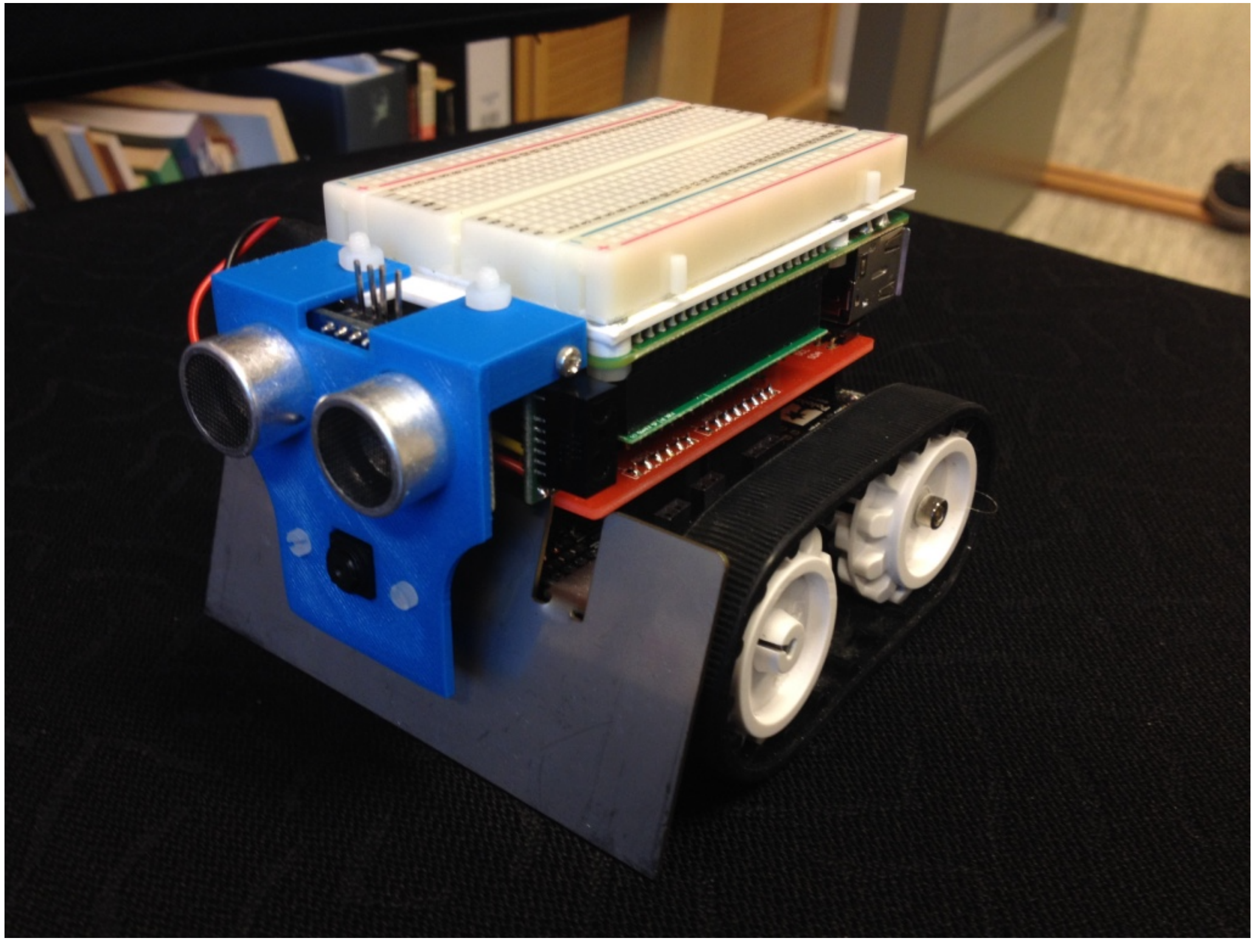
Project 6:
• Behavior based robot control☺



