Julia + Lego Mindstorms

Robin Deits
Dec. 7, 2015

Lego Mindstorms

1998

[http://www.legomindstormsrobots.com]

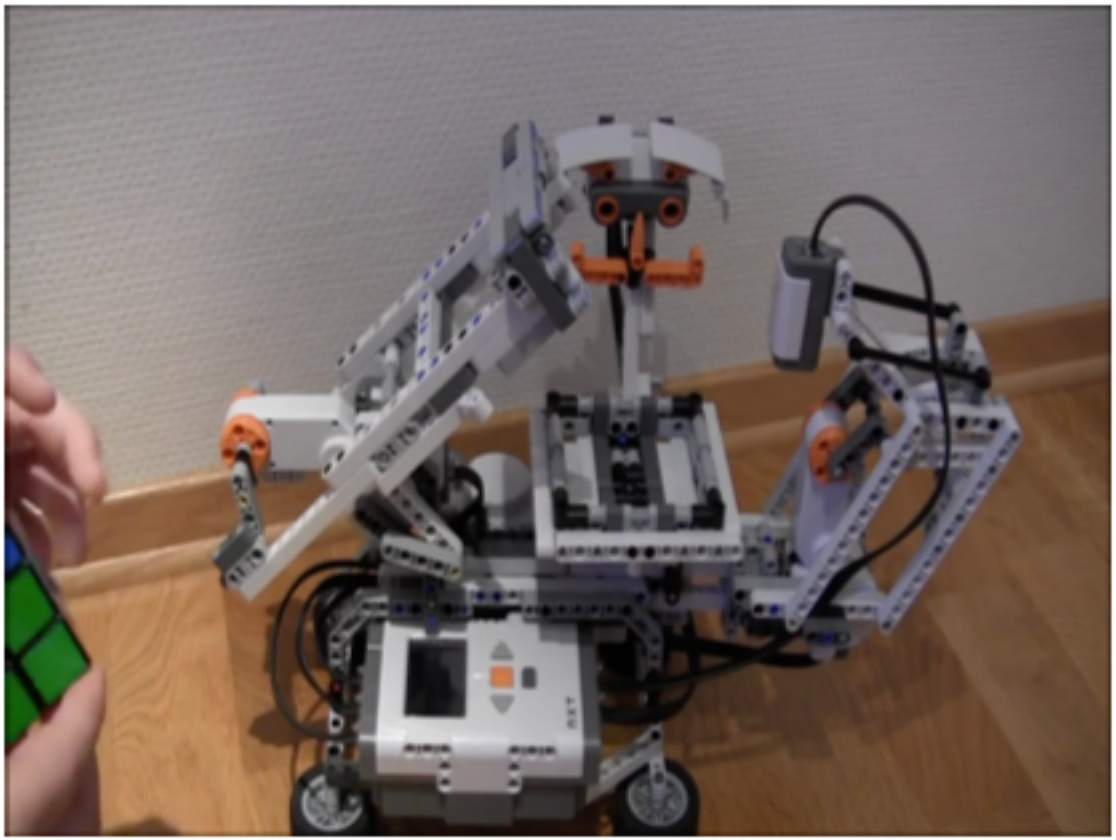
2006

2013

[www.amazon.com]

Mindstorms Projects

[http://www.legomindstormsrobots.com/]



