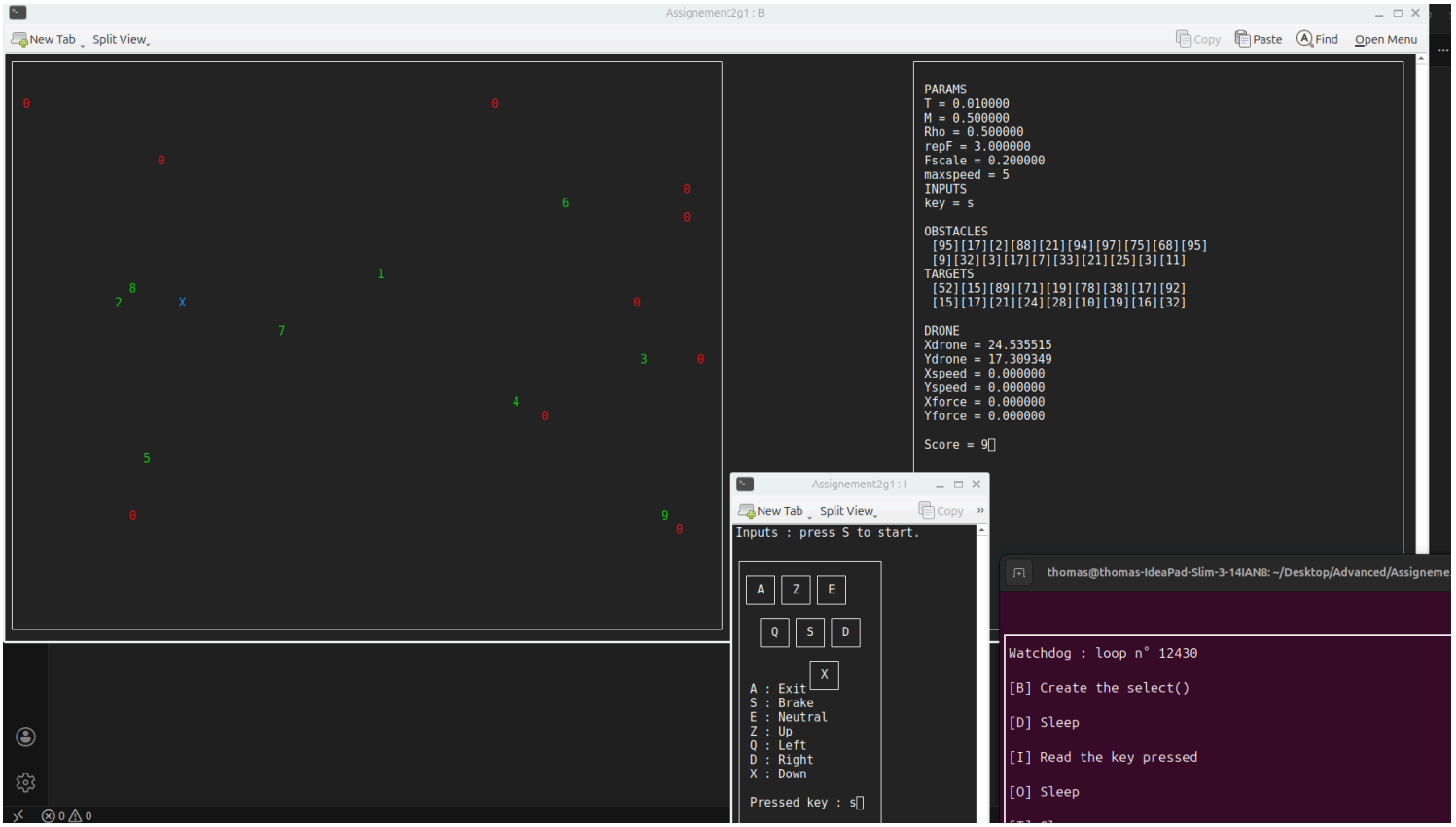
DRONE SIMULATOR PROJECT

Groupe ERASMUS n°1

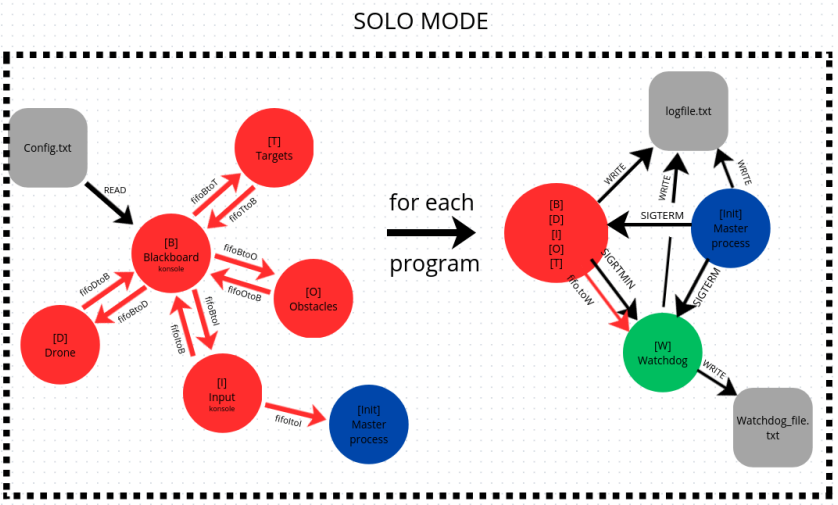
Assignment 3

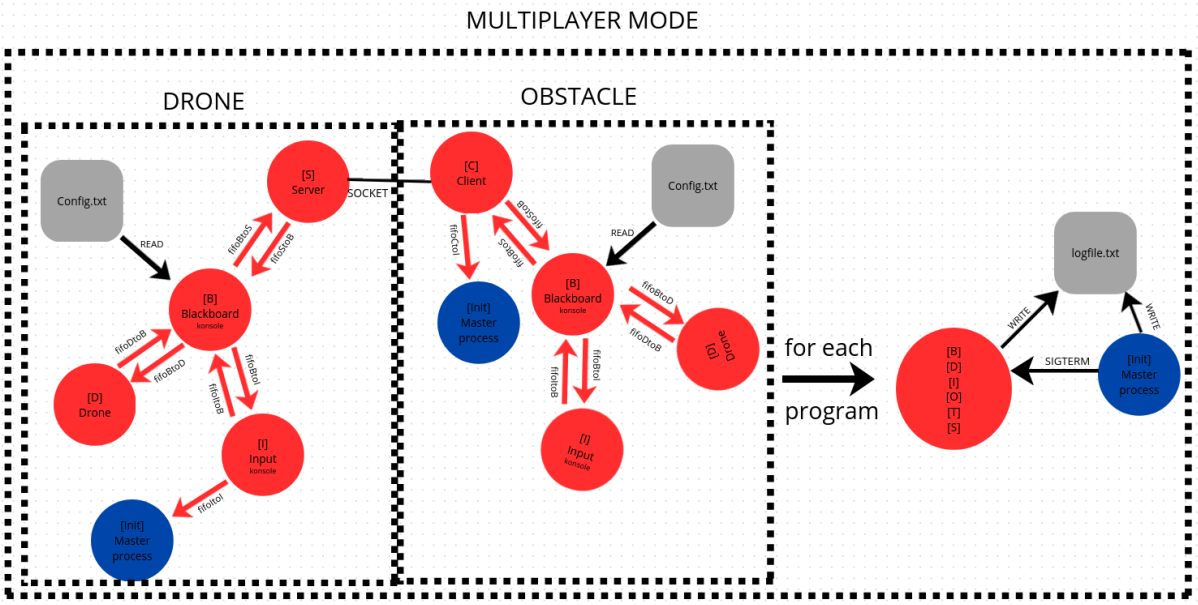


Project members :

GRESSARD Josselin
PRADINAT Thomas
ROSSIGNOL Victor

1. sketch of the architecture





2. short definitions of all active components :

In bold : New functions of the 3rd assignment

Master process [Master] :

- **read the simulation mode in network_setup.txt**
- **reset the logfile**
- **reset the pid-file**
- **use the functions fork()+exec() to launch one of the 3 possible [Init] program, depending on the mode**
- **wait for all the Init program to terminate**
- **terminate cleanly**
- **write in the logfile (errors, entry into a relevant area of the code)**

—-----SOLO MODE —----------

Init process [Init] :

- create all the fifos of this project
- use the functions fork()+exec() to launch the 5 other programs
- write all the PID of every program in a pid-file
- wait a message from [I] when the user want to quit
- send a signal to all the other programs to terminate
- wait for all the programs to terminate
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)

Blackboard [B] : Store all the values of the project and communicate it to the different programs.

