Artificial Life Summer 2025

Subsumption Architecture Artificial Life Extras

Master Computer Science [MA-INF 4201]
Mon 14:15 – 15:45, HSZ, HS-2

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn

Subsumption Architecture, 1985, R.Brooks

Rodney A. Brooks (1985)

„ *A Robust Layered Control System for a Mobile Robot* “

A.I Memo 864, 1985
Massachusetts Institute of Technology
Artificial Intelligence Laboratory

http://www.uv.edu.mx/mia/ingreso/documents/Brooks.pdf

Behavior Based Robotics

R. Brooks defines a number of requirements of a control system for an intelligent autonomous mobile robot.
They each put constraints on possible control systems that one might build and employ.

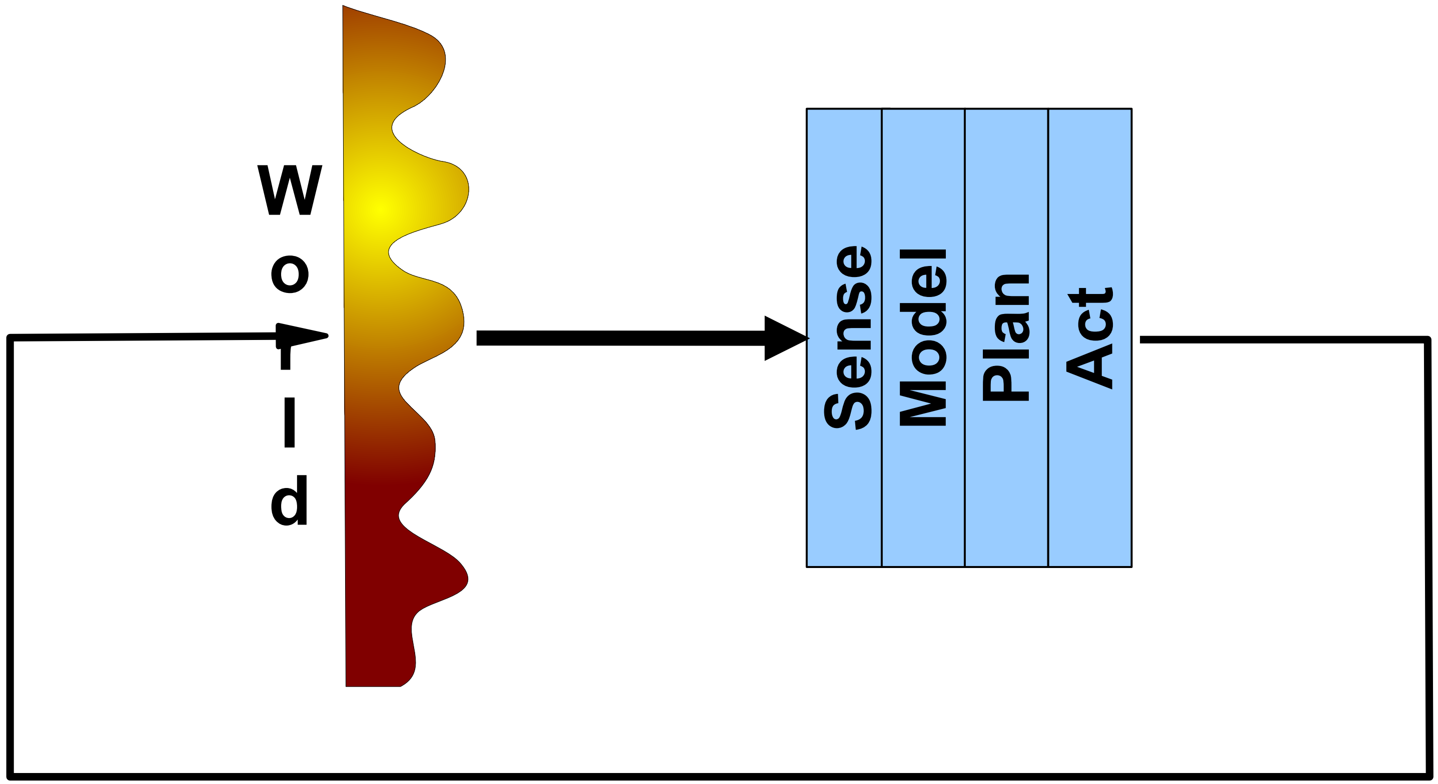
A robotics wish list for Santa Claus:

- Multiple Goals
- Multiple Sensors
- Robustness
- Additivity

See publication of R.Brooks (1985) for details.

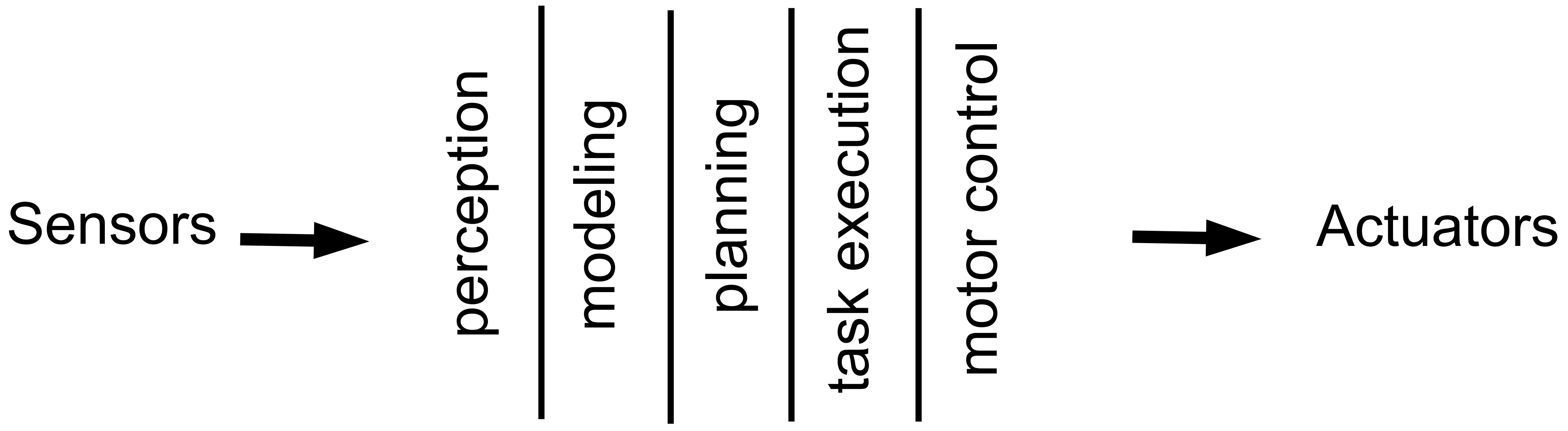
SMPA Architecture - reminder

The sensory input (**Sense**) obtained from the outer world is processed by the SMPA architecture onto commands controlling the robot/or the environment (**Act**).



SMPA Architecture - once more

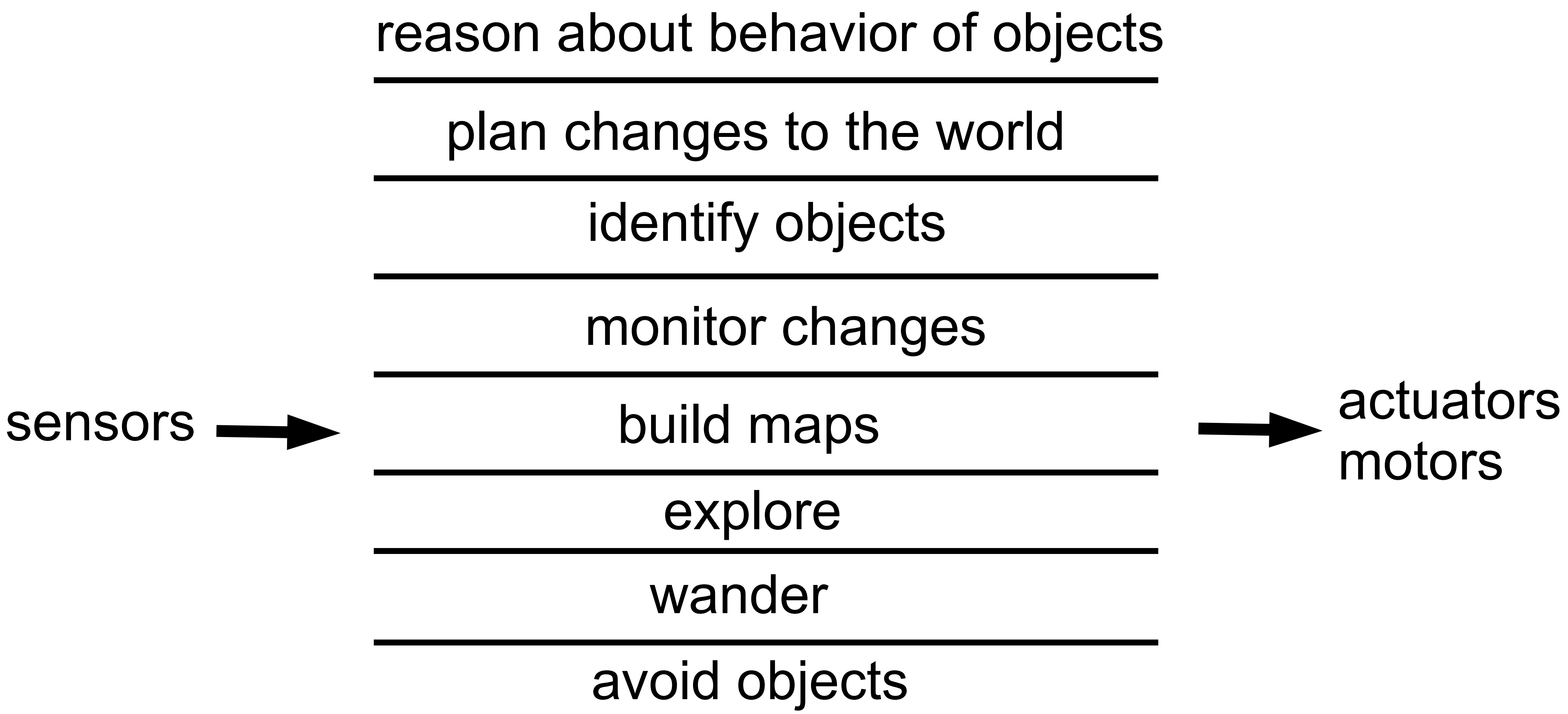
A traditional decomposition of a mobile robot control system into functional modules *(Brooks 1985)*



The classical SMPA architecture in a more detailed version:

Task Achieving Behaviors

A decomposition of a mobile robot control system based on task achieving behaviors *(Brooks 1985)*



Nine Dogmatic Principles

Brooks bases the design of the robot on **nine dogmatic principles**.

- Complex (and useful) behavior
- Things should be simple
- Map making is of crucial importance
- Three dimensional environment
- Relational maps for the robot
- No artificial environment for the robot, (no exact world model)
- Visual data for the robot (not just sonar ranger data)
- Self calibrating robot system, self calibration at all time
- Robots must be self sustaining

Levels of Competence, Levels of Behavior

The approach of Rodney Brooks decomposes the robot control problem into a number of ***levels of competence.***

A ***level of competence*** is an informal specification of a desired class of behaviors for the robot over all environments it will encounter.

A higher level of competence implies a more specified desired class of behaviors.

Thus the different levels of behavior are used to structure the robot controller.

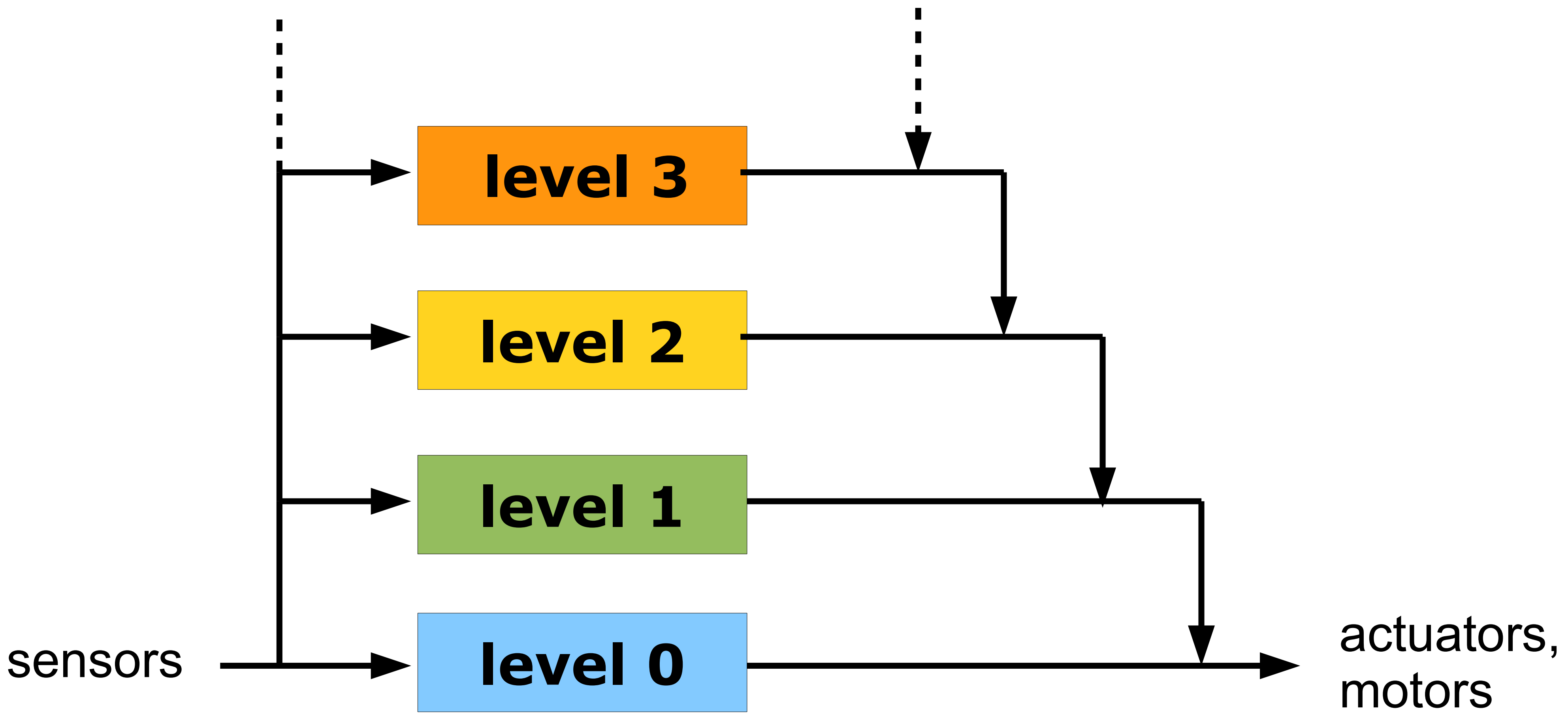
Levels of Competence, Levels of Behavior

The following 8 levels of competence were proposed (Brooks):

0. Avoid contact with objects
1. Wander aimlessly around (without hitting things)
2. Explore the world
3. Build a map, and plan routes within this map
4. Notice changes in this (static) environment
5. Reason about the world
6. Formulate and execute plans changing the world
7. Reason about the behavior of other objects in the world

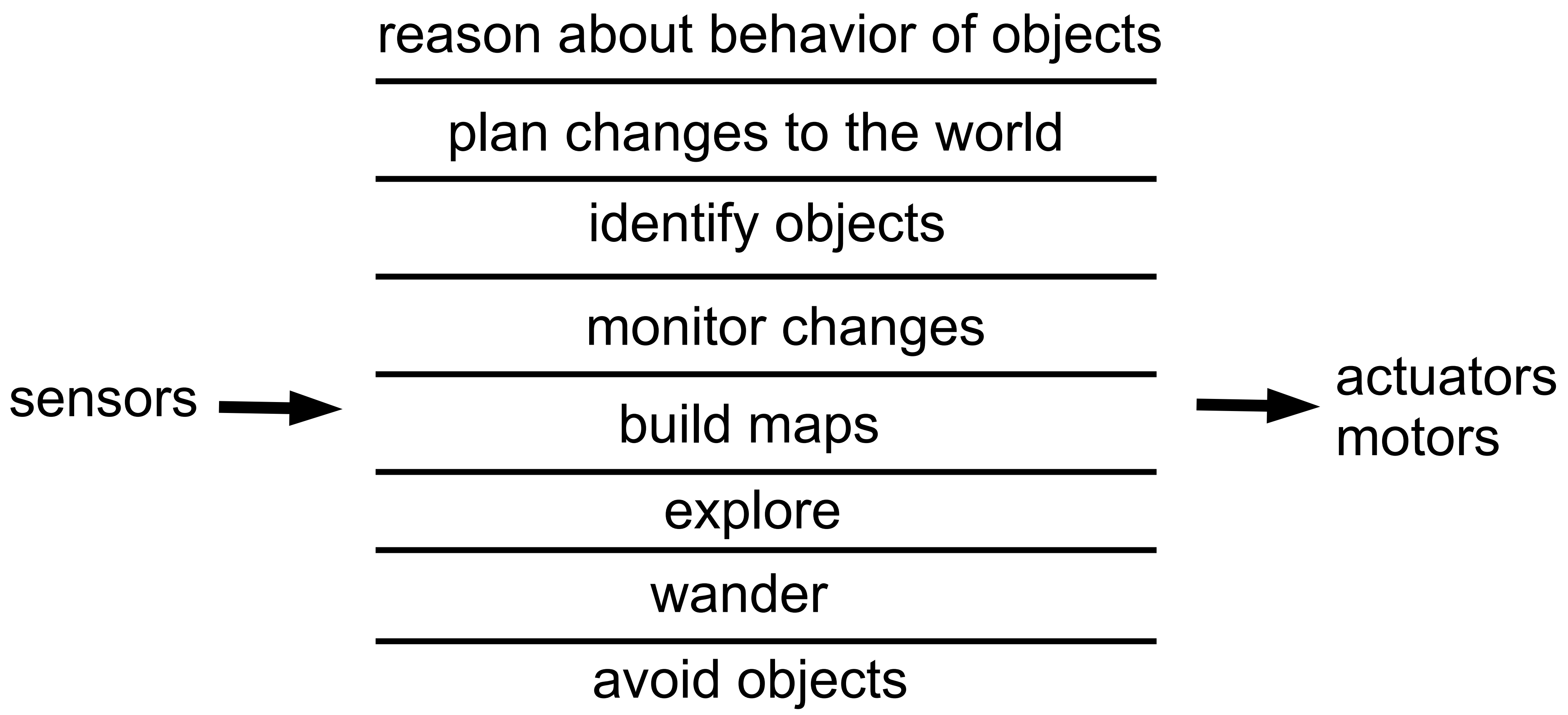
Layered Control

Control is layered with higher level layers subsuming the lower levels.



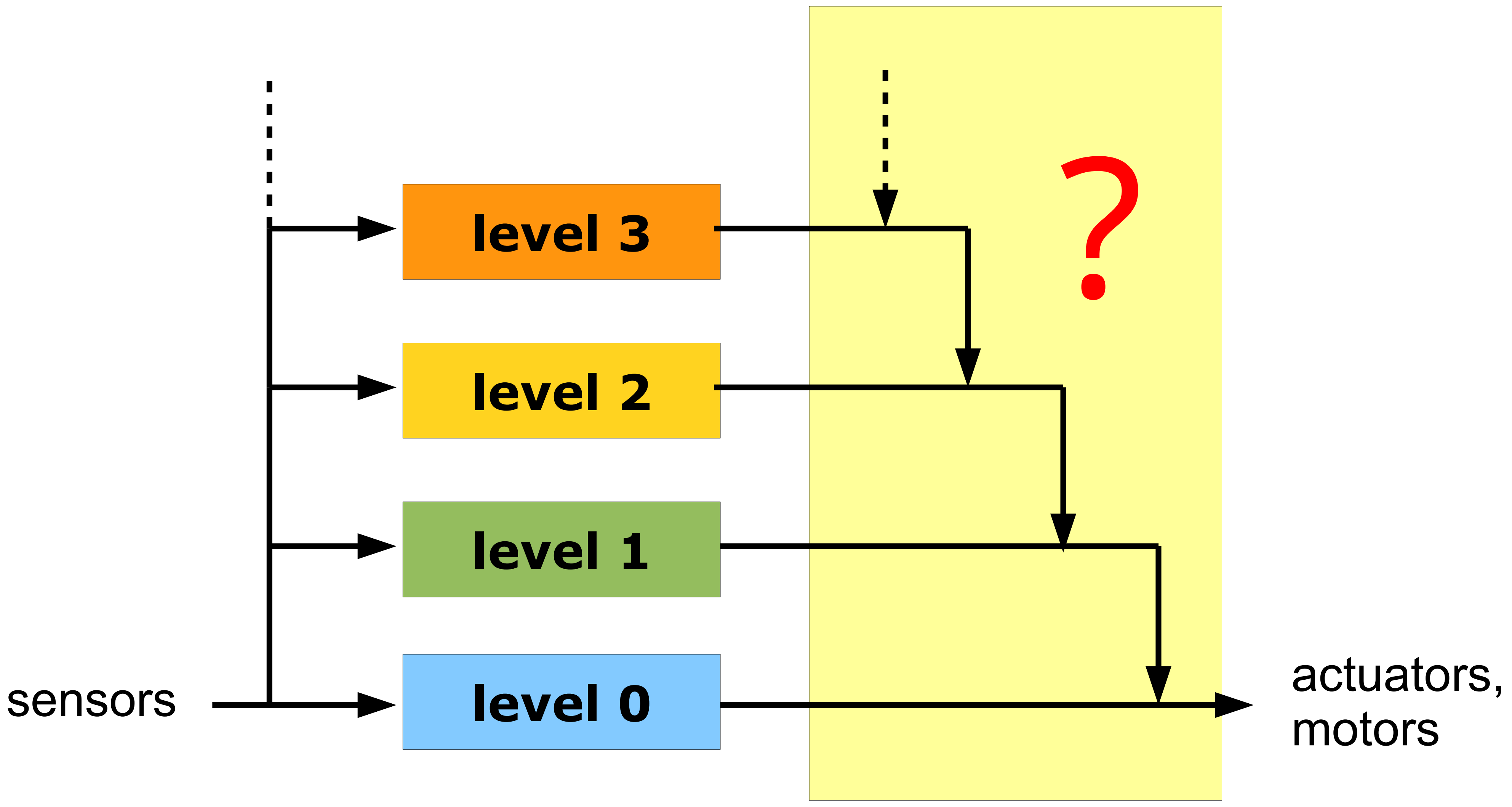
Task Achieving Behaviors

A decomposition of a mobile robot control system based on task achieving behaviors *(Brooks 1985)*



Layered Control

Control is layered with higher level layers subsuming the lower levels.



Subsumption Control

If necessary, or appropriate for the task, the higher level layers subsum the effect of the lower layers.

Thus, the higher level behaviors can dominate the lover level behaviors.

And as an additional positive effect:

If, in any case some part of this layered control structure is not producing any commands (busy, or damaged, or ...)
the lower levels would still be operational and would thus implement a working controller.

Implementing the Layers

The layers are build up following a modular approach. They consist of simple structured control units (modules).

The modules are as simple as a finite state machine.

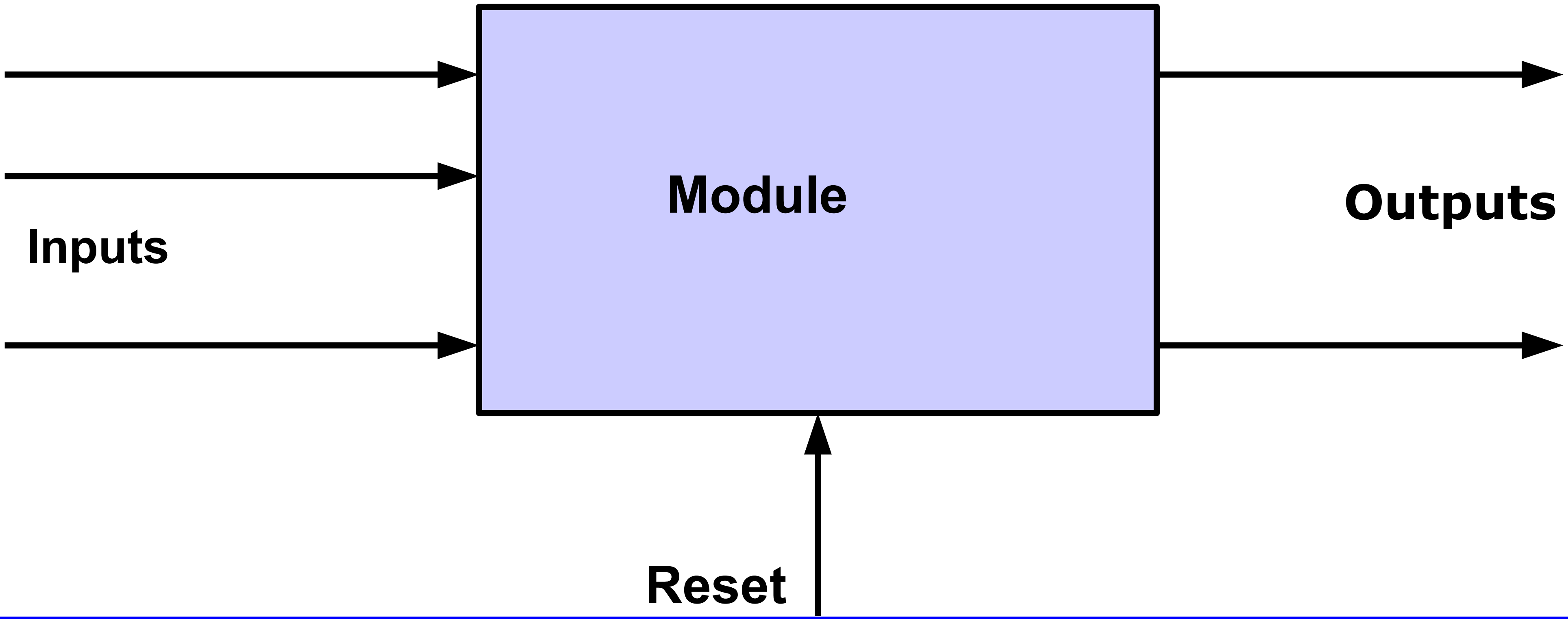
(In fact, R.Brooks has explicitly designed them to be finite state machines, for easily implementing them in hardware.)

