Advanced Robot Programming Labs
C++ Programming

Lab 1: Open projects

1 Content of this lab

The goal of the first C++ labs are to discover the language by building small projects. Many projects are listed below and you are encouraged to try and build several of them. Most of them can be done without using evolved concepts of C++. You can also build your own project if you have a clear idea of the goal to be reached.

You are encouraged to use functions, classes or even templates if you think it will lead to a better code that is easier to understand and to write. You are also encouraged to separate your code in several files. If you are not sure about classes or functions, you can always change it later and compare the same program with different code designs.

You may look for mathematical solutions, problem formulation or problem variations on Internet, but of course the actual code should come from yourselves.

1.1 How to compile

We use CMake tools to compile our projects. The content of the CMakeLists.txt file should follow the structure of your code directory (or preferably sub-directory): The structure of your code directory (or sub-directory should be like:

```
MyProject
├── include
│   ├── a_class.h
│   ├── some_functions.h
│   └── a_basic_header.h
├── src
│   ├── main.cpp
│   ├── a_class.cpp
│   └── some_functions.cpp
└── CMakeLists.txt
```

```
cmake_minimum_required (VERSION 2.6)
project (MYPROJECT)

add_compile_options(-std=c++11)

include_directories(include)

add_executable(mygame
  src/main.cpp include/a_basic_header.h
  include/a_class.h a_src/class.cpp
  include/some_function.h src/some_function.cpp)
```

The CMake file is used to generate a Makefile. The steps are:

1. Create a build directory: `mkdir build`
2. Go inside: `cd build`
3. Call CMake: `cmake ..`
4. Call Make: `make`
5. Then launch your program: `./mygame`

1.2 How to use QtCreator

The IDE[1] that will be used for C++ (and all my other labs) is Qt Creator. It is an open source, multi-platform IDE based on the Qt framework (which is used in the KDE Linux desktop). Before loading a project in Qt Creator, you have at least to write a basic CMakeLists.txt file (even without executables) and to create the `build` folder. You can then run Qt Creator:

1. Load the CMakeLists.txt file through `File...open project`
2. Give the path to your `build` directory
3. Qt Creator will then execute CMake and display the source files found in CMakeLists.txt (if any)
4. To add a file (.cpp or .h), you can create an empty one with `File... new file`
5. Add this file in the CMakeLists.txt and right-click to run CMake to tell Qt Creator that the structure f the project has changed
6. Compilation is done by clicking the bottom-left hammer
7. Run your program with the green triangle. However, if you need user input, you will need to run the program from the console.

2 Single player games

In the single player games, the opponent is the computer which plays with a basic or advanced AI depending on your programming motivation (and game AI possibilities...).

2.1 Find the number

Here the computer start by choosing randomly a number between 1 and 100. The player has to find this number with trial and error, the only answer being whether the number is greater or lesser than the goal. This can also be done in reverse, where the computer has to guess.

- **C++ skills:** Input/output
- **AI tips:** Random numbers depend on seed

2.2 Rock-paper-scissors

The player is asked to choose between rock, paper or scissors. The computer also does a choice and the winner depends on the two choices: Rock beats Scissors, Paper beats Rock, Rock beats Scissors, Draw in the case of equal choices.

- **C++ skills:** Input/output, Random numbers
- **AI tips:** none, this game is completely random

[1] Integrated Development Environment

3 Turn-based text games

In these two-player games, the players play one at a time. The goal is usually to reach the winning situation before the other. They can also be programmed to play against an artificial intelligence. This time the AI can be much more complex than for the first games.

3.1 21 sticks game

In this game, a given umber of sticks (typically 21) are displayed. Each player has to pick up 1 to 3 sticks, then the other player can pick up his sticks. The goal is not to have to pick the last stick.

- **C++ skills:** Input/output
- **AI tips:** If you can start correctly you may never lose

3.2 Battleship

In this game each player has a number of boats that are placed on a 10×10 grid. The goal is to sink all the opponent's boats. Each player begins by placing their boats according to the following:

- One Aircraft carrier [A] of length 5
- One Cruiser [C] of length 4
- One Destroyer [D] of length 3
- One Submarine [S] of length 3
- One Minesweeper [M] of length 2

The player grids may be entered manually or with a text file (check grid consistency), or randomly generated. Each player then enters coordinates of the targeted grid cell. A boat can be hit, sunk or it may be a miss.

The current grids should be displayed with:

- A dot for a non-targeted cell
- A cross for missed shots
- The boat letter for hit or sunk boats

	0	1	2	3	4	5	6	7	8	9
0	x	.	.	.	.	.	.	.	.	.
1	.	.	.	.	C	C	x	.	.	.
2	.	x	.	.	.	.	.	M	.	.
3	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.
5	.	.	x	.	A	A	A	A	A	.
6	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	.	.	.	x	.	.
9	.	.	.	.	.	.	.	.	.	.

