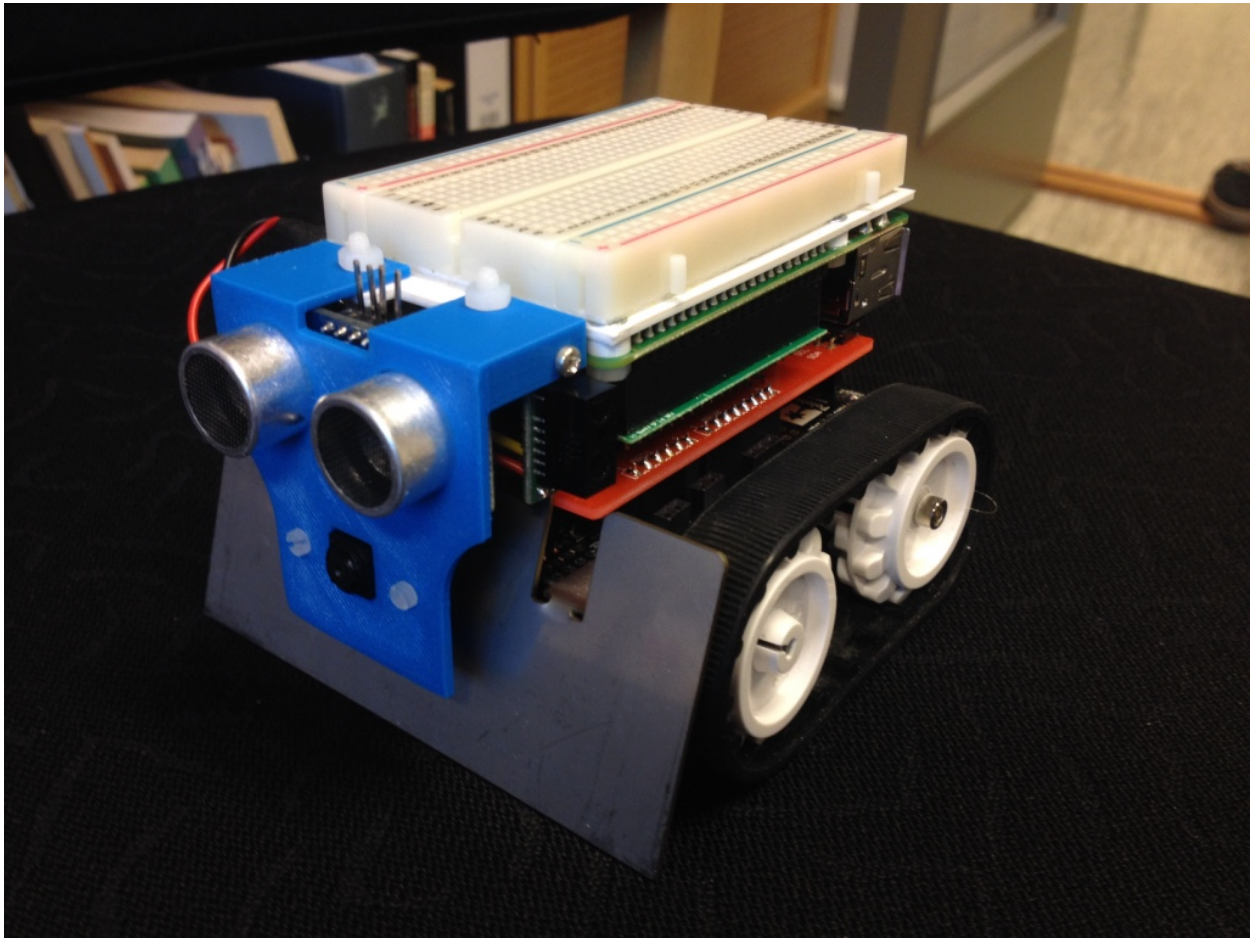
