Design of a multimodal engine

Objectives

The aim of this project is to **specify, design and implement a multimodal engine** to interact with a drawing palette that has no buttons. To create and move shapes on the palette you will use the following methods:

1. **Speech recognition** using the speech recognition engine (e.g. with the use of the ivy sra5 agent)
2. **Gesture recognition** using the 2D gesture recognition palette (your ivy $1Recognizer or ICAR agent - see below -)
3. **Pointing** device (mouse) on the drawing palette

The objective is to **develop a merge engine for the different modalities** that will make it possible to approach the famous "put that there" (see link below), one of the first multimodal interaction techniques proposed by MIT some 40 years ago.

Démonstration: http://www.youtube.com/watch?v=RyBEUyEtxQo

Architecture

The tools should communicate according to the following or equivalent architecture (see Figure 1):

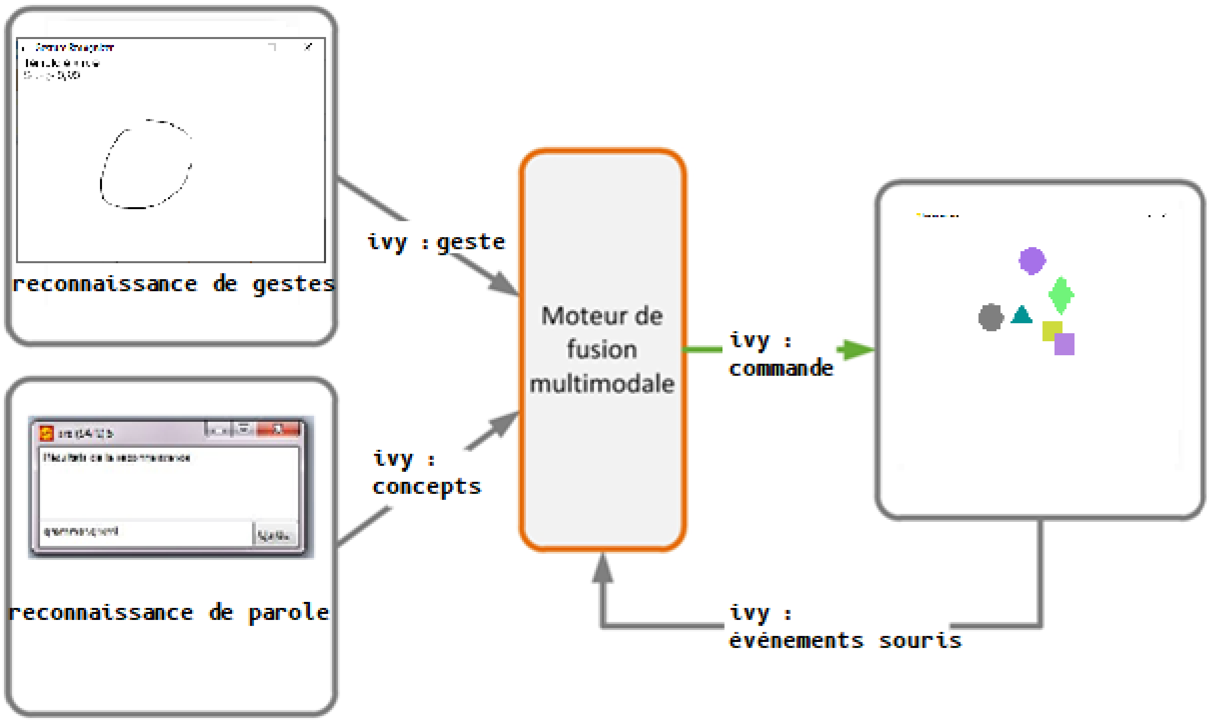


Figure 1 – possible software architecture

Specification of requirements:

From these three types of interaction, you are asked to create a multimodal merge engine that allows you to perform the following actions on the drawing palette.

Create a shape

The merge engine should allow this action to be carried out in different ways:

Create	*rectangle* | *circle* | *triangle*	here	
Create	*rectangle* | *circle* | *triangle*	*red* | *green* | *blue*	
Create	*rectangle* | *circle* | *triangle*	of that colour	
Create	*rectangle* | *circle* | *triangle*	here	of that colour
Create	*rectangle* | *circle* | *triangle*	*red* | *green* | *blue*	here
Etc.			