A module has input and output lines connecting the modules among each other.

The interconnected modules combine to a network.

Structure of a Module

Each module is implemented as a **finite state machine** augmented with some instance variables

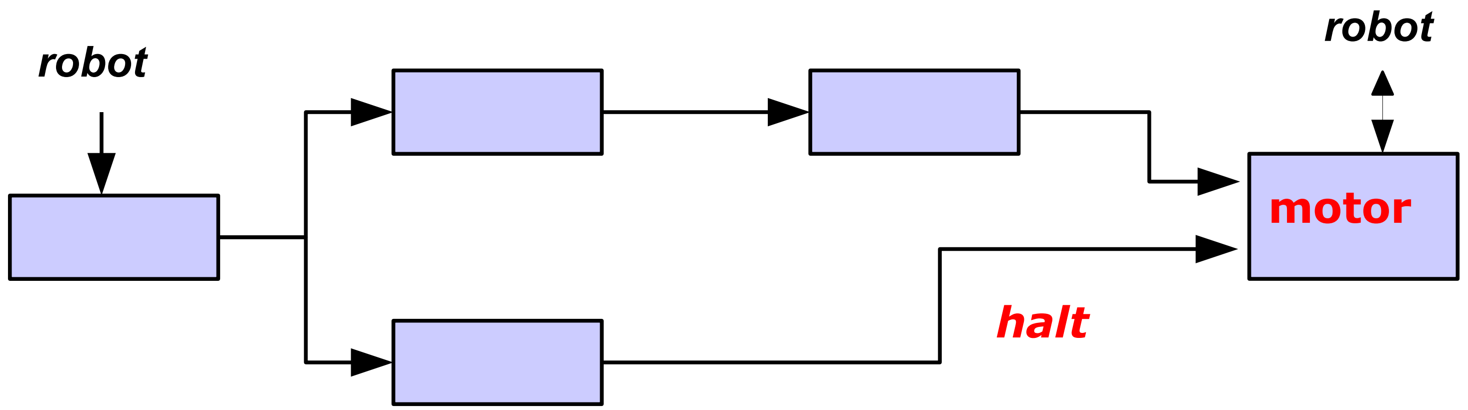


Modules are Connected

The output of one module can be connected to the input of one of the other modules.

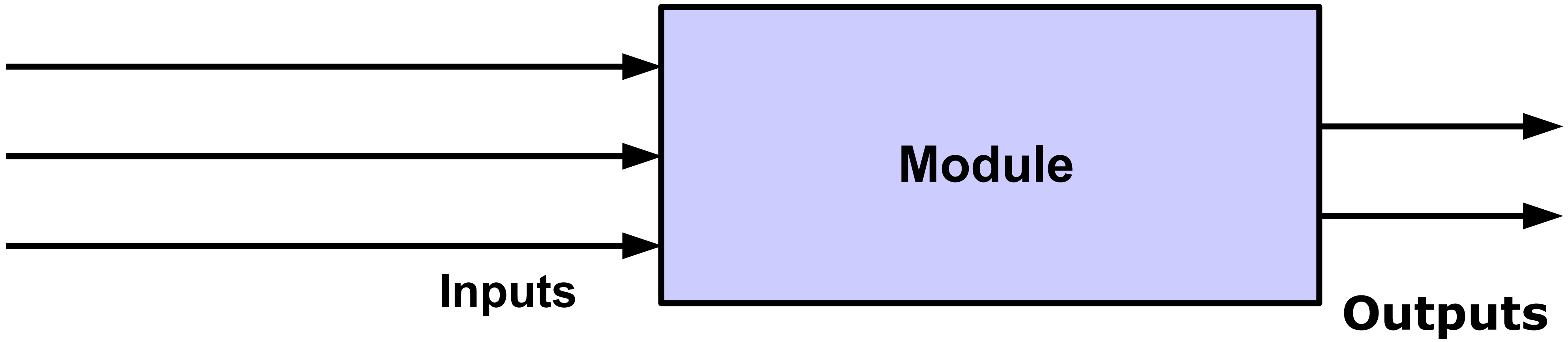
Each module is responsible for a special subtask:
e.g. ***motor*** control

The signals between the modules correspond to messages between the subtasks: e.g. ***halt***



Suppress and Inhibit

Beside the connection of input and output lines between the modules, the output of a module can alter the signal on another connection by 2 methods:

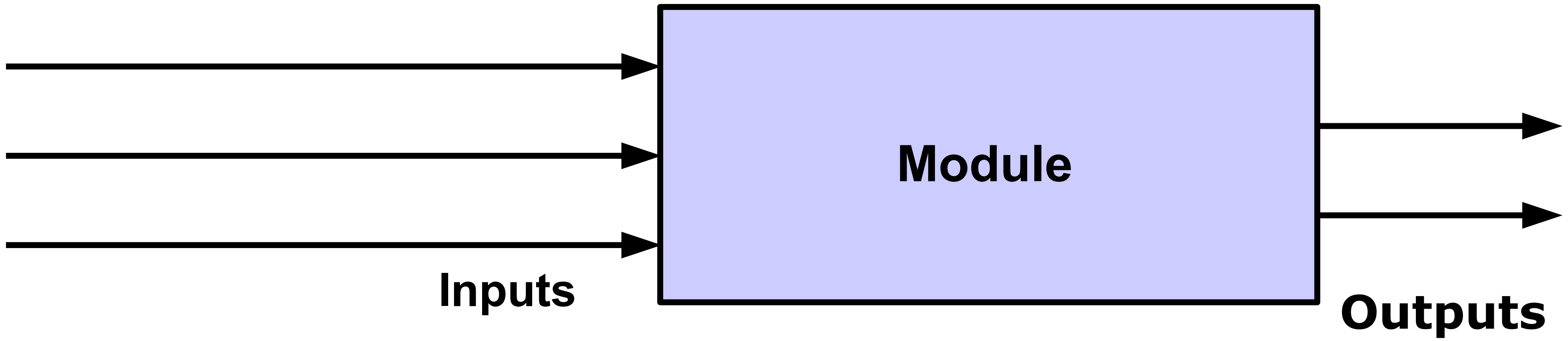


Suppress and Inhibit

Beside the connection of input and output lines between the modules, the output of a module can alter the signal on another connection by 2 methods:

Inhibiting:

Suppressing:

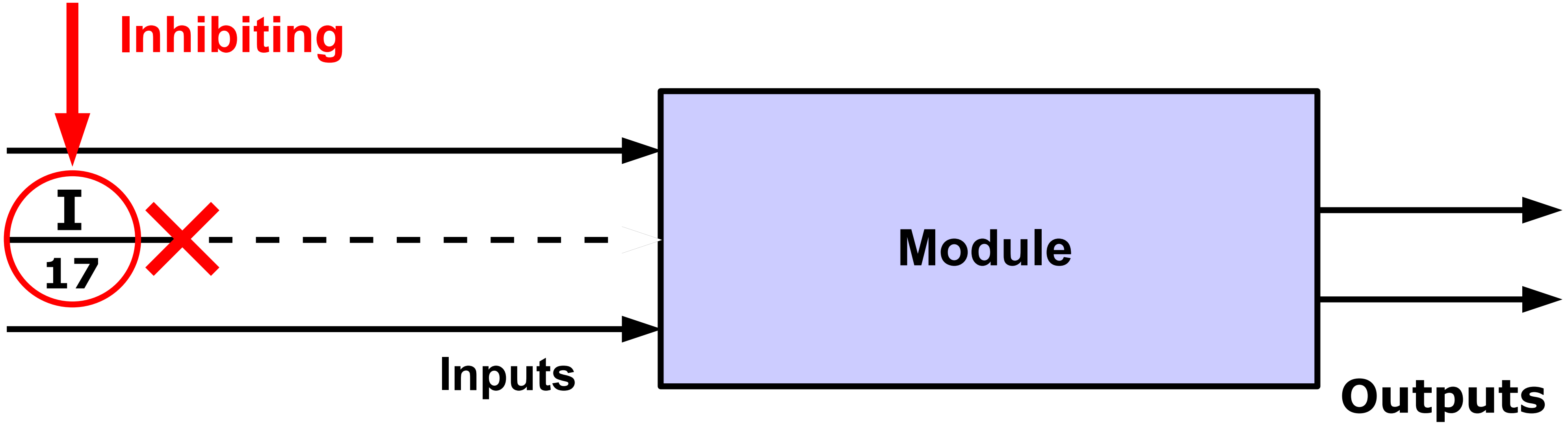


Suppress and Inhibit

Beside the connection of input and output lines between the modules, the output of a module can alter the signal on another connection by 2 methods:

Inhibiting: canceling signals for **z** time steps

Suppressing:

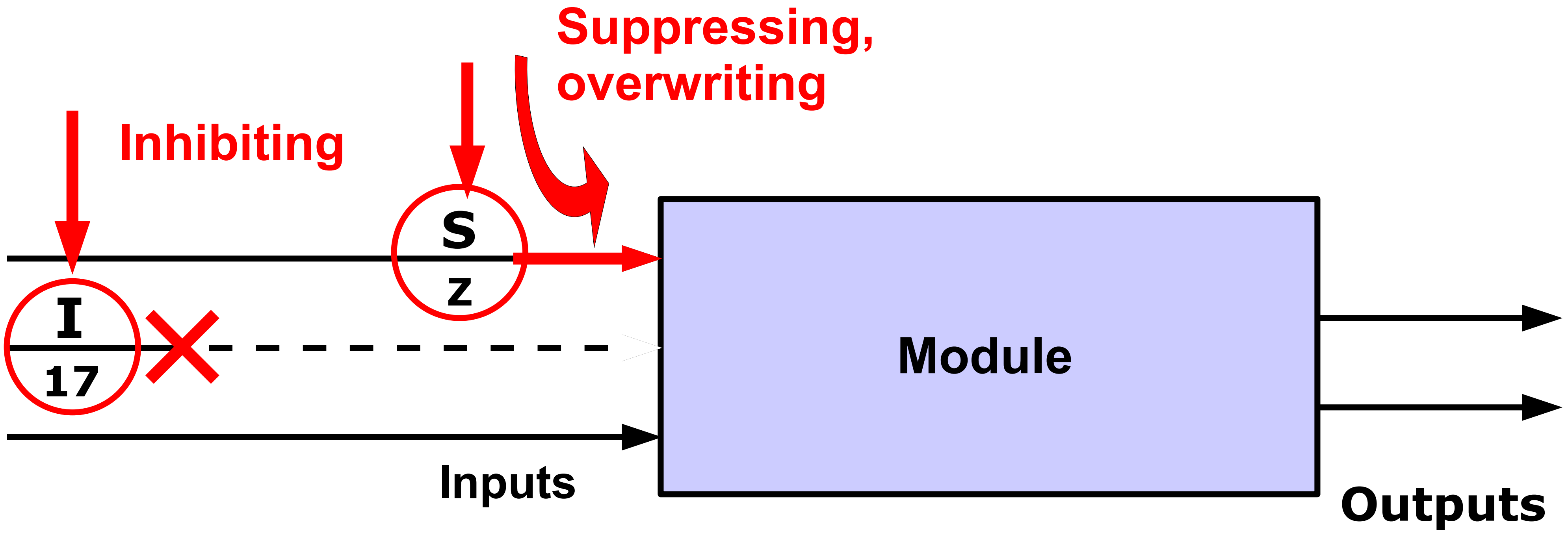


Suppress and Inhibit

Beside the connection of input and output lines between the modules, the output of a module can alter the signal on another connection by 2 methods:

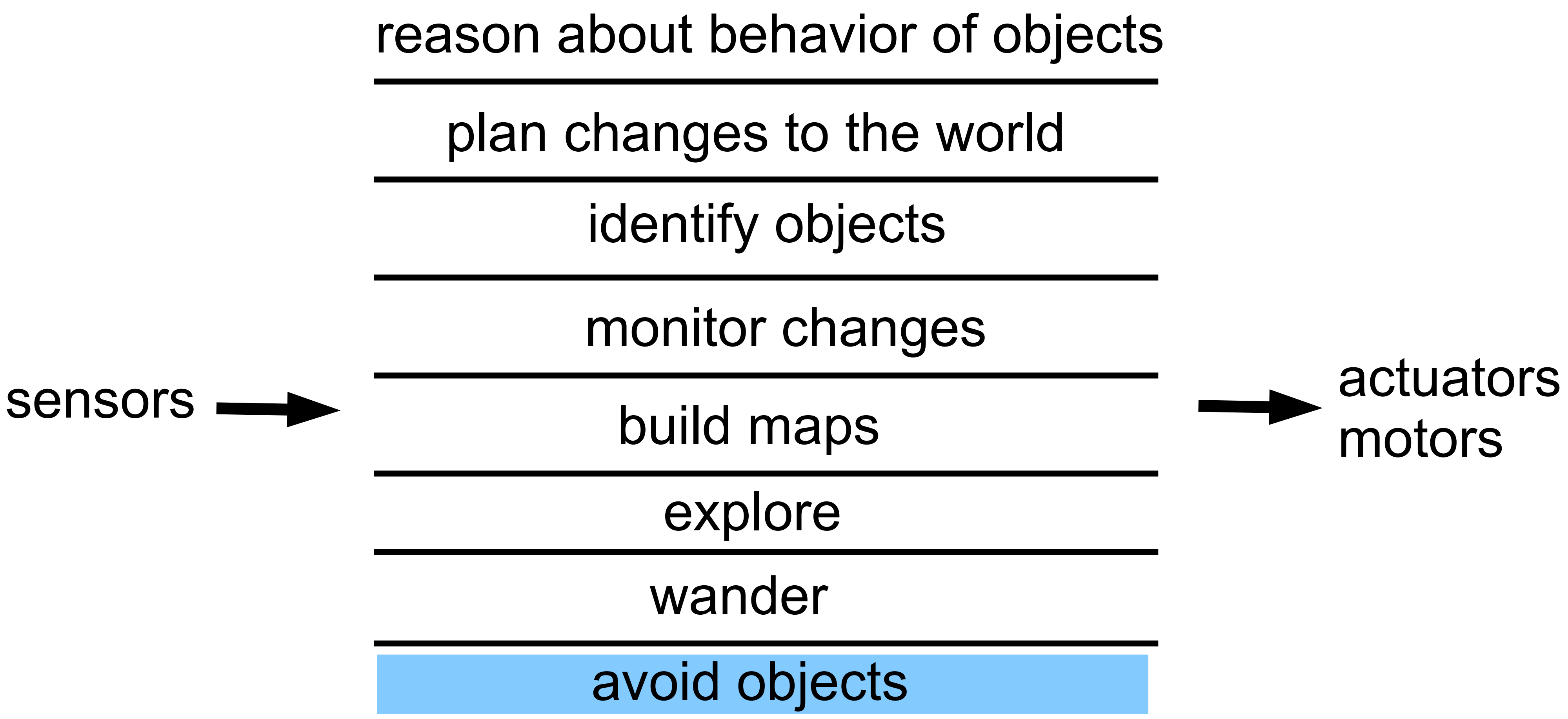
Inhibiting: canceling signals for **z** time steps.

Suppressing: overwriting signal lines for **z** time steps



Task Achieving Behaviors

A decomposition of a mobile robot control system based on task achieving behaviors *(Brooks 1985)*



Level 0: *Avoid Objects*

The designated behavior for level 0 is to:
Avoid the contact between the robot and any objects.

Level 0: *Avoid Objects*

The designated behavior for level 0 is to:
Avoid the contact between the robot and any objects.

Therefore, two mechanisms have been implemented:

Stop the robot if a collision is detected: **halt**

If something approaches the robot - move away: **runaway**

Level 0: *Avoid Objects*

The designated behavior for level 0 is to:
Avoid the contact between the robot and any objects.

Therefore, two mechanisms have been implemented:

Stop the robot if a collision is detected: **halt**

If something approaches the robot - move away: **runaway**

sonar
robot
motor
robot

Level 0: *Avoid Objects*

sonar:
the sonar module takes a vector of sonar readings, filters them for invalid readings, and effectively produces a robot centered map of obstacles in polar coordinates.

sonar
robot
motor
robot

Level 0: *Avoid Objects*

motor:
the motor module communicates with the actual robot.
The communication is going in both ways, so the motor controller can be implemented as closed loop control.
It accepts a command specifying angle and turning velocity, magnitude of forward movement and velocity.

sonar
robot
motor
robot

Level 0: *Avoid Objects*

The designated behavior for level 0 is to:
Avoid the contact between the robot and any objects.

Therefore, two mechanisms have been implemented:

Stop the robot if a collision is detected: **halt**

If something approaches the robot - move away: **runaway**

sonar
robot
motor
robot

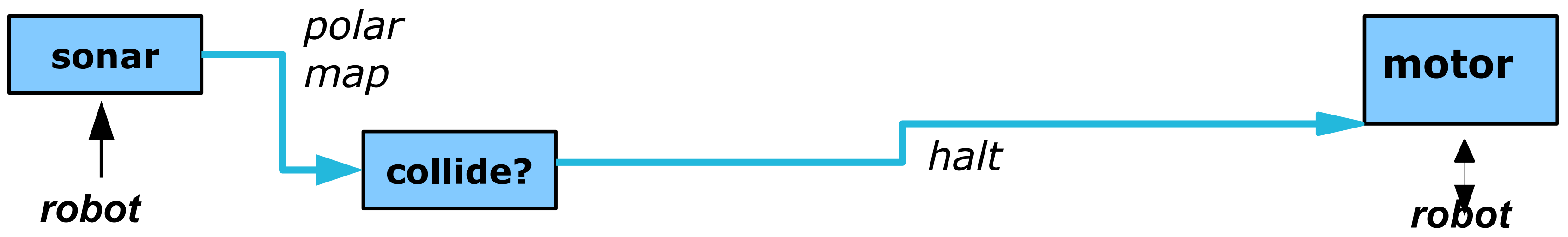
Level 0: *Avoid Objects*

The designated behavior for level 0 is to:
Avoid the contact between the robot and any objects.

Therefore, two mechanisms have been implemented:

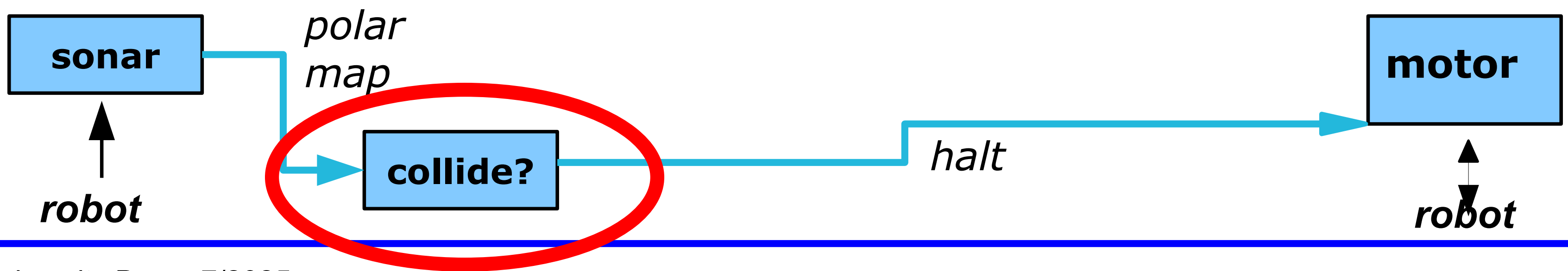
Stop the robot if a collision is detected: **halt**

If something approaches the robot - move away: **runaway**



Level 0: *Avoid Objects*

collide?:
the collide module monitors the sonar map and if it detects objects dead ahead it sends a signal on the halt line to the motor module. The collide module does not know or care whether the robot is moving Halt messages sent while the robot is stationary are essentially lost.



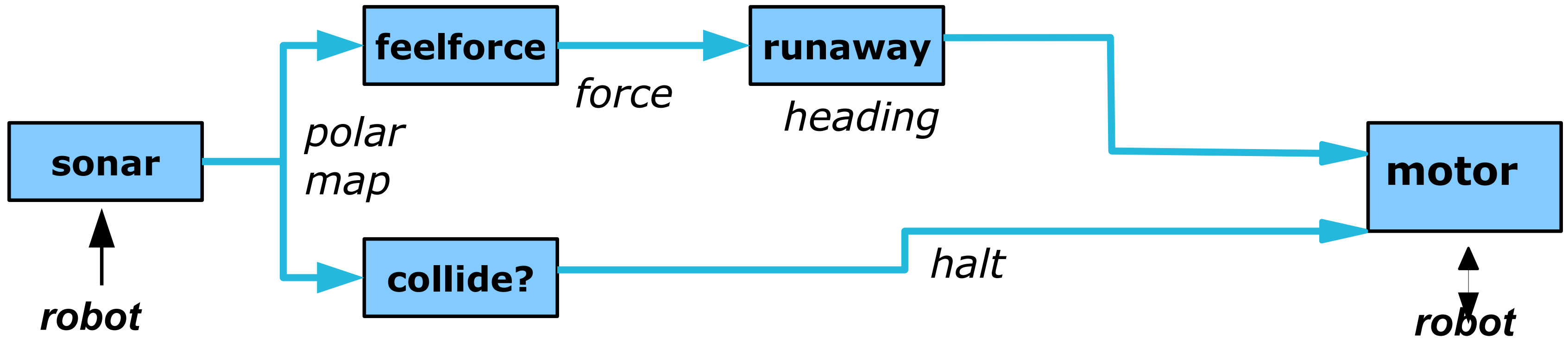
Level 0: *Avoid Objects*

The designated behavior for level 0 is to:
Avoid the contact between the robot and any objects.

Therefore, two mechanisms have been implemented:

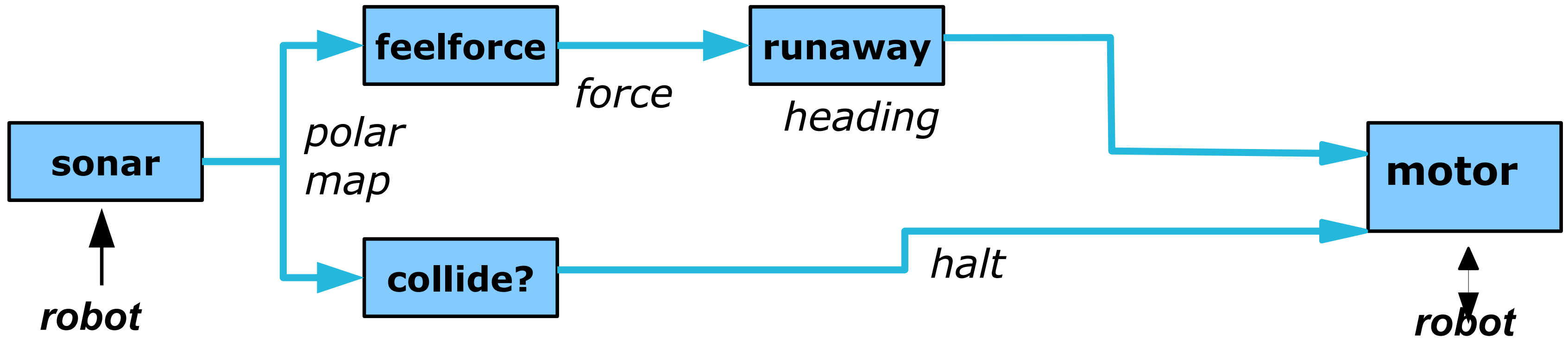
Stop the robot if a collision is detected: **halt**

If something approaches the robot - move away: **runaway**



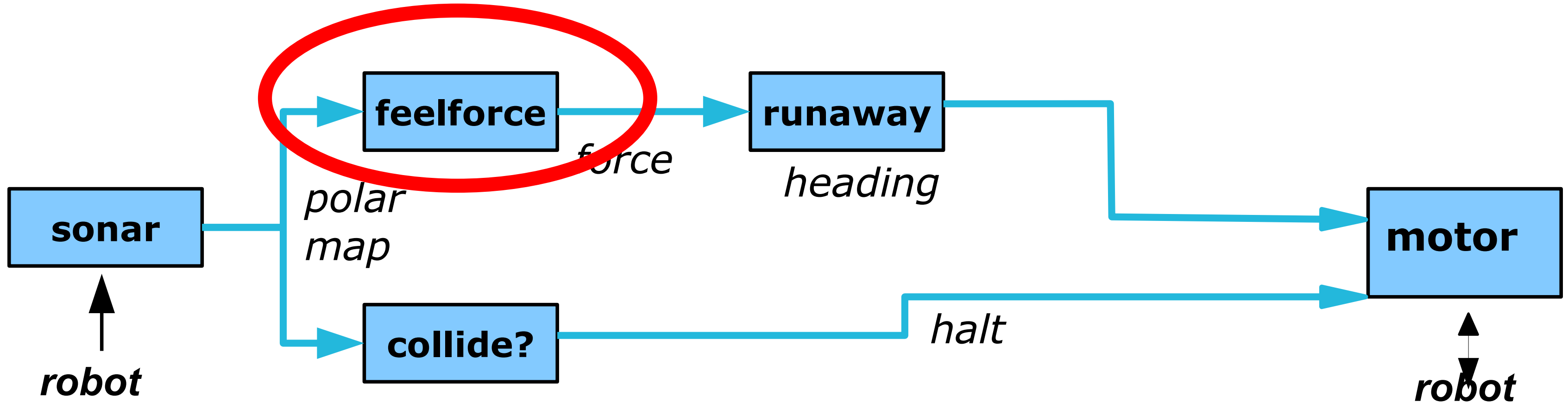
Level 0: *Avoid Objects*

sonar:
the sonar module takes a vector of sonar readings, filters them for invalid readings, and effectively produces a robot centered map of obstacles in polar coordinates.