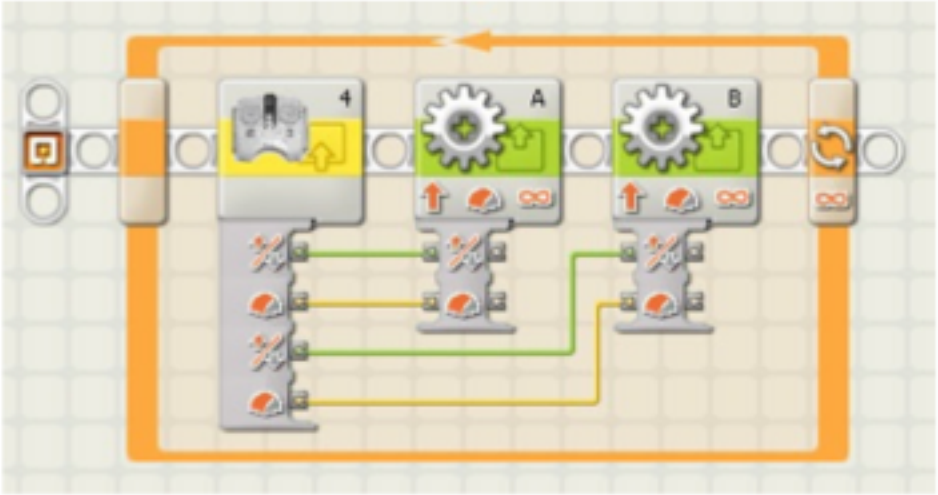
https://www.youtube.com/watch?v=dreTvumjNyw

Mindstorms Interfaces

1998

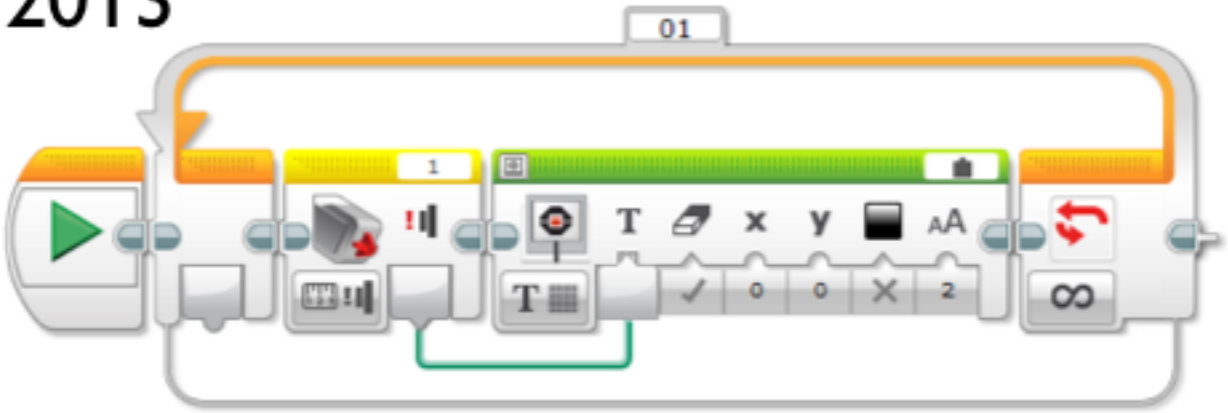
[http://www.techno-stuff.com/swmux.htm]

2006



[http://www.hitechnic.com]

2013



http://www.iberobotics.com

Mindstorms CPUs

- 1998: RCX
  - Processor: 16 MHz
  - RAM: 32 KB
  - Communication: custom infrared port
- 2006: NXT
  - Processor: 48 MHz
  - RAM: 64 KB
  - Communication: USB
- 2013: EV3
  - Processor: 300 MHz
  - RAM: 64 MB
  - Communication: ssh + USB, WiFi, Bluetooth, etc.
  - **Runs Linux!**

It Gets Even Better: ev3dev

- ev3dev is a custom build of Linux (based on Debian Jessie) that runs directly on the Mindstorms EV3 brick
- Uses sysfs to map motors & sensors to the file system
  - Running a motor is as easy as:

```
echo “run-forever” > /sys/class/tacho-motor/motor0/command
```

