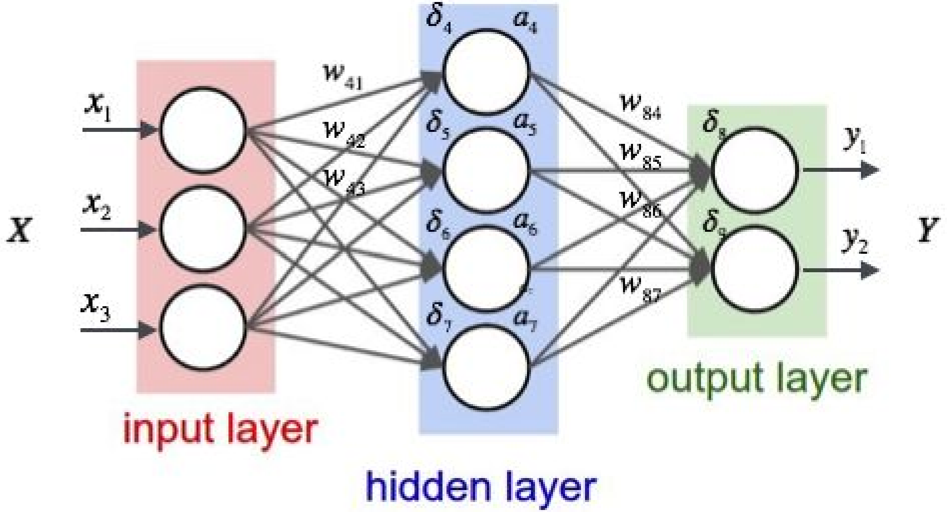
Great example - https://www.youtube.com/watch?v=GOFws_hhZs8&t=254s

Evolution parameters

- Each problem has its own representation
  - Have to choose proper cross-over, mutation and mainly fitness function
- Mutation regarding neural nets
  - Change in a weight of a synapse
  - Change of the type of the activation function
  - Adding/removing synapses
  - Adding/removing neurons
- Other important parameters
  - Population size
  - Probability of a cross-over
  - Probability of a mutation
  - Size of the elite group

Simple neural net

- Possible model for a robot
  - Output layer of 2 neurons represents two wheel signals
  - Input layer represents robot's sensors



Line follower evolution

for Arduino robot

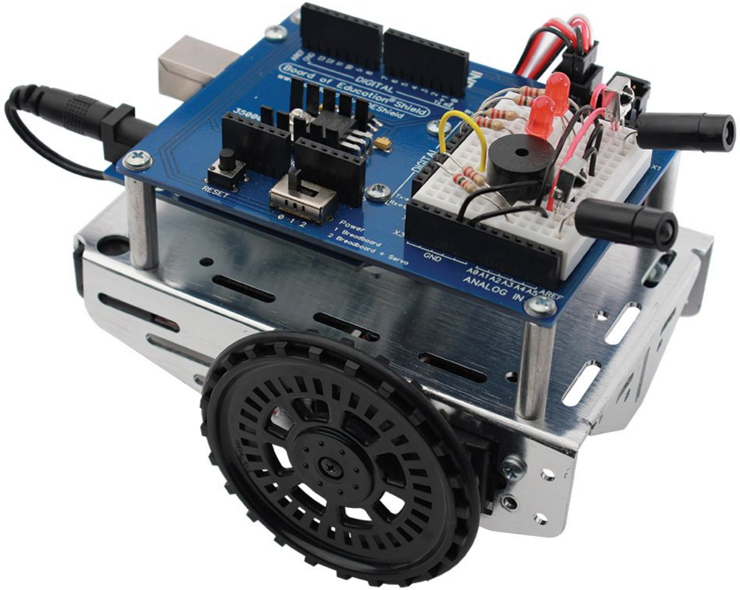
Line follower problem

- Something like Hello World for new programmers
- The goal of the exercise is to develop a robot that can properly follow a black line on a white sheet of paper (with harder version which can have turns and signals etc.)
- Not a trivial task but a good one to introduce the robotics to the newcomers
- Programming this robot takes time and people often have to learn new technologies because of the robot's architecture