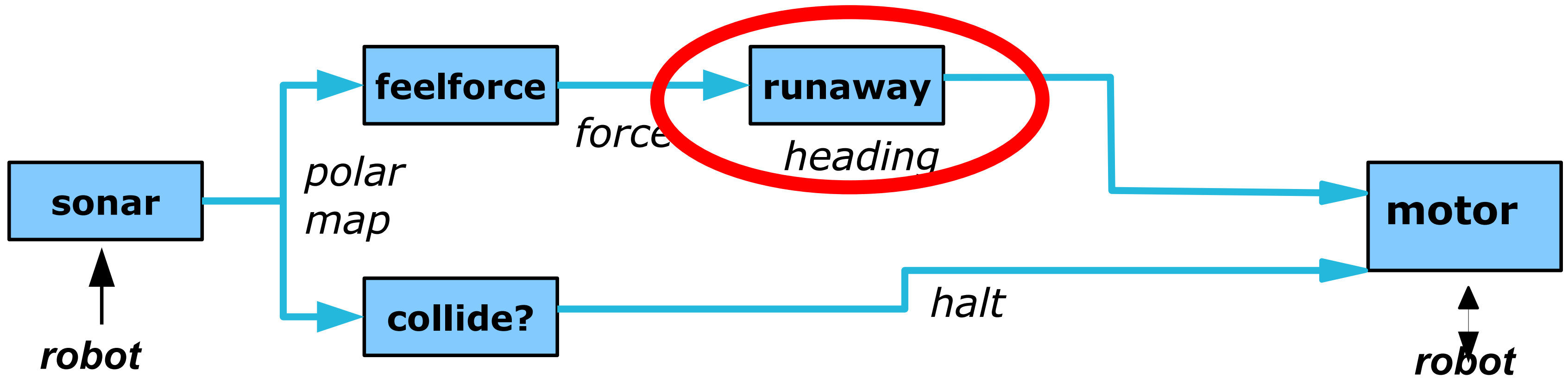
Level 0: *Avoid Objects*

feelforce:
the feelforce module sums the results of considering each detected object as a repulsive force, generating a simple resultant force.



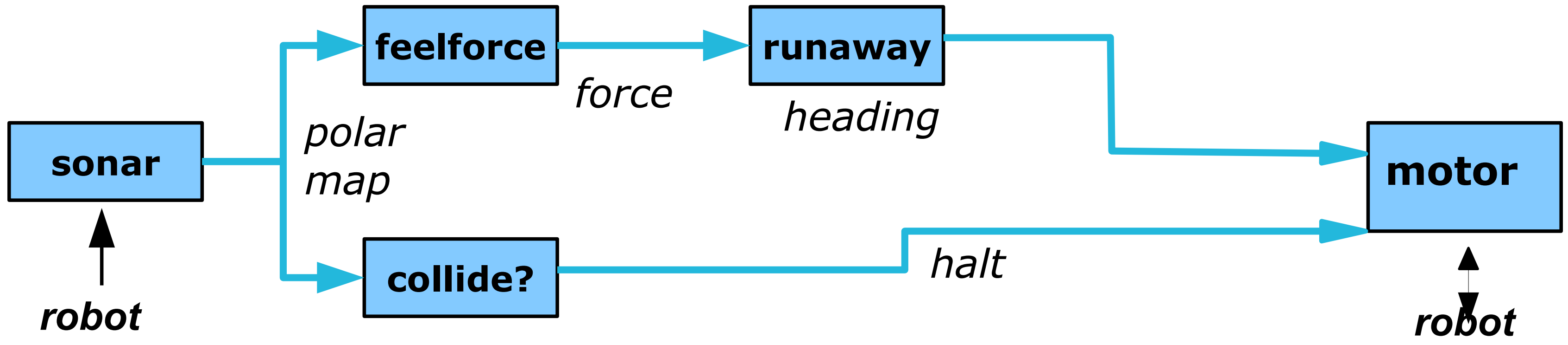
Level 0: *Avoid Objects*

runaway:
the runaway module monitors the force produced by the sonar detected obstacles and sends command (e.g. heading) to the motor module if it ever becomes significant.



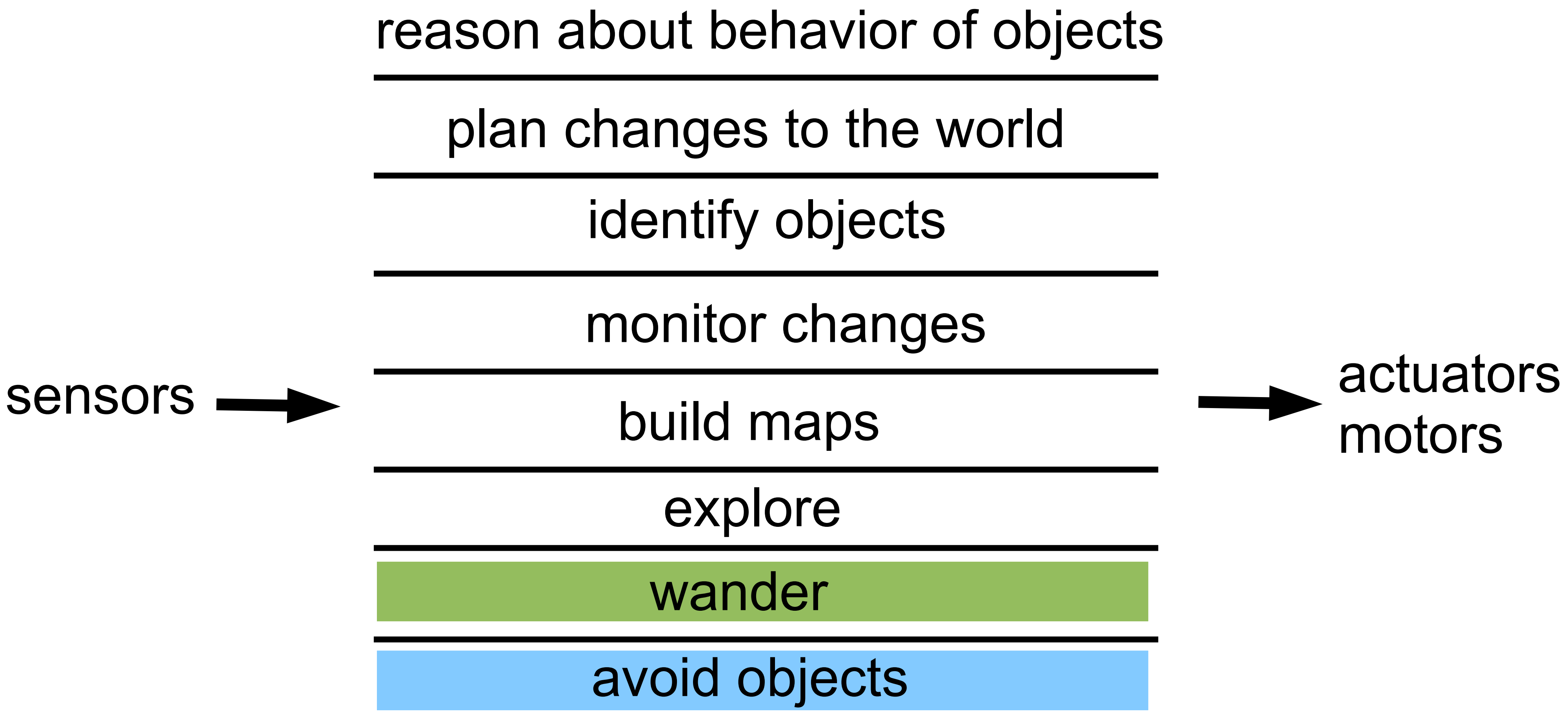
Level 0: *Avoid Objects*

motor:
the motor module communicates with the actual robot.
The communication is going in both ways, so the motor controller can be implemented as closed loop control.
It accepts a command specifying angle and turning velocity, magnitude of forward movement and velocity.



Task Achieving Behaviors

A decomposition of a mobile robot control system based on task achieving behaviors *(Brooks 1985)*



Level 1: *Wander*

The designated behavior for **level 1** is to:

Wander around

The first level layer of control, when combined with level 0, imbues the robot with the ability to wander around aimlessly without hitting obstacles.

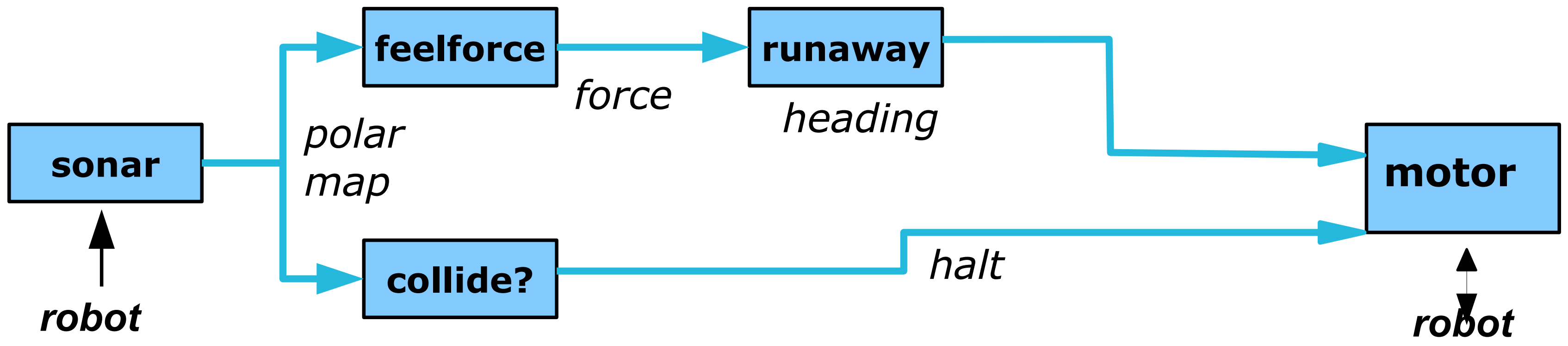
This control level relies in a large degree on the level 0 aversion to hitting obstacles.

Level 1: *Wander*

The designated behavior for **level 1** is to:

Wander around

The first layer level of control will add new modules to the existing layer 0.



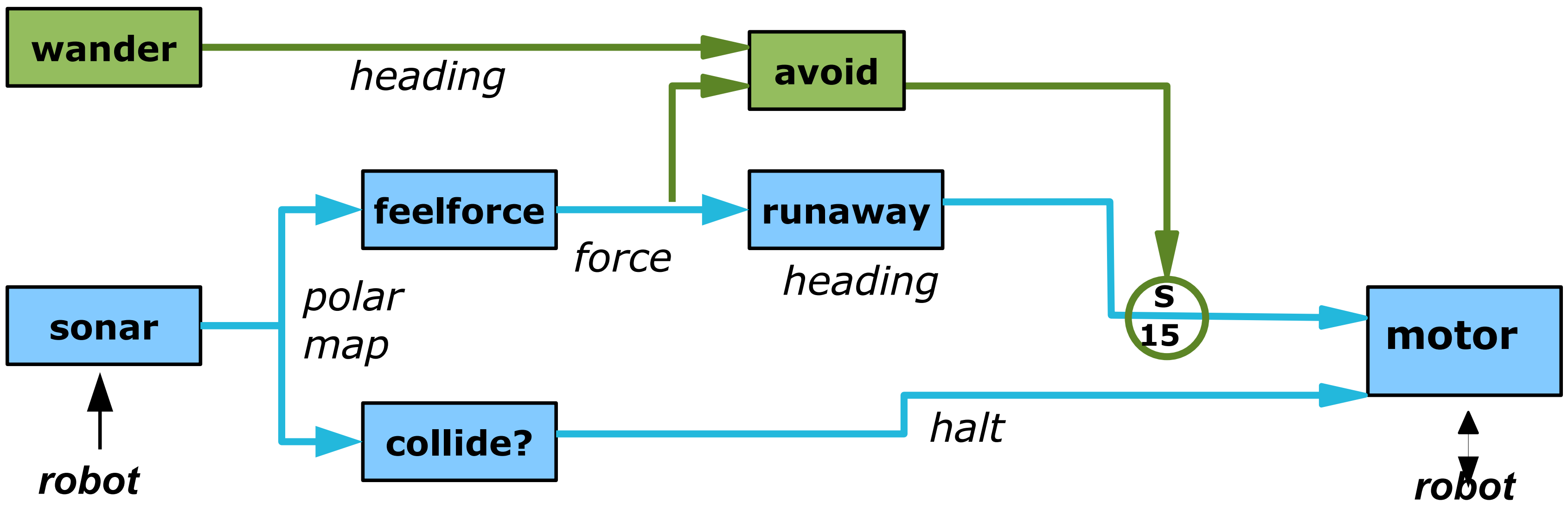
Level 1: *Wander*

The designated behavior for **level 1** is to:

Wander around

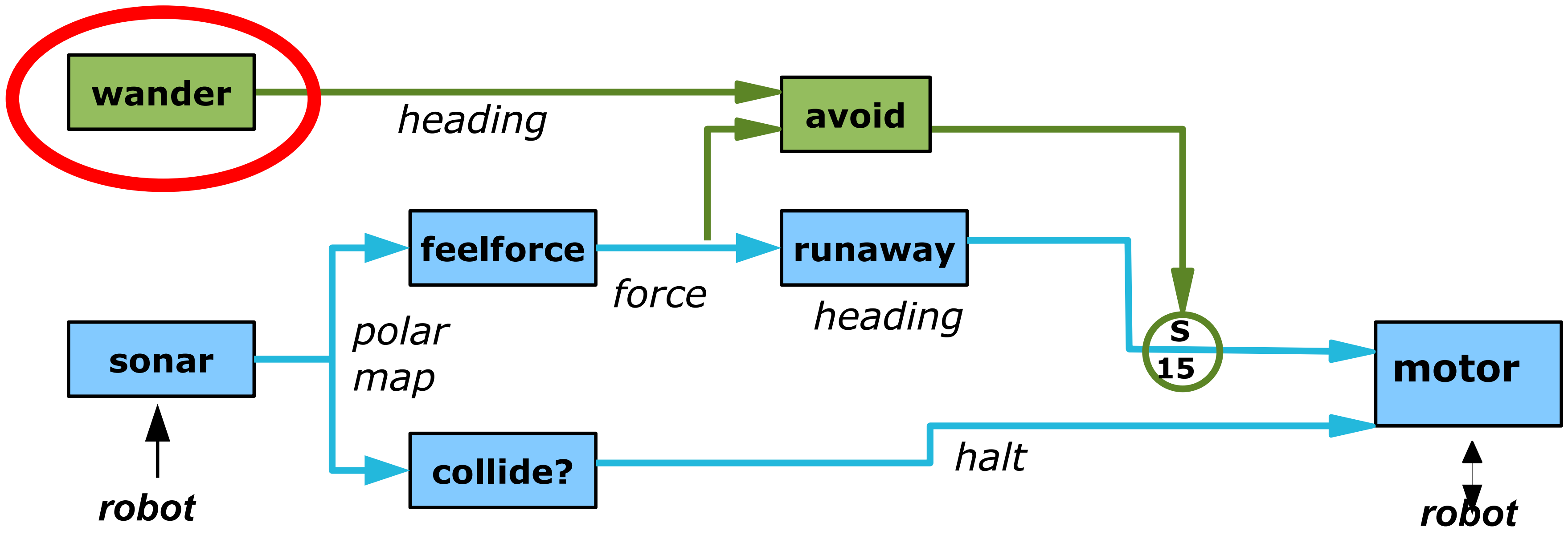
The first layer level of control will add new modules to the existing layer 0.

Level 1 modules: **wander** and **avoid**



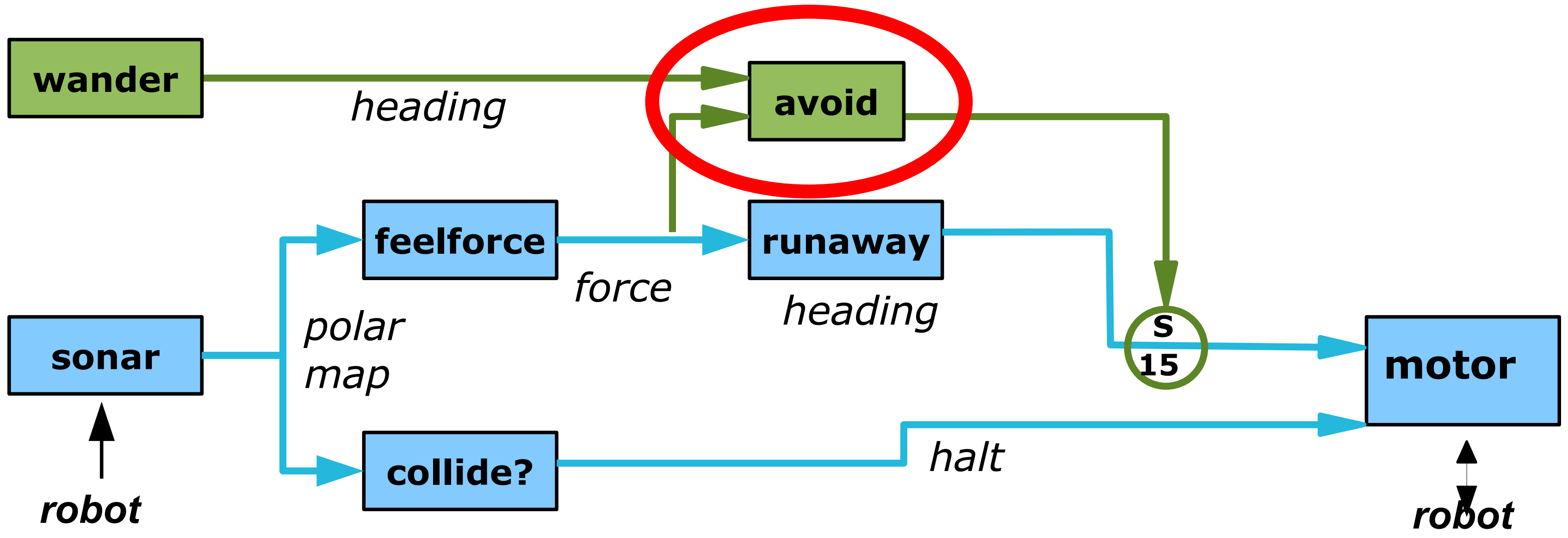
Level 1: *Wander*

wander:
the wander module generates a new heading for the robot in a fixed time interval (e.g. every 10 seconds or so).



Level 1: *Wander*

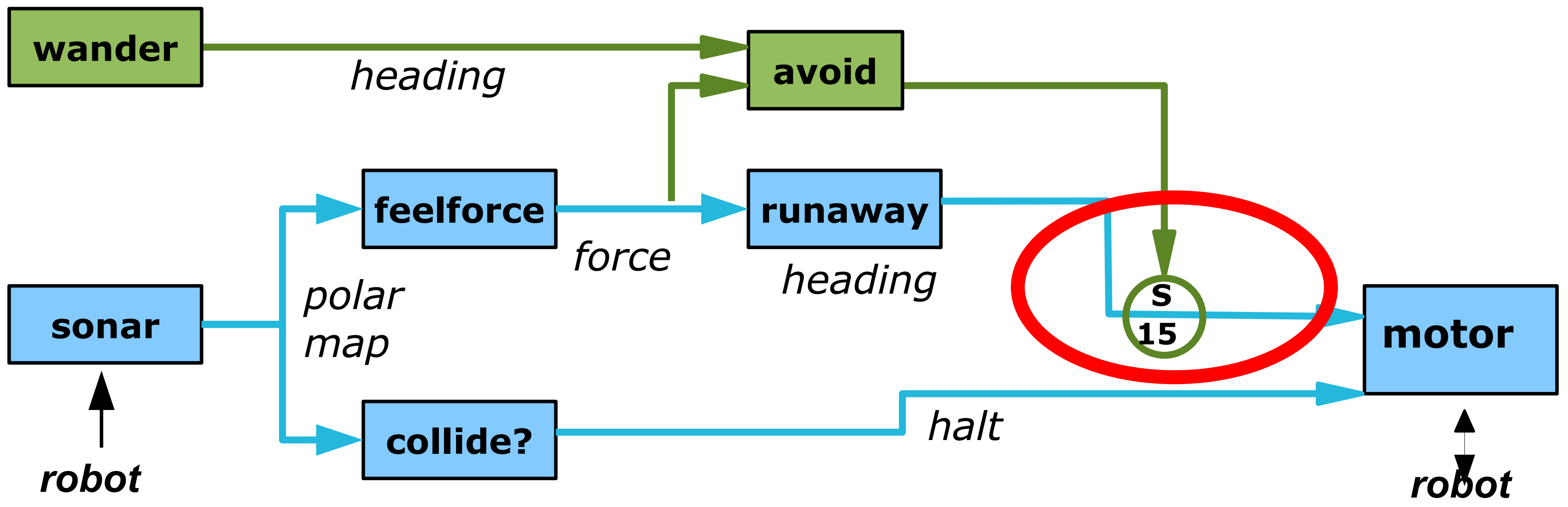
avoid:
the avoid module takes the result of the force computation from level 0, and combines it with the desired heading to produce a modified heading which usually points in roughly the correct direction, but perturbed to avoid any obvious obstacles.



Level 1: *Wander*

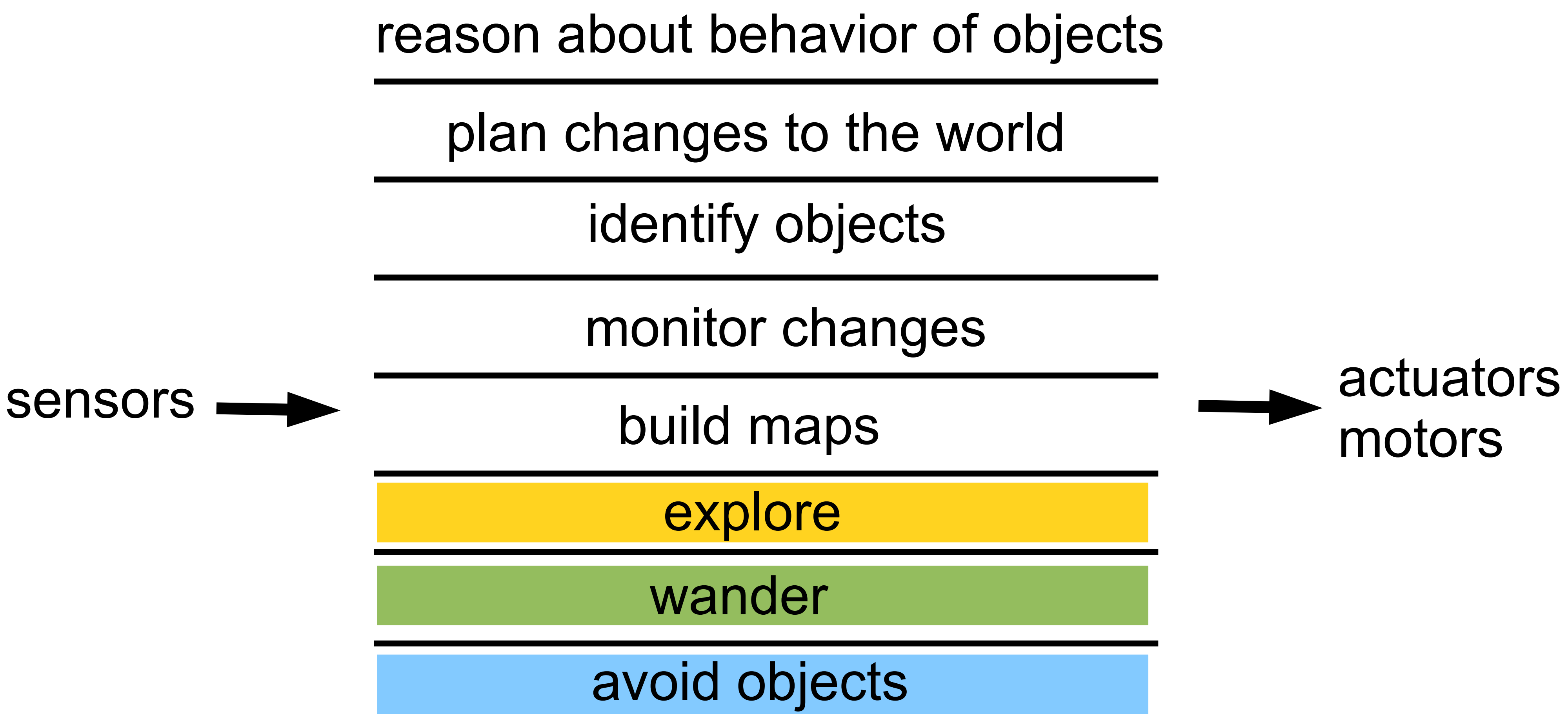
avoid:
the output form the avoid module suppresses the output from the runaway module of level 0, before entering the motor module.