- **C++ skills:** Input/output, Vectors, File input, Data representation
- **AI tips:** Do not target the same cell twice, finish what you started

3.3 4 in a row

This game is played on a 6×7 grid. Each player has to drop a token in one of the column, that will fall to the bottom or above potential existing tokens in this column. The goal is to be the first to align 4 tokens, either vertically, horizontally or diagonally.

.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	o	.	x	.	.
.	o	x	x	o	o	.

- **C++ skills:** Input/output, Vectors, Data representation
- **AI tips:** Do not let the other player align 4 token (that is just trying not to lose), or try to guess best possible choices for the next 1, 2, 10... turns.

4 Algorithms

In this section, we program the computer to solve a given problem through an algorithm. Similarly to a cooking recipe, an algorithm is a sequence of basic steps (like finding the minimum value in a vector) that lead to an overall more complex behavior.

The problem data may be defined in a file or be randomly generated. The use of the STL and of the `math.h` functions will be of good use.

4.1 1D function minimization

Function minimization consists in finding the minimum value of a function, and where this value is reached. A classical algorithm to do so is the gradient descent. The corresponding algorithm is as follow:

Data: derivative f', starting point x, max iterations, gain λ, min gradient g_m
Result: x, the point where f reaches its minimum value
iter←0;
$g \leftarrow 2 \times g_m$;
while $iter < iter_{max}$ *and* $g < g_m$ **do**
  $g \leftarrow f'(x)$;
  $x \leftarrow x - \lambda.g$;
end
return x;

Algorithm 1: Gradient descent

This algorithm relies on the function derivative $f'(x)$ that must be defined. A simple case can be to find the minimum of the function $f(x) = ax^2 + bx + c$ where $a > 0$, starting from any value of x.

- **C++ skills:** Functions

4.2 nD function minimization

Here the function to be minimized is still a scalar but may take several arguments. A typical case is a quadratic function over $\mathbb{R}^n$: $f(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}\mathbf{x}$. The algorithm is the same as in 1D minimization except that $\mathbf{x}$ has several components. Instead of a function for the derivative, we need thus a function to compute the Jacobian of f that will return a n-D vector for all derivatives along the x_i's.

- **C++ skills:** Functions, Vector

4.3 Tower of Hanoi

This one is a classical mathematical puzzle. A given number of disks of different sizes are placed on three rods. The only constraint is that a bigger disk cannot be above a smaller one.

The goal is to move the disks one at a time (only the top disk of a rod may be moved to another rod) in order to have them all on a single rod. The Wikipedia page will give you strong hints to solve this problem.

The starting position of the disks may be randomly generated or written in a file. The goal is of course to use a minimum number of moves.

- **C++ skills:** File input, output, Vectors, Data representation, Recursive programming (functions)

4.4 Robust fitting with Hough lines

This method will be taught in Computer Vision. We assume we have a number of (x, y) points and want to find the line or lines that pass through them. A method was proposed by Hough and consists in finding all possible lines that pass through each point, and then count the similar

lines. The ones that have the highest number of counts (or votes) are considered to be actual lines in the image. Do to so, we write a line equation as:

$$x\cos\theta + y\sin\theta = \rho$$

We then consider a finite set of possible θ and ρ, typically:

- θ can go from $-\pi/2$ to $+\pi/2$ with 100 intermediary values
- ρ can go from 0 to half the diagonal of the image with 100 intermediary values

We thus have a 100×100 grid of all potential (ρ, θ) corresponding to 10000 lines.

In practice, at the beginning of the program you can define X as [-10, .., 10] and chose one, then two values for (a, b). Then Y is computed as $Y = aX + b$ and you have your set of X and Y coordinates. You can also add some random points to make it more difficult to the algorithm.

The algorithm is given below.

Data: X and Y, min. votes n
Result: (ρ, θ) for best lines
init ρ_v and θ_v vectors from desired range and step;
RT←zero matrix of dim. 100×100;
for *i in range(100)* **do**
  for *(x,y) in (X,Y)* **do**
    $\rho \leftarrow x\cos\theta_i + y\sin\theta_i$;
    j←index of ρ_v nearest to ρ;
    RT[j,i]+= 1;
  end
end
(ρ, θ) are the values where RT[i,j] > n;

Algorithm 2: Hough line detector

- **C++ skills:** Output, Vectors, Sort and Find maximum (STL)

4.5 Path planning with A*

The A* algorithm is a popular path planning method that can easily find the shortest path from a starting to a desired position in a graph. A graph is a set of nodes (positions or situations) linked by elementary steps (for example, going from (0,0) to (0,1)). It may be a classical 2D or 3D space, but also any kind of graph where we want to find the shortest path.

- **C++ skills:** File input and output, Vectors, Data representation, Sort and find minimum (STL), Classes, Inheritance, Templates or References

4.5.1 General A* algorithm

The algorithm is based on two functions:

- The travel function $g(i)$ expresses the distance between the starting position and the evaluated one.
- The heuristic function $h(i)$ tries to estimate the cost from the current position to the desired one.