- endless loop, except if received the terminate signal
- read the content of the file at each iteration and update the values of variables
- use the select() function with all the input fifos
- use a special two-pipe protocol to communicate with the programs
- can send or receive data
- calculate the score of the player
- use ncurses to print the scene in color on the screen and additional informations
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)
- read the pid file to find the pid of the Watchdog
- periodically send signal and message to watchdog

Drone [D] : Update the values of the drone representation with interactive dynamic equations.

- endless loop, except if received the terminate signal
- interaction period can be changed through config file
- get value from the blackboard
- use dynamic equations to calculate new values
- send the new values to the blackboard
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)
- read the pid file to find the pid of the Watchdog
- periodically send signal and message to watchdog

Input [I] : Read the key pressed.

- endless loop, except if received the terminate signal
- use ncurses to read the key pressed
- the key must be one of the keys selected for this project (a,z,q,s,..)
- send the key value to the blackboard
- if the key is the one to terminate, send a message to the master process
- terminate cleanly

- write in the logfile (errors, entry into a relevant area of the code)
- read the pid file to find the pid of the Watchdog
- periodically send signal and message to watchdog

Obstacles [O] : Generate obstacles.

- endless loop, except if received the terminate signal
- long period
- generate new coordinates for the obstacles with random values from /dev/urandom
- send the coordinates to the blackboard
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)
- read the pid file to find the pid of the Watchdog
- periodically send signal and message to watchdog

Targets [T] : Generate targets.

- endless loop, except if received the terminate signal
- long period
- generate new coordinates for the targets with random values from /dev/urandom
- send the coordinates to the blackboard
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)
- read the pid file to find the pid of the Watchdog
- periodically send signal and message to watchdog

Watchdog [W] : Watch all the programs.

- endless loop, except if received the terminate signal
- receive signals of the other programs
- read the messages from the programs if the message is received
- print the messages in the terminal
- write the messages in a watchdog-file
- terminate cleanly
- write in the logfile (errors, entry into relevant area of the code)

—-----MULTIPLAYER MODE : DRONE—----------

Init process [Init] :

- create all the fifos of this project
- use the functions fork()+exec() to launch the **4** other programs
- write all the PID of every program in a pid-file
- wait a message from [I] when the user want to quit
- send a signal to all the other programs to terminate
- wait for all the programs to terminate
- terminate cleanly

- write in the logfile (errors, entry into a relevant area of the code)

Blackboard [B] : Store all the values of the project and communicate it to the different programs.

- endless loop, except if received the terminate signal
- read the content of the file at each iteration and update the values of variables
- use the select() function with all the input fifos
- use a special two-pipe protocol to communicate with the programs
- can send or receive data
- use ncurses to print the scene in color on the screen and additional informations
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)

Drone [D] : Update the values of the drone representation with interactive dynamic equations.

- endless loop, except if received the terminate signal
- interaction period can be changed through config file
- get value from the blackboard
- use dynamic equations to calculate new values
- send the new values to the blackboard
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)

Input [I] : Read the key pressed.