The signal from level 0 is suppressed (overwritten, replaced) by the level 1 commands for 15 time steps



Task Achieving Behaviors

A decomposition of a mobile robot control system based on task achieving behaviors *(Brooks 1985)*



Level 2: *Explore*

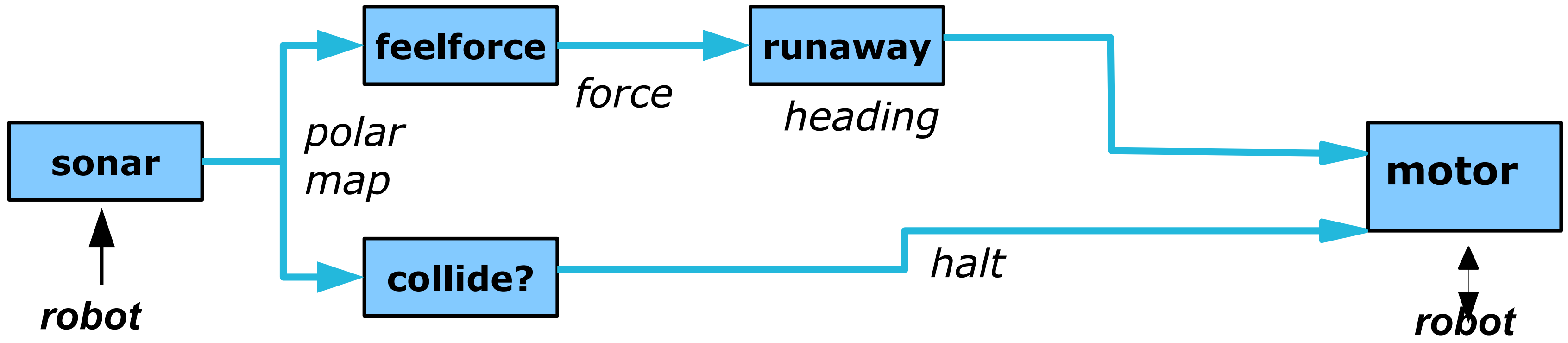
The designated behavior for **level 2** is to:

Explore the environment

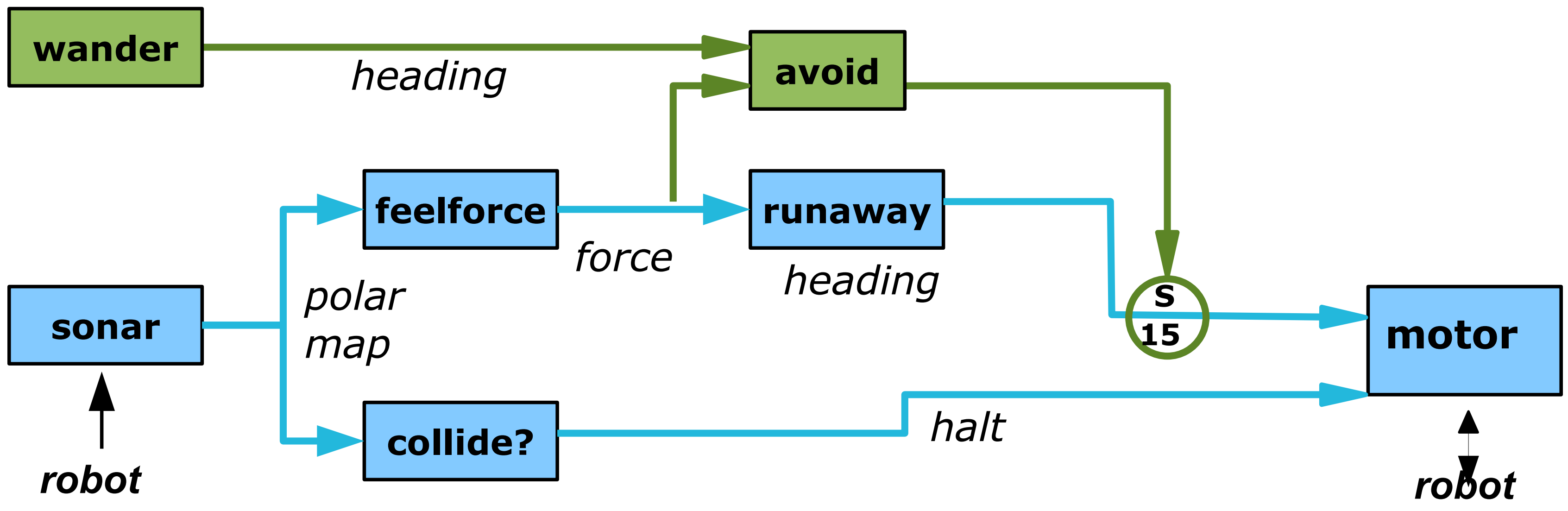
Level 2 is meant (by R. Brooks) to add an exploratory mode of behavior to the robot, using visual observations to select interesting places to visit.
(At the time of the publication in 1985, R.Brooks could only implement the non-vision aspects of this level)

A total of 5 new modules, 10 new signals, 17 connections, 4 places of signal inhibition and 3 places of signal supression are added in level 2 to the existing level 0 and level 1.

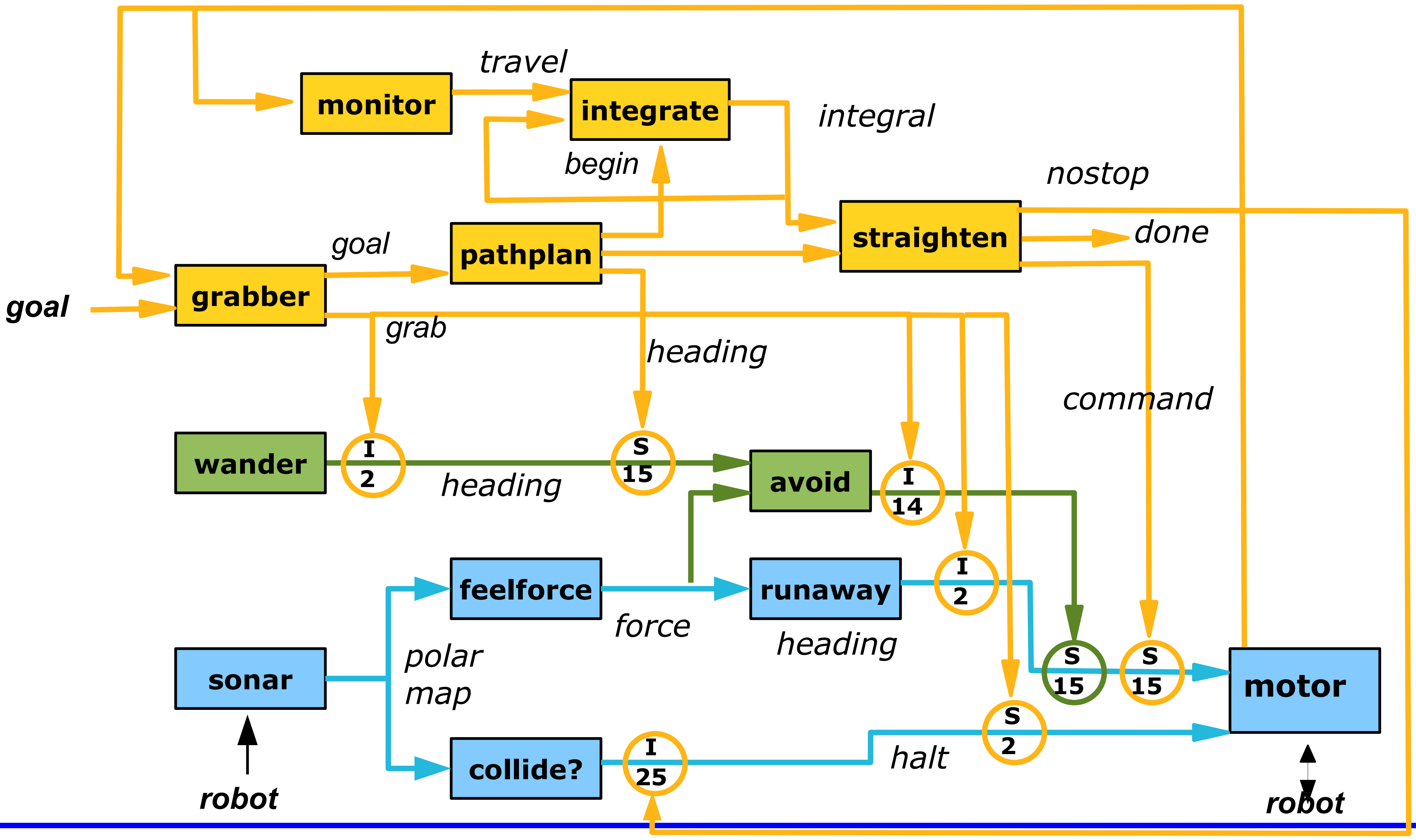
Level 2: *Explore*



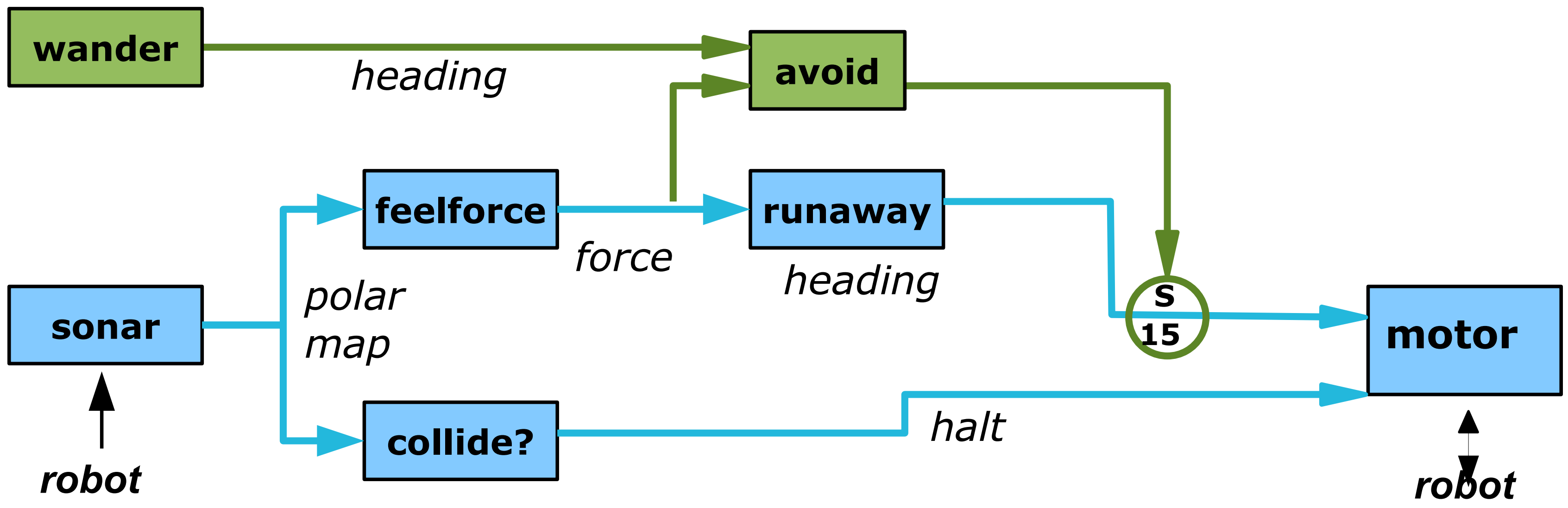
Level 2: *Explore*



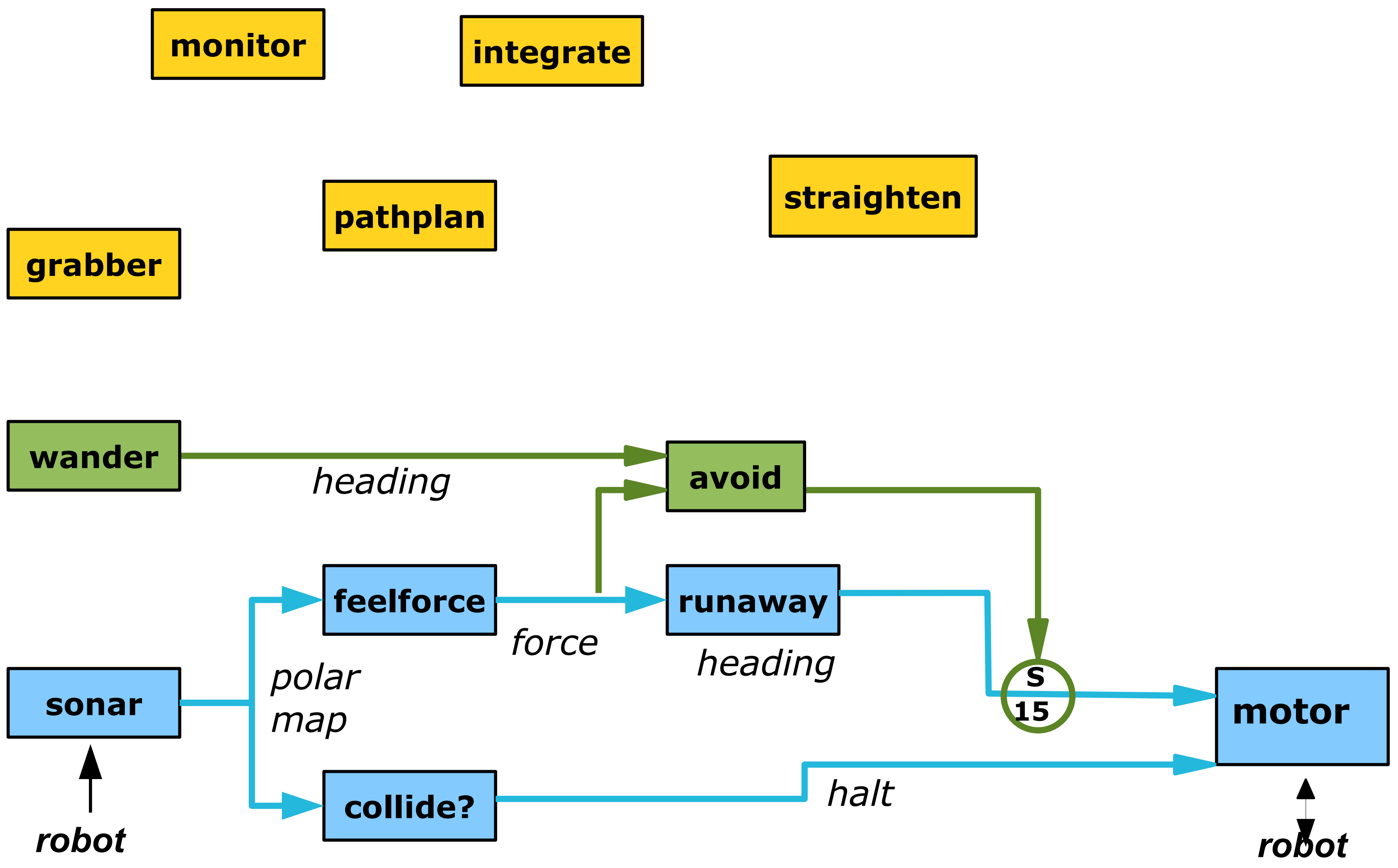
Level 2: *Explore*



Level 2: *Explore*

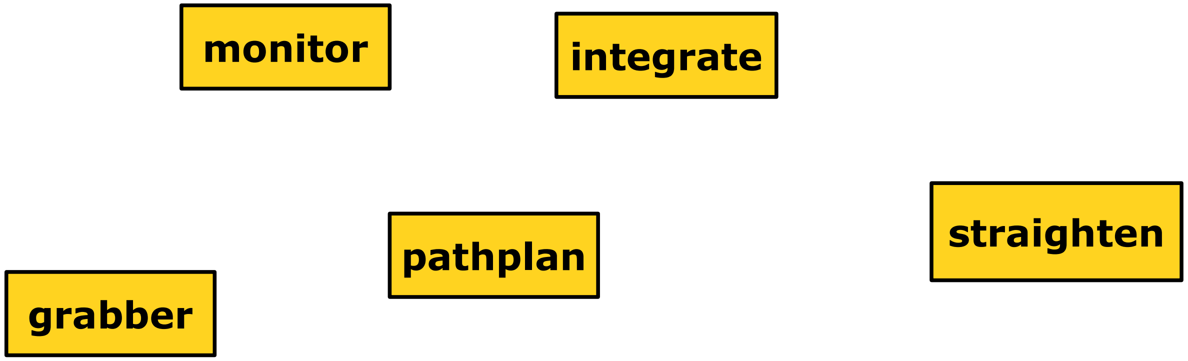


Level 2: *Explore*



Level 2: *Explore*

Level 2 implies the following 5 new modules:



grabber module

monitor module

integrate module

pathplan module

straighten module

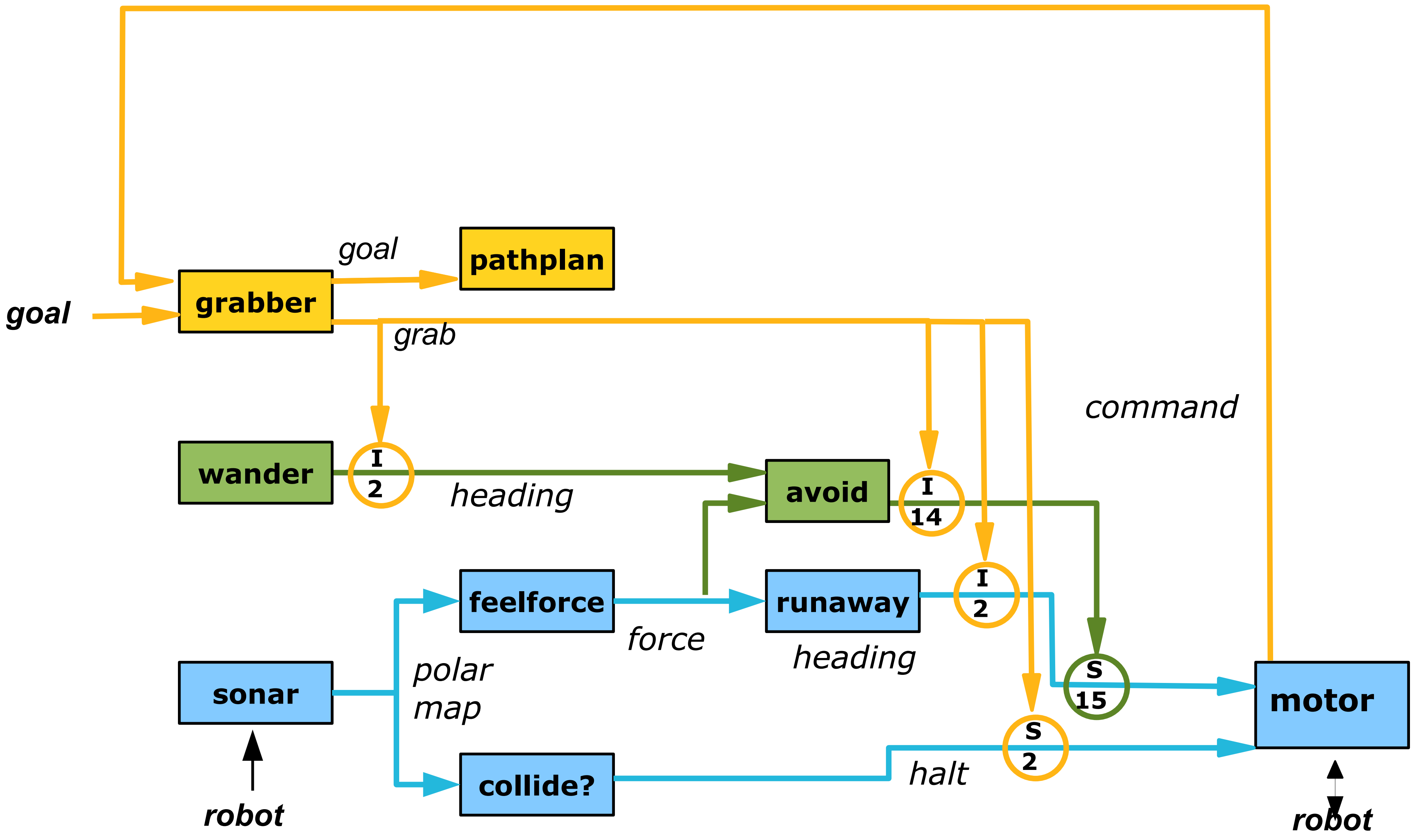
Level 2: *Explore*

grabber:
the grabber (video grabber) module ensures that level 2 has control of the motors by sending a halt signal to the motor module, then temporarily inhibiting a number of communication paths in the lower levels so that no new actions can be initiated, and waiting for the motor module to indicate that it is no longer controlling the robot motion.

At this point the (visual) sensor will be giving stable readings sufficient to plan a detailed motion, so a goal can be sent to the pathplan module.

Please notice: while stopping the other levels, the robot will be unable to avoid approaching objects for a brief 2 seconds.

Level 2: *Explore*



Level 2: *Explore*

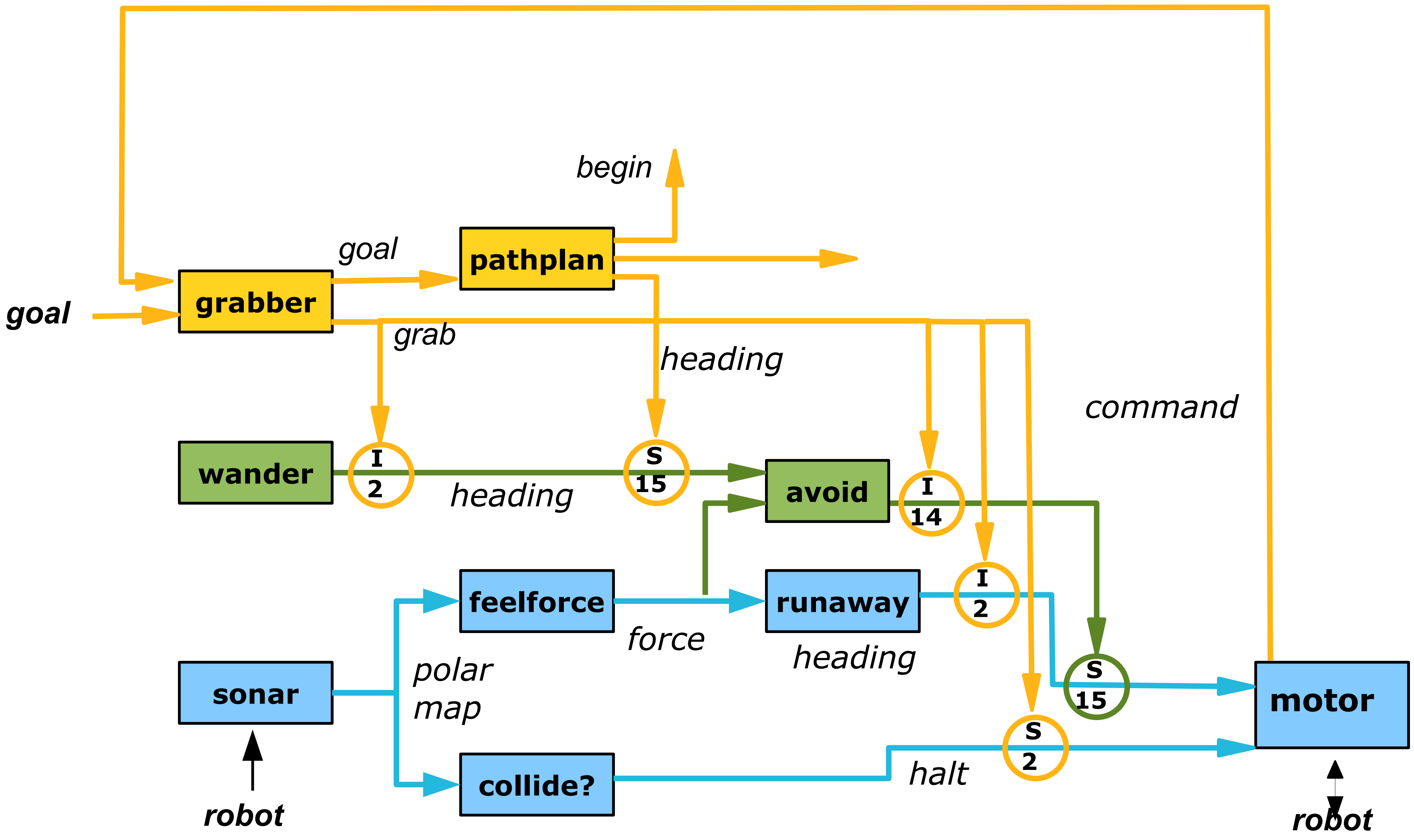
pathplan:
the pathplan module takes a goal specification and attempts to reach that goal.

To do this, it sends headings to the avoid module, which may perturb them to avoid local obstacles, and monitors its integral input which is an integration of actual motions.

The messages to the avoid module suppress random wanderings of the robot, so long as the higher level planner remains active.

When the position of the robot is close to the desired position it outputs the goal to the straighten module.

Level 2: *Explore*

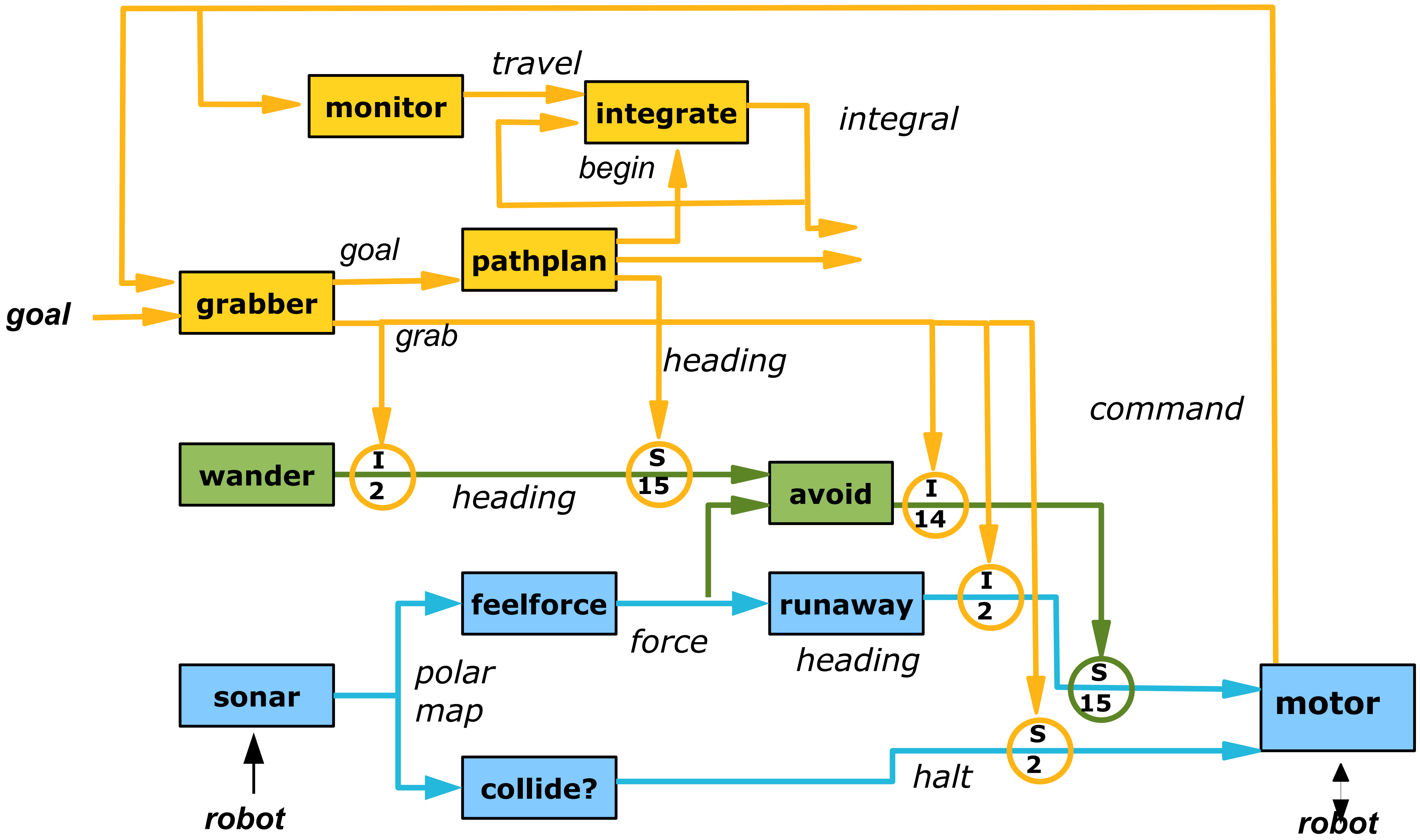


Level 2: *Explore*

monitor:
the monitor module continually monitors the status of the motor module. As soon as the module becomes inactive the monitor module queries the robot via a direct connection to get a reading from its shaft encoders on how far it has traveled

Thus it is able to track each motion, whether it terminated as intended or if there was an early hat due to looming obstacles.