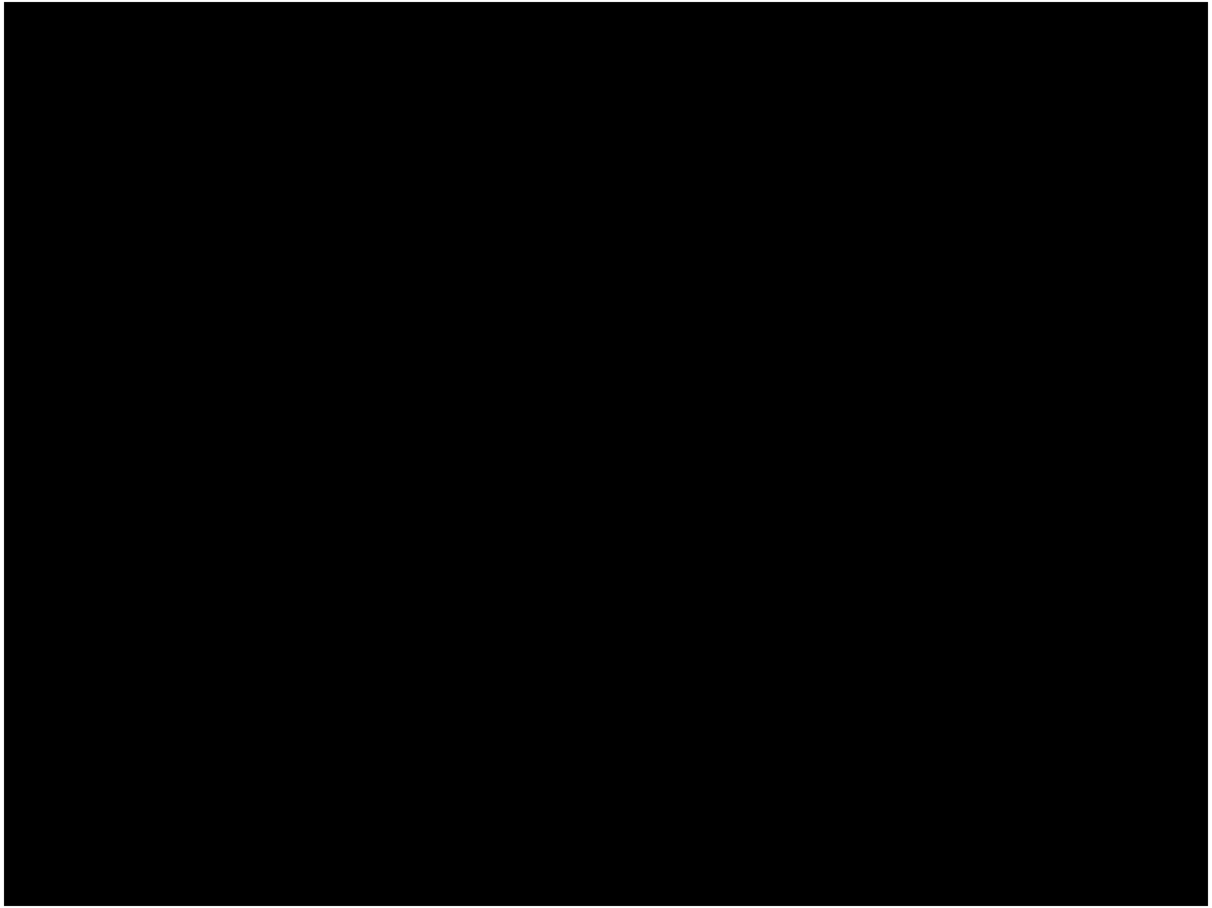
- Can lazy people solve it with evolution?

Goals

- implement a simulator for an Arduino robot used in the subject *Introduction to Robotics (NAIL028)*
- develop a line following robot using evolution
- somehow port to the real robot?



Motivation

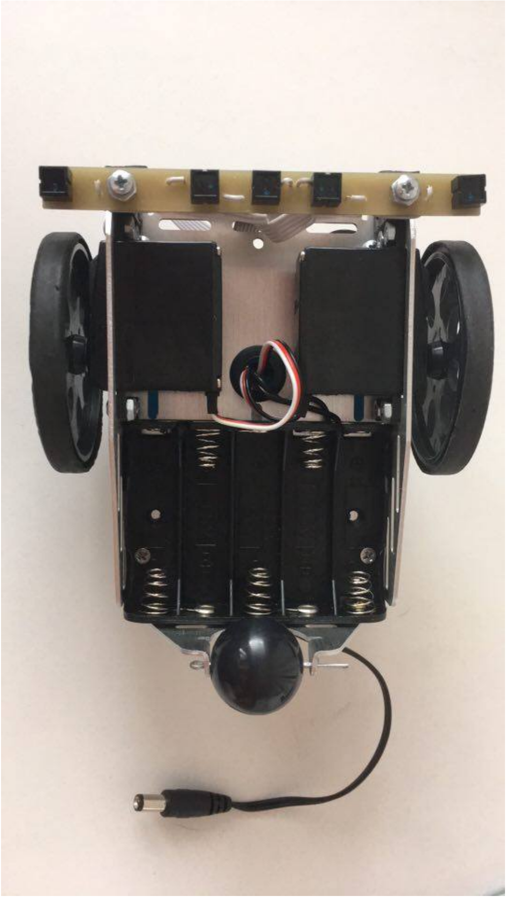


What we worked with

- Arduino Parallax Boe-Bot
- JavaScript/Webpack/Node.js
- Neataptic.js library
  - Implemented NEAT algorithm in JavaScript

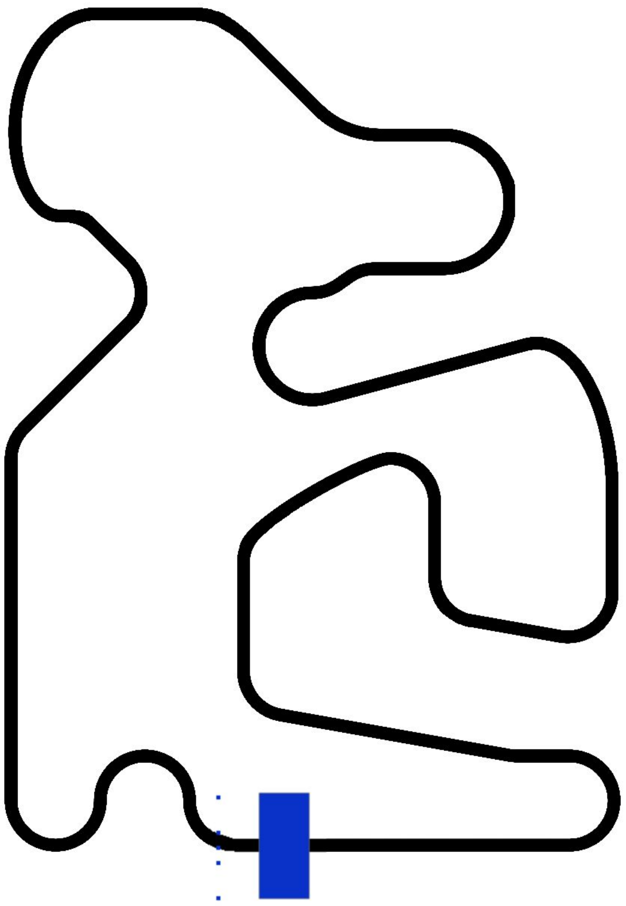
Robot

- 5 sensors on a ramp
  - digital input (0/1 ~ black/white)
  - analog input (0–1000 ~ grayscale)
- 2 wheels
  - possible values: 1300–1700 (from full clockwise to full counterclockwise)
  - examples
    - (1300, 1700) → robot moves forward at full speed
    - (1700, 1300) → robot moves backward at full speed
    - (1700, 1700) → robot rotates in place
    - (1500, 1500) → robot does not move