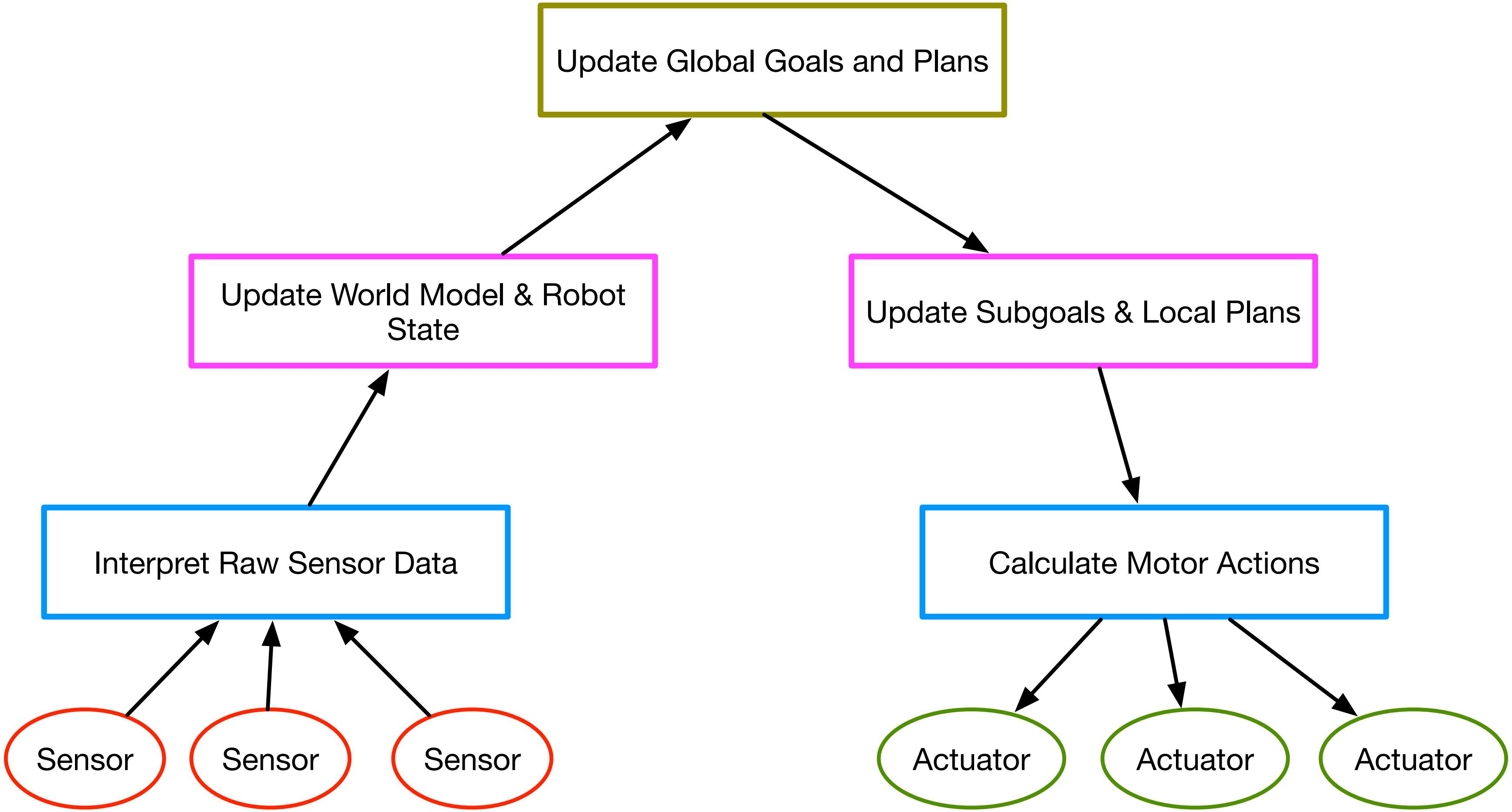
Must have behavior:

- Wander
- Line follower
- Avoid walls

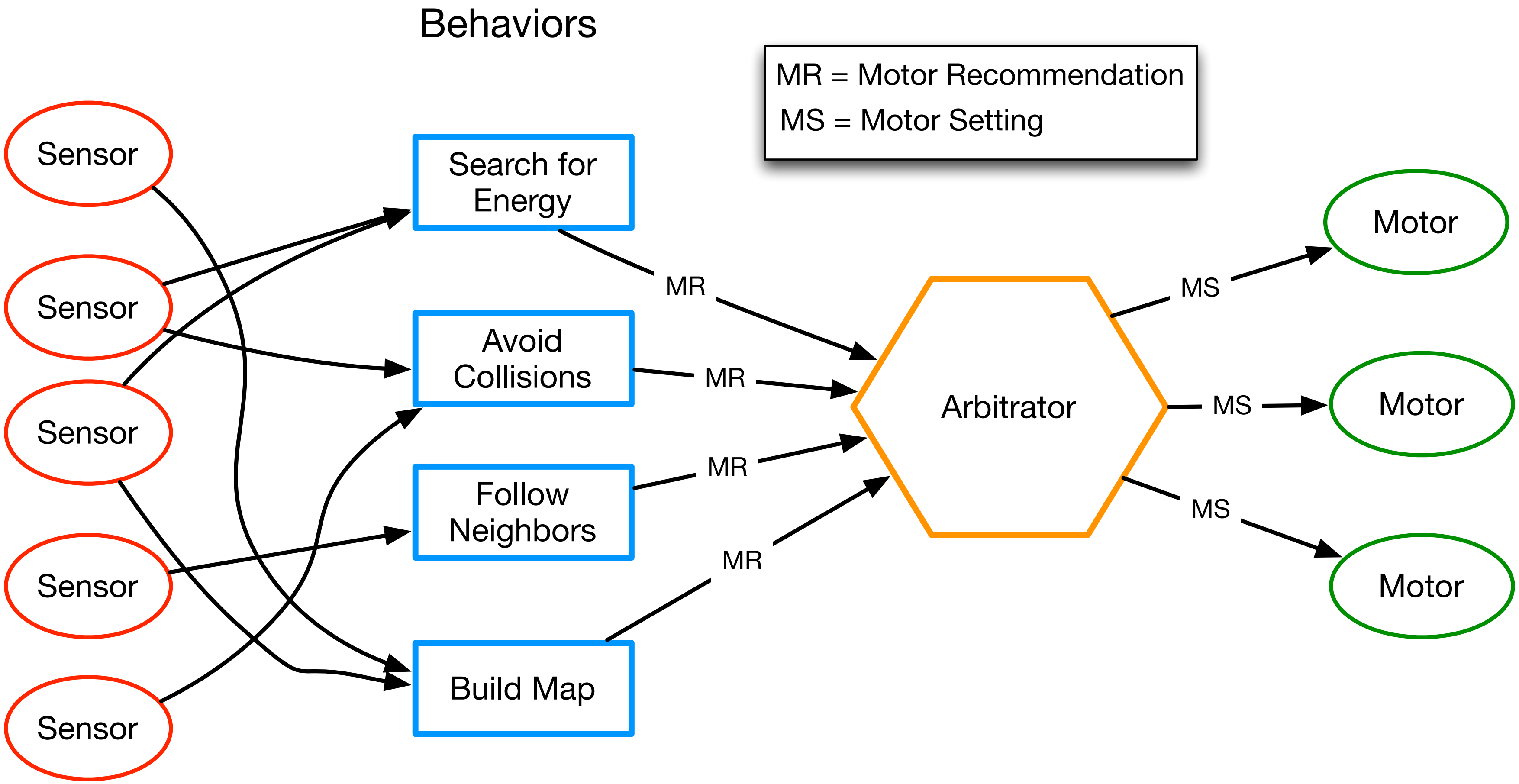
And may:

- ”Build map”?
- etc

Typical example of the classic AI robotic control hierarchy

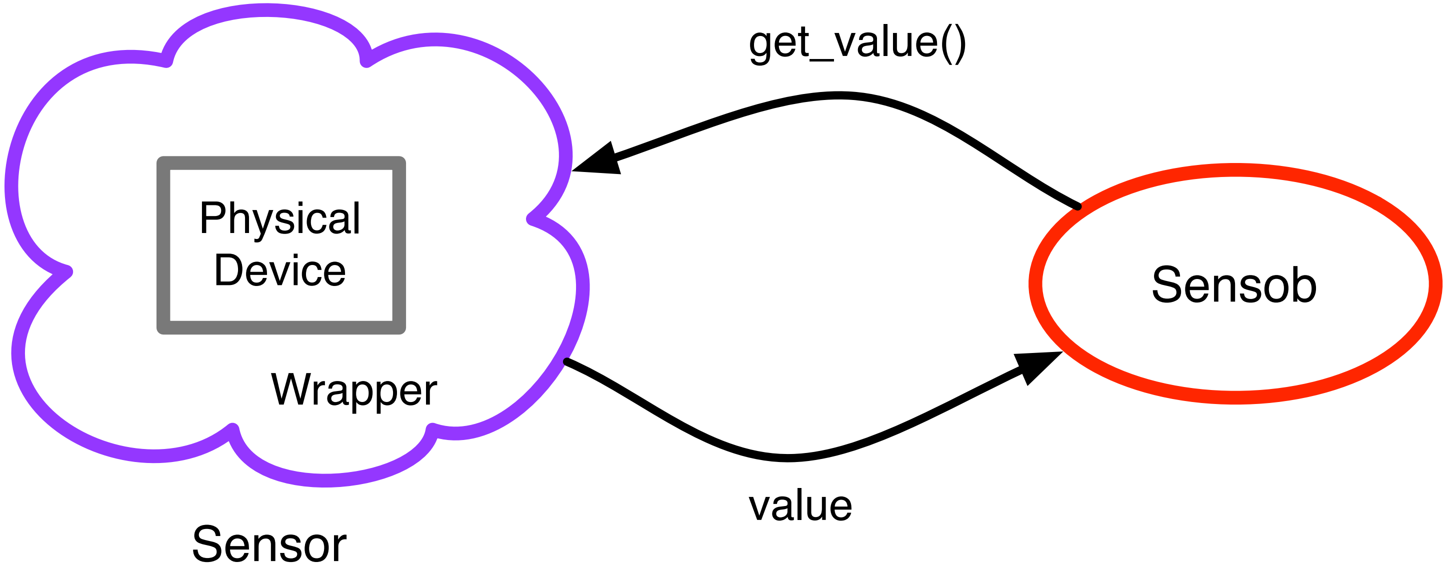

Update Global Goals and Plans
Update World Model & Robot State
Update Subgoals & Local Plans
Interpret Raw Sensor Data
Calculate Motor Actions
Sensor
Sensor
Sensor
Actuator
Actuator
Actuator