Additional constraints:

1. No proposed interaction should be **monomodal**!
2. The action of creating an object should be performed using gesture recognition (first the word "create", followed by the gesture to choose the shape).
3. Colour and position designation should be optional, and the order of the two should be flexible
4. If a colour is specified, this should be done via speech
5. In the case of a designation ("of this colour", "here"), this is done by voice and must be completed by pointing/clicking on the drawing palette

Move a shape

This action allows the user to move a previously created object. The user must be able to specify the move action either by gesture or speech. The designation will be done as for the creation of an object.

Move this	*rectangle* | *cercle* | *triangle*	here	
Move this	*rectangle* | *cercle* | *triangle*	*red* | *green* | *blue*	here
etc.			

Nota: In the second case, the addition of the colour makes it possible to disambiguate a case where there would be 2 rectangles at the location of the designation.

Other actions

It would be possible to define other actions (delete, change the colour). This is not required in this study, but you can go further if you have time.

Colours and shapes

You can limit yourself to example shapes (rectangle, ellipse, triangle, diamond...). It is also sufficient to choose three colours as examples.

Useful tools

Recommended programming language

Code in Processing.org or Java language via an IntelliJ IDEA project, Eclipse, ... (or Python if you really prefer it 😊).

Communication between agents: Ivy

The available tools (Palette / $1Recognizer or icar / sra5) are Ivy agents.

Ivy agents communicate via text messages over the network. They subscribe to the types of messages that interest them. These types are defined by regular expression. The reception of a message triggers a callback/listener call.

Display messages on Ivy

1. An ivy viewer

The viewer allows you to manage the agents and messages that transit on the Ivy bus in a graphical way. This is particularly useful when developing an application using the Ivy bus.

It can be downloaded at this address:
https://github.com/truillet/upssitech/blob/master/SRI/3A/IHM/TP/Outils/visionneur_1_2.zip

2. Probe on CLI

Alternatively, you can use the CLI tool (see ivy documentation seen in distributed interaction lecture)

Speech recognition:

1. Simulation

During the development, speech recognition can be simulated. In order to simulate speech recognition, a Java swing panel with several *JButtons* can be used. Each button will represent a voice command to be recognised ("this object", "here", "put this on", ...). A click on the button will send the corresponding Ivy message. In order to easily interchange this panel with the sra5 application, the buttons will have to send Ivy messages of the following form:

```
sra5 Parsed='resReco' Confidence='proba' NP='id' Num_A='id'
```

Example for the recognition of the expression "this object" we will send:

```
sra5 Parsed='thisobjet' Confidence=0.8 NP=1 Num_A=0
```

[**Nota** : `Confidence` specifies the confidence of the recognised concept(s); NP and Num_A specify the number of items recognised since sra5 launch and the recognition alternative. You can set both options to 0 in all cases].

[**Nota 2** : t is possible to code this interface in python, Processing.org or via ... the language of your choice]

2. Using speech recognition

Later on, we will use speech recognition with the *sra5* module (see Tutorial 1). You will therefore need to modify the recognition grammar as required for the project.

Gesture recognition: $1Recognizer or Rubine's algorithm

OneDollarIvy (which implements the $1Recgnizer algorithm) consists of a Processing.org application (see Figure 2).