- endless loop, except if received the terminate signal
- use ncurses to read the key pressed
- the key must be one of the keys selected for this project (a,z,q,s,..)
- send the key value to the blackboard
- if the key is the one to terminate, send a message to the master process
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)

Server [S] : Talk with a client to exchange data

- **read the communication port in the network_setup file**
- **create a socket, wait, then connect to a client**
- **use a specific protocol to communicate**
- **send the scene size to the client**
- **endless loop, except if received the terminate signal**
- **get value from the blackboard**
- **exchange the values with the client**
- **send the new values to the blackboard**
- **terminate cleanly**
- **tell the client to quit**
- **write in the logfile (errors, entry into a relevant area of the code)**

—-----MULTIPLAYER MODE : DRONE—----------

Init process [Init] :

- create all the fifos of this project
- use the functions fork()+exec() to launch the **4** other programs
- write all the PID of every program in a pid-file
- wait a message from [**C**] when the user want to quit
- send a signal to all the other programs to terminate
- wait for all the programs to terminate
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)

Blackboard [B] : Store all the values of the project and communicate it to the different programs.

- endless loop, except if received the terminate signal
- read the content of the file at each iteration and update the values of variables
- use the select() function with all the input fifos
- use a special two-pipe protocol to communicate with the programs
- can send or receive data
- use ncurses to print the scene in color on the screen and additional informations
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)

Drone [D] : Update the values of the drone representation with interactive dynamic equations.

- endless loop, except if received the terminate signal
- interaction period can be changed through config file
- get value from the blackboard
- use dynamic equations to calculate new values
- send the new values to the blackboard
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)

Input [I] : Read the key pressed.

- endless loop, except if received the terminate signal
- use ncurses to read the key pressed
- the key must be one of the keys selected for this project (z,q,s,..)
- send the key value to the blackboard
- terminate cleanly
- write in the logfile (errors, entry into a relevant area of the code)

Client [C] : Talk with the server to exchange data

- **read the communication port and server address in the network_setup file**
- **create a socket and connect to a the server**

- **use a specific protocol to communicate**
- **receive the scene size from server and transmit it to the blackboard**
- **endless loop, except if received the terminate terminate message from the server**
- **get value from the blackboard**
- **exchange the values with the server**
- **send the new values to the blackboard**
- **if received the terminate message, send a message to the master process**
- **terminate cleanly**
- **write in the logfile (errors, entry into a relevant area of the code)**

—----OTHERS—-----

Parameter file [Config.txt] : Store the parameters values for the simulation.

Log-file [logfile.txt] : Get all the log messages of every program of the project.

Watchdog file [Watchdog_file.txt] : Get all the watchdog messages (message received from the other programs and alert messages from the watchdog).

Pid file [PID_file.txt] : Store the Pid numbers of every program.

Network file [network_setup.txt] : Interface for the user to choose the mode, the server IP address and the port number

3. list of components, directories, files

One directory : “Assignement**3**” :

- a “bin” directory with all the binary files :
    - **a “Solo” directory : Init ; B ; D ; I ; O ; T ; W.**
    - **a “MultiDrone” directory : Init ; B ; D ; I ; S.**
    - **a “MultiObst” directory : Init ; B ; D ; I ; C.**
    - **a “Master” directory : Master.**
- a “config” directory with two text files : Config.txt **; network_setup.txt**.
- a “log” directory with three text files : logfile.txt ; Watchdog_file.txt ; PID_file.txt.
- a “src” directory with all the code files :
    - **a “Solo” directory : Init.c ; B.c ; D.c ; I.c ; O.c ; T.c ; W.c.**
    - **a “MultiDrone” directory : Init.c ; B.c ; D.c ; I.c ; S.c.**
    - **a “MultiObst” directory : Init.c ; B.c ; D.c ; I.c ; C.c.**
    - **a “Master” directory : Master.c.**
- a make file : makefile.
- the README file : README.pdf.

4. instructions for installing and running

The project requires the following components to compile and run correctly:

System Requirements

- A Unix/Linux operating system
- Access to the special device file /dev/urandom for random number generation
- POSIX-compliant system calls, including fork(), exec(), and wait()

Libraries

- ncurses (for terminal-based display and input) on Debian/Ubuntu systems, install with: “sudo apt install libncurses5-dev libncursesw5-dev”
- math library (math.h)
- Standard C library (stdio, stdlib, string, etc.)

Build Tools

- gcc
- make

5. operational instructions

Start program :

- make
- **choose the mode and the network address and port in the /config/network_setup.txt file.**
- ./bin/**Master/Master**

Problem with window size :

Sometimes, if the program is started for the first time, it is possible that the konsole window is too small, and the print of the scene is bad. In this case, change the size of the konsole to the maximum (but no full screen), then close the program in the terminal with Ctrl+C and restart it.