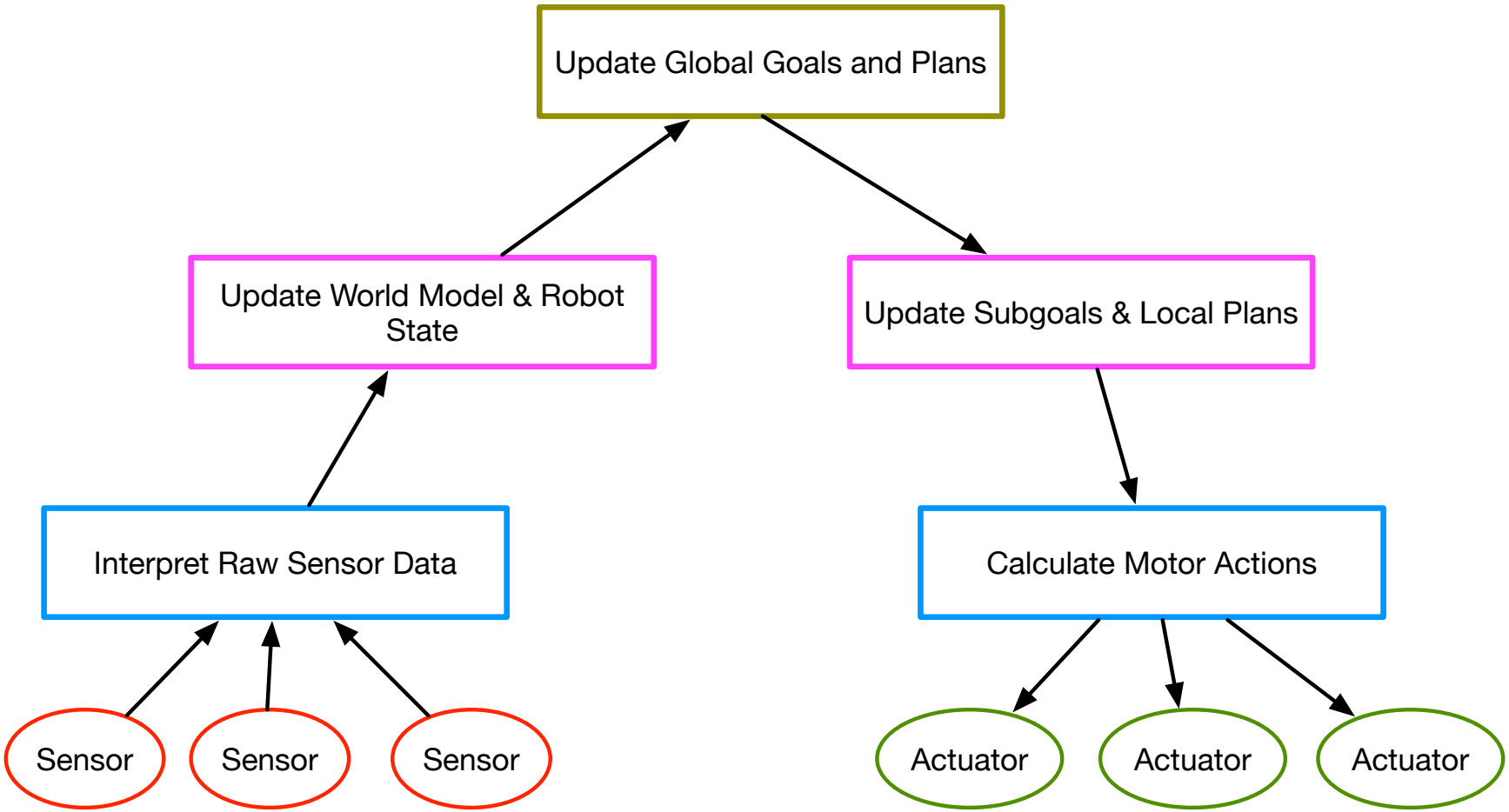
Project 6:

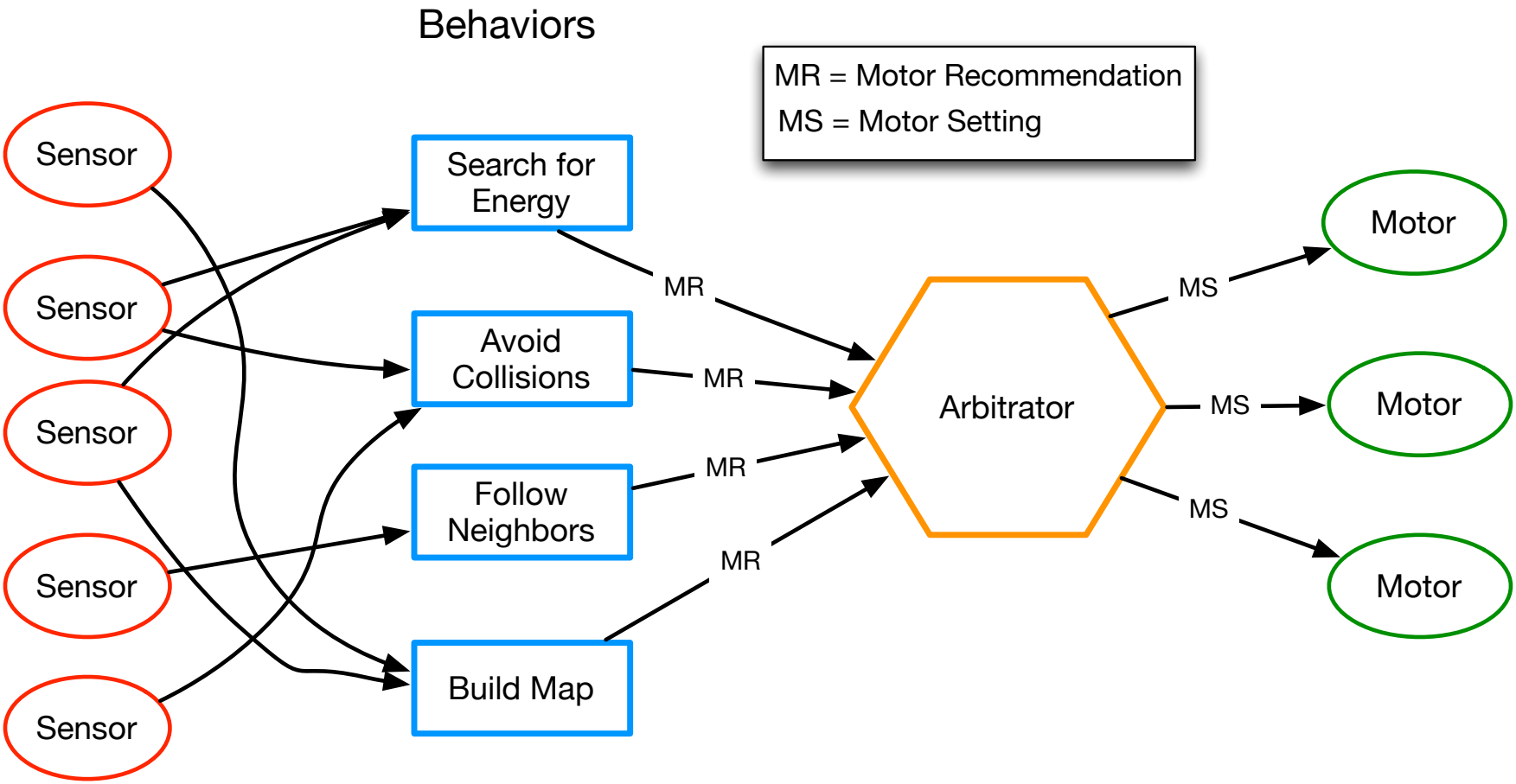
- Behavior based robot control☺



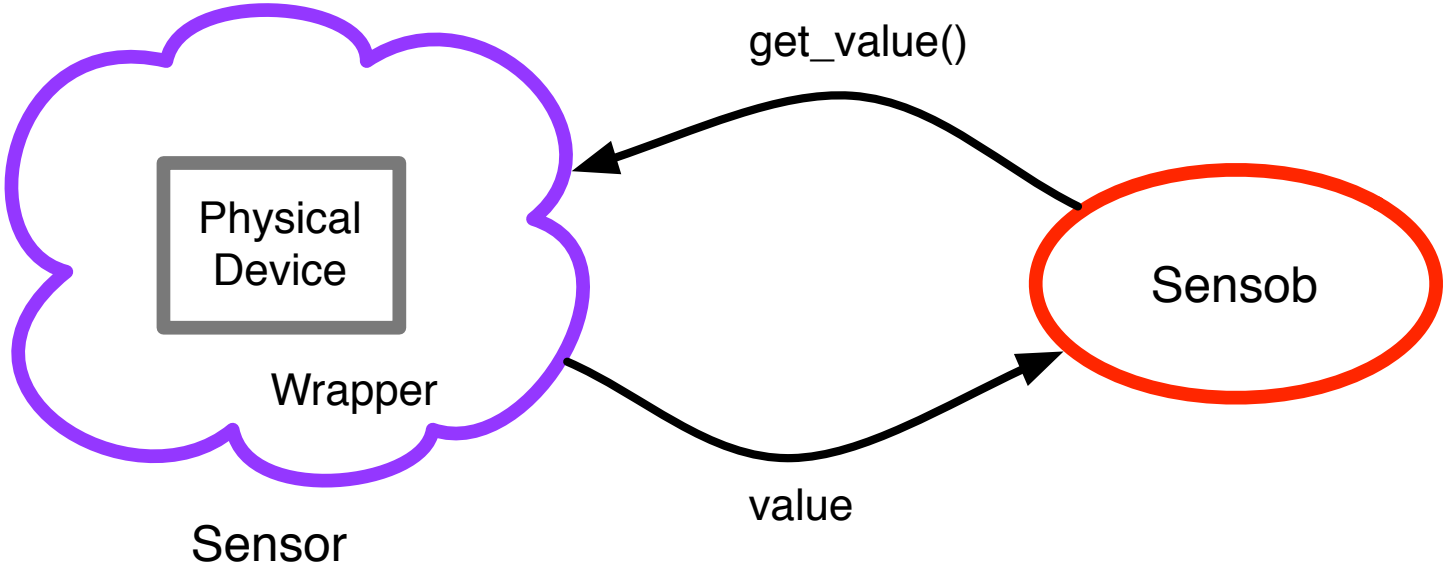
Typical example of the classic AI robotic control hierarchy