Example of behavior-based robotic control



MR = Motor Recommendation
MS = Motor Setting

Key components for Sensor

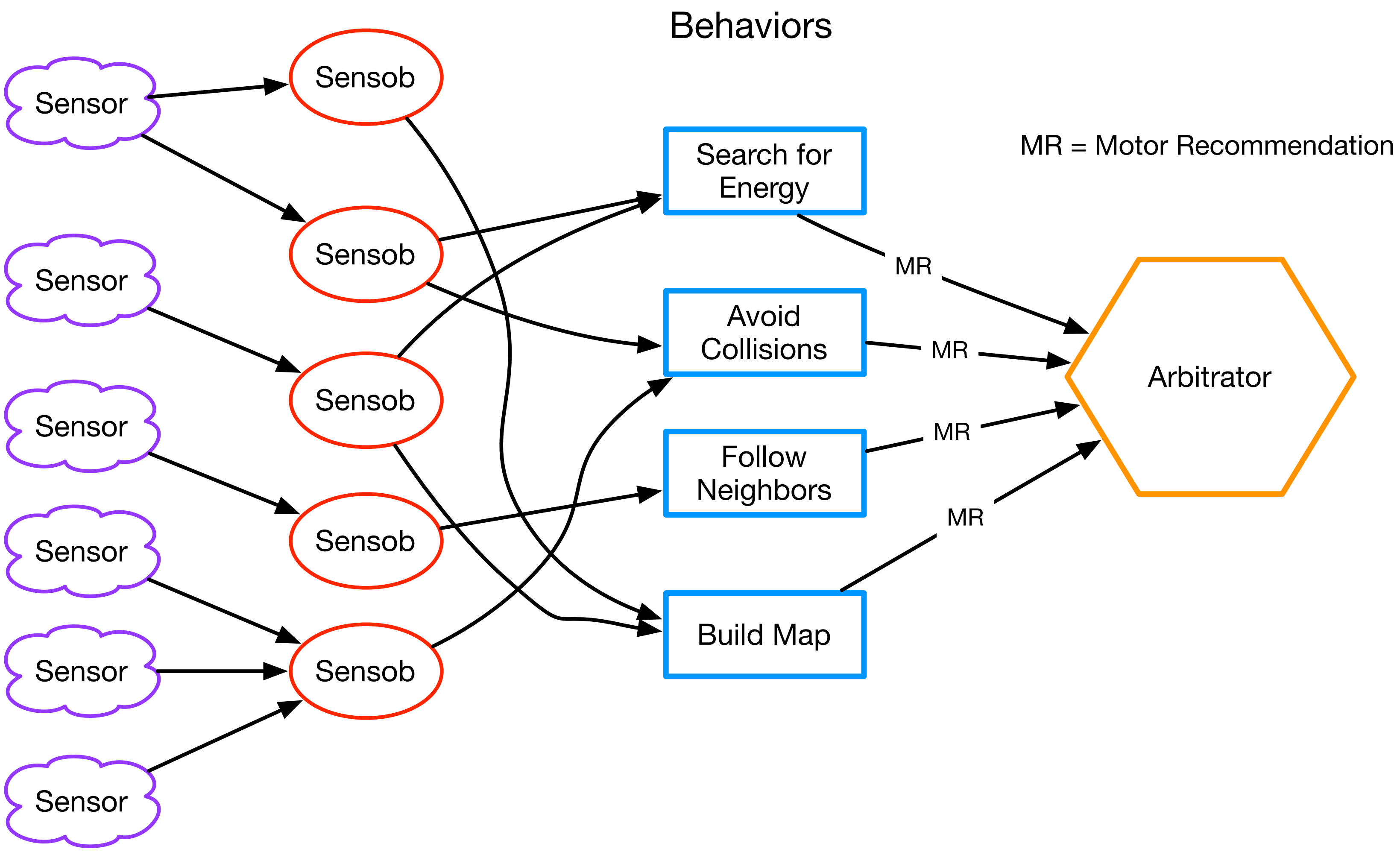


Sensor methods:

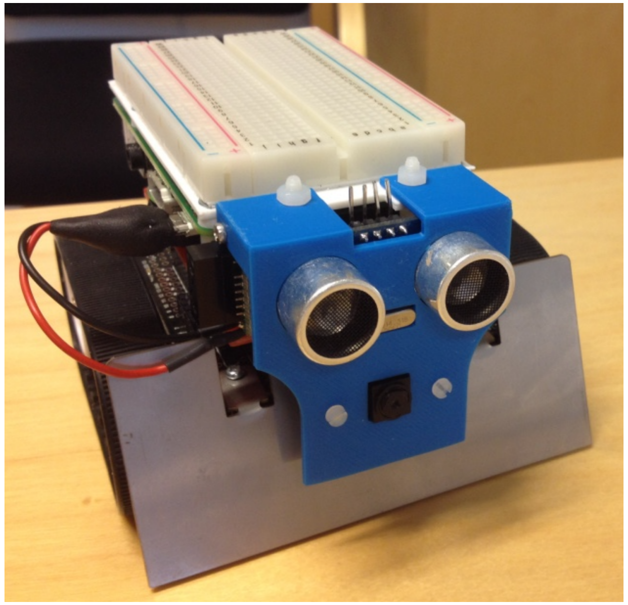
- update
- get_value
- reset

See helpcode

Class Sensob

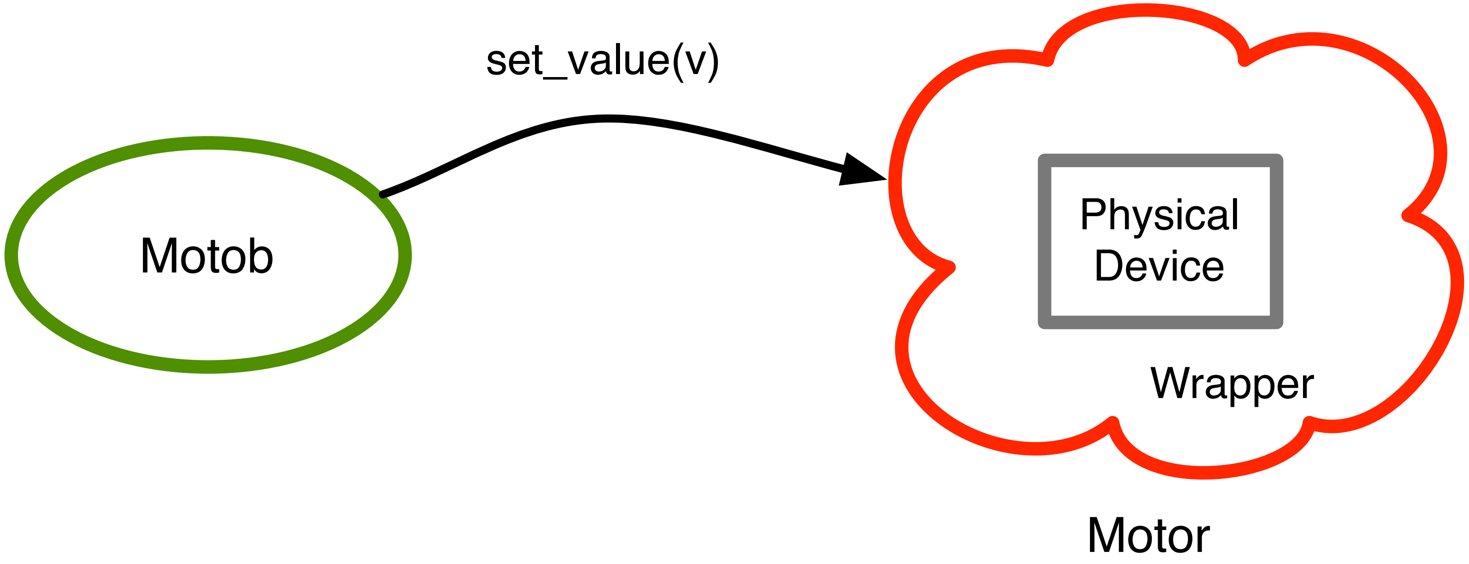


Class Sensob



- Main instance variables:
  - **associated_sensors**
  - **value**
- Main method:
  - **update** fetch sensor values and convert into sensob values