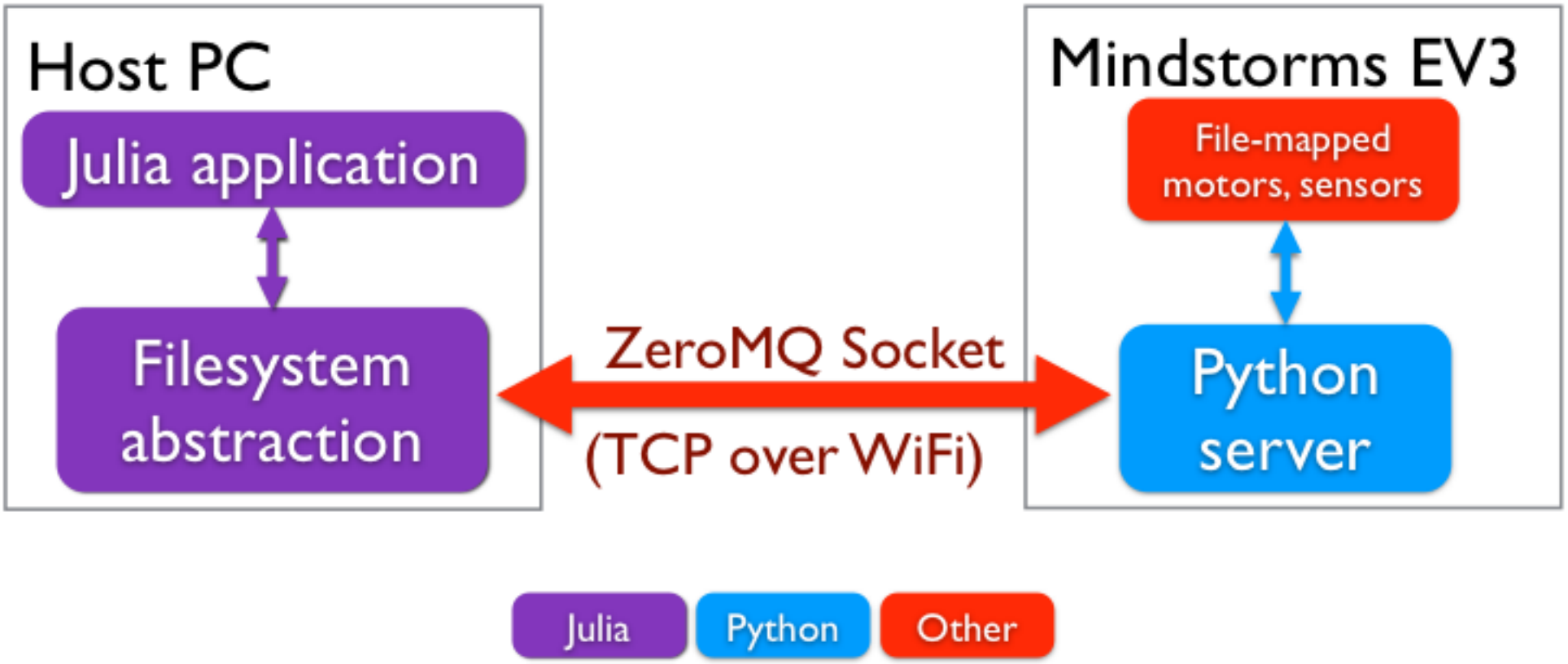
- Provides bindings for C++, Python, etc.

Julia on the EV3

- So, can we just run Julia directly on the EV3?
  - No (or at least not yet).
  - The EV3 processor has no hardware floating-point support, which has so far made building a working Julia impossible
  - Currently Julia builds but segfaults on startup

Running Julia Off-board

- Instead, we can run Julia off-board and use ZeroMQ to communicate with the EV3:



Creating Julia Bindings

- Many signals to read and write for various devices:
  - `command, position, value, driver_name, fw_version, etc…`
- Leads to lots of repetitive code
- Ev3dev's Python bindings use Liquid Templating language (in Ruby) and an additional build system to generate Python code

Creating Julia Bindings

- Julia's macros, on the other hand, make generating code easy:

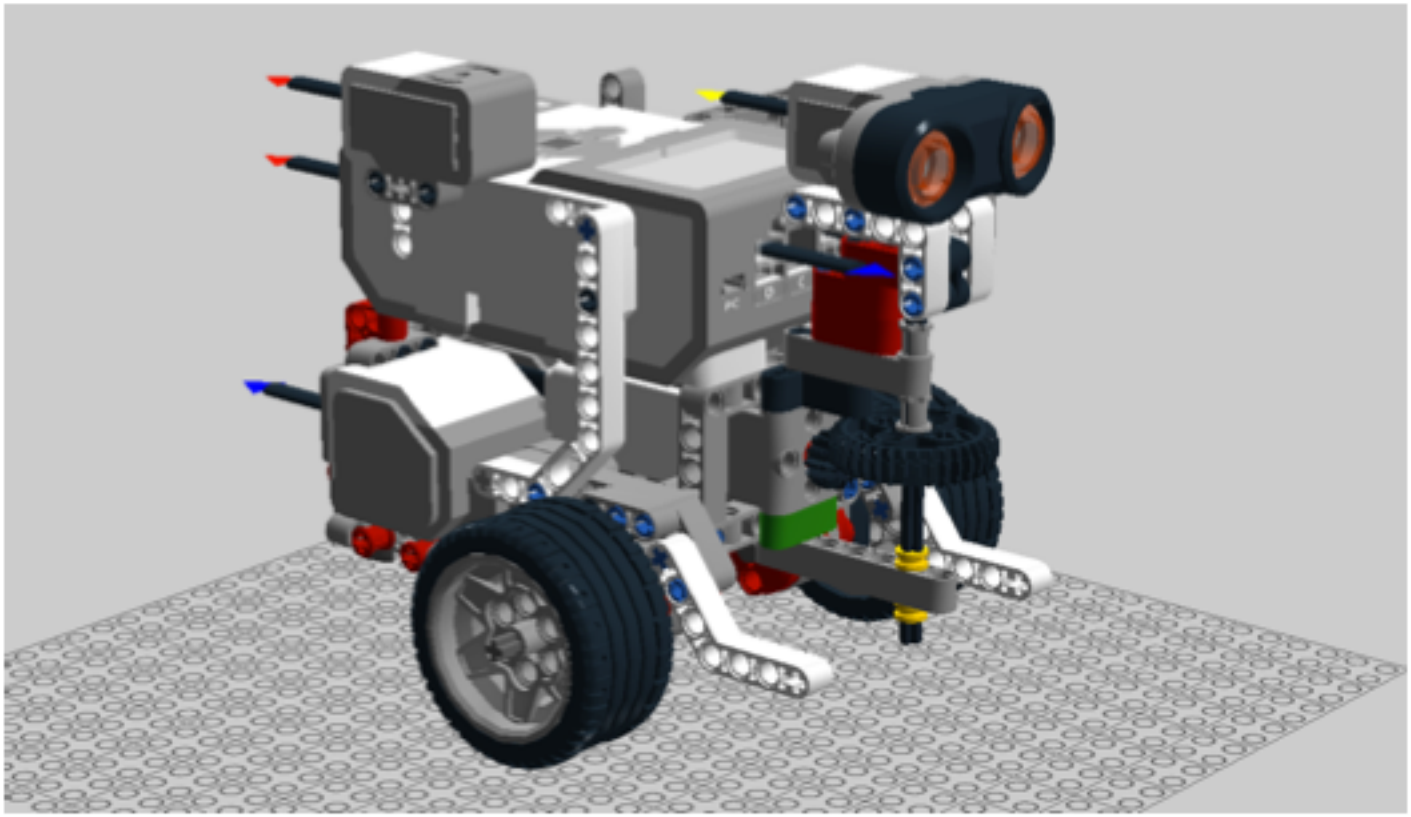
```
macro readable(name, T, parser)
    return quote
        function ($(esc(name)))(dev::($T))
            $(parser)(read(dev, $(esc("$(name)"))))
        end
    end
end

@readable driver_name AbstractDevice as_string
@readable fw_version AbstractDevice as_string
@readable value0 Sensor as_int
@readable position Motor as_int
```