Simulator

- custom made in Javascript
- tailored to the Arduino robot
- users can use any black and white image (with size specification) which is transformed to pixel representation and used in the simulator
- http://robot-simulator.herokuapp.com
- 1 metre in real life is represented by 1000 pixels in the simulator
    - 1mm precision



Simulator

- adjustable parameters:
  - track size in real world
  - starting position
  - interval of sensor detection
  - starting and max speed
  - Arduino servo specifics (values range, which servo signal means stopping)
  - robot’s initial rotation
  - wheel gauge (distance of the centers of the wheel)
  - sensors’ positions in regard to the center of the wheel axis
  - sensor radius
  - controller of the robot (a neural net in our case)
- metres and seconds used as units

Robot controlling system

- neural net with 5 input neurons and 2 output neurons
  - however, the simulator can theoretically work with any controller capable of mapping a 5-dimensional input vector to a 2-dimensional output vector
- we used digital (=binary) inputs from the sensors (0/1 for black/white)
- outputs are expected in the interval (-1,1) and are subsequently mapped to (1300, 1700) by the simulator

Evolution

- **NEAT** used to evolve networks
  - JS library **neataptic** (https://github.com/wagenaartje/neataptic)
- starting with a random 2-layer network with 3 hidden neurons and sigmoid activation function on all nodes
- **Configurations**
  - population: 100-200
  - max generation: 200-500
  - mutation rate: 0.8
  - experimented with various mutation strategies and fitness

Mutations

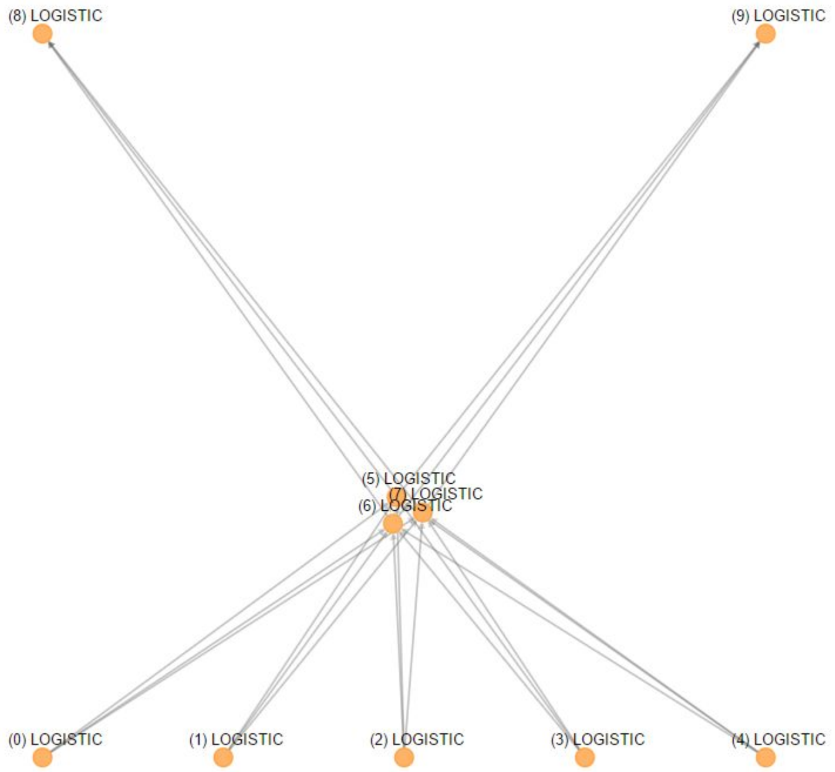
- possible mutations:
  - modify weight of a synaptic connection
  - modify threshold of a neuron
  - modify the neuron’s activation function
  - add a connection
  - remove a connection
  - add a neuron
  - remove a neuron

Results

First steps

- **mutation:** only change of synaptic weights and neural thresholds allowed (no structural mutations)
- **fitness:** travelled distance
- **stopping condition:** robot is out of track (all sensors see white)