Level 2: *Explore*

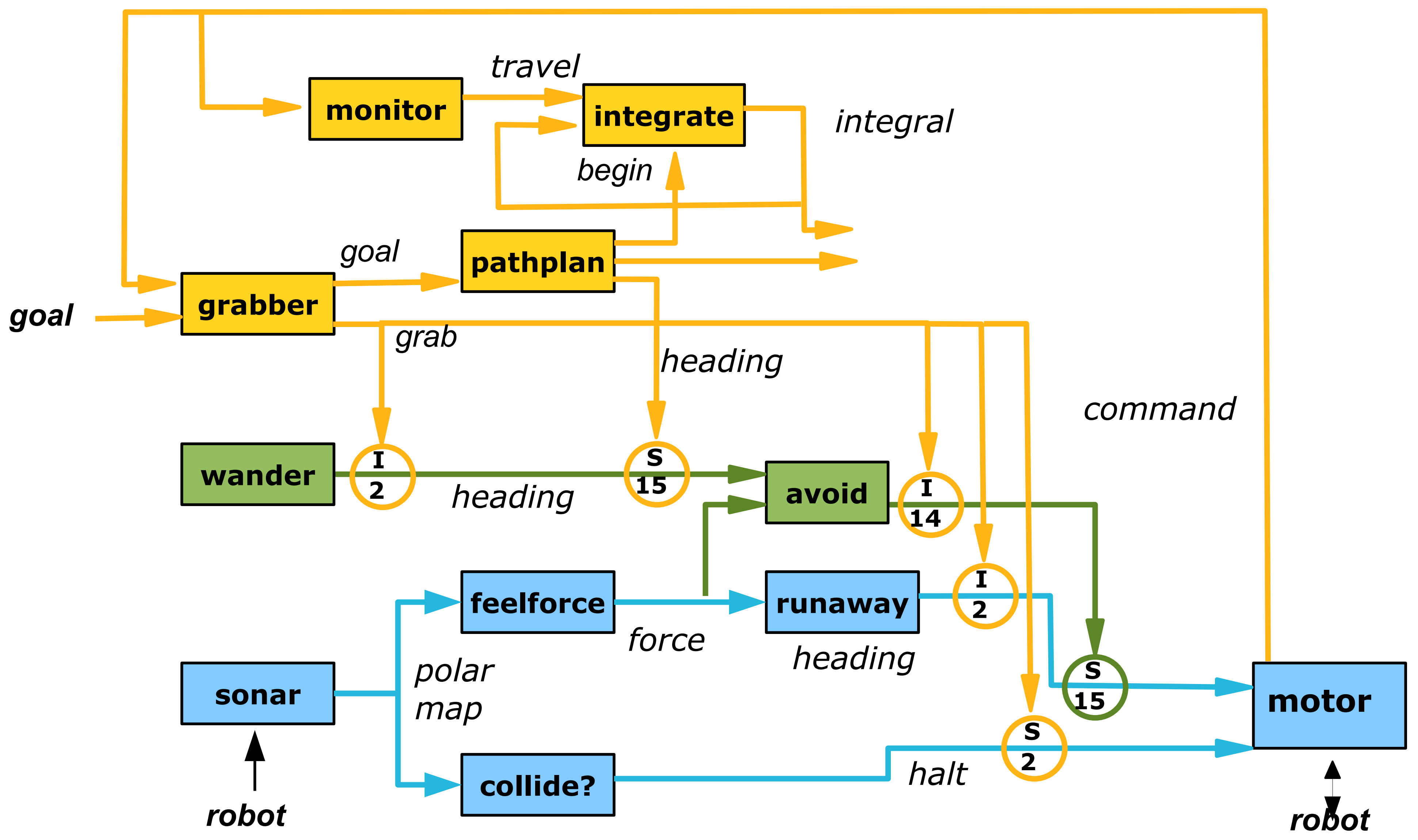


Level 2: *Explore*

integrate:
the integrate module accumulates reports of motions from the monitor module and always sends its most recent result out on its integral line.

It gets restarted by application of a signal to its reset input.

Level 2: *Explore*



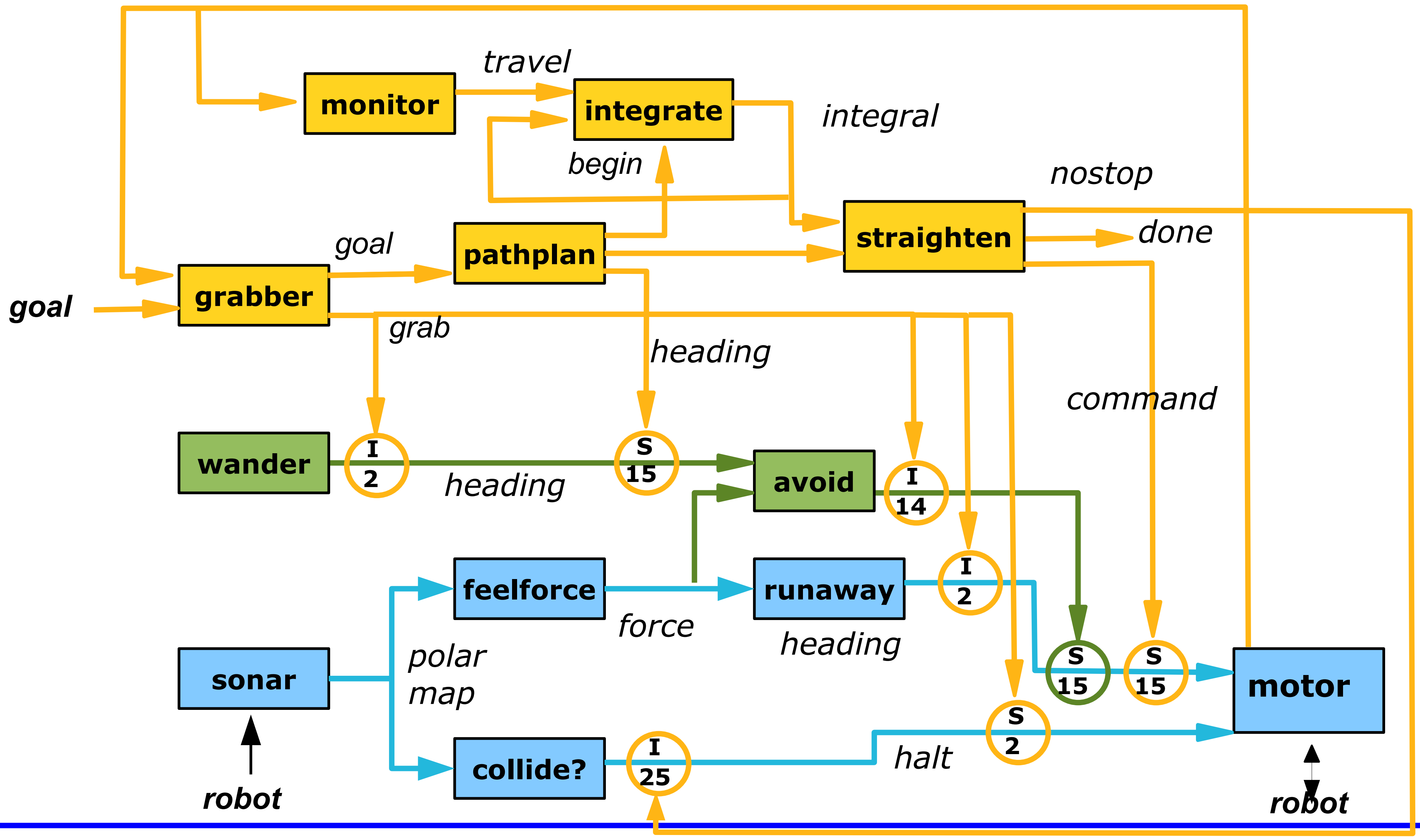
Level 2: *Explore*

straighten:
the straighten module is responsible for modifying the final orientation of the robot.

Any command with just an angular heading will not get through the avoid module as it filters out small motions, since the pressure of forces from remote obstacles would otherwise make it „buzz“ with large turns and small forward motions.

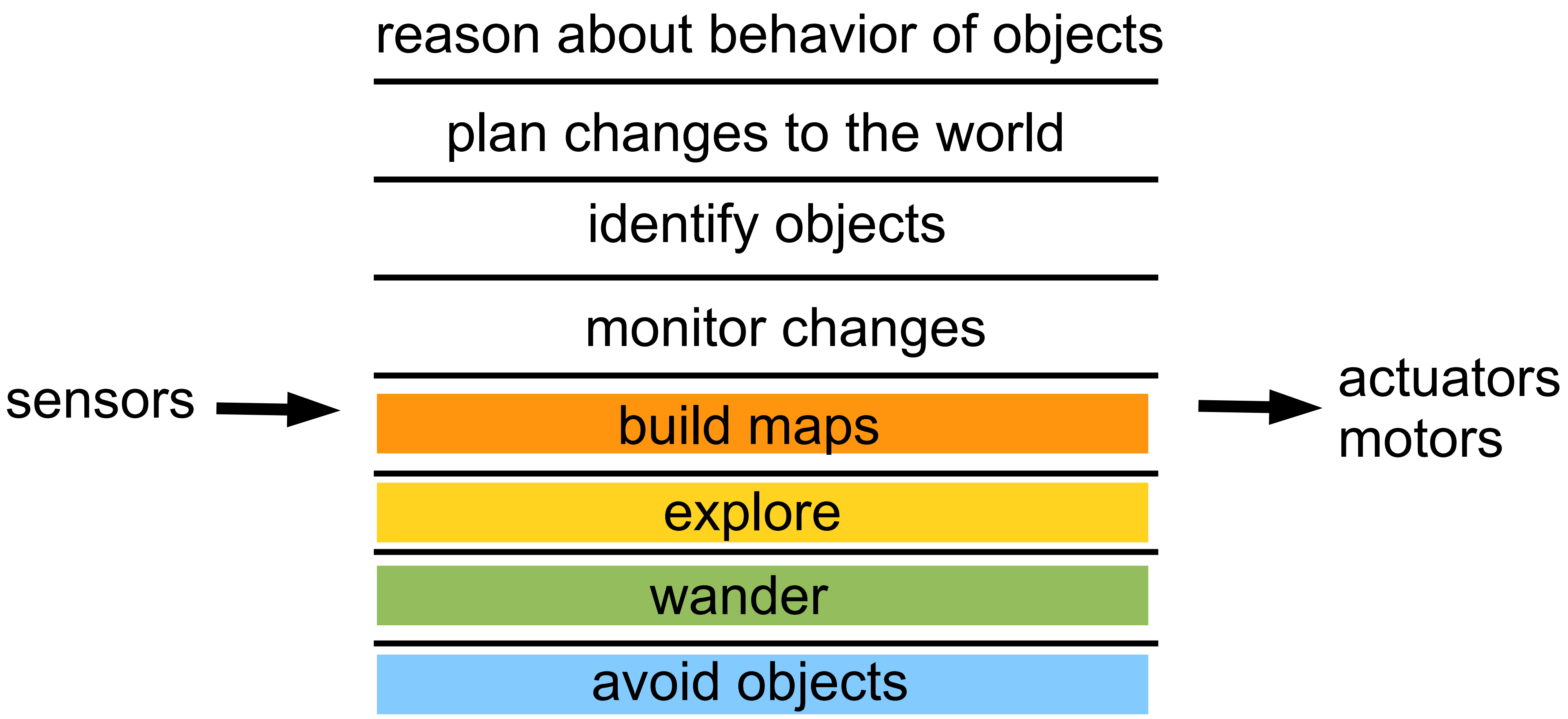
The straighten module sends its commands directly to the motor module, and monitors the integral line to make sure it is successful. For good measures it also inhibits the collision reflex, since there is no danger of a collision from forward motion (straighten only turns the robot).

Level 2: *Explore*



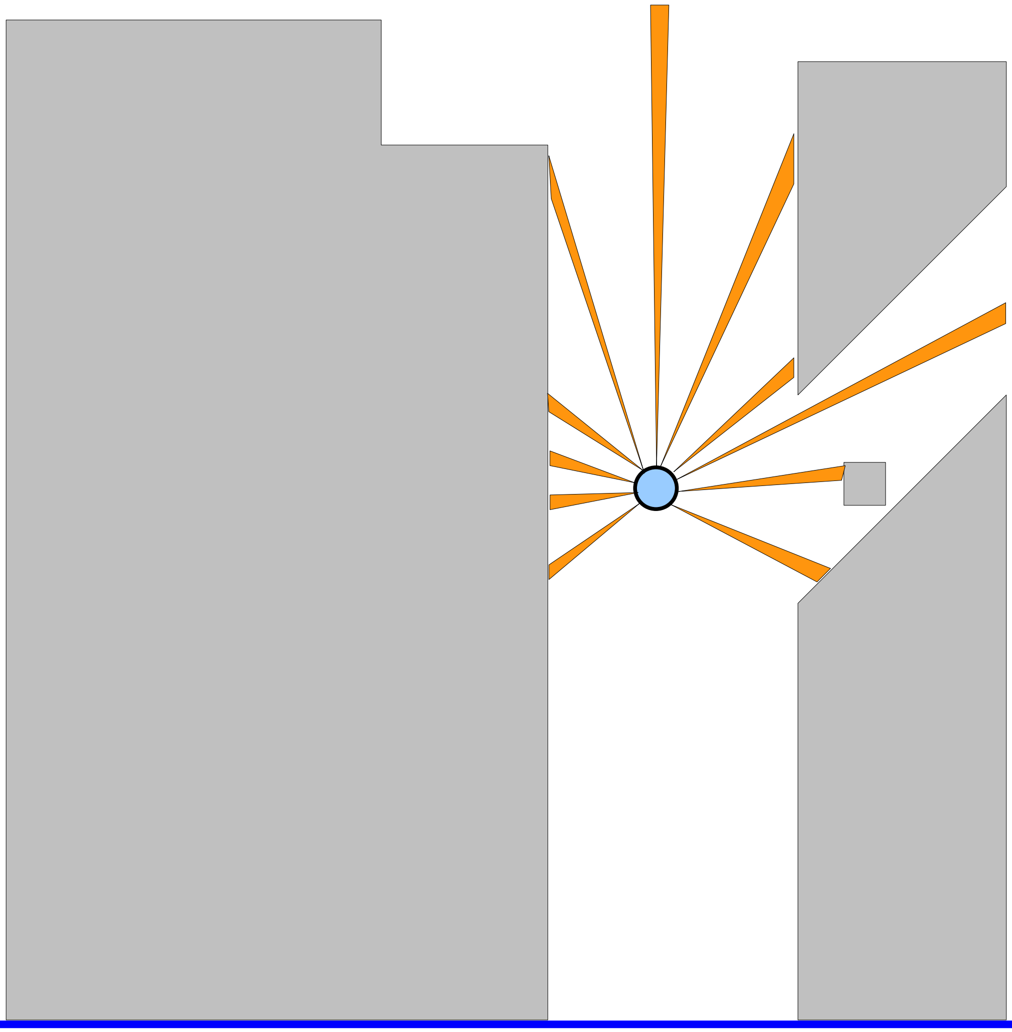
Task Achieving Behaviors

A decomposition of a mobile robot control system based on task achieving behaviors *(Brooks 1985)*



Subsumption Architecture

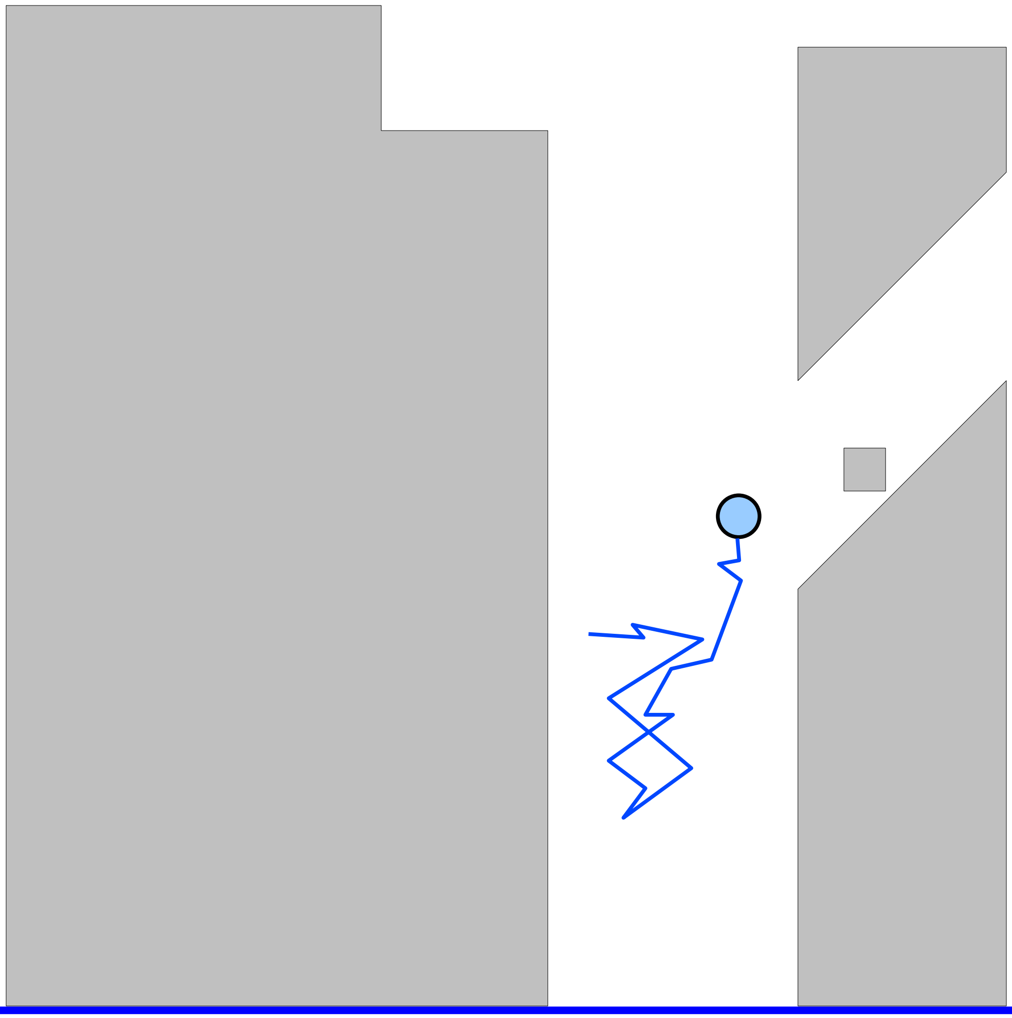
R.Brooks' test world for the subsumption controlled robot.



Robot is equipped with sonar based distance sensors.

Subsumption Architecture

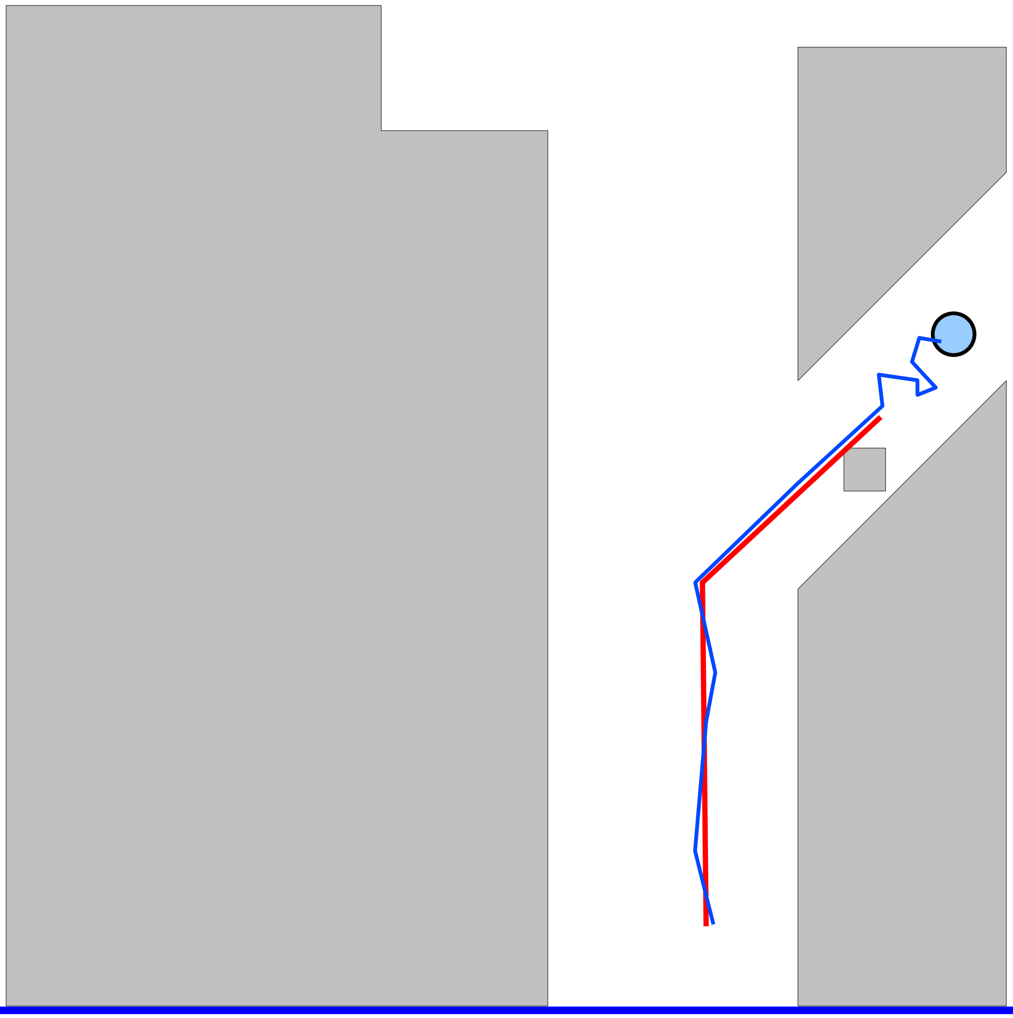
R.Brooks' test world for the subsumption controlled robot.



Under levels 0 and 1 control the robot wanders around aimlessly.
It does not hit obstacles

Subsumption Architecture

R.Brooks' test world for the subsumption controlled robot.

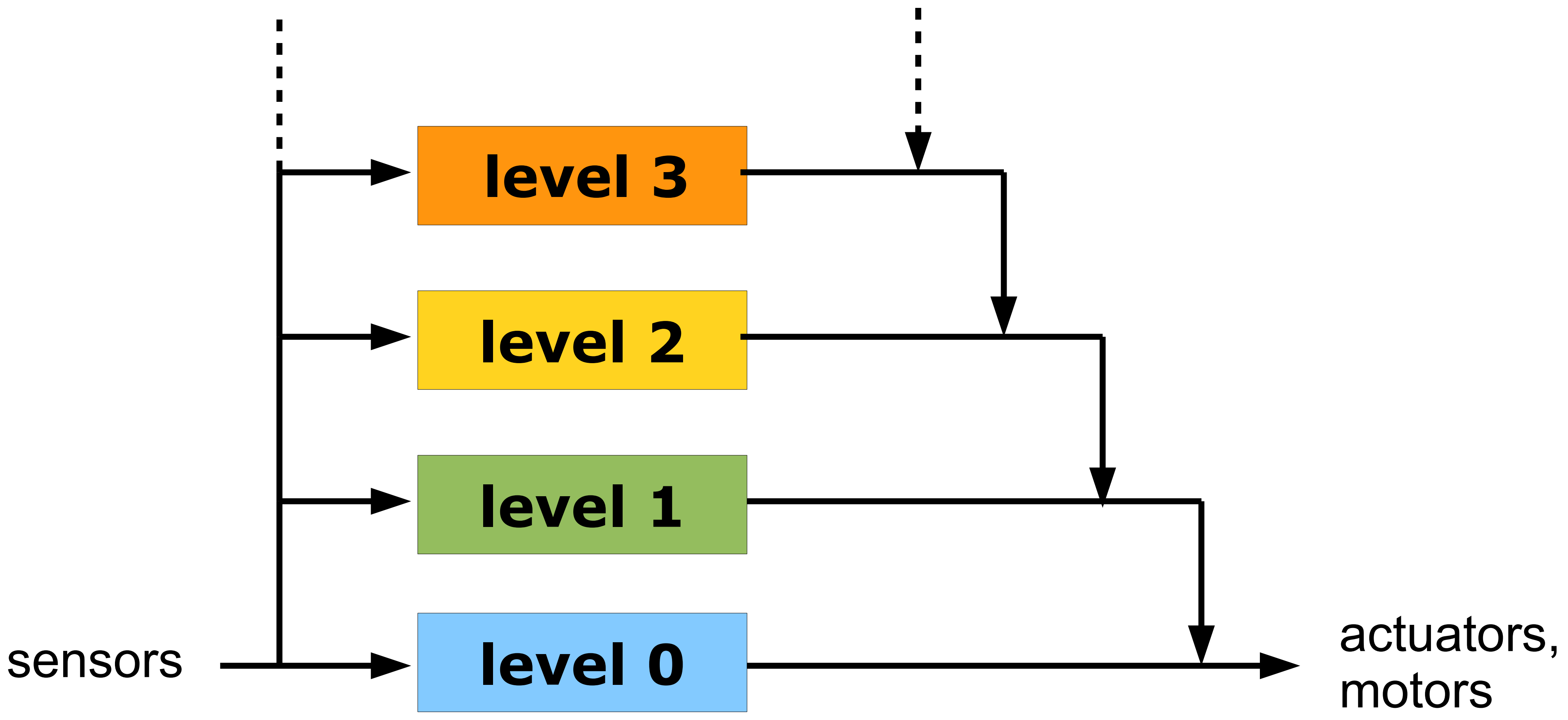


With level 2 control the robot tries to achieve command goals. The nominal goals are the two straight (red) lines.