Example Task: Mapping

- To demonstrate parallel robotics with Julia and Mindstorms EV3, we'll perform cooperative mapping of an environment
- Each robot will:
  - Drive around
  - Use its ultrasound sensor to detect and avoid walls
  - Use odometry and inertial sensors to estimate its position
  - Build a local map of the walls it observes

Mapping Hardware

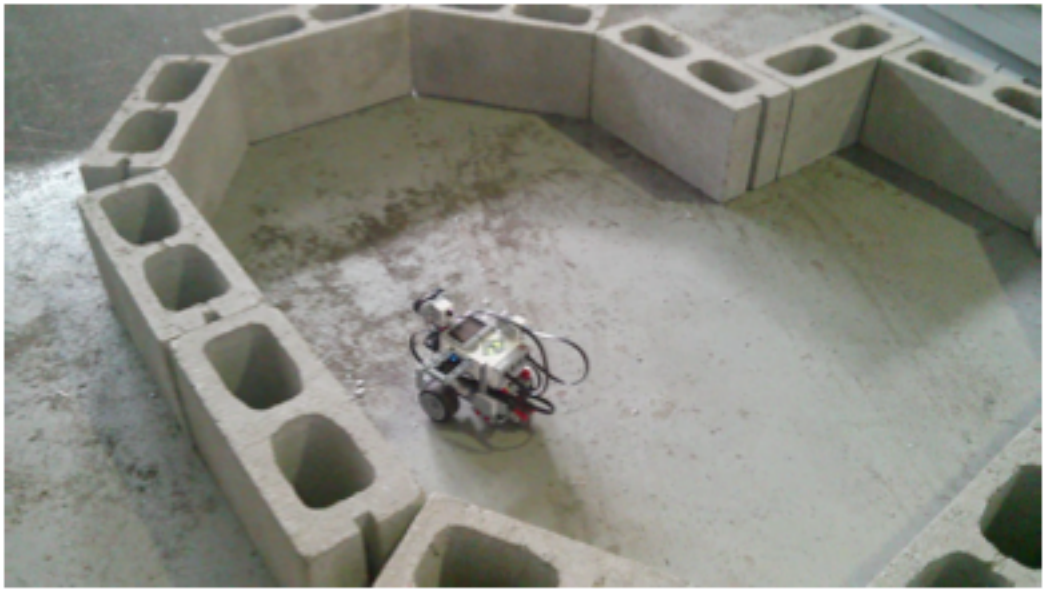
- Custom design, based on a standard Lego 2-wheeled chassis:



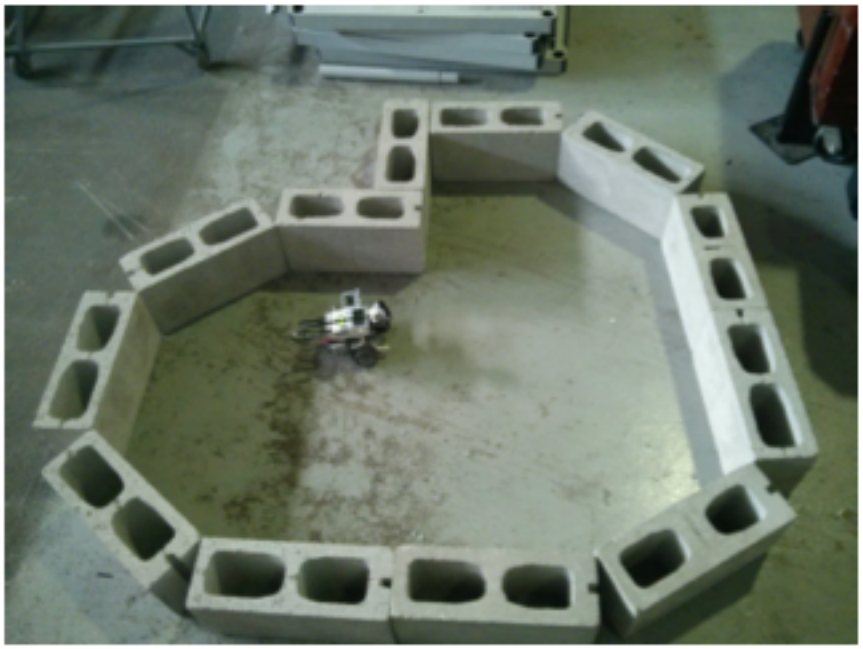
Mapping Software

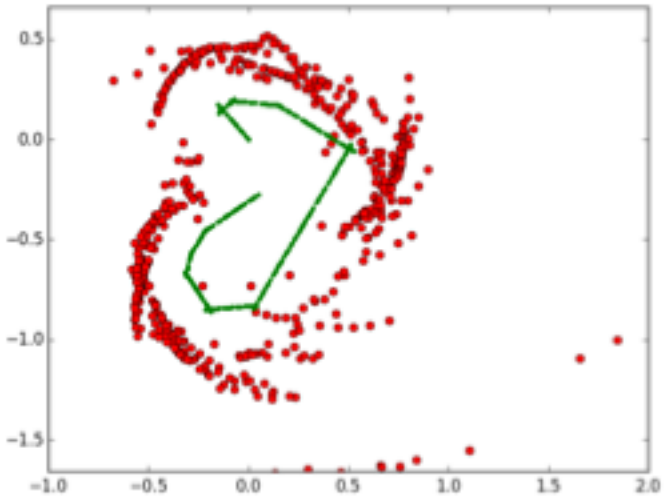
- Ev3.jl: low-level sensor and motor drivers
- Behaviors.jl: simple finite state machine behavior engine
- Mapping.jl: dead-recking state estimation and pointcloud collection

Results (Serial)



Results (Serial)