Key components for Motor



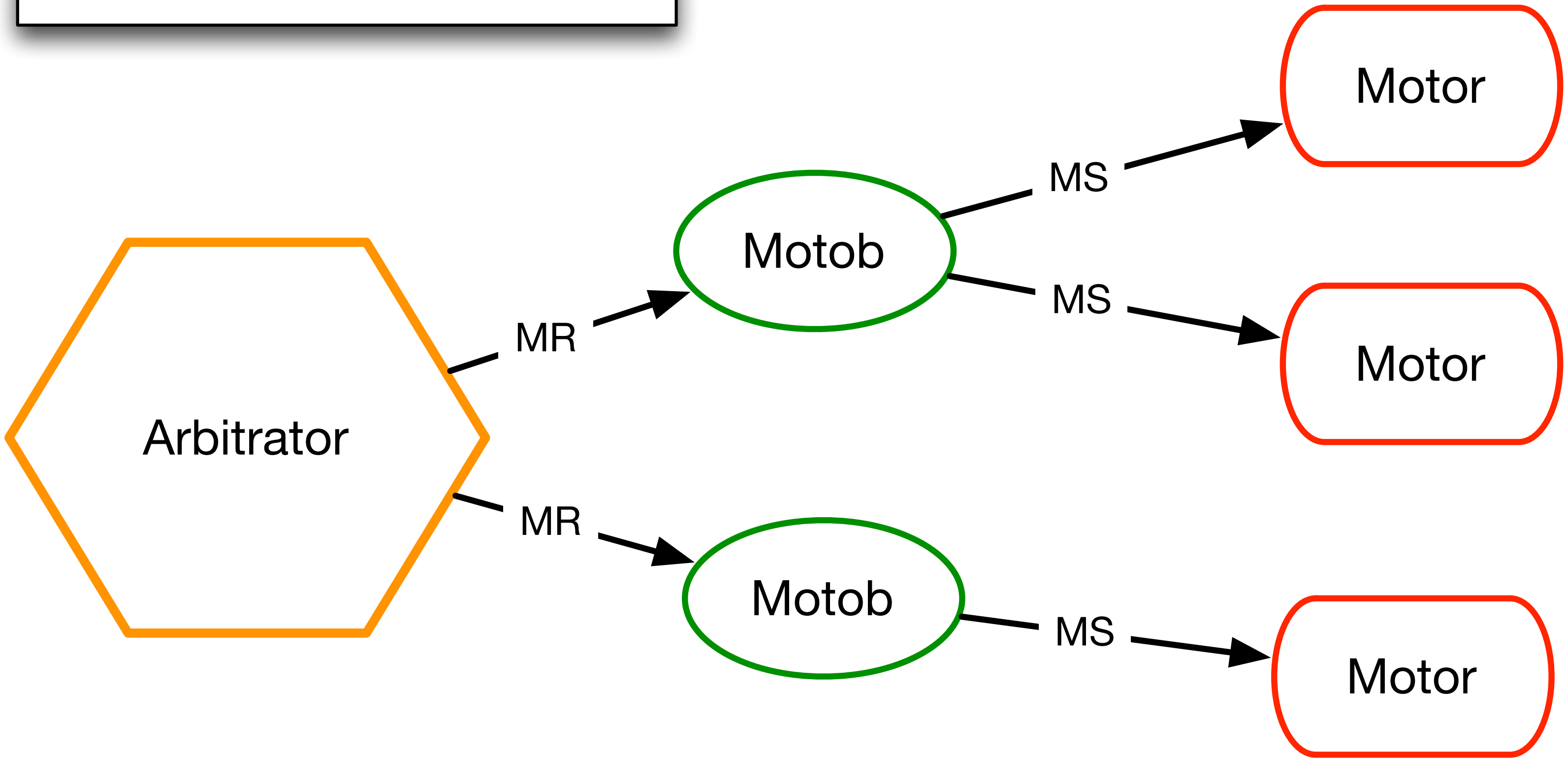
Class Motob

- Instance variables:
  - **motors** - a list of the motors whose settings will be determined by the motob.
  - **value** - a holder of the most recent motor recommendation sent to the motob.
- Methods:
  - **update** - receive a new motor recommendation, load it into the value slot, and operationalize it.
  - **operationalize** - convert a motor recommendation into one or more motor settings, which are sent to the corresponding motor(s).