After reaching the second (final) goal the robot reverts to aimless level 1 behavior.

Layered Control

Control is layered with higher level layers subsuming the lower levels.



Some Literature on Subsumption Architecture

- **R. A. Brooks (1986)**,
  "*A Robust Layer Control System for a Mobile Robot*",
  IEEE Journal of Robotics and Automation RA-2, 14-23.
- **R. A. Brooks (1987)**,
  "*Planning is just a way of avoiding figuring out what to do next*",
  Technical report, Working Paper 303, MIT Artificial Intelligence Laboratory.
- **R. A. Brooks (1991),**
  "*Intelligence Without Reason*",
  in Proc. of the 1991 Int. Joint Conf. on Artificial Intelligence, pp. 569–595.
- **R. A Brooks (1991)**,
  "*Intelligence Without Representation*",
  Artificial Intelligence 47 (1991) 139-159.
- **Brooks, R.A. (1992)**
  "*Artificial Life and Real Robots*".
  In F. J. Varela and P. Bourgine, editors, "*Toward a Practice of Autonomous Systems*": Proceedings of the First European Conference on Artificial Life, pages 3-10. MIT Press/Bradford Books, Cambridge, MA, 1992.

A working robot with Subsumption Architecture

In 1991 the company iRobot has been funded by three MIT researchers (MIT roboticists Colin Angle, Helen Greiner and Rodney Brooks).

The company iRobot has launched a series of robots,
e.g. Ghengis, Ariel and Packbot.

A working robot with Subsumption Architecture

In 1991 the company iRobot has been funded by three MIT researchers (MIT roboticists Colin Angle, Helen Greiner and Rodney Brooks).

The company iRobot has launched a series of robots, e.g. Ghengis, Ariel and Packbot.

In 2002 iRobot launches the Roomba® floor vacuuming robot, which is one of their most selling robots.

A working robot with Subsumption Architecture

In 1991 the company iRobot has been funded by three MIT researchers (MIT roboticists Colin Angle, Helen Greiner and Rodney Brooks).

The company iRobot has launched a series of robots, e.g. Ghengis, Ariel and Packbot.

In 2002 iRobot launches the Roomba® floor vacuuming robot, which is one of their most selling robots.

Major parts of the roomba behavioral system is implemented using the subsumption architecture from Rodney Brooks.

A variation of the roomba became the iRobot Create platform which has been explicitly developed for educational and research purposes.

Roomba® 105 Combo Roboter – Weiß

A working robot with Subsumption Architecture

In 1991 the company iRobot has been funded by three MIT researchers (MIT roboticists Colin Angle, Helen Greiner and Rodney Brooks).

The company iRobot has launched a series of robots, e.g. Ghengis, Ariel and Packbot.

In 2002 iRobot launches the Roomba® floor vacuuming robot, which is one of their most selling robots.

Major parts of the roomba behavioral system is implemented using the subsumption architecture from Rodney Brooks.

A variation of the roomba became the iRobot Create platform which has been explicitly developed for educational and research purposes.

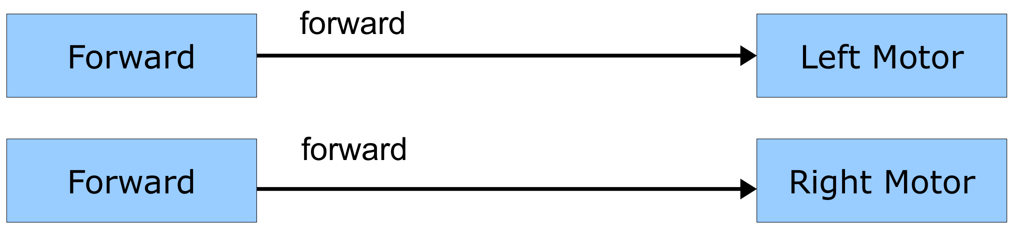
The vacuuming robot roomba is the basis for the ***RoomRider***, one of the teaching and research platforms of the AIS research group, Computer Science, University of Bonn.

Subsumption Architecture, Example

Adapted from:Andrej Lúĉny
Comenius Uiniversity Bratislava
Robotic Summer School 2009
www.microstep-mis.com/~andy

Subsumption Architecture, Example

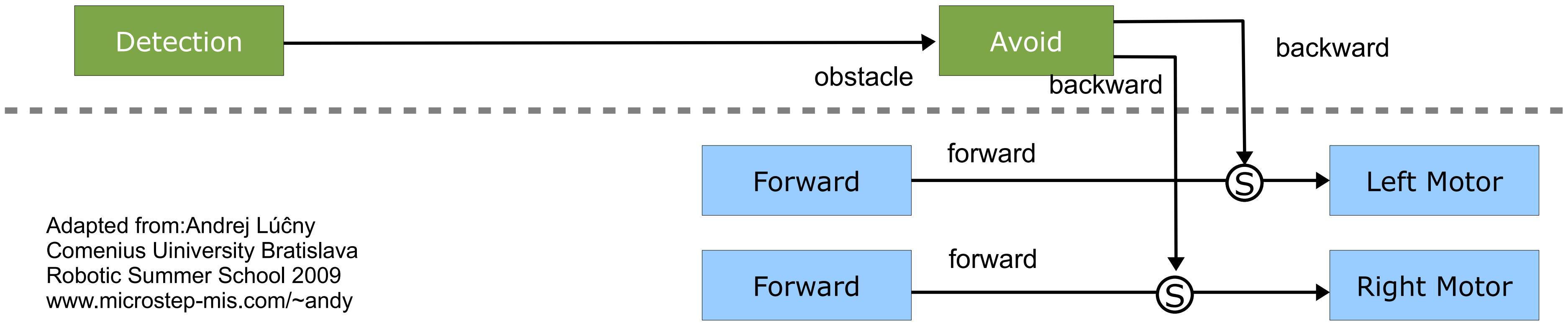
Start with a robot which can just go forward.



Adapted from:Andrej Lúĉny
Comenius Uiniversity Bratislava
Robotic Summer School 2009
www.microstep-mis.com/~andy

Subsumption Architecture, Example

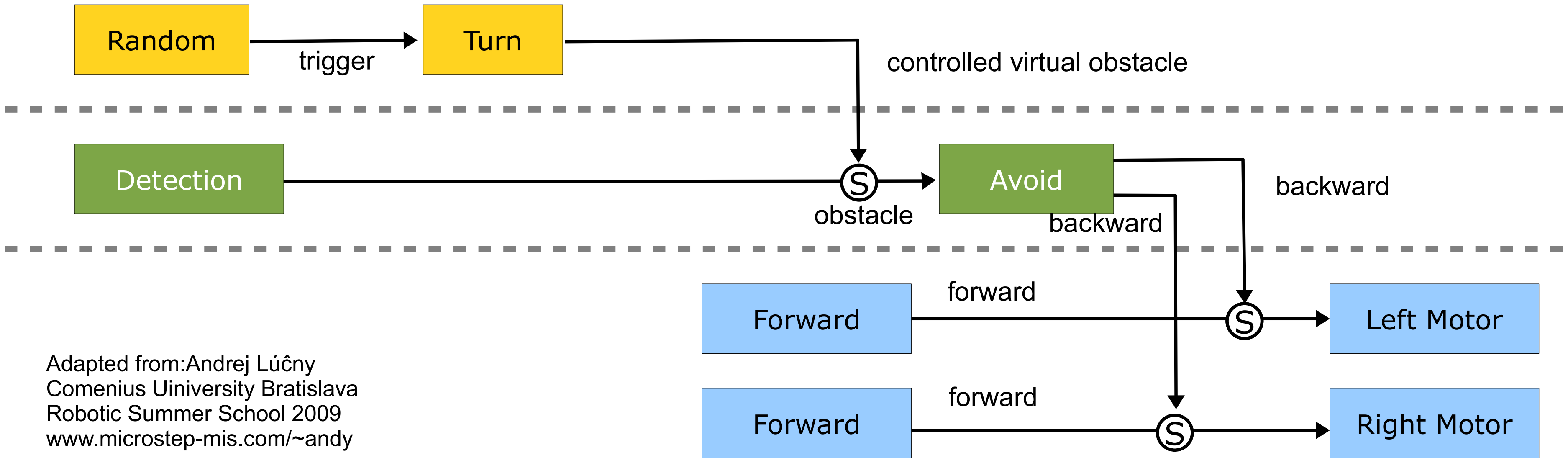
Add a layer which recognizes obstacles and while they are detected, the layer generates a message for one wheel to go backward to prevent the robot from colliding.



Adapted from:Andrej Lúĉny
Comenius Uiniversity Bratislava
Robotic Summer School 2009
www.microstep-mis.com/~andy

Subsumption Architecture, Example

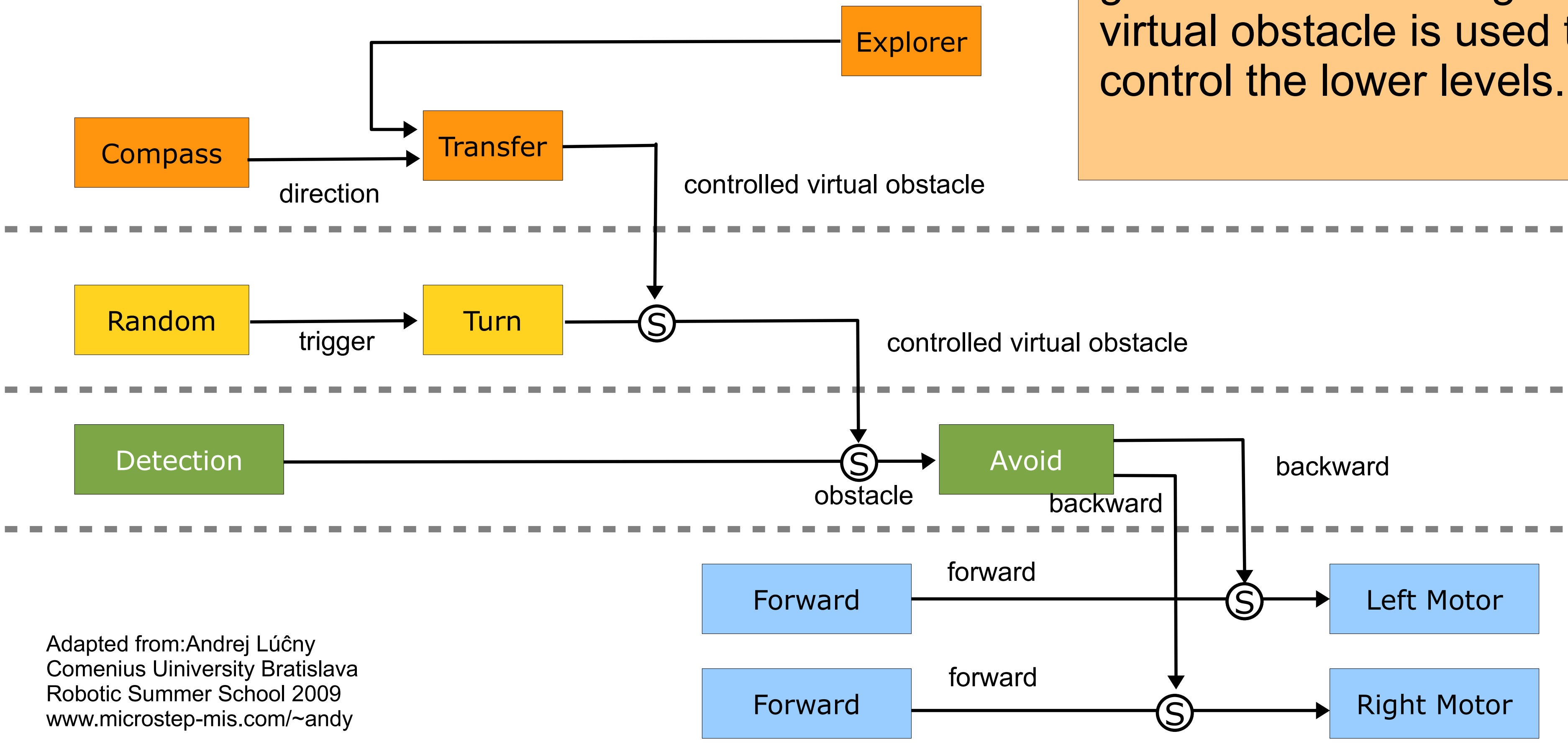
Add a layer which sometimes causes the robot to perform random turns. Making the lower level believe that there is an obstacle it is caused to do an avoidance movement.



Adapted from:Andrej Lúĉny
Comenius Uiniversity Bratislava
Robotic Summer School 2009
www.microstep-mis.com/~andy

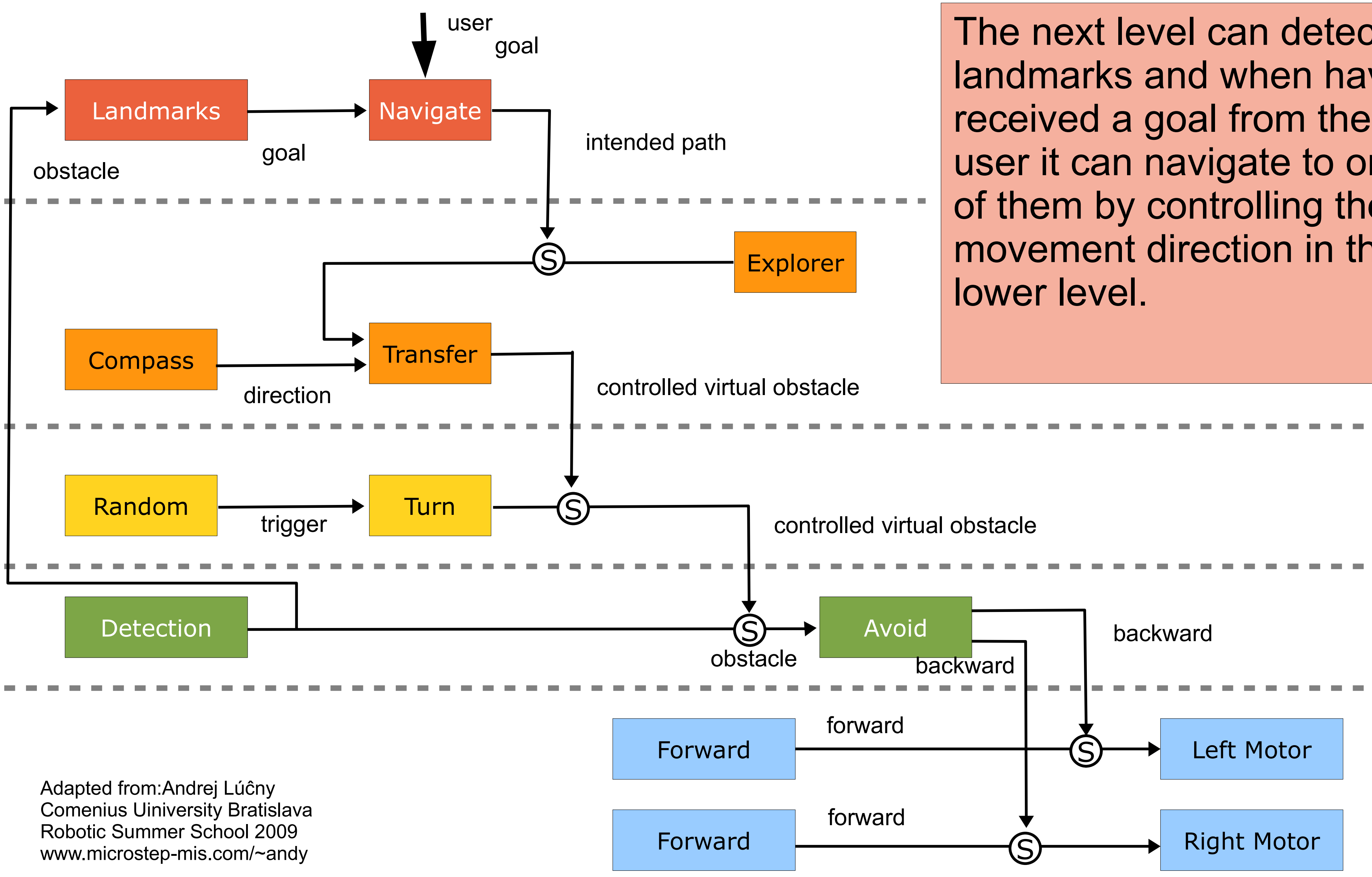
Subsumption Architecture, Example

The next layer implements an exploratory behavior by generating sequences of headings adjusted with a global orientation. Again a virtual obstacle is used to control the lower levels.



Adapted from:Andrej Lúĉny
Comenius Uiniversity Bratislava
Robotic Summer School 2009
www.microstep-mis.com/~andy

Subsumption Architecture, Example



The next level can detect landmarks and when having received a goal from the user it can navigate to one of them by controlling the movement direction in the lower level.

Adapted from:Andrej Lúĉny
Comenius Uniiversity Bratislava
Robotic Summer School 2009
www.microstep-mis.com/~andy

Summary Subsumption Architecture

The subsumption architecture is:

- A hierarchical control architecture
- Organized in layers of competence
- Each layer is working autonomously (unless)
- Controlled by higher layers
- Inhibition and suppression control lower layers
- The layers consist of simple structured modules
- Rather robust against damage or module failure
- Implementation can be very complex

Control Architectures

Examples for control architectures:

- reactive control
- open-loop / closed loop control
- SMPA architectures
- Subsumption architecture
- Learning control (Neural Networks, Fuzzy, ...)
- Blackboard architectures
- ACT-R
- Hierarchical control
- ... ... and many more interesting approaches ...

Artificial Life
Summer 2025

Subsumption Architecture
Artificial Life Extras

Master Computer Science [MA-INF 4201]
Mon 14:15 – 15:45, HSZ, HS-2

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn

Outline:

- Cellular Automata Simulator
- Langtons's Ant
- Game of Life Simulator
- Langton's Loop
- Core Wars, Redcode
- Co-Evolution, Predator-Prey Behavior
- Evolutionary Robotics
- Evolving Cars
- Evolving Morphology
- Bionics, Bionic Robots (Festo)

Cellular Automata Simulator

A wide variety of simulation tools for Cellular Automata exist today for all typical computing architectures and operating systems, and most of them are freely available.

Some of them are accompanied with a large library of patterns.

Some of them are designed to simulate cellular automata, in one and 2 dimensions.

The list on the next page is is neither complete, nor should it be considered as being a judgment;
these are just some of my personal favorites.

Game of Life Simulators

A wide variety of simulation tools for Conway's Game of Life exist today for all typical computing architectures and operating systems, and most of them are freely available.

Some of them are accompanied with a large library of patterns.

Some of them are designed to simulate cellular automata, and “just” include Conways Game of Life.

The list on the next page is is neither complete, nor should it be considered as being a judgment;
these are just some of my personal favorites.

Game of Life Simulators

The list is neither complete, nor should it be considered as being a judgment; these are just some of my personal favorites.

Mirek's Cellebration, MCell 4.20
1D and 2D Cellular Automata explorer by Mirek Wojtowicz
http://psoup.math.wisc.edu/mcell/

Winlife32, Version 2.3
A system for manipulating Conway's Game of Life
http://www.winlife32.com/

Golly, Version 2.6, Mac OS X 10.6 or later.
An open source, cross-platform Game of Life Simulator
http://www.macupdate.com/info.php/id/19622/golly

Golly, Version 3.2, Windows, Linux
An open source, cross-platform Game of Life Simulator
https://sourceforge.net/projects/golly/c

Langtons's Ant

A nice Windows based freeware simulator for Langton's Ant can be found at:
www.markuswelz.de/freeware

Another simulator is web based and can be found at:
www.langtonant.com
There, it is possible to choose different behaviors of the ant:
like RL(original one), or RLR, or LLRRRLRLRLLR or …

The CA Simulator Golly, Version 3.2, Windows, Linux,
an open source, cross-platform CA Simulator, provides a nice implementation for Langton's Ant
https://sourceforge.net/projects/golly/c

Core Wars, Redcode

Core War is a 1984 programming game created by D. G. Jones and A. K. Dewdney in which two or more battle programs (called "warriors") compete for control of a virtual computer.
These battle programs are written in an abstract assembly language called Redcode.
The first version has been published in Scientific American in 5/84.

At the beginning of a game, each battle program is loaded into memory at a random location, after which each program executes one instruction in turn.
The goal of the game is to cause the processes of opposing programs to terminate (which happens if they execute an invalid instruction), leaving the victorious program in sole possession of the machine.

https://en.wikipedia.org/wiki/Core_War

Core Wars, Redcode

Redcode is the programming language used in Core War. It is executed by a virtual machine known as a **M**emory **A**rray **R**edcode **S**imulator, or **MARS**.
The design of Redcode is loosely based on actual CISC assembly languages of the early 1980s, but contains several features not usually found in actual computer systems.

The earliest published version of Redcode defined only eight instructions, which later (1986 and 1988) has been increased to 11. Currently, the ICWS-94 standard has 16 instructions.
However, Redcode supports a number of different addressing modes and modifiers yielding a total of 7168 possible operations.

https://en.wikipedia.org/wiki/Core_War

Core Wars, Redcode

Here is a simple Redcode instruction set:

Encoded Form	Mnemonic Symbol	Argument		Action
0	DAT		B	Initialize location to value B.
1	MOV	A	B	Move A into location B.
2	ADD	A	B	Add operand A to contents of location B and store result in location B.
3	SUB	A	B	Subtract operand A from contents of location B and store result in location B.
4	JMP		B	Jump to location B.
5	JMZ	A	B	If operand A is 0, jump to location B; otherwise continue with next instruction.
6	DJZ	A	B	Decrement contents of location A by 1. If location A now holds 0, jump to location B; otherwise continue with next instruction.
7	CMP	A	B	Compare operand A with operand B. If they are not equal, skip next instruction; otherwise continue with next instruction.

https://users.obs.carnegiescience.edu/birk/COREWAR/DOCS/guide2red.txtr

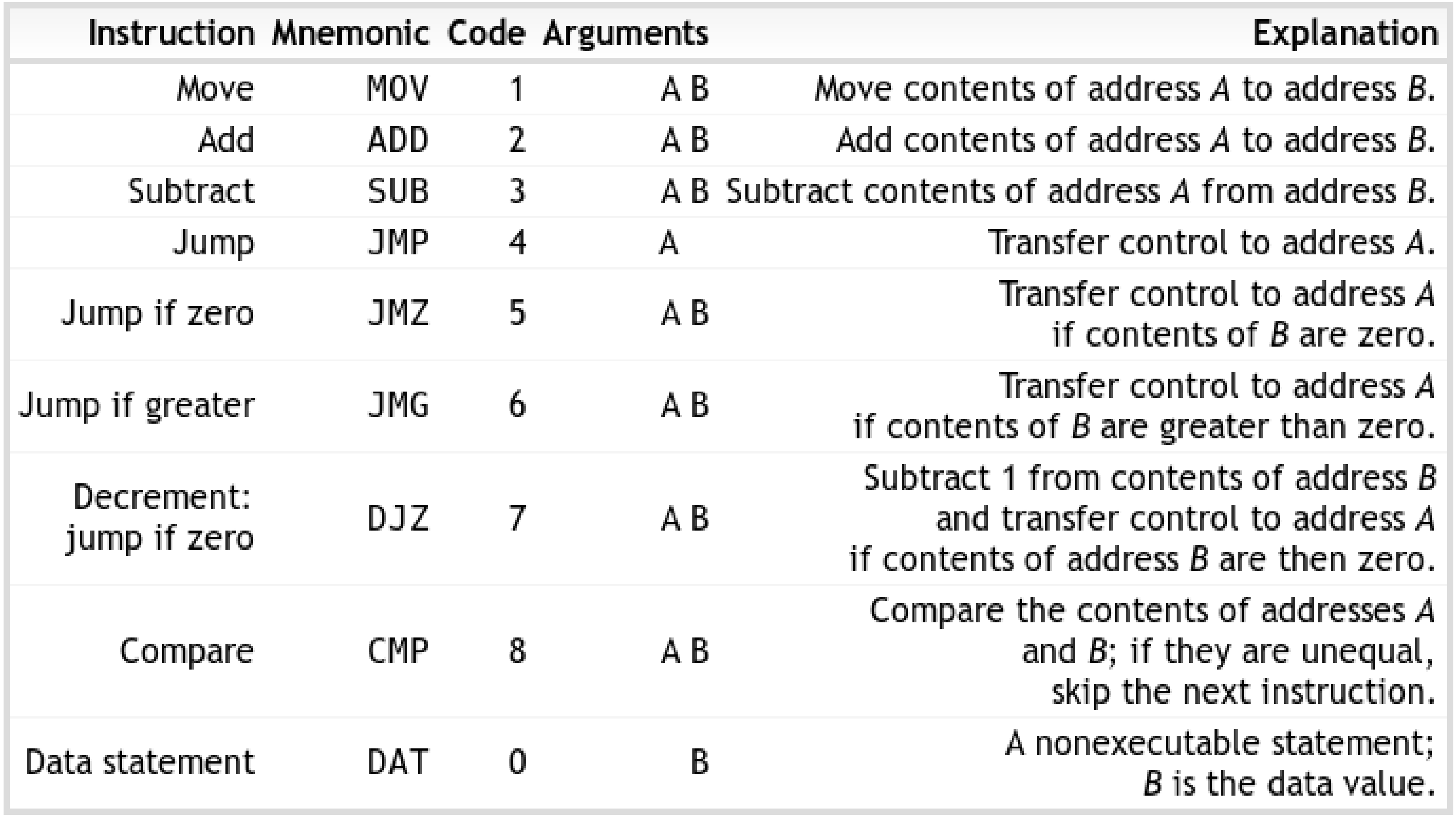
Core Wars, Redcode

Redcode/MARS Adressing modes:

Encoded Form	Mnemonic Symbol	Name	Meaning
0	#	**Immediate**	The number following this symbol is the operand.
1	<none>	**Relative**	The number specifies an offset from the current instruction. Mars adds the offset to the address of the current instruction; the number stored at the location reached in this way is the operand.
2	@	**Indirect**	The number following this symbol specifies an offset from the current instruction to a location where the relative address of the operand is found. Mars adds the offset to the address of the current instruction and retrieves the number stored at the specified location; this number is then interpreted as an offset from its own address. The number found at this second location is the operand.