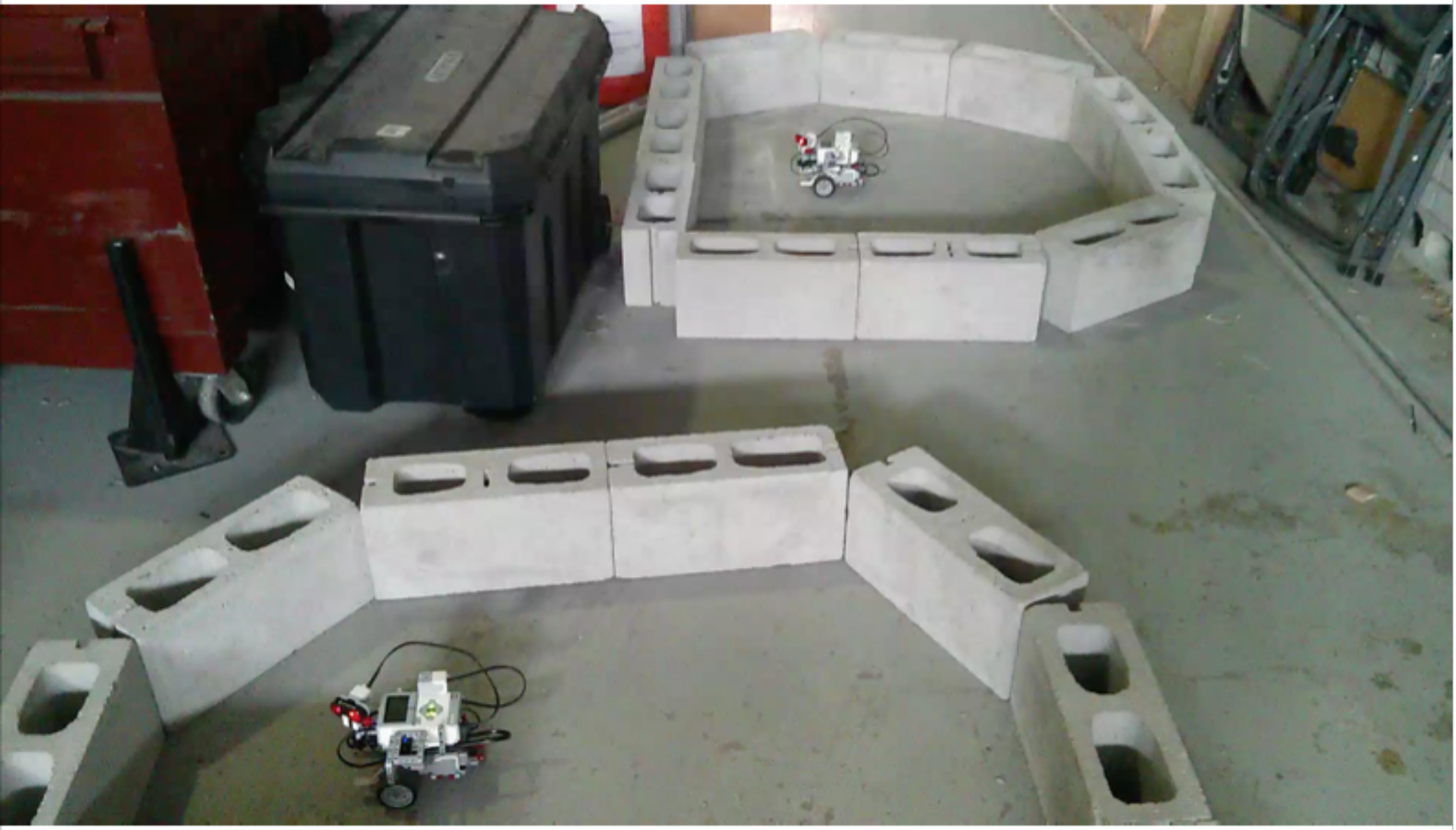


Parallel Mapping

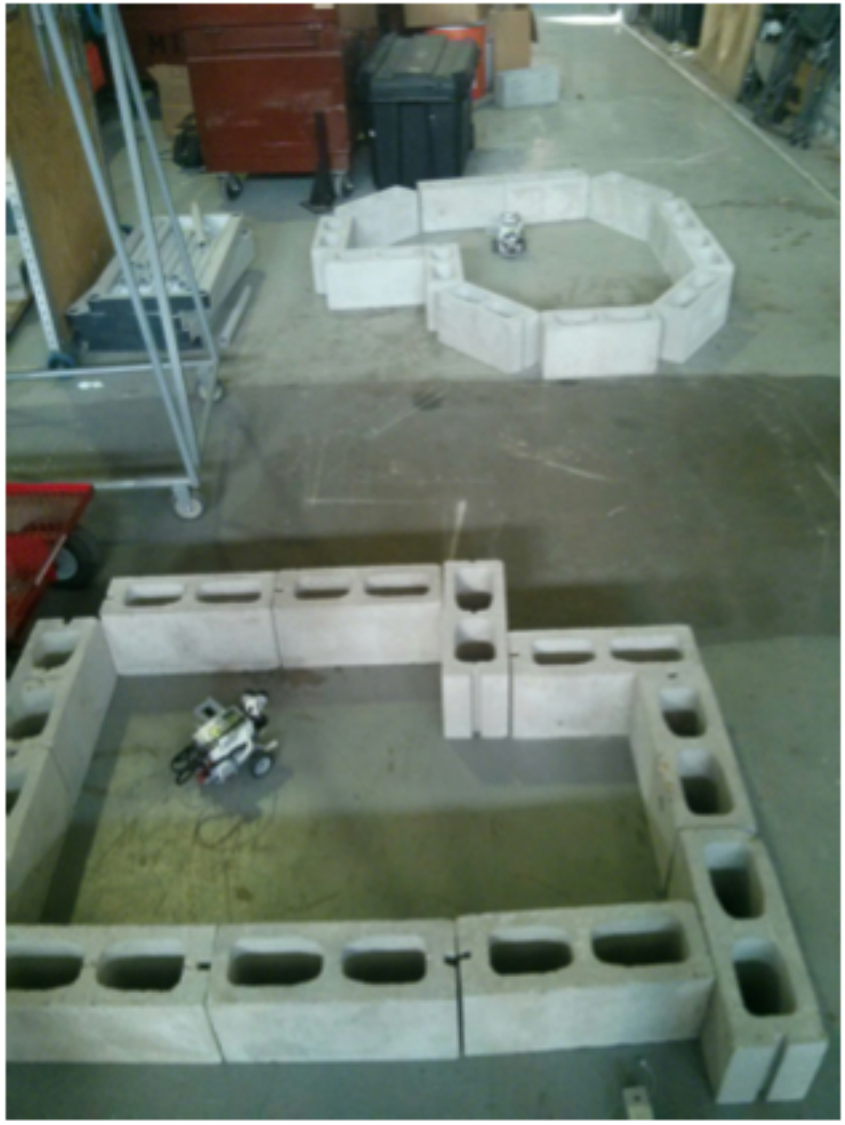
- It couldn't be easier:

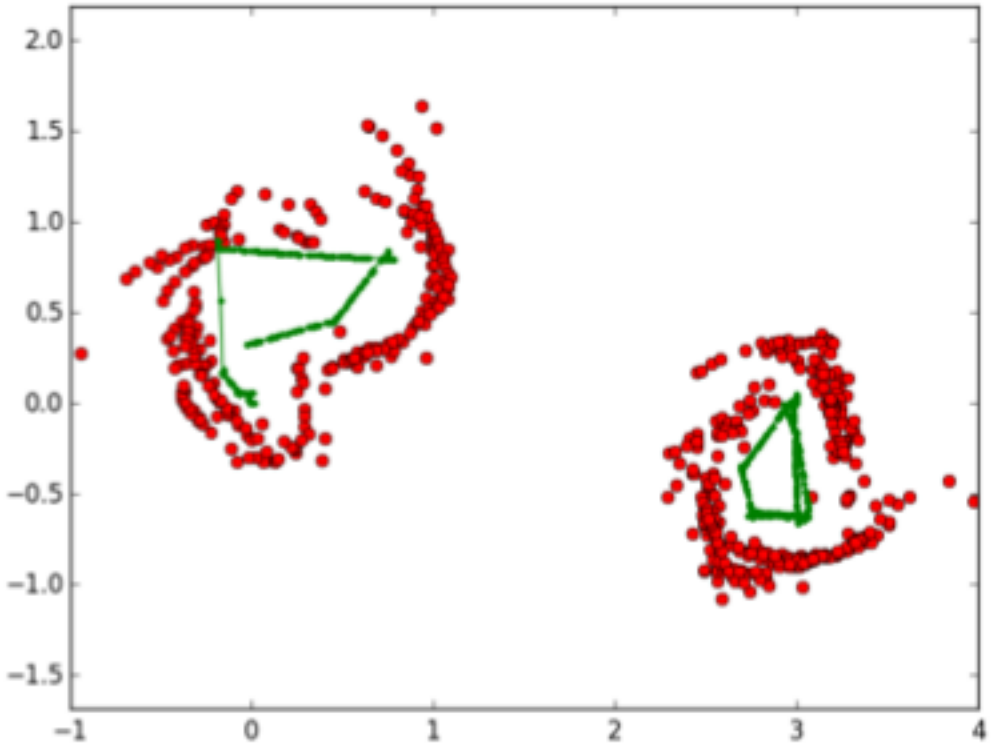
```
addprocs(2)
@everywhere hostnames = ["192.168.1.27", "192.168.1.25"]

maps = @sync @parallel (vcat) for i = 1:2
    robot = construct_robot(hostnames[i])
    run_mapping(robot)
end
```

Results (Parallel)



Results (Parallel)





Future Work

- Keep trying to build Julia for the EV3
- Refactor out clean Ev3.jl, Behaviors.jl, & Mapping.jl, and document them
- More exciting applications