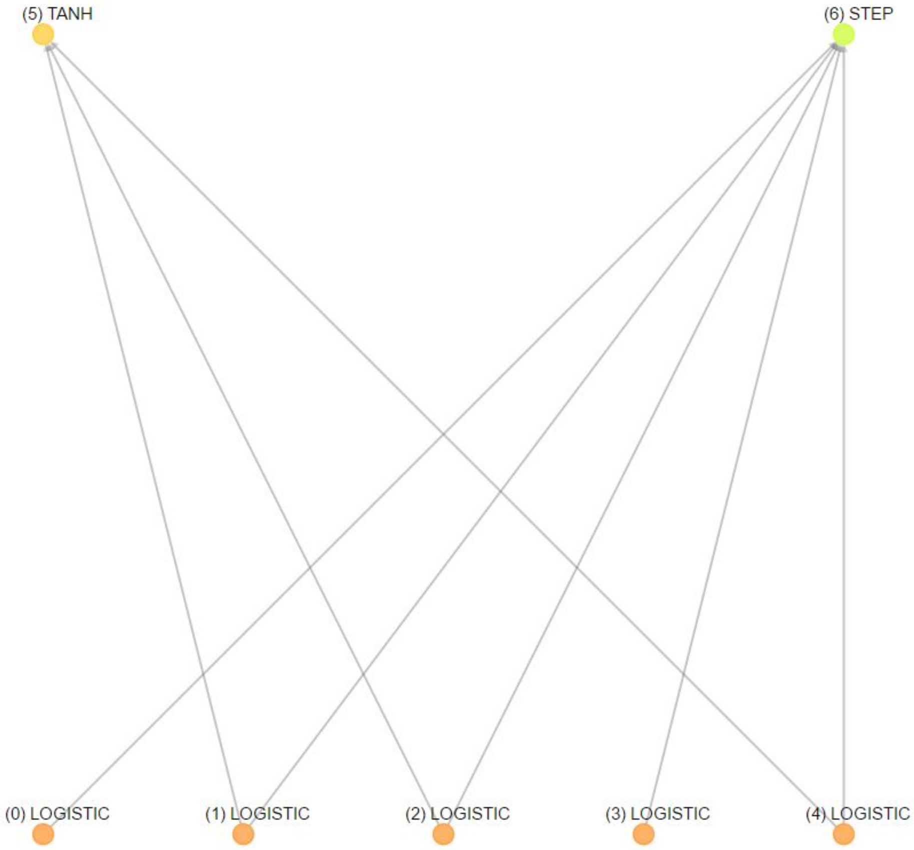
Result:
Failure. Can’t even make the first turn.



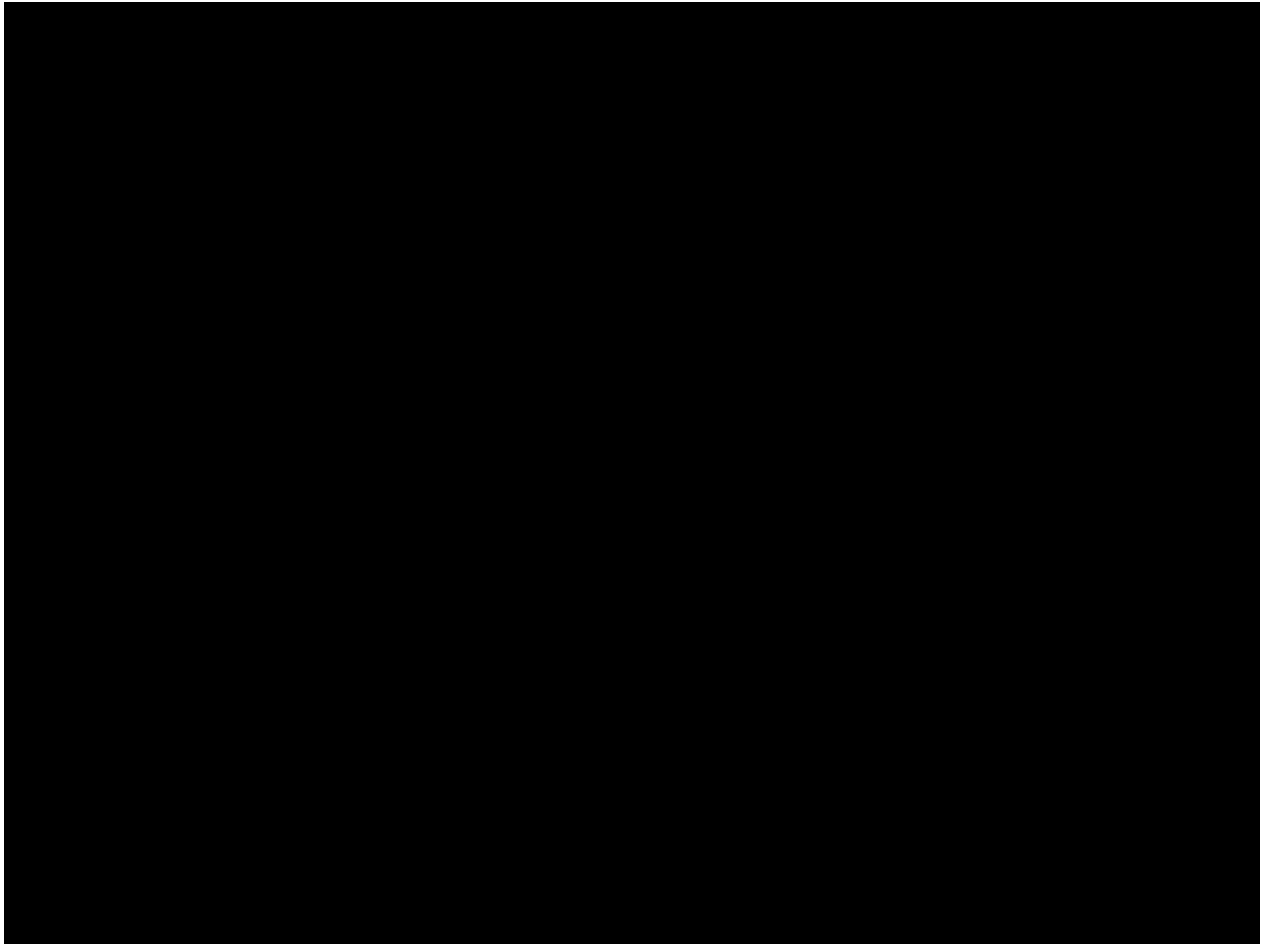
Partial success

- **mutation:** ***all allowed, but:***
  - ***input neurons fixed on LOGISTIC (=sigmoid activation)***
- **fitness:** travelled distance ***& middle sensor on the line***
- **stopping condition:** robot is out of track (all sensors see white) ***for a certain period of time***

Result:
Can navigate about a quarter of the track before failing.