https://users.obs.carnegiescience.edu/birk/COREWAR/DOCS/guide2red.txtr

Core Wars, Redcode

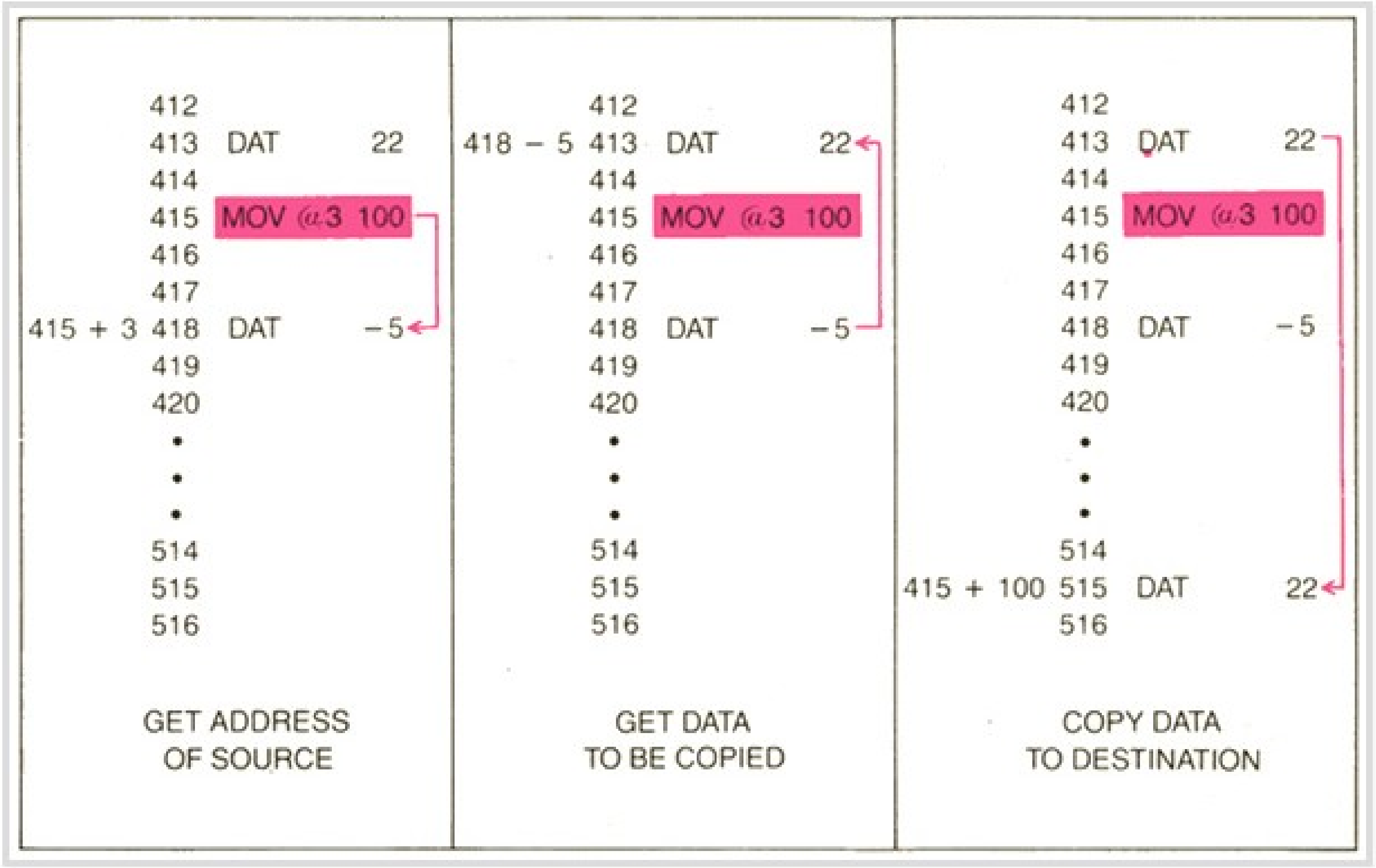


Instruction	Mnemonic	Code	Arguments	Explanation
Move	MOV	1	A B	Move contents of address *A* to address *B*.
Add	ADD	2	A B	Add contents of address *A* to address *B*.
Subtract	SUB	3	A B	Subtract contents of address *A* from address *B*.
Jump	JMP	4	A	Transfer control to address *A*.
Jump if zero	JMZ	5	A B	Transfer control to address *A* if contents of *B* are zero.
Jump if greater	JMG	6	A B	Transfer control to address *A* if contents of *B* are greater than zero.
Decrement: jump if zero	DJZ	7	A B	Subtract 1 from contents of address *B* and transfer control to address *A* if contents of address *B* are then zero.
Compare	CMP	8	A B	Compare the contents of addresses *A* and *B*; if they are unequal, skip the next instruction.
Data statement	DAT	0	B	A nonexecutable statement; *B* is the data value.

The instruction set of Redcode, an assembly language for Core War

http://corewar.co.uk/dewdney/1984-05.htm

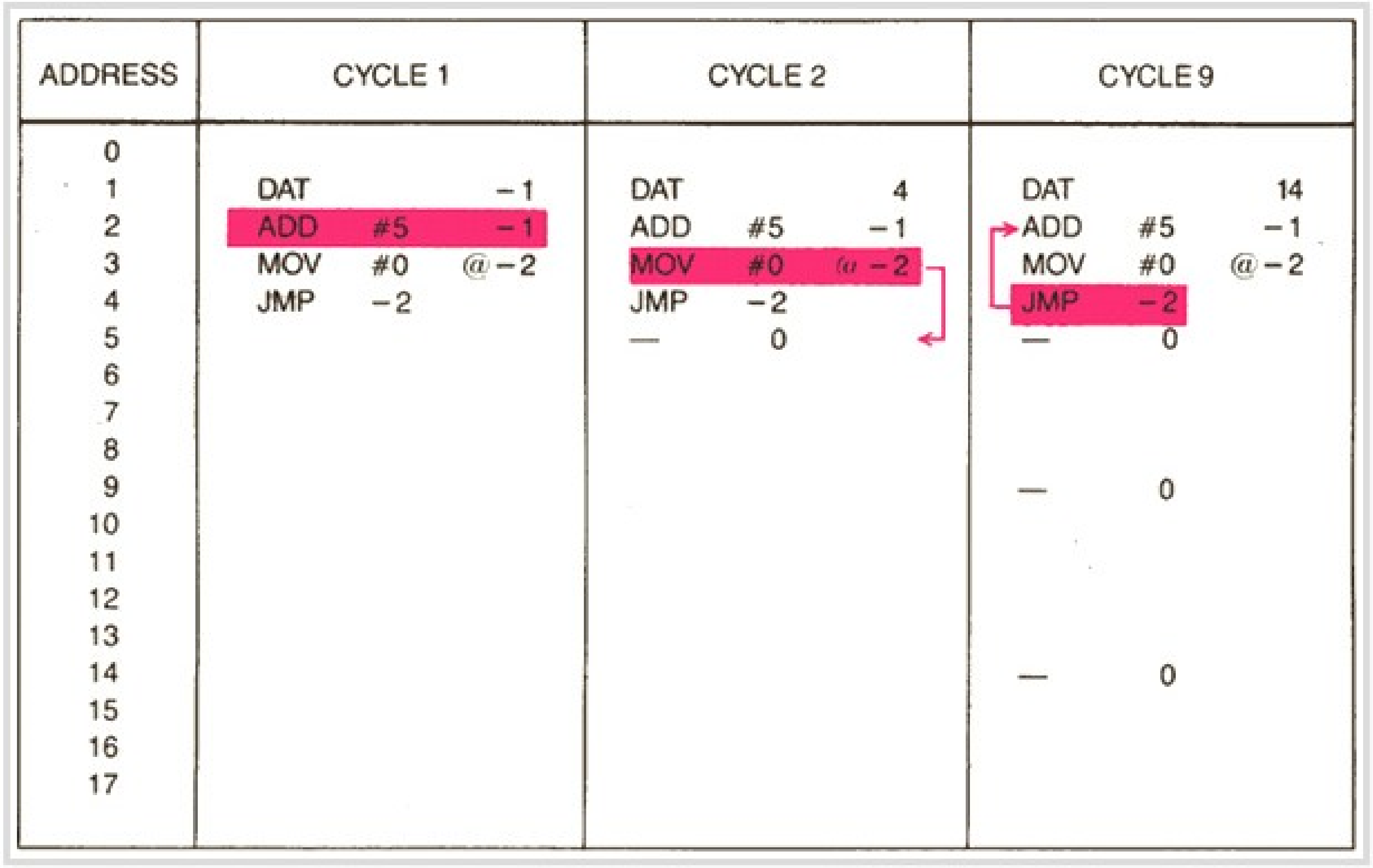
Core Wars, Redcode



The three-step mechanism of indirect relative addressing

http://corewar.co.uk/dewdney/1984-05.htm

Core Wars, Redcode



ADDRESS	CYCLE 1	CYCLE 2	CYCLE 9
0			
1	DAT -1	DAT 4	DAT 14
2	ADD #5 -1	ADD #5 -1	ADD #5 -1
3	MOV #0 @-2	MOV #0 @-2	MOV #0 @-2
4	JMP -2	JMP -2	JMP -2
5		— 0	— 0
6			
7			
8			
9			— 0
10			
11			
12			
13			
14			— 0
15			
16			
17			

Dwarf, a battle program, lays down a barrage of "zero bombs"

http://corewar.co.uk/dewdney/1984-05.htm

Core Wars, Redcode

Two little nasty programs: *Imp* and *Dwarf*

Imp: Copys itself into the next memory location.

```
MOV   0   1
```

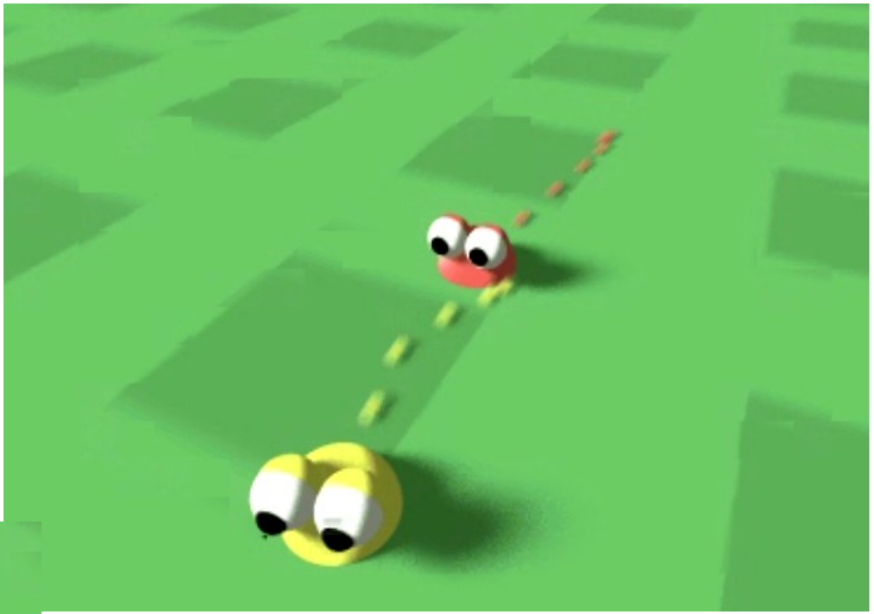
Dwarf: Works its way through the memory array bombarding every fifth address with a zero.

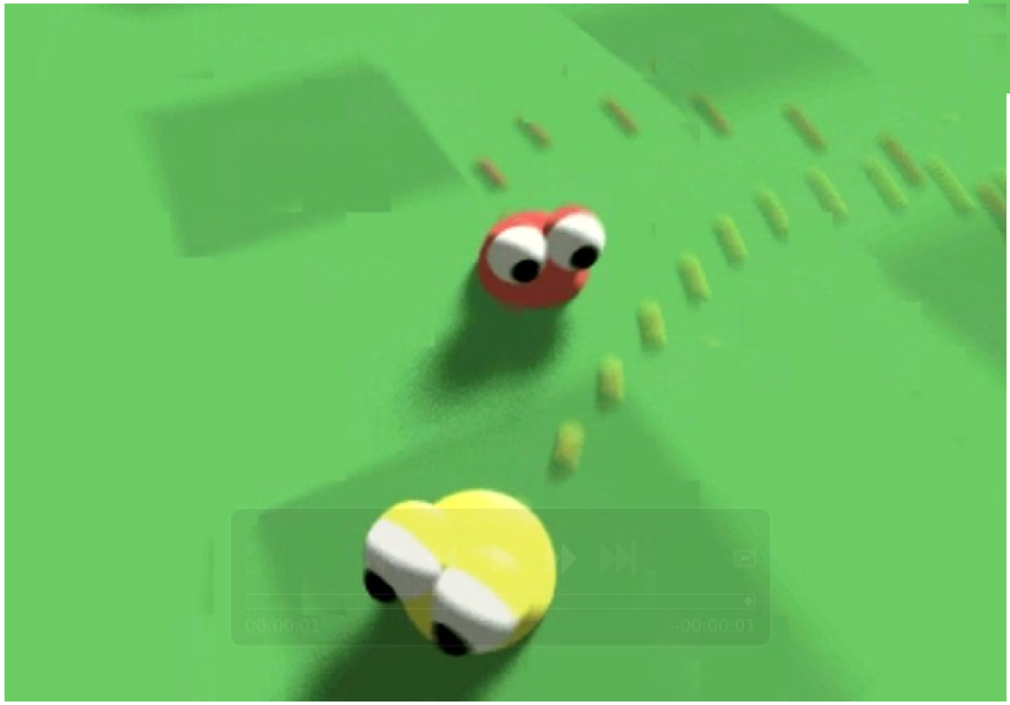
```
DAT           -1
ADD   #5      -1
MOV   #0     @-2
JMP   -2
```

http://corewar.co.uk/dewdney/1984-05.htm

Evolving Predator-Prey Behavior

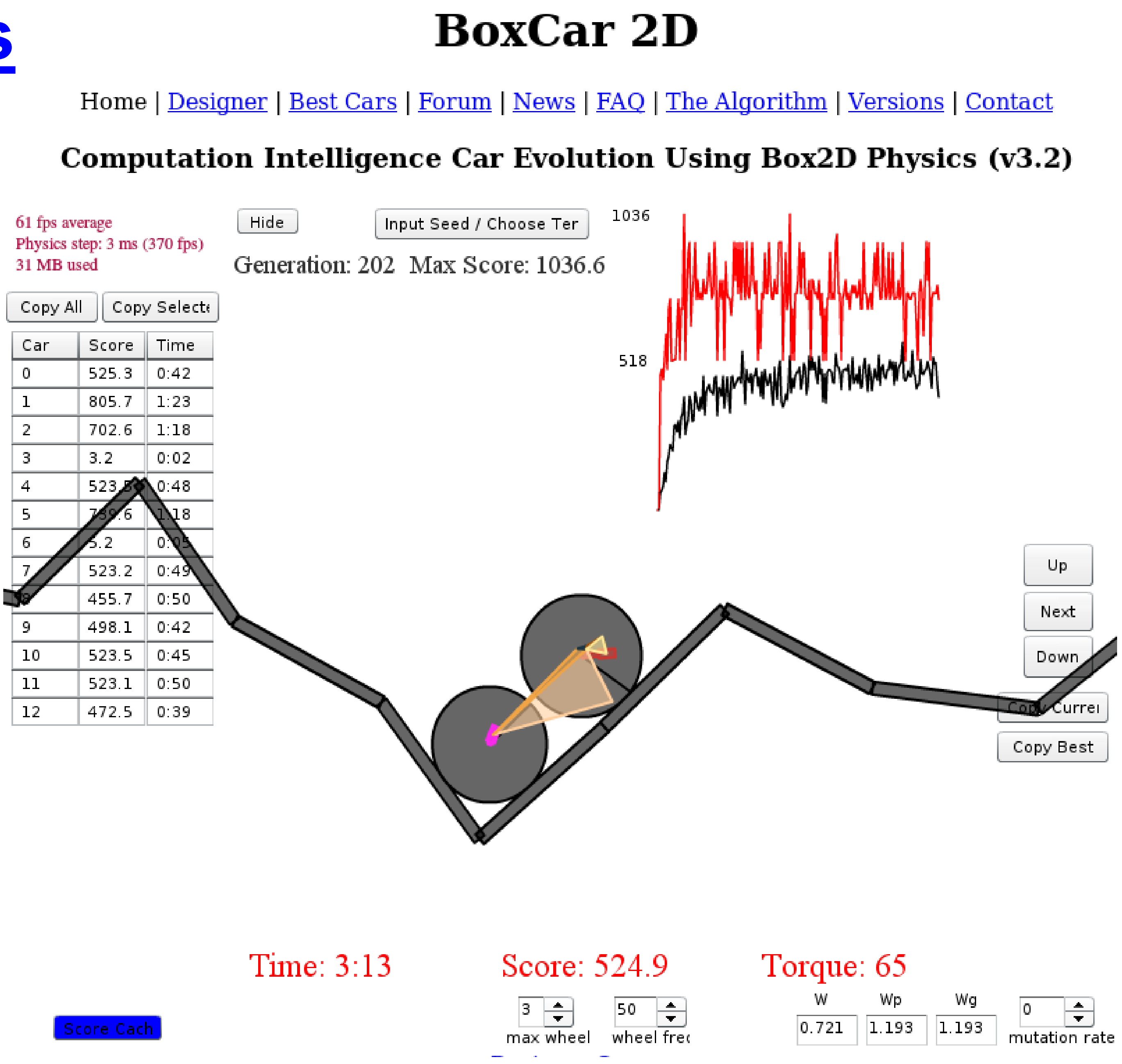
The predator tries to catch the prey, while the prey is trying to escape. During the evolutionary process the capabilities of both increase.





Evolving Cars

http://boxcar2d.com/



BoxCar 2D

Home | Designer | Best Cars | Forum | News | FAQ | The Algorithm | Versions | Contact

Computation Intelligence Car Evolution Using Box2D Physics (v3.2)

61 fps average
Physics step: 3 ms (370 fps)
31 MB used

Generation: 202 Max Score: 1036.6

Car	Score	Time
0	525.3	0:42
1	805.7	1:23
2	702.6	1:18
3	3.2	0:02
4	523.5	0:48
5	739.6	1:18
6	5.2	0:05
7	523.2	0:49
8	455.7	0:50
9	498.1	0:42
10	523.5	0:45
11	523.1	0:50
12	472.5	0:39

Time: 3:13 Score: 524.9 Torque: 65

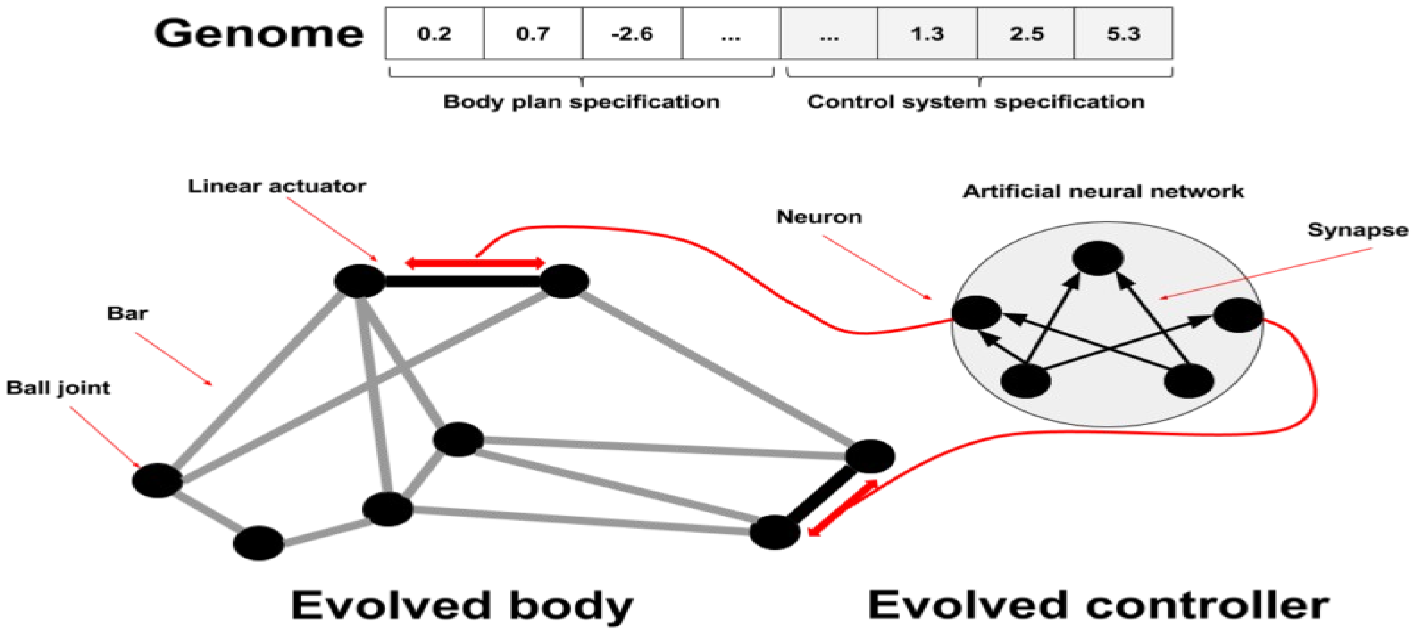
Evolutionary Robotics

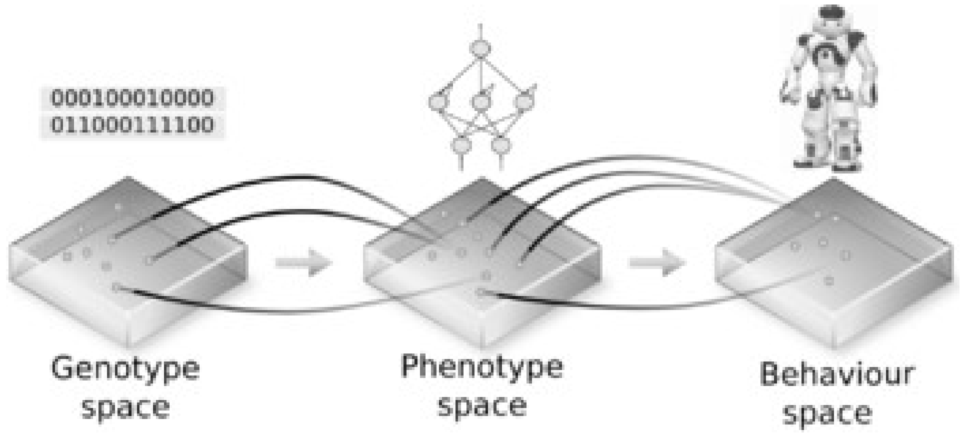
Evolutionary Robotics is a fascinating new combination of Evolutionary Computation and Robotics.

Evolutionary robotics is a field of research that employs evolutionary computation to generate robots that adapt to their environment through a process analogous to natural evolution. The generation and optimisation of robots are based on evolutionary principles of blind variations and survival of the fittest, as embodied in the neo-Darwinian synthesis (Gould, 2002).

Evolutionary robotics is typically applied to create control system for robots. Although less frequent, evolutionary robotics can also be applied to generate robot body plans, and to coevolve control systems and body plans simultaneously (Lipson and Pollack, 2000) (http://www.scholarpedia.org/article/Evolutionary_Robotics)

Evolutionary Robotics





(http://www.scholarpedia.org/article/Evolutionary_Robotics)

Evolutionary Robotics (Karl Sims)

Karl Sims is a digital media artist and visual effects software developer. He was the founder of GenArts, Inc., a creator of special effects software tools for the motion picture industry.
He previously held positions at Thinking Machines Corporation, Optomystic, and Whitney/Demos Productions.
Karl studied computer graphics at the MIT Media Lab, and Life Sciences as an undergraduate at MIT.
He is the recipient of various awards including two ARS Electronica Golden Nicas and a MacArthur Fellowship Award
(http://www.karlsims.com/)

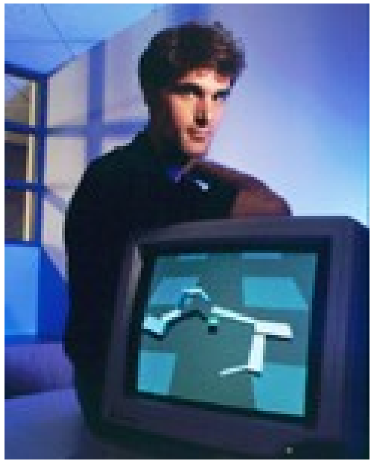


Photo by Hank Morgan
www.karlsims.com

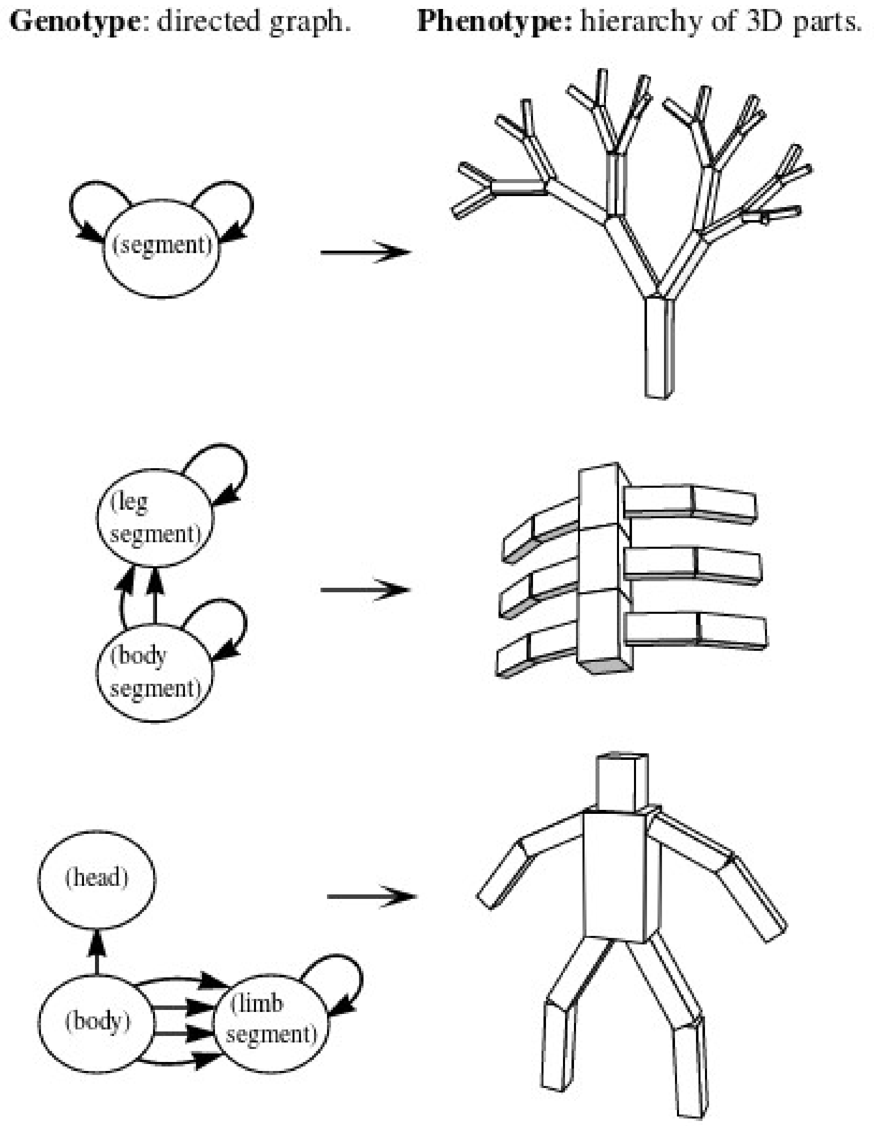
Evolutionary Robotics (Karl Sims)

Among his work in Artificial Life research is a series of publications w.r.t evolutionary Robotics:

Karl Sims: “Evolving 3D Morphology and Behavior by Competition”
Artificial Life IV Proceedings, ed. by R. Brooks & P. Maes, MIT Press, 1994, pp28-39
(http://www.karlsims.com/papers/alife94.pdf)

Karl Sims: "Evolving Virtual Creatures"
Computer Graphics (Siggraph '94 Proceedings), July 1994, pp.15-22.
(http://www.karlsims.com/papers/siggraph94.pdf)

Evolutionary Robotics (Karl Sims)



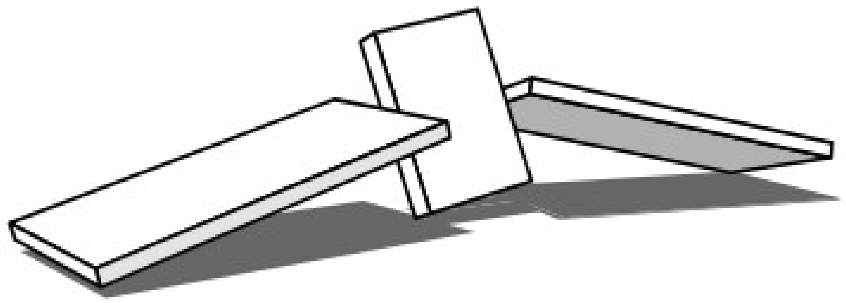


Figure 6a: The phenotype morphology generated from the evolved genotype shown in figure 5.

