Embedded Motion Control

——

Suggestions for the design of robotic tasks and of their software architectures

Herman Bruyninckx
Eindhoven University of Technology
KU Leuven
http://people.mech.kuleuven.be/~bruyninc/

May 4, 2018

Introduction: affordances
Objects come with a “plan” for how to use them

Affordance is a term from psychology

James J. Gibson, 1966.

`https://en.wikipedia.org/wiki/Affordance`

that reflects the fact that humans don’t just **see** objects in the world, but also, **inherently connected** to that **perception**, they know how **to manipulate** them.

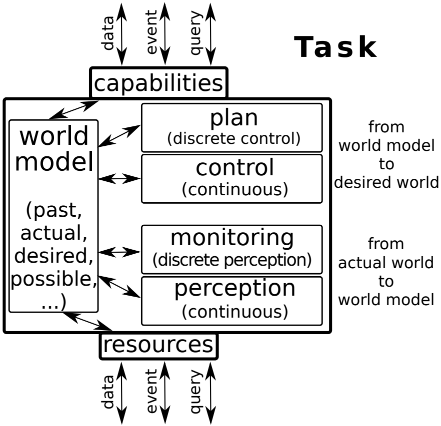
$\rightarrow$ **plan** comes for free with the object!

pick up, push, turn, shove, fill, order, clean, throw, break,...

Discuss **plan** for: *push*; *pick up*

“Task”: formalisation of the “Affordance” concept

And a guideline to design robot applications



A robot application designer must **integrate**:

- *capabilities*: what does the application offer?
- *resources*: what does it rely on?

- *world model*: **state** of “**everything**” that “**everyone**” must **know about**

- *plan*: discrete set of “behavioural” states
- *control*: continuous-time feedback/feedforward
- *monitoring*: system dynamics trigger events
- *perception*: continuous-time sensor processing

The “world model” plays a key role in your design!

Expected capabilities — Available resources

Capability: **escape from the room**.

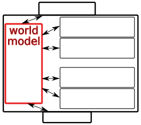
Plan (at *geometric* level):

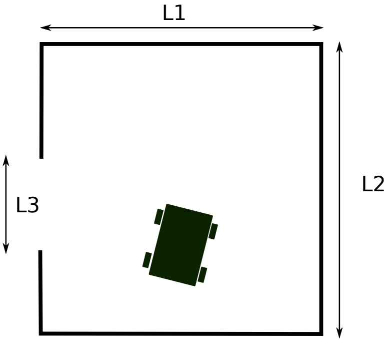
1. initialize sensors and motors
2. move forward till wall is detected
3. move while following wall on the right
4. turn right at first large enough hole
5. stop

Resources:

- laser range finder: series of rays indicating free space, within minimal & maximum measurement range.
- encoders: instantaneous actual velocity of platform.
- velocity control: instantaneous desired velocity of platform.
- effort value: percentage of “full” available power used for robot motion.
- keyboard button: event from keyboard

Initial world model: parameterized room with a hole

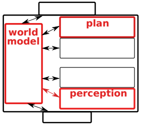


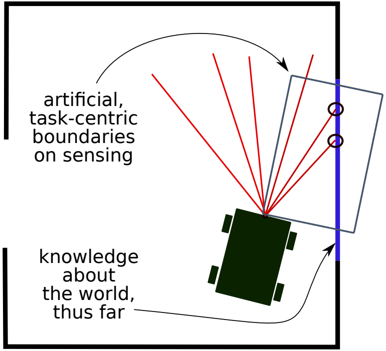


At **initialization**, this is **assumed**:

- the robot is *inside* a room
- the room has a *rectangular* shape as in the figure
- the room has *one door*, with a *width enough* to let the robot pass through
- the position and orientation of the robot in the room are *not known*
- the size of the room is *not known*

Perception ("sensor processing")



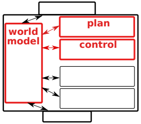


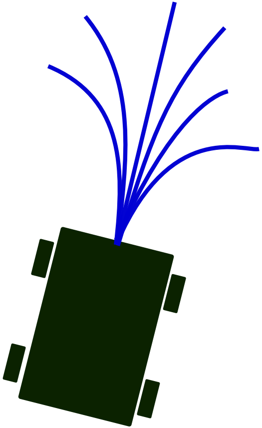
The sensor provides way more data than strictly necessary to do the job:

- select a *region of interest* (grey box) that fits to the *plan* (= only interested in right-hand side)
- fit a *line* through a *large enough* cluster of measurements
- do this over a *time window* of measurements