Also, sometimes, the obstacles are not generated right -> just re-make the project.

If you start the program as the client, be careful with the window size, sometimes the scene print doesn’t work correctly (the window is too small if the server scene is too big ; or the scene appears too small). In that case, just start again after resizing the window.

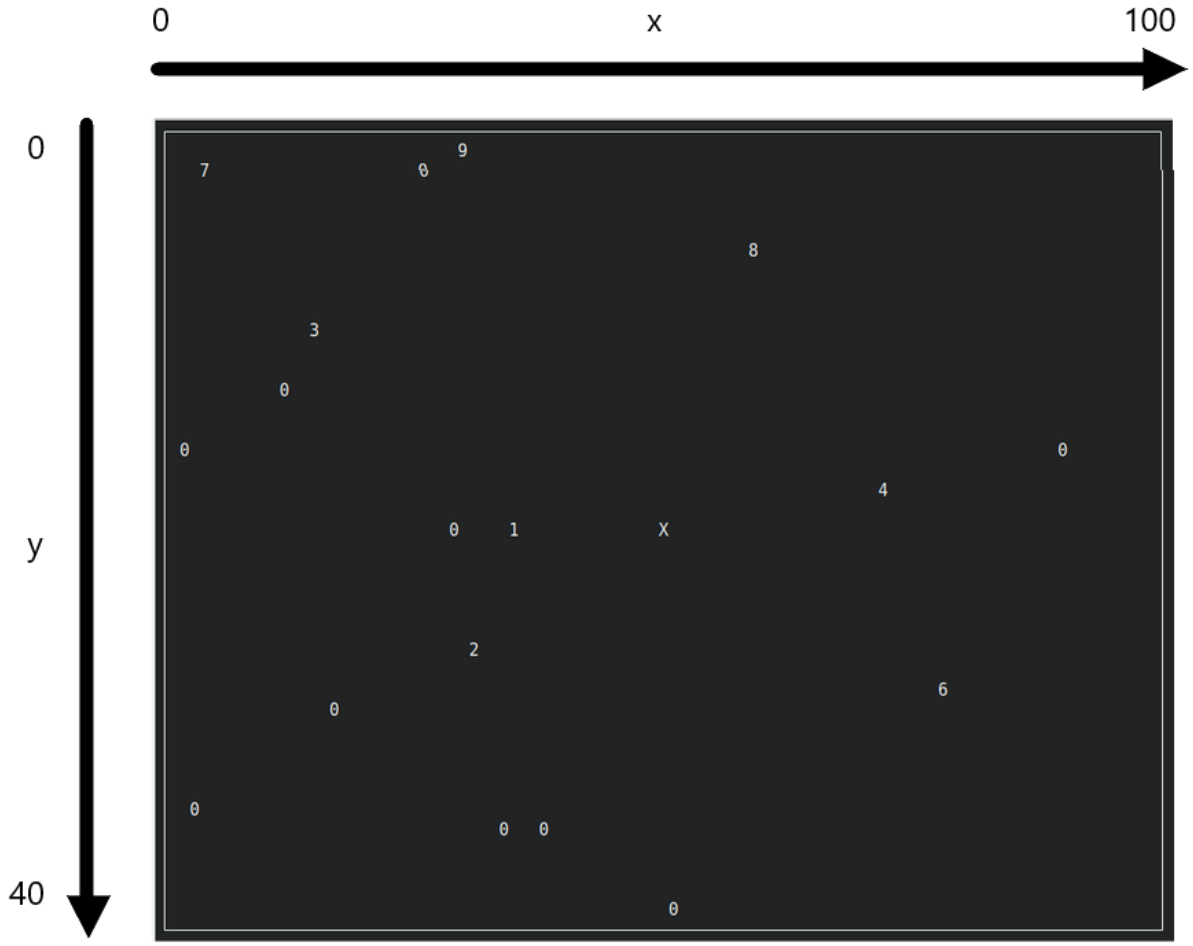
6. useful things

Protocol to communicate with the blackboard :

- The first message is always addressed to the blackboard. It indicates if the other program want to Read or write in the blackboard : ‘r’ or ‘w’
- Then the blackboard answer :
  - the data if the program want to read
  - “ok” if the program wants to write, indicating the blackboard is blocked and is waiting for the values.
- The program then read or write the values

Each program asks to read / asks to write different variables, the buffers are very different between the pipes.