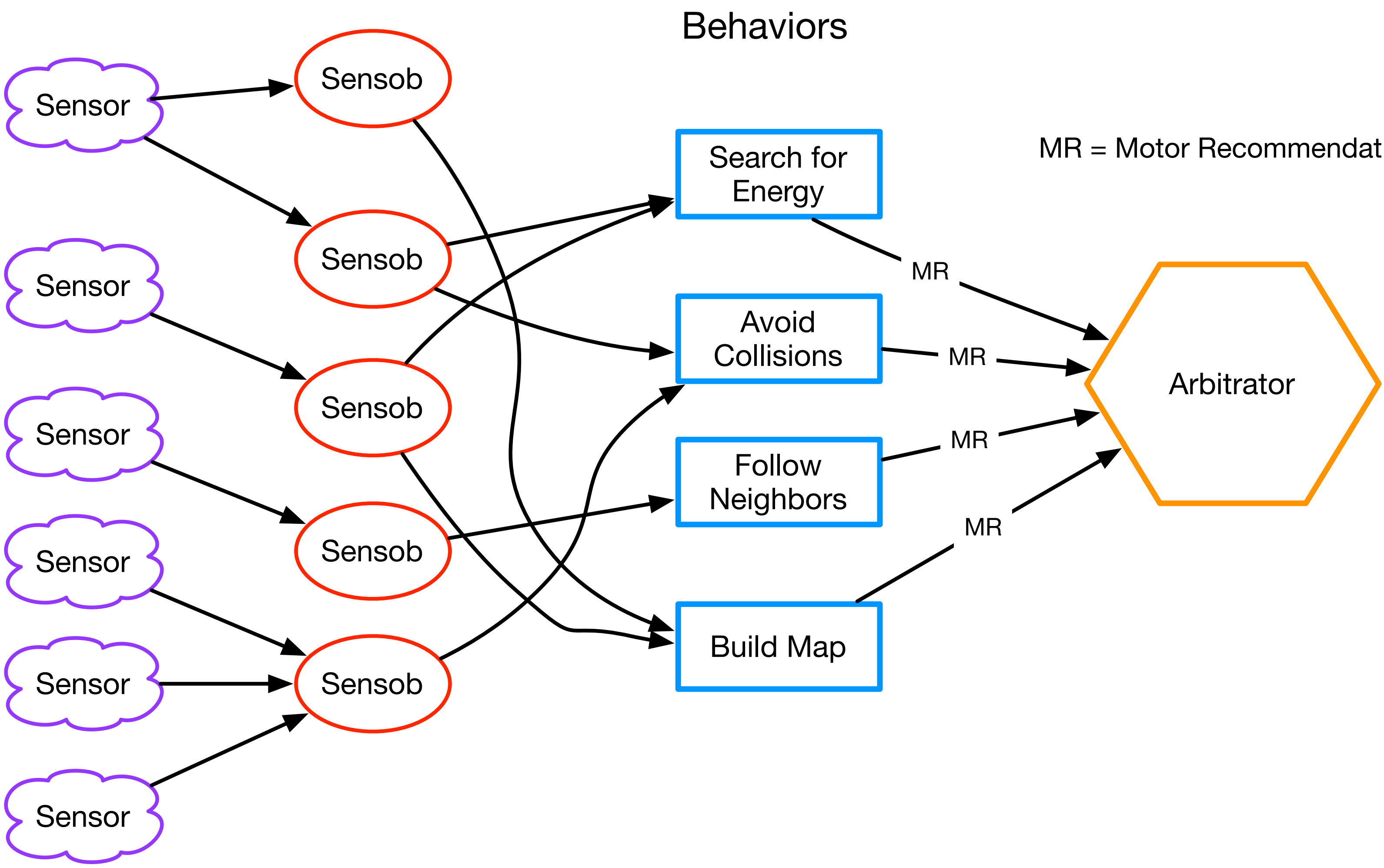
Basic relationships Arbitrator and Motob

MR = Motor Recommendation

MS = Motor Setting



Basic relationships



MR = Motor Recommendation

Class BBCON: instance variables

- Should contain:
  - **behaviors** - a list of all the behavior objects used by the bbcon
  - **active_behaviors** - a list of all behaviors that are currently active.
  - **sensobs** - a list of all sensory objects used by the bbcon
  - **motobs** - a list of all motor objects used by the bbcon
  - **arbitrator** - the arbitrator object that will resolve actuator requests produced by the behaviors.
- Also useful
  - **current_time_step**
  - **inactive_behaviors**
  - **controlled_robot**

Class BBCON: methods

- Should have following simple procedures:
  - **add_behavior** - append a newly-created behavior onto the behaviors list.
  - **add_sensob** - append a newly-created sensob onto the sensobs list.
  - **activate_behavior** - add an existing behavior onto the active-behaviors list.
  - **deactive_ behavior** - remove an existing behavior from the active behaviors list.

- **MUST** have:
  - **run_one_timestep** - which constitutes the **core BBCON activity** (see description on next slide)

Description **run_one_time_step**

- **Update all sensobs** - These updates will involve querying the relevant sensors for their values, along with any pre-processing of those values (as described below)
- **Update all behaviors** - These updates involve reading relevant sensob values and producing a motor recommendation.
- **Invoke the arbitrator** by calling **arbitrator.choose action**, which will choose a winning behavior and return that behavior's motor recommendations and halt request flag.
- **Update the motobs** based on these motor recommendations. The motobs will then update the settings of all motors.
- **Wait** - This pause (in code execution) will allow the motor settings to remain active for a short period of time, e.g., one half second, thus producing activity in the robot, such as moving forward or turning.
- **Reset the sensobs** - Each sensob may need to reset itself, or its associated sensor(s), in some way.

Important requirement for Class Behavior

It **violates the fundamental principles** of BBR to design behaviors that communicate directly with one another.

All interaction occurs indirectly via either the arbitrator or via information posted by one behavior (in the bbcon) and read by a second behavior (from the bbcon).