So, *perception* ⇔ least-squares fitting of a line of limited length through a clever, plan-directed selection of current and previous "hits"

Plan/control: motion trajectories

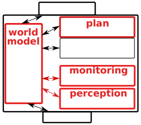


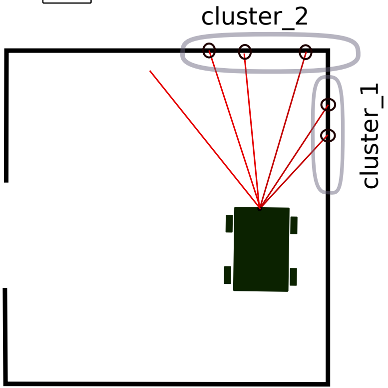


One easy **possibility** for “control”:

- apply a set of constant speeds to each wheel $\rightarrow$ set of known trajectories of the robot in the near future, to choose from
- sparsity/density of trajectories can be chosen, in a plan-directed way
- time/space horizon of trajectories can be chosen, in a plan-directed way
- **control** can be as simple as **selecting** the “best” trajectory and apply the corresponding wheel velocities during a **long** period

Monitoring to decide to add next wall to control scope

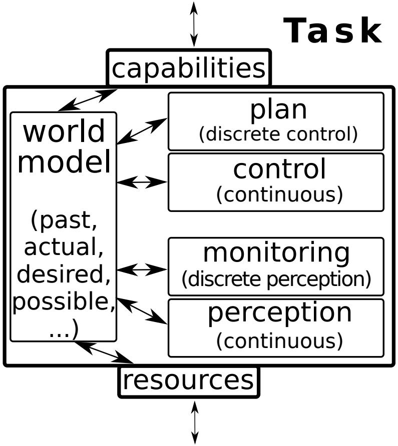




Monitoring has four **hypotheses** to follow:

1. *local* horizon to fit *wall*, on the *right*, as expected by the *task* context, to configure the *control*.
2. further horizon in *forward* direction:
   - 2.1 to monitor whether there is "something", to react to in *plan*;
   - 2.2 to find another *line cluster*, orthogonal to first one, to update the *world model* with a new *corner*.
3. the leftmost rays can be discarded
   → reduces the computational load.
4. *all* measurements *could* be *neglected* until needed again, based on *planned* speed of motion.

Task model — Revisited



The *task* of the robot relies on:

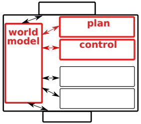
- *plan*, *control*, *perception* and *monitoring* models.
- all share information via *world model*.

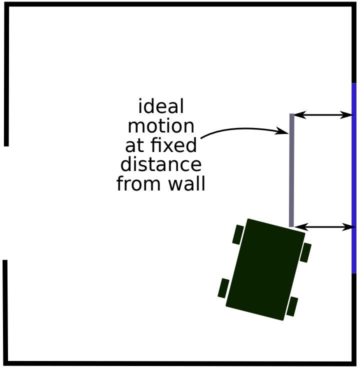
The *resources* provide **constraints** on **how fast, good, accurate,...** the capabilities **can** be realised.

The *capabilities* provide **tolerances** on **how well** control & perception **must** be.

Use this task template to structure your design discussions!

Motion specification for control
First simple example

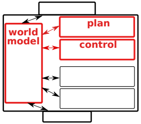


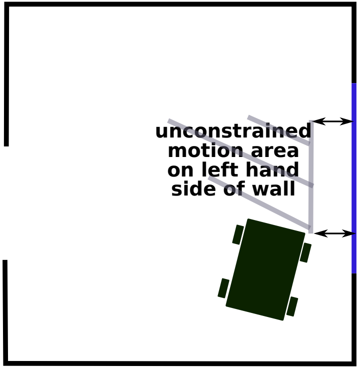


- *plan* places **reference trajectory** (grey line) in world model
- at fixed offset with respect to the best fitting wall line
- and with **goodness of fit** function for the actual robot motion

→ the controller need not change when representation and/or location of reference trajectory change

Motion specification for control
Second simple example

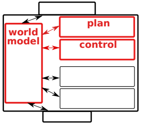


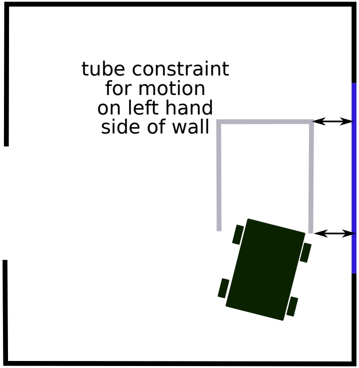


Another easy possibility:

- the robot is allowed to move "anywhere" left of the wall.