Example of behavior-based robotic control



Key components for Sensor

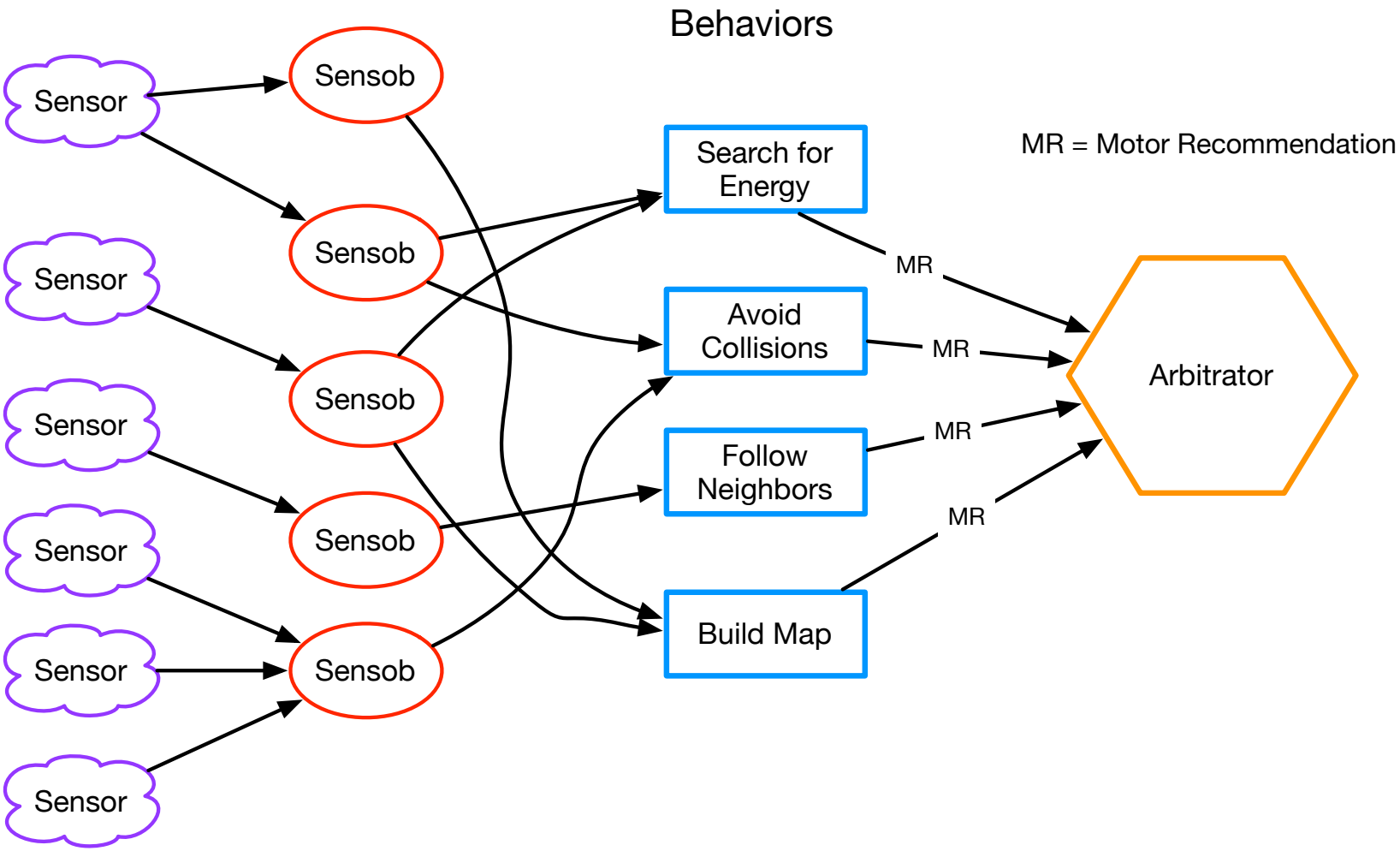


Sensor methods:

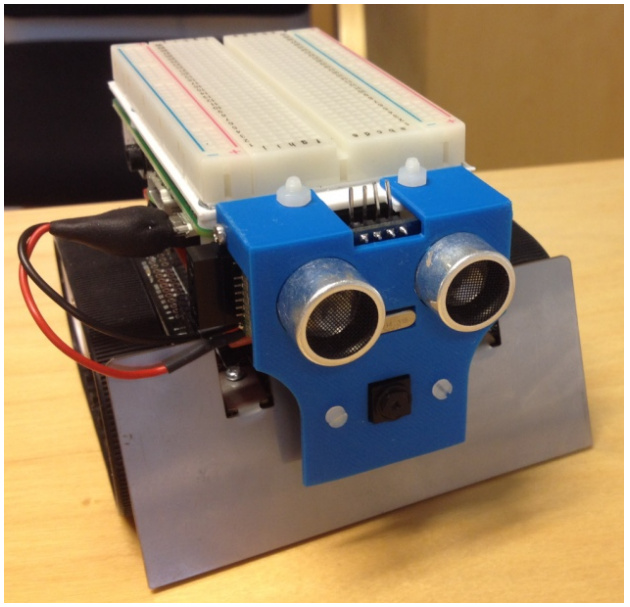
- update
- get_value
- reset

See helpcode

Class Sensob

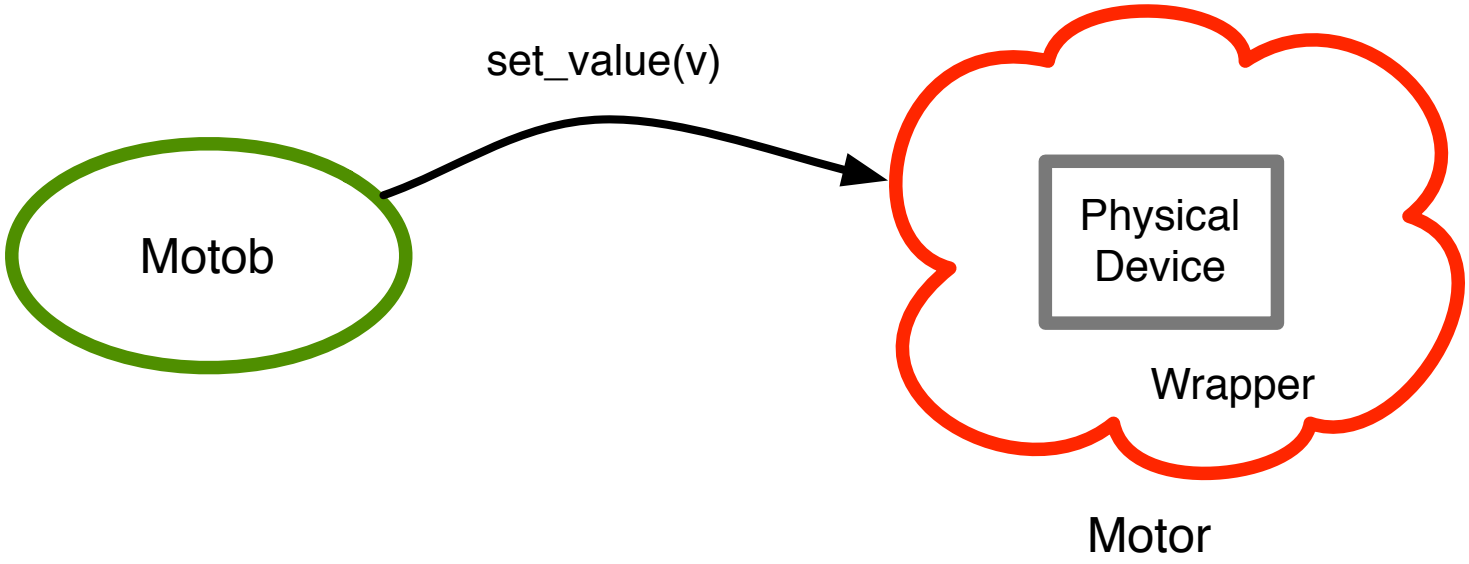


Class Sensob