One important condition for receiving a passing mark on this project is that your group's code obey's this simple, yet extremely important, principle

Class Behavior: Primary instance variables

- **bbcon** - pointer to the controller that uses this behavior.
- **sensobs** - a list of all sensobs that this behavior uses.
- **motor recommendations** - a list of recommendations, one per motob, that this behavior provides to the arbitrator. In this assignment, we assume that ALL motobs (and there will only be one or a small few) are used by all behaviors.
- **active flag** - boolean variable indicating that the behavior is currently active or inactive (analyse sensor information and MAKE motor recommandations)
- **halt request** - some behaviors can request the robot to completely halt activity (and thus end the run).
- **priority** - a static, pre-defined value indicating the importance of this behavior.
- **match degree** - a real number in the range [0, 1] indicating the degree to which current conditions warrant the performance of this behavior.
- **weight** - the product of the priority and the match degree, which the arbitrator uses as the basis for selecting the winning behavior for a timestep.

Class **Behavior**: methods

- **consider_deactivation** - whenever a behavior is active, it should test whether it should deactivate.
- **consider_activation** - whenever a behavior is inactive, it should test whether it should activate.
- **update** - the main interface between the bbcon and the behavior (update activity status, call sense_and_act, update behavior weight)
- **sense_and_act** - the core computations performed by the behavior that use sensob readings to produce motor recommendations (and halt requests).

Summary: class **Behavior**

- In general, behaviors can perform many operations, but they **MUST**:
  - consider activation or deactivation
  - produce motor recommendations
  - update the match degree

- and they **MUST NOT** communicate directly with other behaviors.

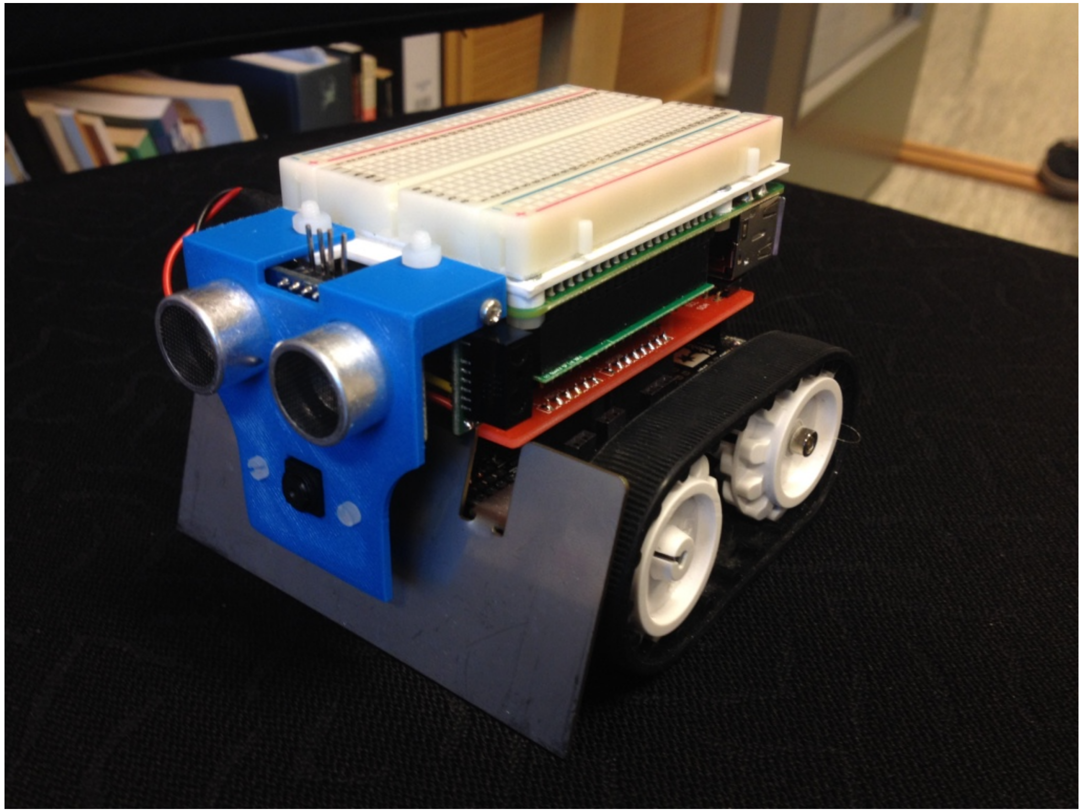
Class **Arbitrator**

- Instance variables:
  - **bbcon** – pointer to BBCON to fetch all active behaviors
- Methods:
  - **choose_action** which returns motor recommandation and halt flag

Deterministic or stochastic (weight based)

Requirements behavior

- **Must have**:
  - Line follower
  - Avoid walls
  - Wander
  - ”build map” ?



GPIO – bruk av ekstra pinner

- Se beskrivelse ledige pinner på **wiki**
  -
- Hjelpekode Oppgave 6 viser bruk av GPIO
  - Se for eksempel **irproximity.py**

Getting started with Zumo

- Connect robot to ethernet with an ethernet cable
- How to know IP-address?
  - *see IP-document on It´s learning/wiki (use MAC-address)*
  - *folk.ntnu.no/haakongi/TDT4113/get_ip_from_mac.php*

MAC: b8:27:eb:1a:36:50

IP: 129.241.111.162

Getting started with Zumo (2)

- Use ssh to access the robot

  ssh pi@<your IP address>

username (pi)
password (raspberry)

- Better to login as root:

  ssh root@<your IP address>

password (raspberry)