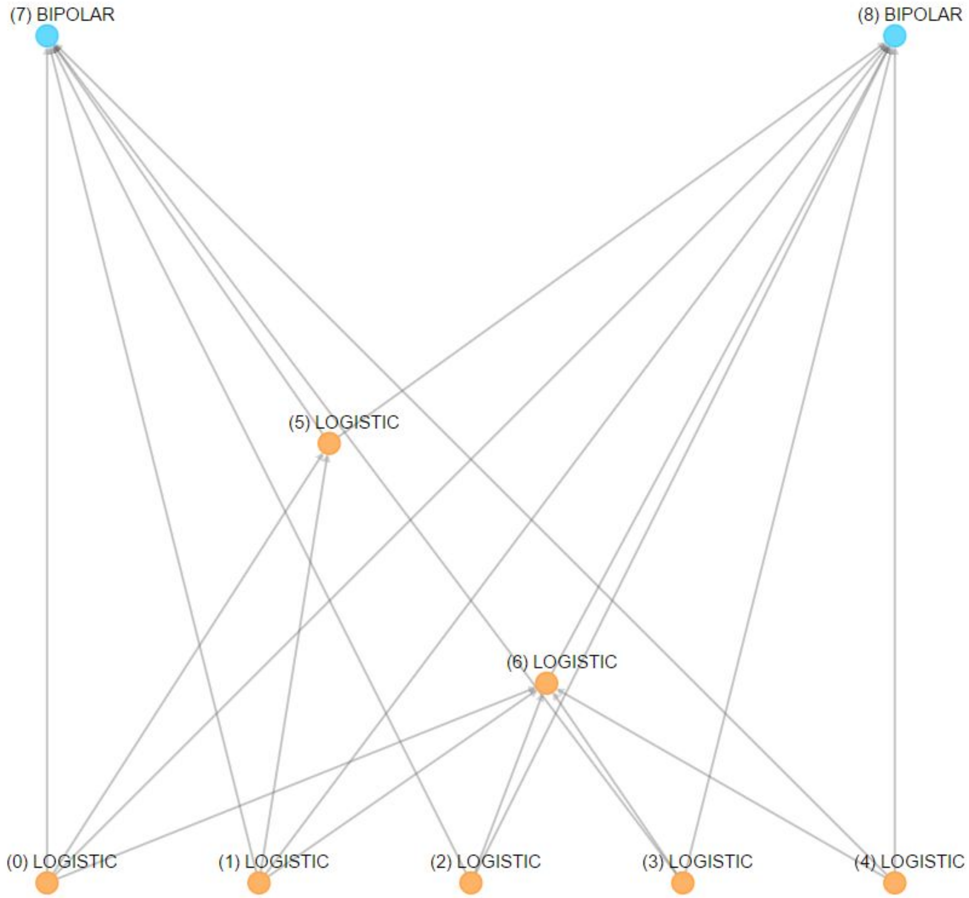
Video



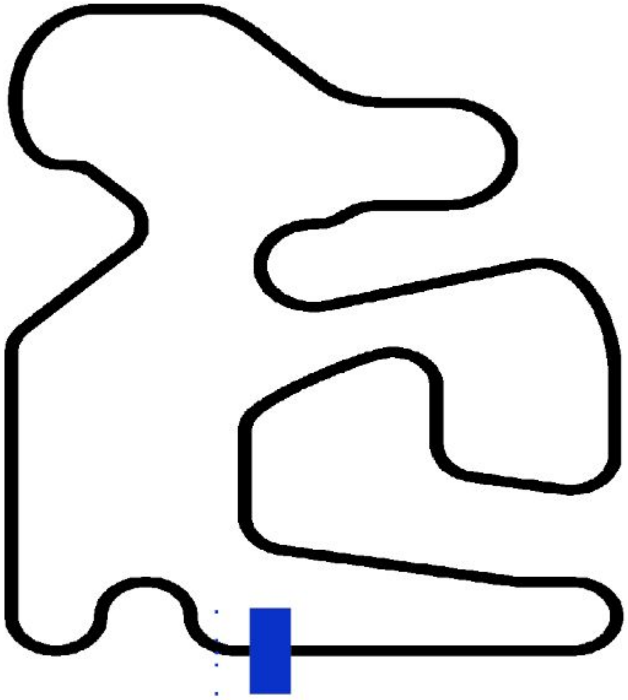
Cheating robot

- **mutation:** all allowed, but:
  - input neurons fixed on LOGISTIC (=sigmoid activation)
- **fitness:** travelled distance & middle sensor on the line ***& speed***
- **stopping condition:** robot is out of track (all sensors see white) for a certain period of time

Result:
Well...



Video



Looks like an image instead of a video?