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

Key components for Motor



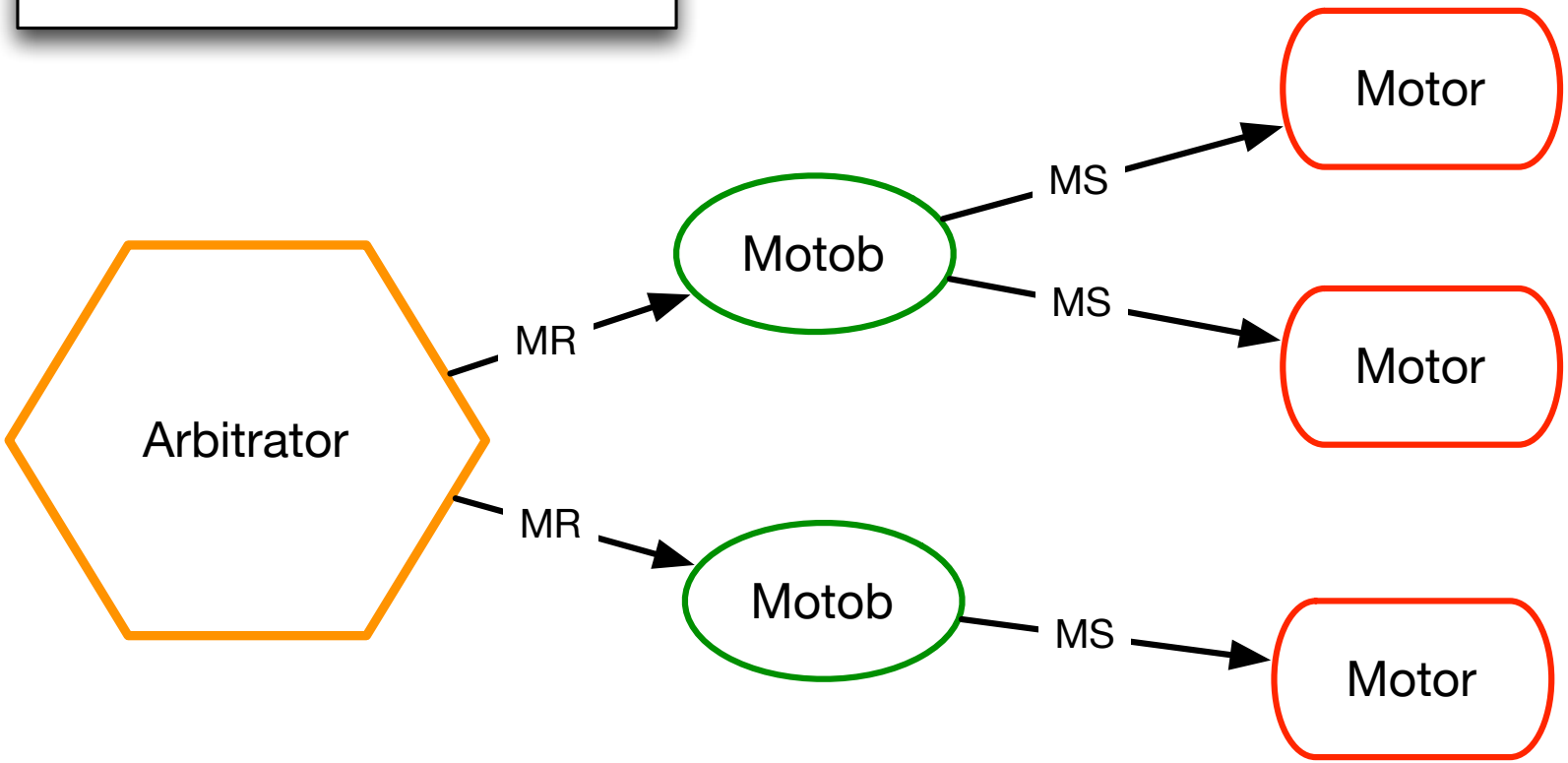
Class Motob

- 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).