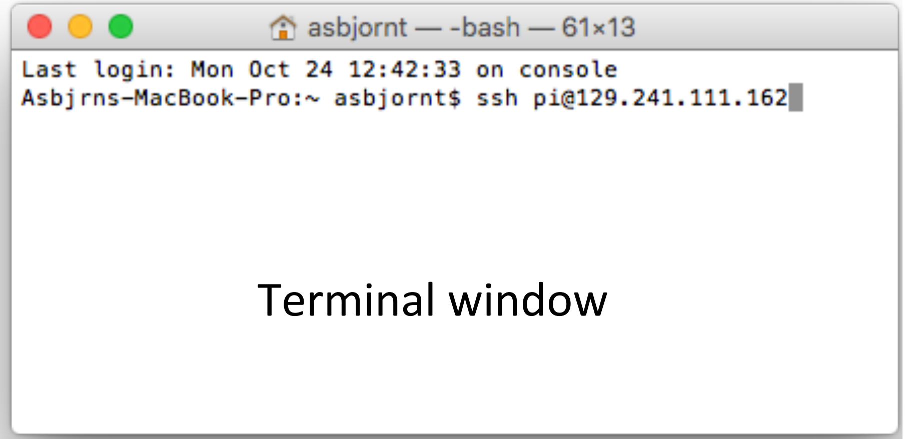
- If personalize robot (add new user)

  sudo useradd monte

  sudo passwd monte

  ssh monte@<your IP address>

  sudo passwd



```
Last login: Mon Oct 24 12:42:33 on console
Asbjrns-MacBook-Pro:~ asbjornt$ ssh pi@129.241.111.162
```

Terminal window

Transfer programs from laptop to Zumo

- Navigate to laptop directory *mylaptop/robot*

  sftp root@<your IP address>

- Navigate to *home/pi/robot* (use ls, pwd, cd)
- Use **put** and **get** commands in ftp

  put my controller.py

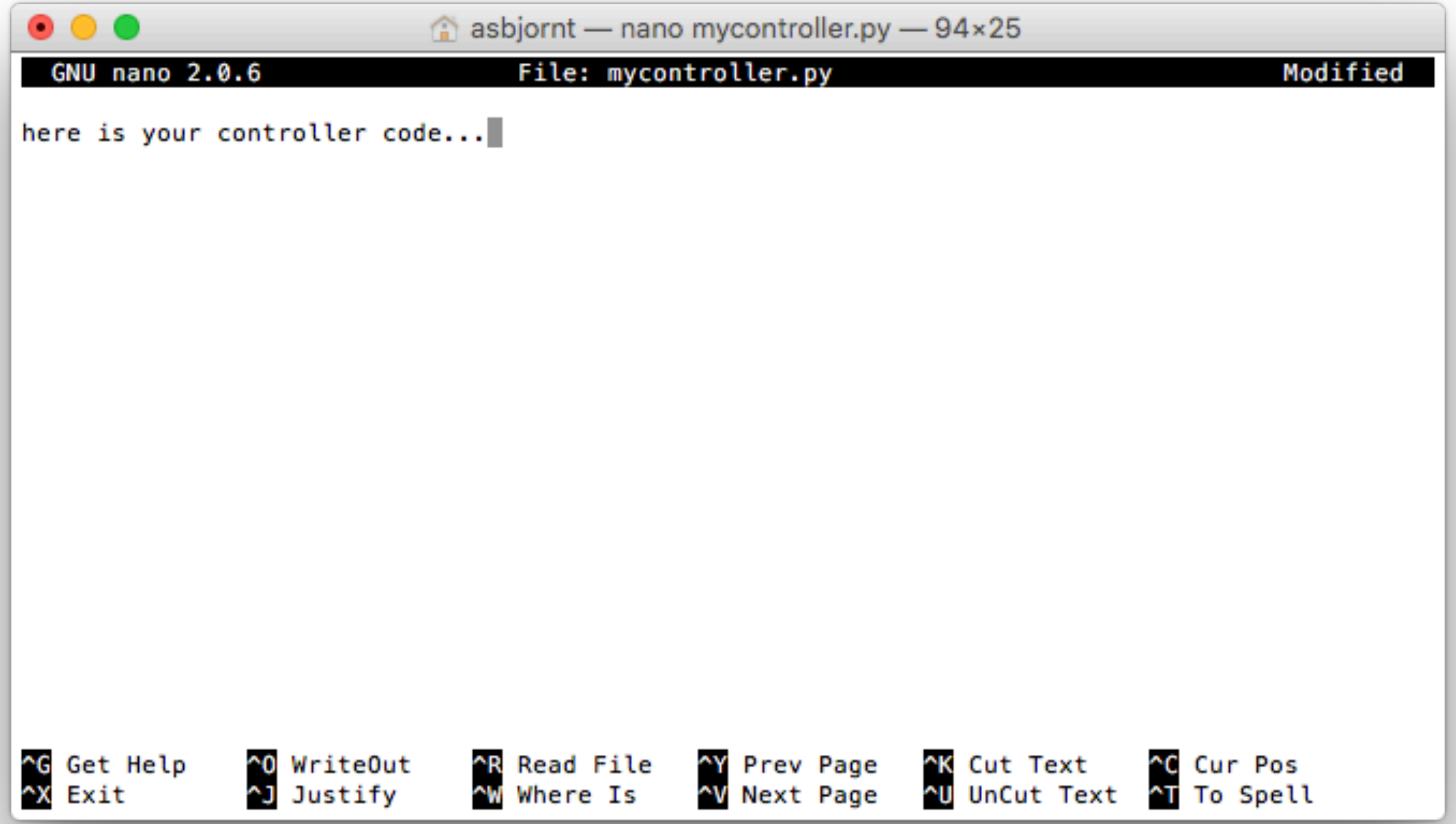
  get image.png (robot image to laptop)

Alternative for windows: **Putty** and **Filezilla**

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
https://filezilla-pro ject.org/download.php?typ e=client

Local editing on the robot

- Use Nano in robot´s terminal window



```
  GNU nano 2.0.6                   File: mycontroller.py                              Modified

here is your controller code...

^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```