Karl Sims: "Evolving Virtual Creatures" Computer Graphics (Siggraph '94 Proceedings), July 1994, pp.15-22.

Evolutionary Robotics (Karl Sims)

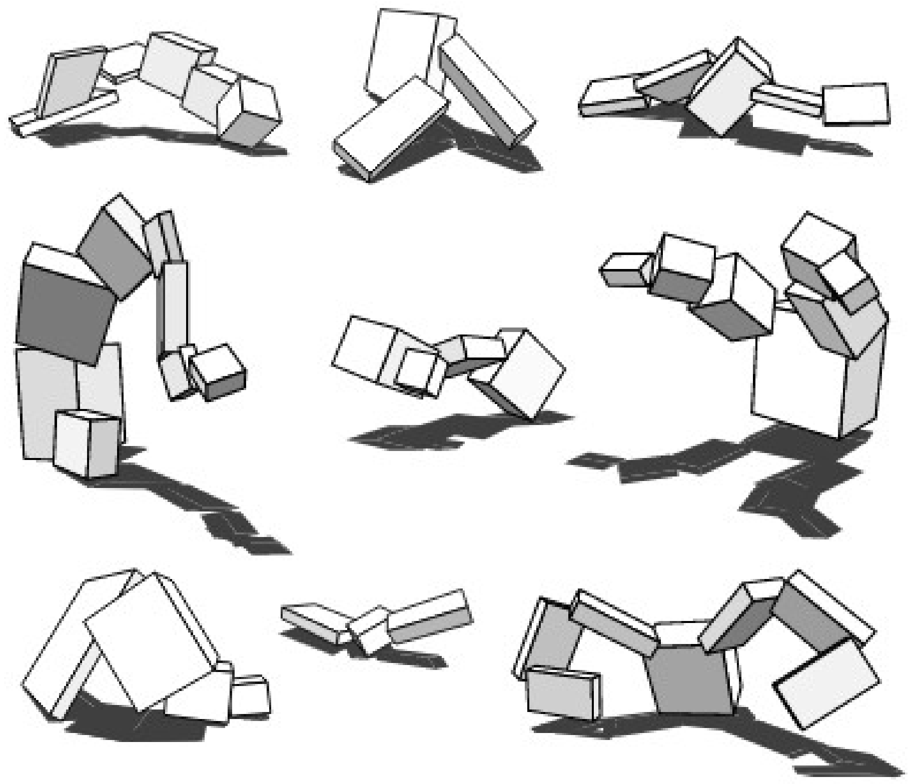


Figure 7: Creatures evolved for walking.

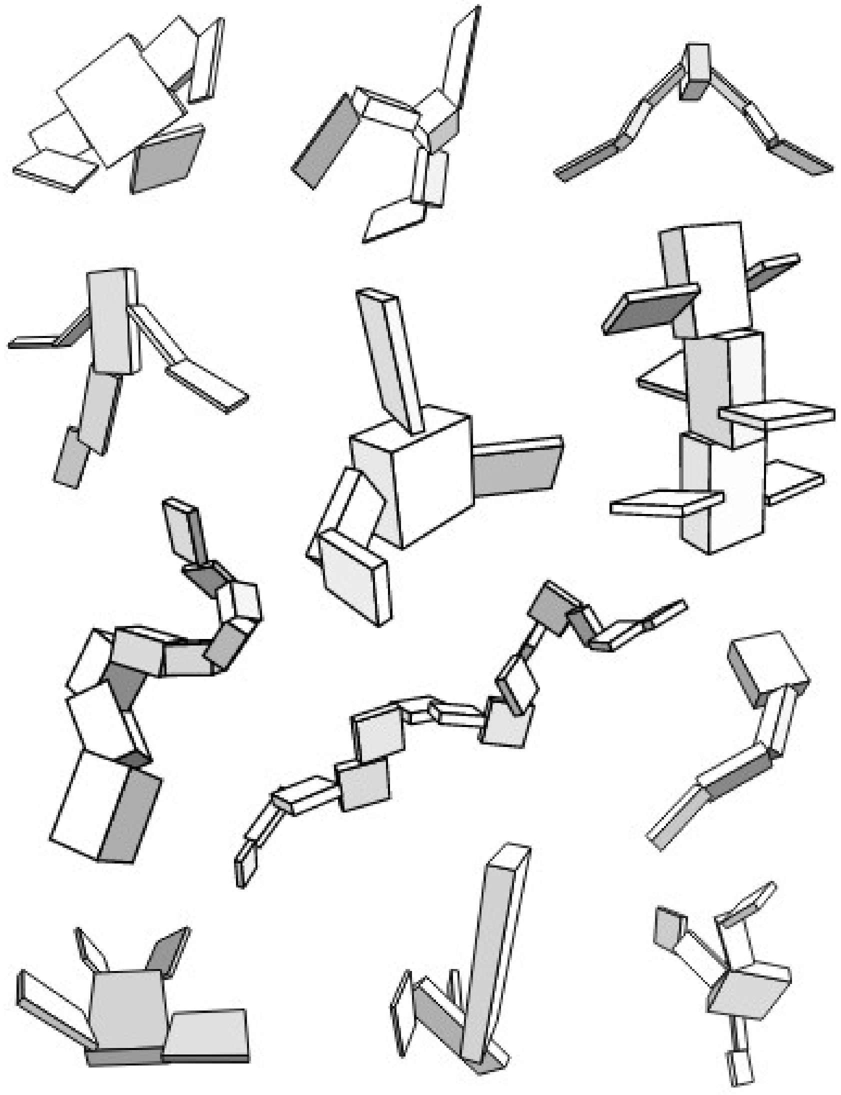
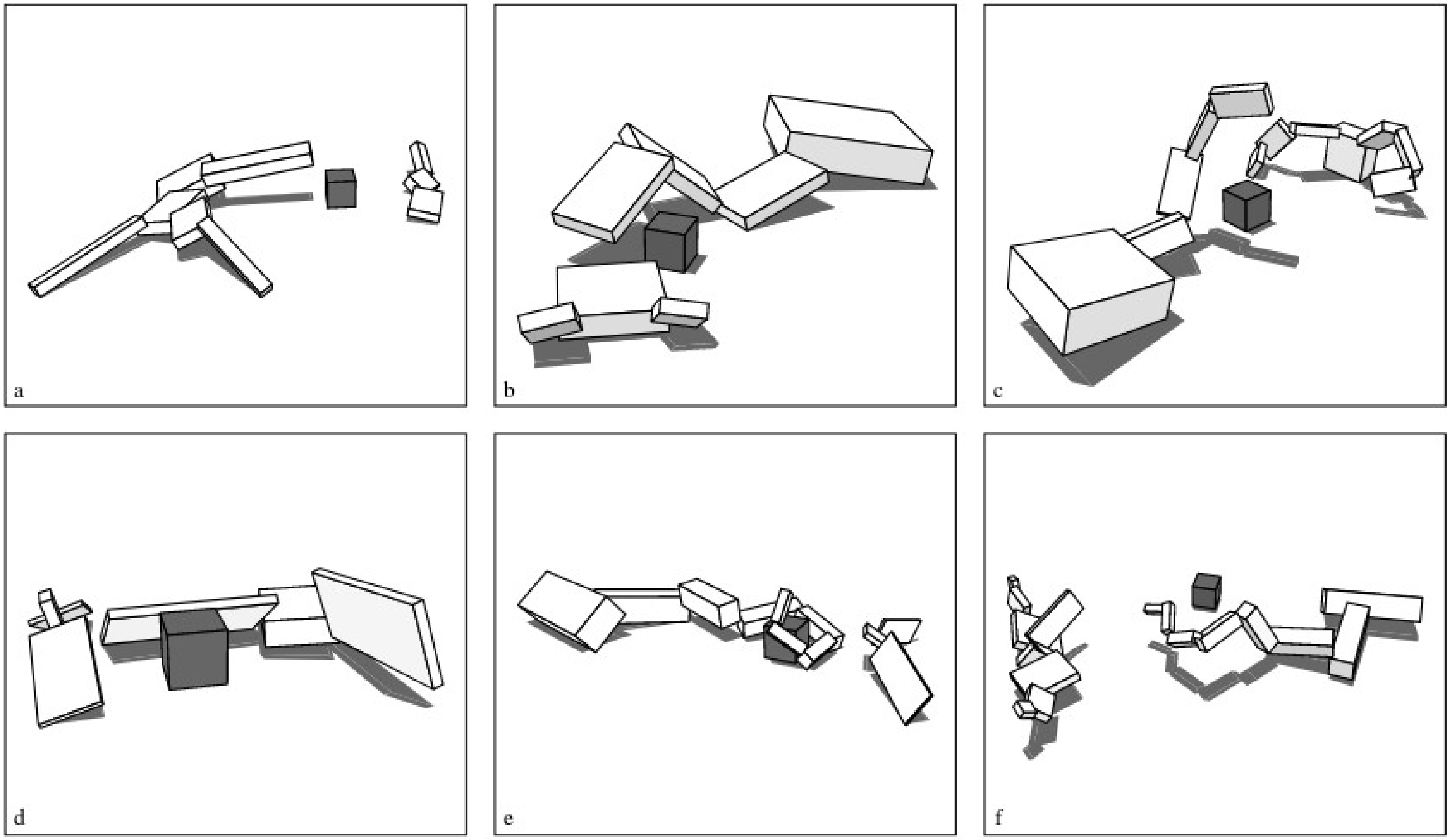


Figure 6: Creatures evolved for swimming.

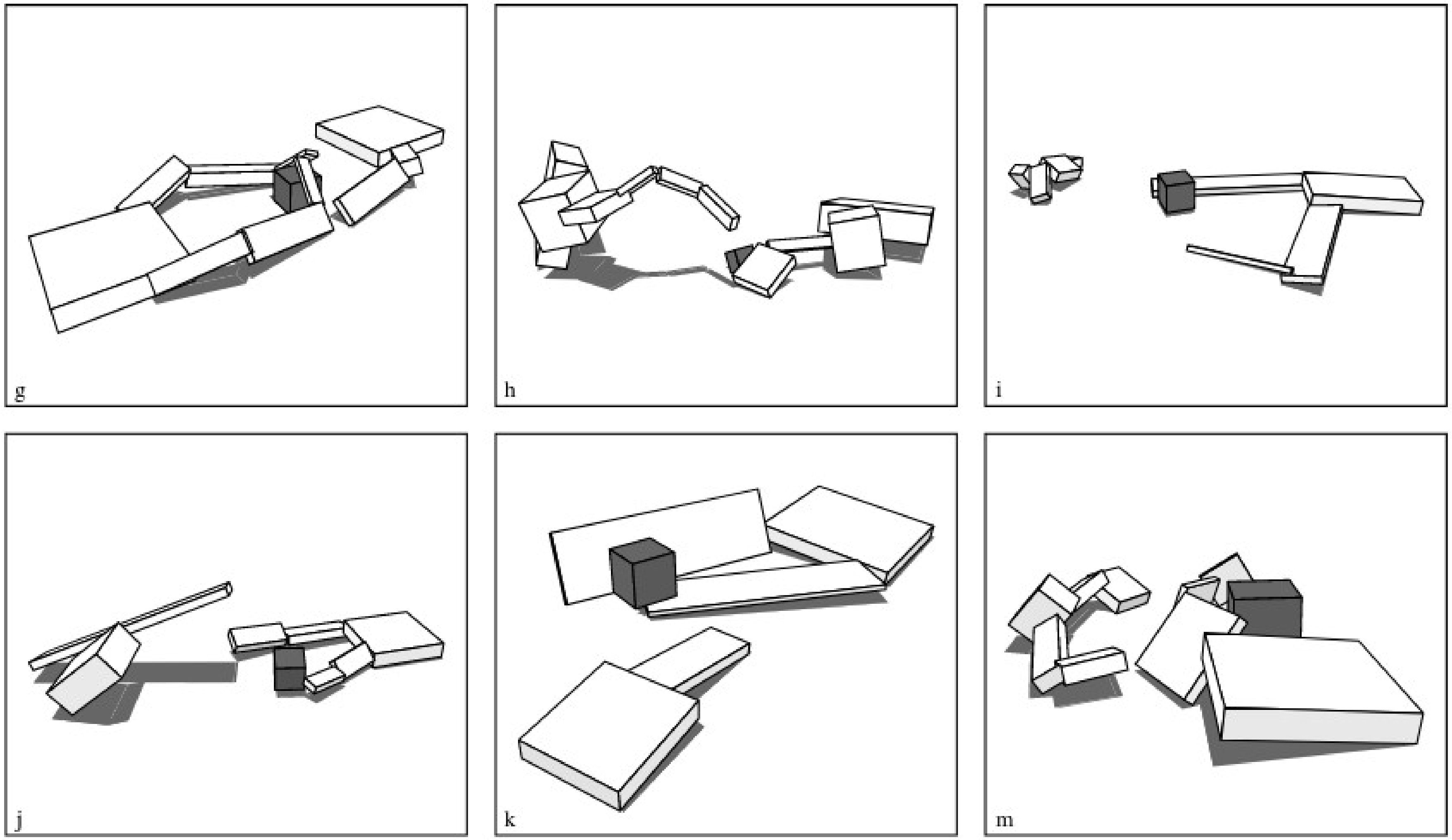
Karl Sims: "Evolving Virtual Creatures" Computer Graphics (Siggraph '94 Proceedings), July 1994, pp.15-22.

Evolutionary Robotics (Karl Sims)



Karl Sims: "Evolving Virtual Creatures"
Computer Graphics (Siggraph '94 Proceedings), July 1994, pp.15-22.

Evolutionary Robotics (Karl Sims)



Karl Sims: "Evolving Virtual Creatures"
Computer Graphics (Siggraph '94 Proceedings), July 1994, pp.15-22.

Bionics

Bionics or Biologically inspired engineering is the application of biological methods and systems found in nature to the study and design of engineering systems and modern technology.

The word bionic was coined by Jack E. Steele in 1958, possibly originating from the technical term bion (pronounced BEE-on; from Ancient Greek: βίος bíos), meaning 'unit of life' and the suffix -ic, meaning 'like' or 'in the manner of', hence 'like life'.

https://en.wikipedia.org/wiki/Bionics

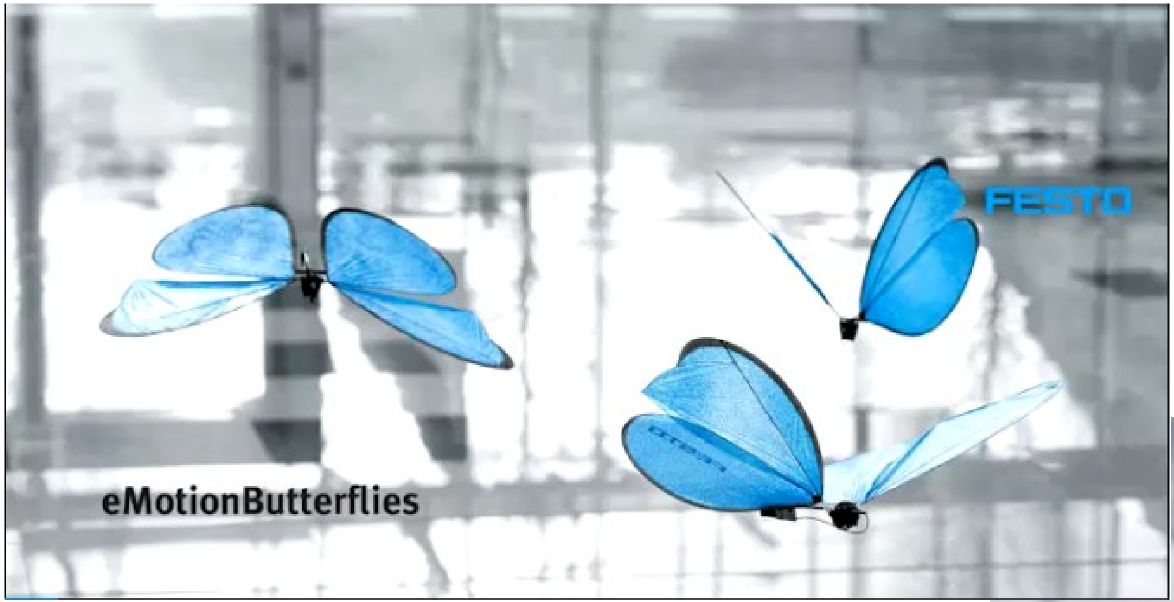
Bionics, Bionic Robots (FESTO)

The Company FESTO has developed a series of biology like robotic systems that are extremely innovative.

Since 2006 they are producing and showing a wide variety of fascinating bionic developments:

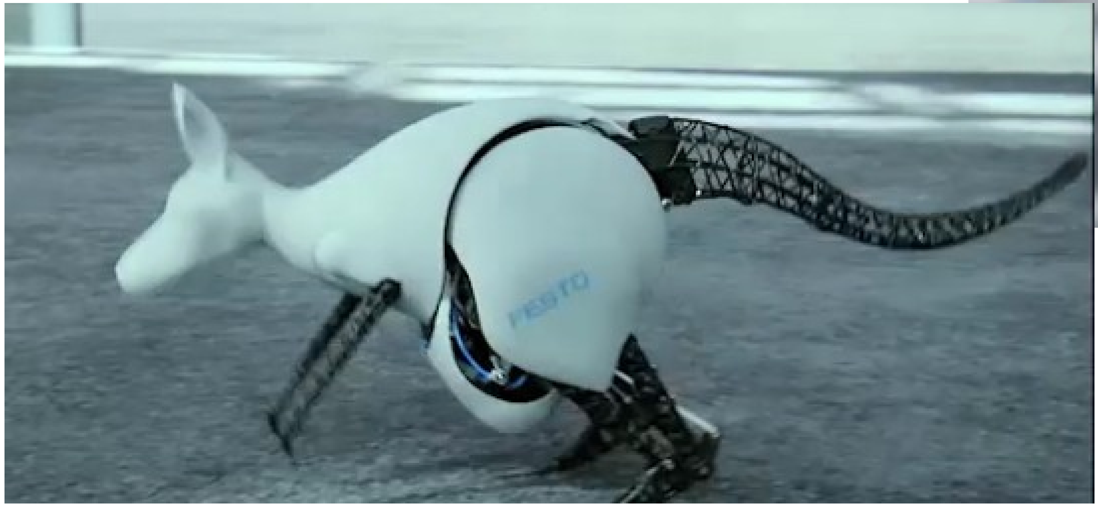
For a gallery of developments see:
https://www.festo.com/group/en/cms/10156.htm
https://www.festo.com/group/en/cms/10218.htm

Bionics, Bionic Robots (FESTO)



FESTO: eMotionButterflies

FESTO: BionicOpter



FESTO: BionicKangaroo

https://www.festo.com/group/en/cms/10218.htm

Artificial Life Summer 2025

Subsumption Architecture Artificial Life Extras

Master Computer Science [MA-INF 4201]
Mon 14:15 – 15:45, HSZ, HS-2

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn

Some information:

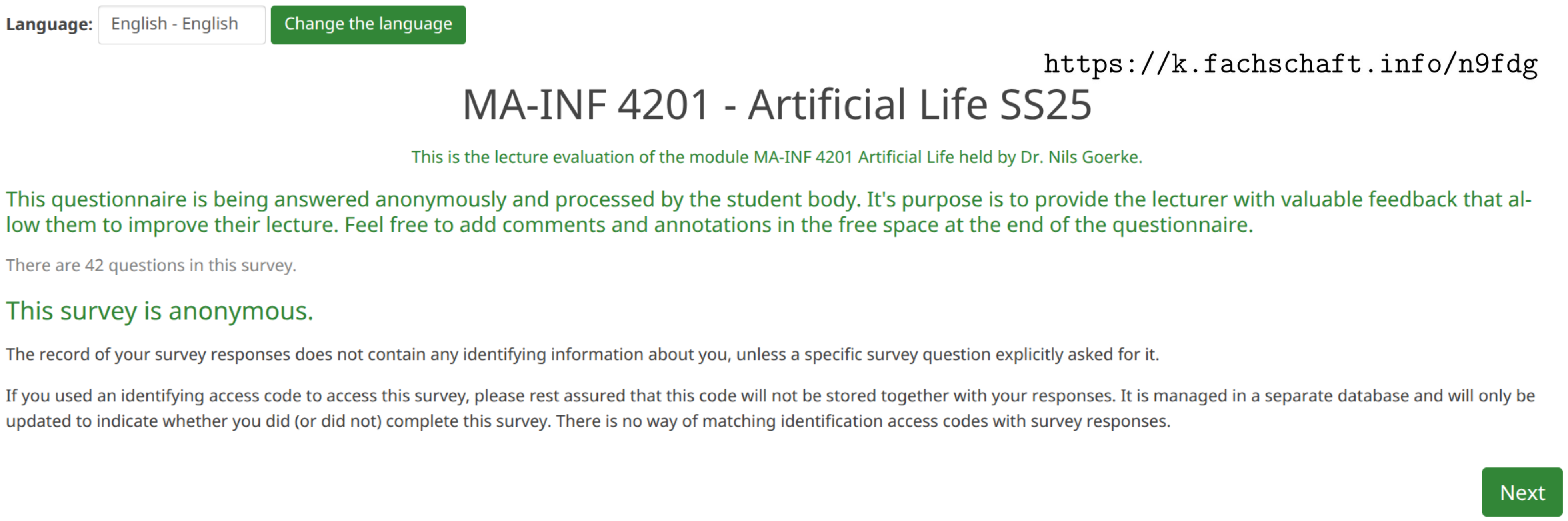
Next/last Artificial Life Lecture :
next **Monday, July 14 2025, 14:15 – 15:45:**

- Is dedicated for **Questions and Answers**.
  Please be prepared for this.
- Preparation for the **examination,**
  tipps, tricks, practicing

Student’s questionaire

The Student Body Computer Science (Fachschaft Informatik) provides every semester an anonymous questionnaire about the lecture.

The link for Artificial Life 2025:
https://k.fachschaft.info/n9fdg



https://k.fachschaft.info/n9fdg

MA-INF 4201 - Artificial Life SS25

This is the lecture evaluation of the module MA-INF 4201 Artificial Life held by Dr. Nils Goerke.

This questionnaire is being answered anonymously and processed by the student body. It's purpose is to provide the lecturer with valuable feedback that allow them to improve their lecture. Feel free to add comments and annotations in the free space at the end of the questionnaire.

There are 42 questions in this survey.

This survey is anonymous.

The record of your survey responses does not contain any identifying information about you, unless a specific survey question explicitly asked for it.

If you used an identifying access code to access this survey, please rest assured that this code will not be stored together with your responses. It is managed in a separate database and will only be updated to indicate whether you did (or did not) complete this survey. There is no way of matching identification access codes with survey responses.

Some information:

Next/last Artificial Life Lecture :
next **Monday, July 14 2025, 14:15 – 15:45:**

- Is dedicated for **Questions and Answers**.
  Please be prepared for this.
- Preparation for the **examination,**
  tipps, tricks, practicing

Artificial Life Summer 2025

Subsumption Architecture Artificial Life Extras

Thank you for listening to me

Master Computer Science [MA-INF 4201]
Mon 14:15 – 15:45, HSZ, HS-2

Dr. Nils Goerke, Autonomous Intelligent Systems,
Department of Computer Science, University of Bonn