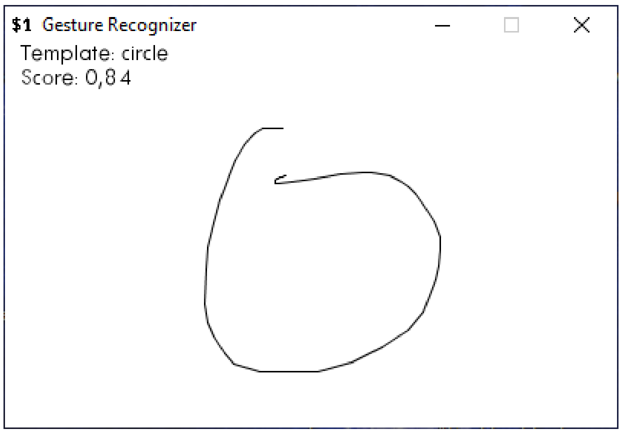


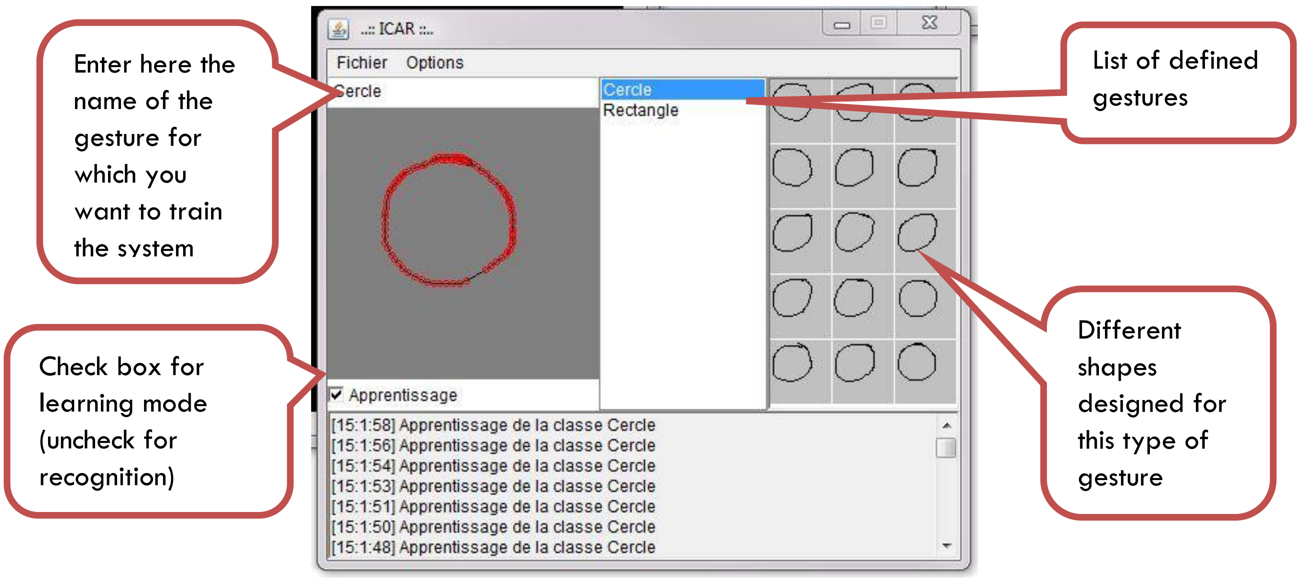
Figure 2 – gesture recognition by OneDollarIvy

OneDollarIvy allows you to create (**L**earn), import (**I**mport), export (**E**xport), list (display **T**emplates) and recognise (gesture **R**ecognition) learned gestures.

ICAR (which implements Rubine's algorithm) can be downloaded here:
https://github.com/truillet/upssitech/blob/master/SRI/3A/IHM/TP/Outils/icar.1.2.zip
The documentation is available here:
https://github.com/truillet/upssitech/blob/master/SRI/3A/IHM/TP/Outils/icar.pdf

ICAR is composed of 2 sub-applications: one for gesture learning and the other for gesture recognition.

1. Learning gesture:


Enter here the name of the gesture for which you want to train the system
Check box for learning mode (uncheck for recognition)
List of defined gestures
Different shapes designed for this type of gesture
..: ICAR :..
Fichier
Options
Cercle
Cercle
Rectangle
Apprentissage

```
[15:1:58] Apprentissage de la classe Cercle
[15:1:56] Apprentissage de la classe Cercle
[15:1:54] Apprentissage de la classe Cercle
[15:1:53] Apprentissage de la classe Cercle
[15:1:51] Apprentissage de la classe Cercle
[15:1:50] Apprentissage de la classe Cercle
[15:1:48] Apprentissage de la classe Cercle
```

Figure 3 – Gesture learning interface by ICAR

The dictionary editor "*IcarAdmin*" allows you to save a dictionary of gestures (see Figure 3).

[**Nota:** You should define a single dictionary that will contain several different gestures].

Once the dictionary has been created and saved, simply modify the .bat "*IcarIvy*" to use your new dictionary as a parameter.

2. Gesture recognition:

The recognition tool "IcarIvy" recognises gestures according to the dictionary provided as input. When "*IcarIvy*" recognises a gesture, it sends an Ivy message with the following form:

```
ICAR 'nomGesteReconnu'
```

Drawing palette

It responds to Ivy messages, allowing you to create, move, colour, delete, etc. items. In order to communicate with it, your application must also emit ivy messages.