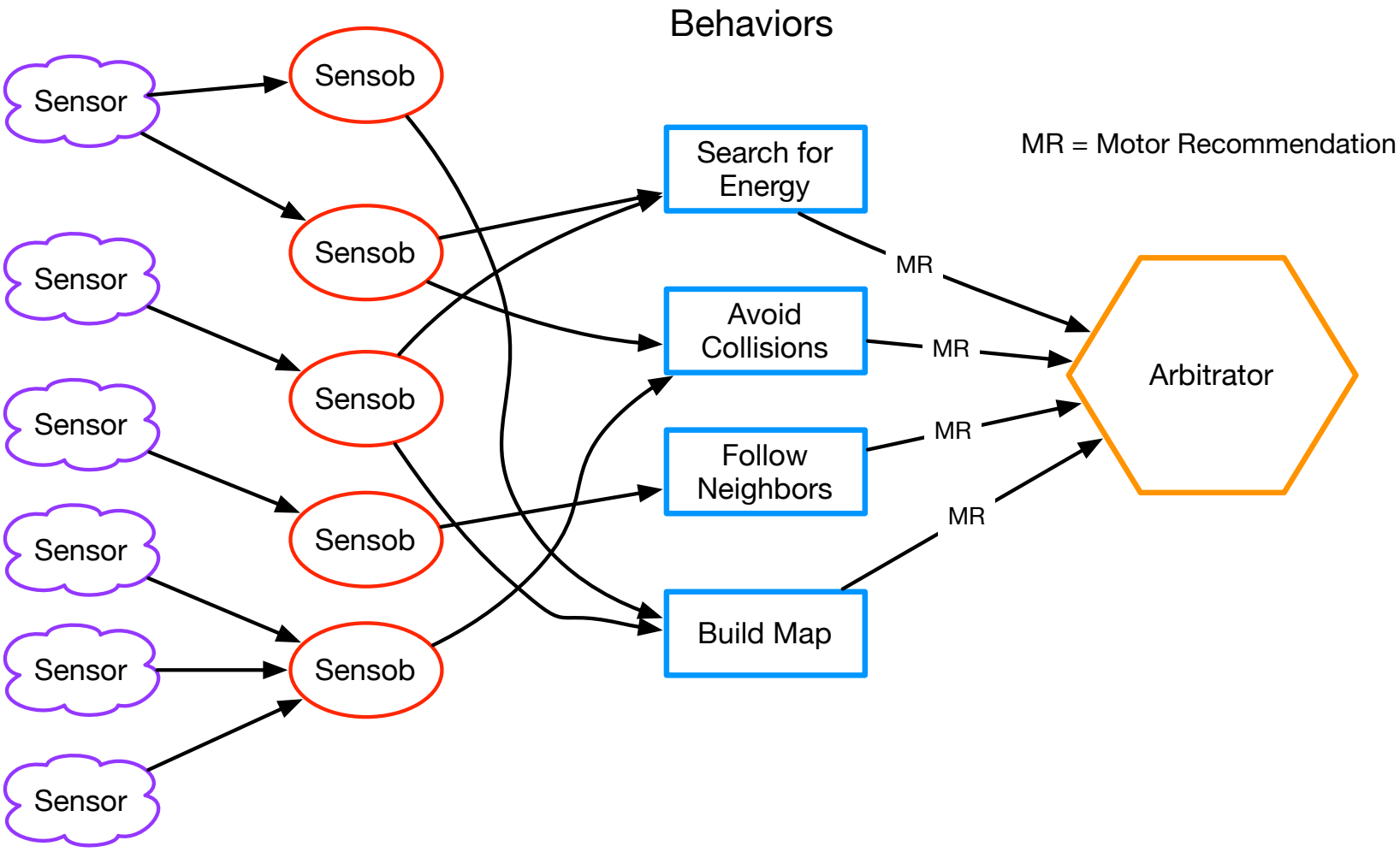
High-level recommandations

Basic relationships Arbitrator and Motob

MR = Motor Recommendation
MS = Motor Setting



Basic relationships



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 (if active: analyse sensor information and MAKE motor recommandations, else: check if can become active)
- **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 urgency and appropriateness this behavior. Calculated for each step.
- **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