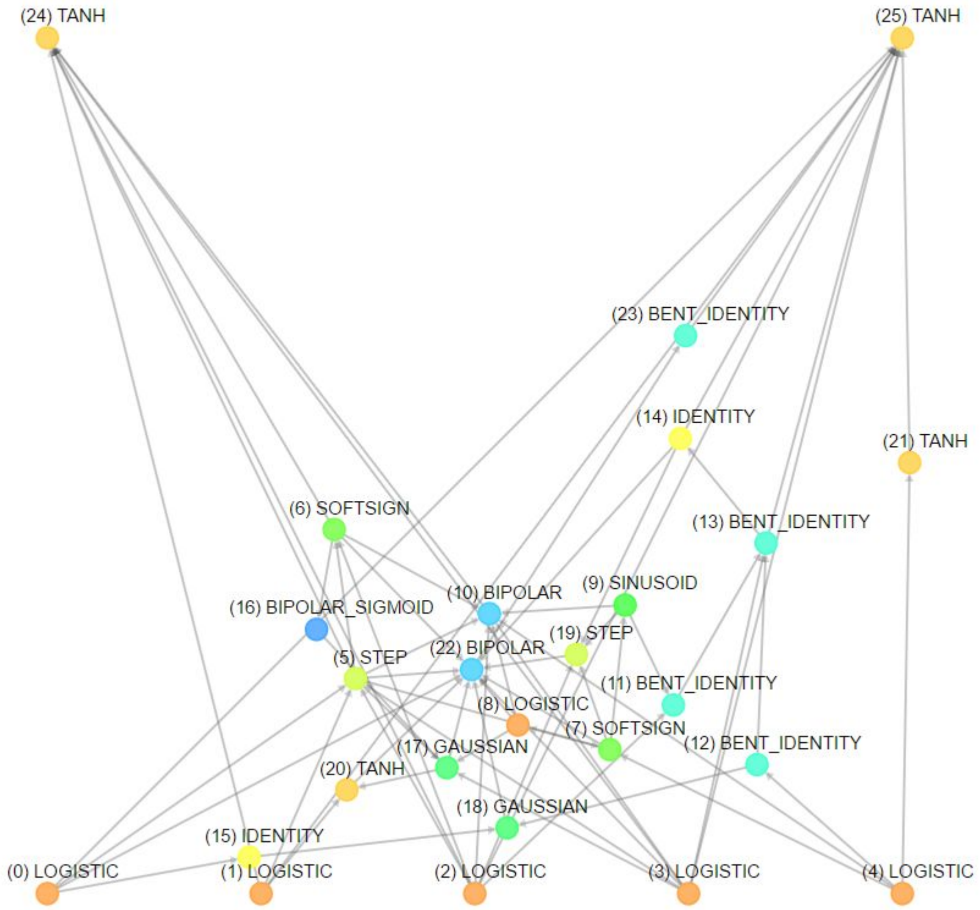
```
▶ Object {left: 0.2, right: 0.2}
▶ Object {left: -0.2, right: -0.2}
▶ Object {left: 0.2, right: 0.2}
▶ Object {left: -0.2, right: -0.2}
▶ Object {left: 0.2, right: 0.2}
▶ Object {left: -0.2, right: -0.2}
▶ Object {left: 0.2, right: 0.2}
▶ Object {left: -0.2, right: -0.2}
```

Every tick, the robot iterates between going full speed ahead and full speed in reverse.

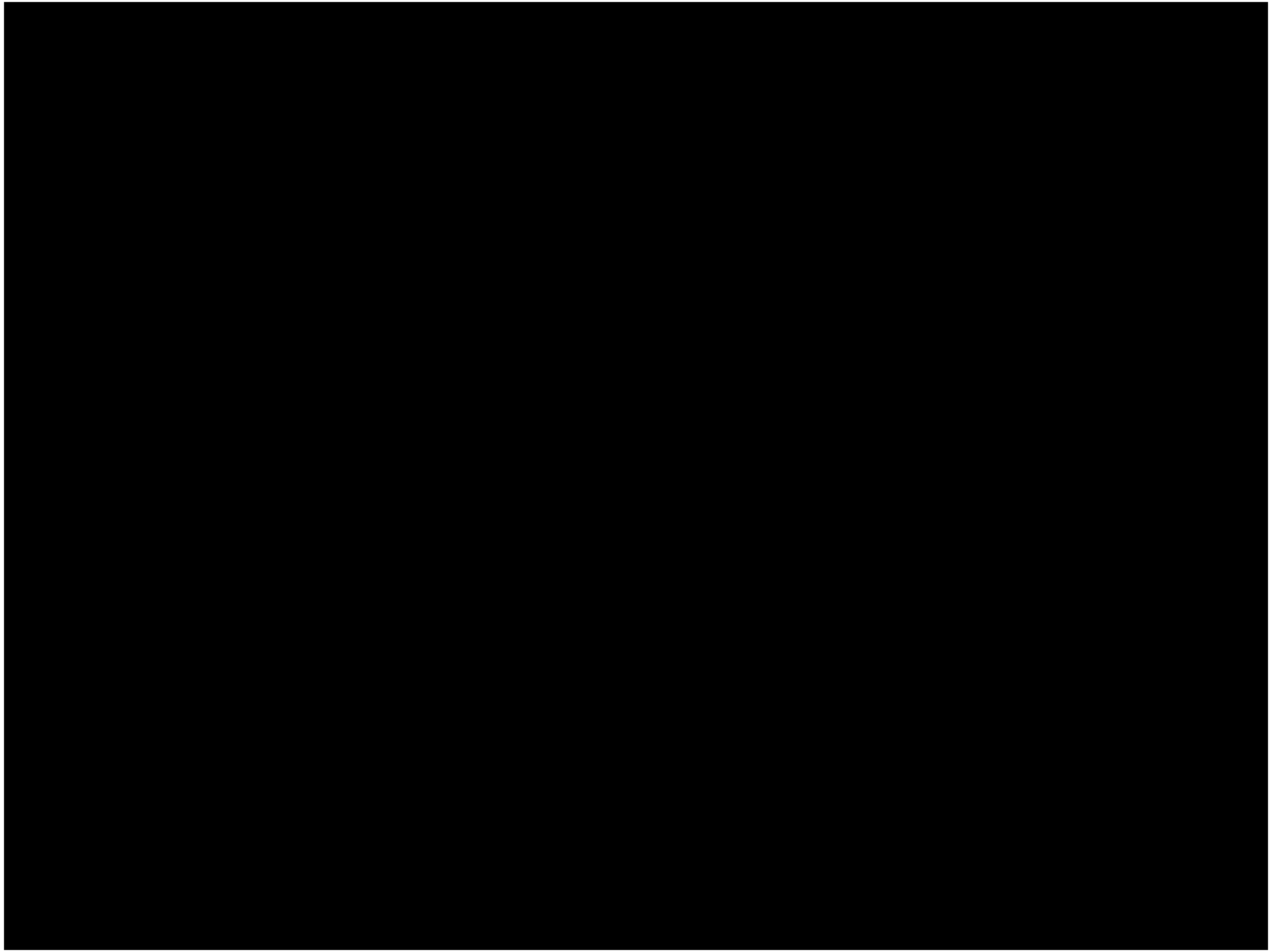
The “winner”