Motion specification for control
Third simple example



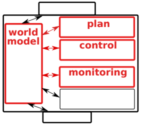


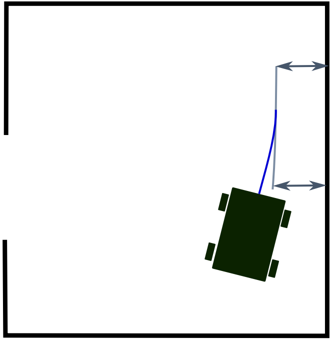
Another easy possibility:

- the robot is allowed to move "anywhere" inside a "tube" at some distance from the wall.

Control

Simple solution: best fitting open loop trajectory



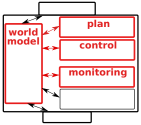


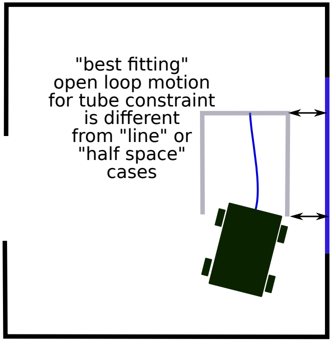
Control is simple:

- generate the "spray" of feedforward trajectories
- select a "good enough" fit
- apply corresponding open loop motor values
- until **monitoring** tells us that deviation becomes "too large"

Control

Other simple solution: best fitting open loop trajectory





The *tube* approach has

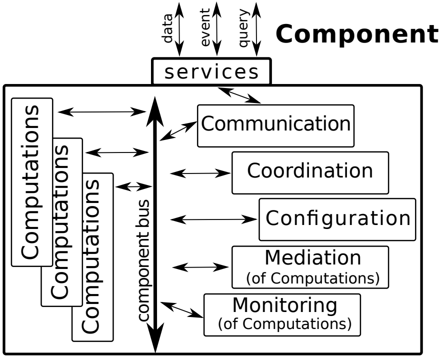
- other **optimal** trajectories
- other **tolerances**
- but same **monitoring**

Slides before: modelling
Now we need to go to software

- *models* must be turned into *data structures* in a programming language.
- *functions* on these data structures must be written, everywhere where the previous slides used *verbs* like "fit", "place", "select",...
- the *order* of the computations ("*schedule*") must be determined
- each *state* in the *plan* corresponds to one set of all of the above
- the *timing* of the computations ("*sampling*") must be determined
- the *execution* of the computations ("*dispatching*") must be done
- the *communication* with sensors, motors, keyboard,... must be realised, via the operating system and/or middleware libraries

The **resource usage** *of all of the above must be* **mediated***!*

"Component": formalisation of the software
And a guideline to implement robot applications



Component model **integrates** the following:

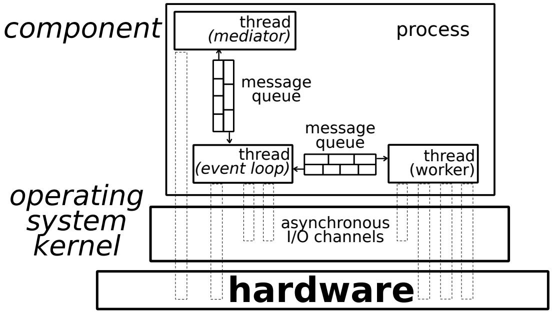
- *Computations*: all data + functions to execute
- *Communication*: read/write I/O data
- *Coordination*: decide to switch plan **state**
- *Configuration*: set right parameters
- *Mediation*: make trade-offs for scarce resources
- *Monitoring*: of CPU, BUS, RAM, IO *resources*

Local "component bus" = read/write access to all *shared* data.

Software pattern of the "Process"

Structures interactions between asynchronous programs

Your application **typically** has three types of **threads**:



- one **main**: runs an **event loop** every "sample time". (See next slide.)
- one or more **workers**: each run an algorithm that can take longer than one sample time, or that can *block* indefinitely.
- one **mediator**: checks resource usage, and *decides* about reconfiguration of *main* thread, when needed.

Software pattern of the "Event Loop"
Recipe for each time one thread is triggered

```
when triggered  // by operating system
 do {
  communicate() // get asynchronous data from workers, OS,...
  coordinate()  // decide what functions to run this time around
  configure()   // set all parameters for the selected functions
  compute()     // execute control, perception, monitoring, plan
                // functions synchronously, one after the other
  mediate()
  configure()
  communicate() // set data to be sent asynchronously
  sleep()       // the loop deactivates itself, until next deadline
 }
```