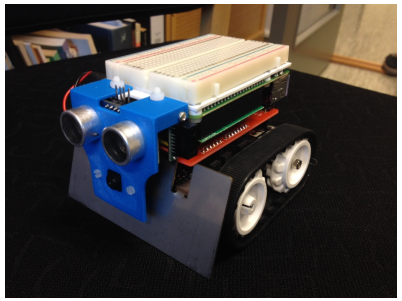
Deterministic or stochastic (weight based)

To receive a passing mark for this project, your group must:

- Build a bbcon (in object-oriented Python) to run your Zumo robot, which will perform some interesting, multi-step task.

- Include the camera and three other types of sensors in the bbcon (e.g. infrared belly sensors, proximity sensors, and an ultraviolet light sensor) , and use each of these sensors in at least one aspect of the multi-step task. The Zumo Button does NOT count as one of the 2 additional sensors.

- Show a working demonstration of your system in which EACH of your bbcon behaviors can be clearly seen to have some effect upon the robot's activity.



GPIO – bruk av ekstra pinner

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

irproximity.py

```
import RPi.GPIO as GPIO


class IRProximitySensor:
    def __init__(self):
        self.value = None
        self.read_pin_1 = 8
        self.read_pin_2 = 10
        self.setup()

    def setup(self):
        GPIO.setmode(GPIO.BOARD)

    def get_value(self):
        return self.value

    def update(self):
        self.value = self.sensor_get_value()
        return self.value

    def reset(self):
        self.value = None

    def sensor_get_value(self):
        GPIO.setup(self.read_pin_1, GPIO.IN)
        GPIO.setup(self.read_pin_2, GPIO.IN)
        read_val_1 = GPIO.input(self.read_pin_1)
        read_val_2 = GPIO.input(self.read_pin_2)
        # Invert the values, so that True means
              something is close
        return [not read_val_1, not read_val_2]
```

Raspbian Stretch

Getting started with Zumo

1. Connect your robot to a power source, insert batteries and turn it on. Connect your robot to the internet via an ethernet cable.

2. In order to use SSH and SFTP (described below) you need to get your robot's IP address. Using the ethernet cables at P15 you can simply use a DNS name instead of an actual IP address:

   rpi-xx.idi.ntnu.no

   where xx is the number printed on the ethernet port on your robot.

.

3. (Optional) If you want to connect to the robot at home, you need to connect your robot to a screen and keyboard. Then login with the username and password (described below) and type

   hostname –I

   The first number in the output will be your robot's IP address.

Getting started with Zumo (2)

- Use ssh to access the robot

  ssh robot@<your IP address>

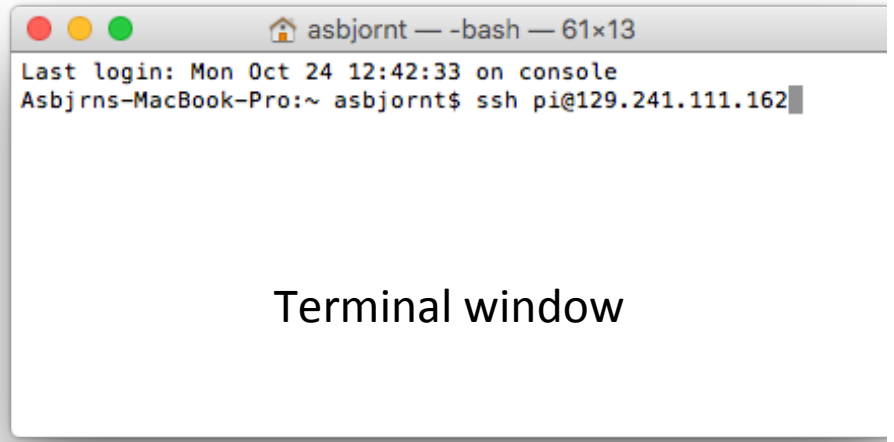
password (D4nc1ngmachin32)

- To change password:

  passwd

Must use: **sudo python3**

sudo pip3



Terminal window

Transfer programs from laptop to Zumo

- Navigate to laptop directory *mylaptop/robot*

  sftp robot@<your IP address>

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

  put mycontroller.py

  get image.png (robot image to laptop)

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

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
https://filezilla-project.org/download.php?type=client

Local editing on the robot

- Use Nano in robot´s terminal window