- **mutation:** all allowed, but:
  - input neurons fixed on LOGISTIC (=sigmoid activation)
  - ***output neurons fixed on TANH***
- **fitness:** travelled distance & middle sensor on the line & speed
  - ***big penalization for going backwards***
- **stopping condition:** robot is out of track (all sensors see white) for a certain period of time

Result:
Flawless on both the training track and a previously unseen, more complicated track.



Video



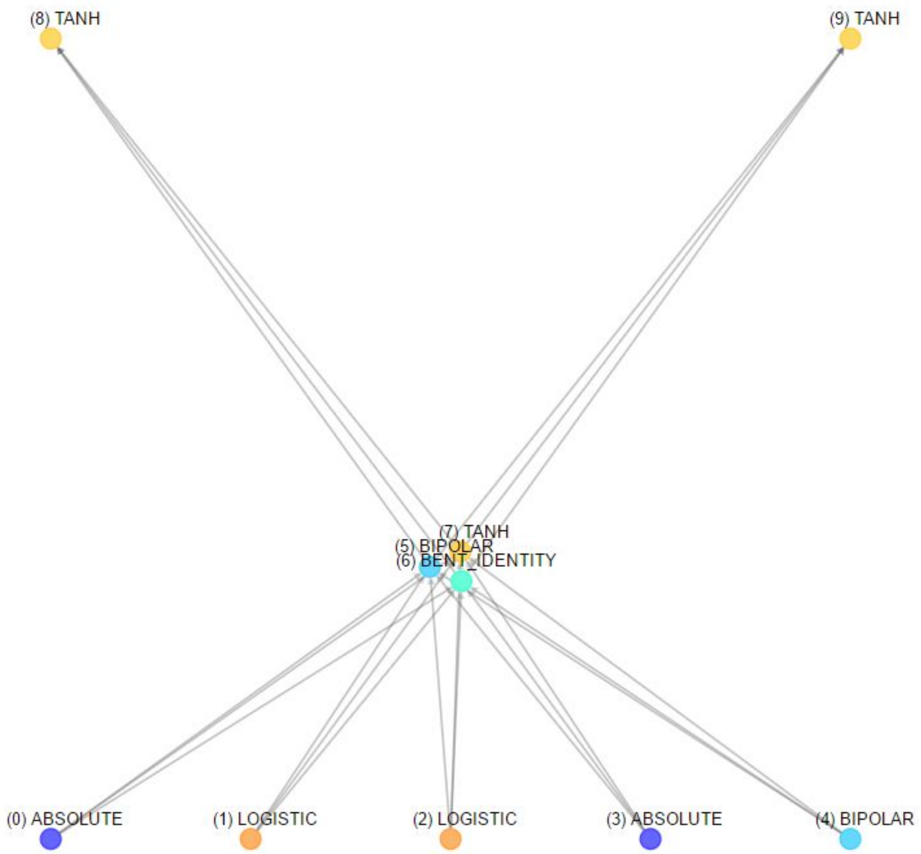
Is such a large network necessary?

- penalization of large network in fitness function
  - always degraded to networks with no hidden neurons
- fix the structure again, but use better fitness and stopping condition