Scene coordinates :

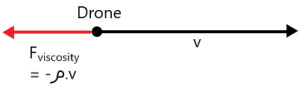


Dynamic calculation of the new coordinates :

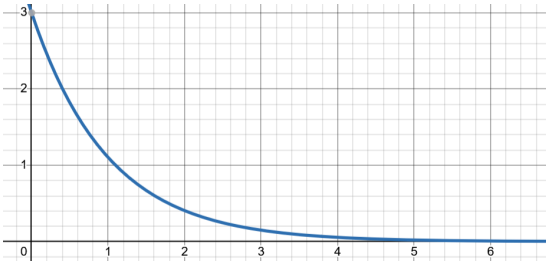
2nd Newton law : $\sum F = m.a \quad so \quad a = \frac{1}{m}\sum F$

$\sum F = F_{motor} + F_{viscosity} + F_{obstacles}$

F_{motor} is determined by the keys ; $F_{viscosity} = -\rho . v$



$F_{obstacle} = . e^{-\frac{distance}{repulsion\ radius}}$

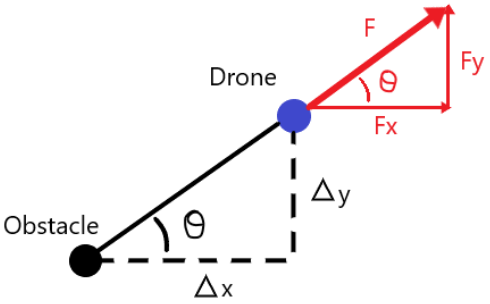


$distance = \sqrt{(x_{drone} - x_{obstacle})^2 + (y_{drone} - y_{obstacle})^2}$

$F_x = F_{obstacle} . cos\ \theta$

$F_y = F_{obstacle} . sin\ \theta$

$\theta = atan((x_{drone} - x_{obstacle})/(y_{drone} - y_{obstacle}))$



Communication protocol between server and client :

client:

```
rcv ok; snd ook
rcv size l, h; snd sok <size>
loop [
        rcv x;
        if (x ==  q) [ snd qok; exit ]
        switch x [
                x == drone; rcv x, y;  snd dok  <drone>
                x == obst; snd x y;   rcv pok <obstacle>
                ]
        ]
```

server:

```
snd ok; rcv ook
snd size l, h; rcv sok <size>
loop [
        if ( <quit> ) [ snd q; rcv qok; exit ]
        snd drone;  snd x, y;  rcv dok <drone>
        snd obst; rcv x, y; snd pok <obstacle>
        ]
```

- **the coordinates are sent/received according to a referential with the origin at the bottom left corner ; with x horizontally and y vertically.**
- **the coordinates transferred can be integers or floating points**
-

Help to understand the socket data transfert :

- **The debug-print are still available in the programs (server and client) un-comment them to see what is passing through the socket and if it is perceived correctly.**
- **We followed exactly the protocol given by the teacher, but if you want to check it, the lines with the protocol orders are highlighted so you can change it if you want.**
- **Both programs can receive floating coordinates, however, if the other program can just read integer coordinates, it is able to switch the type of data by un-commenting/commenting the lines in the programs.**

Test of the 3rd assignment with other groups :

1st try : Thanks to geographical proximity, we first try our program with the 2nd ERASMUS groups. Both sides had to change the program to follow the protocol, we also developed the possibility to change from integer to floating coordinates transmission.
2nd try : During the meeting with italian groups in class, we tried to connect with the other groups. We saw that a coma was missing in our protocol so we fixed it. Then, all the other problems were coming from the other groups that didn't follow the communication protocol. This final test was a complete success !!