We provide a version of Palette developed in Processing.org allowing to manage different shapes (triangles, circles, rectangles and diamonds):
https://github.com/truillet/upssitech/blob/master/SRI/3A/IHM/TP/Code/Palette.zip

You can of course recode a palette according to your wishes.

Procedure

You have **three** two-hour sessions to carry out this design. Here is a possible organisation of the sessions:

Session 1:

1. Specify gesture and speech grammars
2. Learn the gestures ($1Recognizer or ICAR)
3. Describe your **multimodal commands** in the form of timelines (see Figure 4 below). The idea of these timelines is to explore the different possibilities of ordering commands on the ivy bus in order to better predict the flexibility in the system.

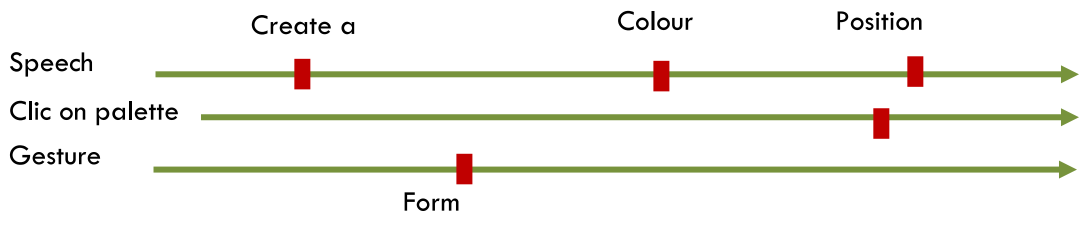


Figure 4 – chronograms of multimodal commands

Session 2:

1. Describe the dialogue controller (state machines, see F. Cabric course).
   This controller is crucial to the operation of your fusion engine and requires some thought!

   [**Reminder:** you must start by describing the possible actions and events, then the automaton and then the state/events matrix]
2. You will need to use a data structure to merge the information. It will be filled in as the messages arrive. This will be implemented in a separate class. What data and methods should it contain? When should the reset take place? Specify and code this structure.

Session 3:

1. Code the multimodal engine. You will also ensure that you apply the techniques of systematic design and implementation of a dialogue controller (deduction of code from a state machine).
2. Test and validate your work.
3. Write the report.

Work to be done

You will send by email a link to a git repository or to a zip archive containing

1. Your design on how to merge of modalities should be argued in a report of a few pages. In it you will describe
   a. a. temporal aspects of multimodal fusion (chronograms)
   b. the software design of your system (class diagram)
   c. the whole state machine
   d. Illustrate the use of your application with one or more examples (screenshots for instance).
2. Source code
3. An executable or entry point for your project (e.g. a .bat or .sh file that runs all the tools needed to run the application). Alternatively, describe the mode of use in the report].
4. Optionally, you can provide a video with an example of the application running.

Deadline: sunday 28 november 2021, 23h55 UTC+1

The work will be sent to Philippe.Truillet@univ-tlse3.fr
(If you have large files to send, you can use a cloud or transfer service like https://www.wetransfer.com)

Each day of delay will result in a 0.25 point penalty.

Bibliography

- Richard Bolt, "Put that there": Voice and Gesture at the Graphics Interface, Proceedings of SIGRAPH'80, **https://dl.acm.org/citation.cfm?id=807503**
- Denis Lalanne, Laurence Nigay, Philippe Palanque, Peter Robinson, Jean Vanderdonckt, Jean-François Ladry, "Fusion Engines for Multimodal Input: A Survey", Proceedings of ICMI-MLMI'09, **http://iihm.imag.fr/publs/2009/FinalSurvey.pdf**
- Sharon Oviatt, "Ten Myths of Mutimodal Interaction", Communication of the ACM, november 1999, **https://pdfs.semanticscholar.org/440a/4e4e842968c58a45ac1e920abfda1c4803bc.pdf**
- Dean Rubine, "Specifying gestures by example", Proceedings of SIGRAPH'81, **https://dl.acm.org/citation.cfm?id=122753**
- Marcos Serrano, « Interaction multimodale en entrée : Conception et Prototypage », Thèse en Informatique de l'Université de Grenoble, 2010, **https://tel.archives-ouvertes.fr/tel-01017242/document**