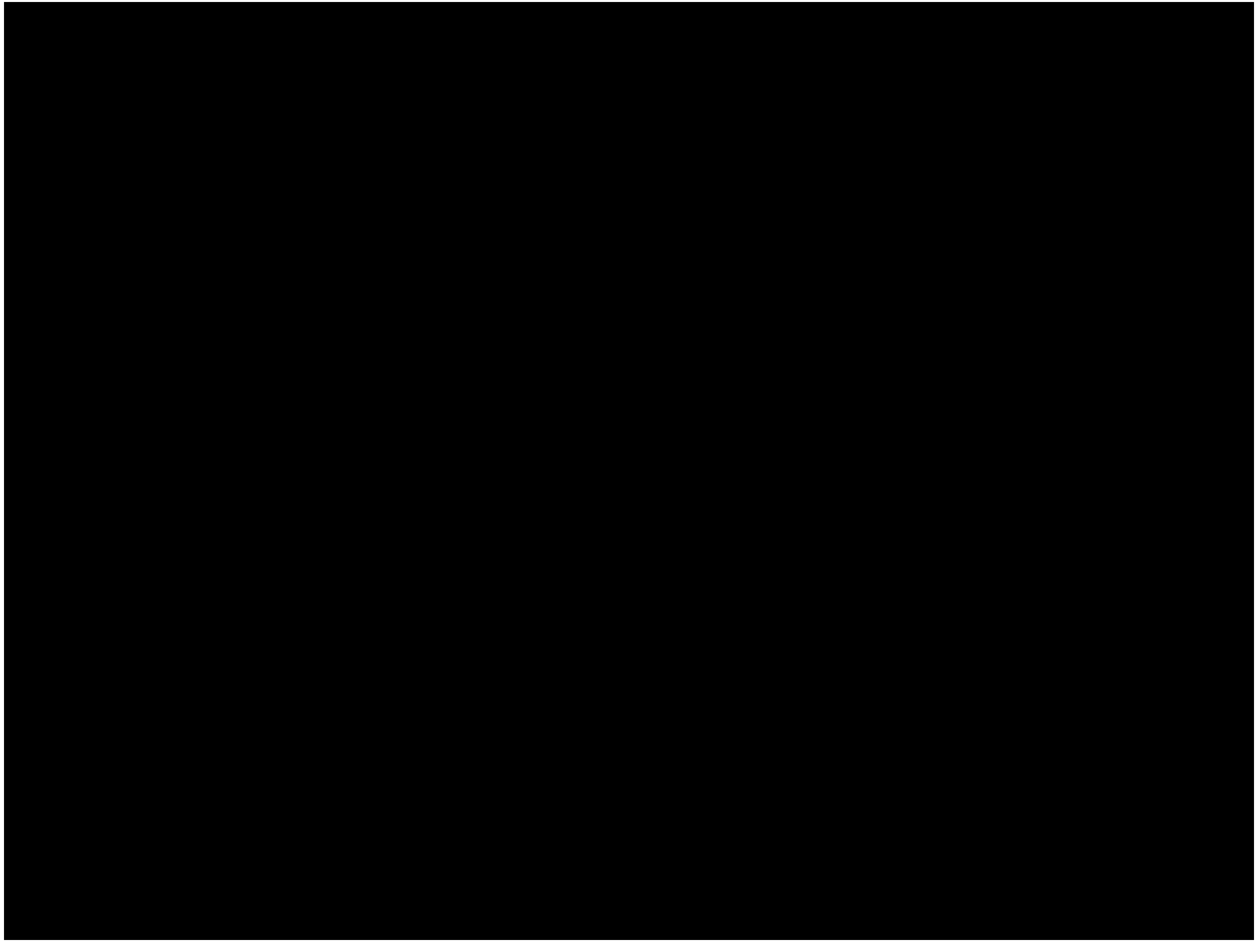
Final experiment

- **mutation:** ***only non-structural,*** but
  - output neurons fixed on TANH
  - ***input neurons are not fixed anymore***
- **fitness:** travelled distance & middle sensor on the line & speed
  - big penalization for going backwards
- **stopping condition:** robot is out of track (all sensors see white) for a certain period of time

Result:
Flawless on the training track, but problems on the previously unseen track.



Video



Intermezzo - what could be done better?

- simulator
  - more parameters
    - friction, acceleration
- neuroevolution
  - better fitness function
  - penalization for large networks
  - train the neural net on multiple tracks at once
  - or even better, dynamically generate path and train the neural net on it

Arduino implementation

- Uses C
  - How to implement the neural net?
- Using Neataptic.js, we can get the list of nodes and edges and then do the computation ourselves
  - https://pastebin.com/xGHupN7C
- Error prone, takes too long
- Can it be done better?
  - YES!
  - https://pastebin.com/7i2zXzaM

Real world results

- We have the neural net implemented in C
- Simulator works
  - Fairy tale environments where nothing can go wrong

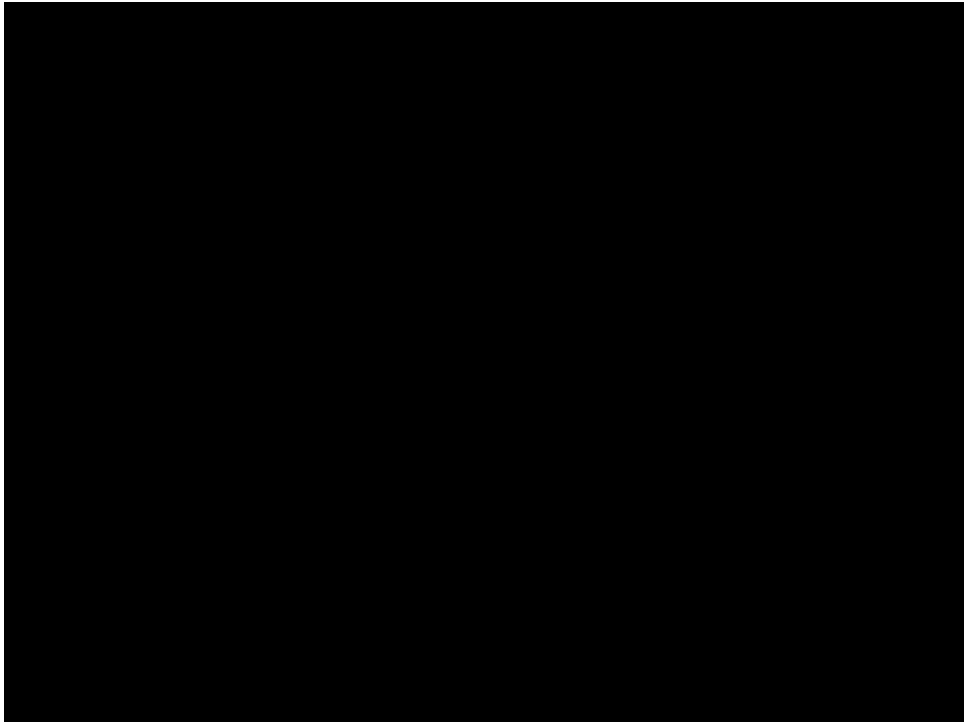
    No outer influences

    No friction

    Instant acceleration
- Would the robot controlling neural net work just by porting it to the real world robot?

Real world results

- Yes.



Thank you for your attention.

Questions?