The total cost function is the sum of g and h. It is also using two sets of nodes:

- The closed set regroups all nodes that have been evaluated (starts at empty)
- The open set regroups all nodes that are to be evaluated (starts with the initial position)

As we see, the algorithm does not depend on the type of the nodes, as long as the following can be defined:

- The h function from the node to the goal
- The *neighboor* function that gives the nearest nodes to the current node (`vector`?)
- The == operator that checks whether two nodes are the same

This is perfect for classes as the actual algorithm can be written in a generic way. I suggest you write it for a given class to test it without using templates.

4.5.2 Classical 2D path planning

This first A* problem consists of a 20×20 grid where a robot must go from $(0, 0)$ to $(19, 19)$. Obstacles exist of course in the grid. They can be defined randomly or in a text file that the program will read. In this case:

- The h function is the Euclidean distance to the goal.
- The *neighboor* function should give all the nodes reachable from a ± 1 move on x or y.
- The == operator corresponds to having equal x and y.

4.5.3 The 8-puzzle or 15-puzzle problem

In this problem, we have a 3×3 grid with cells numbered from 1 to 8[2], and an empty cell. The goal is to reach the following configuration:

1	2	3
4	5	6
7	8	.

We go from one position to another one by sliding one cell in the empty cell. For example, the two configurations that may lead to the desired one are:

[2] can also be done on a 4×4 grid with numbers 1..15

```
Data: start and goal positions
Result: Sequence of nodes that define the shortest path
closedSet← [];
openSet← [start];
inside ← true;
start.g← 0;
start.compute_h(goal);
start.compute_f();
while openSet not empty do
    candidate ← node in openSet with lowest f score;
    if candidate == goal then
        return candidate;
    end
    openSet.remove(candidate);
    closedSet.add(candidate);
    for each neighboor of candidate do
        union←openSet + closedSet;
        if neighboor is in union then
            twin←neighboor from union set ;
            if twin.g > neighboor.g then
                twin.set_parent(candidate);
                twin.compute_f();
                openSet.add(twin);
            end
        else
            neighboor.compute_h(goal);
            neighboor.compute_f;
            openSet.add(neighboor);
        end
    end
end
```

Algorithm 3: A* algorithm

1	2	3
4	5	6
7	.	8

1	2	3
4	5	.
7	8	6

In this case:

- The h function can be the Manhattan distance, that is:

$$h(x, y) = \sum_{i=1}^{8} (|x_i - x_i^*| + |y_i - y_i^*|)$$

where (x_i, y_i) is the position of cell i in the grid and (x_i^*, y_i^*) is the position of cell i in the desired grid.

- The *neighboor* function should give all the grids reachables from the current one.
- The == operator corresponds to having the cells at the same positions.

Note that not all starting grids can reach the desired one. The best way to build a starting position is to start from the desired one and randomly move cells for 30 to 40 steps. You will be surprised the algorithm will usually find a much shorter path to come back.

You should write the different steps in a file with the corresponding move and grid, to go from the starting grid to the desired one.

4.6 Genetic algorithms

Sometimes we want to find the minimum of a function that has many input variables and where the gradient descent algorithms cannot work because we have a lot of local minima (meaning that quickly the algorithm will stop even if the global minimum is not reached). In this case, Genetic Algorithms are a class of methods that are based on a heuristic search of the minimum.

In the GA method, an input point is called an individual and has a given score (which is simply the function evaluated for this input). We start by generating a set of random individuals, called the population. We then keep a number of best individuals amongst this population, and fill up the new population by re-generating new individuals that may be better. The overall algorithm is represented in Algorithm 4.

As we see, the algorithm by itself only needs three particular functions for individuals:

- The evaluation function that gives the score of the individual.
- The Crossing function that takes two individuals to return a new one. This function can be seen as changing two parents into one child that would be a mix of the parents's properties.
- The Mutate function that slightly changes an individual.

These functions depend on the actual problem that we are solving, two examples are shown below.

- **C++ skills:** File input and output, Vectors, Data representation, Sort and find minimum (STL), Classes, Inheritance, Templates or References

Data: Population size n, function f to be minimized, max iterations that do not change the best individual, number k of individuals that we always keep
Result: Individual that minimizes f
population← n random individuals;
iter←0;
while *iter* < $iter_{max}$ **do**
  best←best individual from population;
  new_population←best k individuals from population;
  for *i from k+1 to n/2* **do**
    n1←random number between 1 and n;
    n2←random number between 1 and n, different from n1;
    new_population← best between population[n1] and population[n2];
  end
  for *i from 1 to n/2* **do**
    n1←random number between 1 and n;
    n2←random number between 1 and n, different from n1;
    new_indiv = cross(population[n1], population[n2]);
    new_indiv.mutate();
    new_population.Add(new_indiv);
  end
  population←new_population;
  new_best←best individual from population;
  if *new_best == best* **then**
    iter←iter+1;
  else
    iter←0;
  end
end

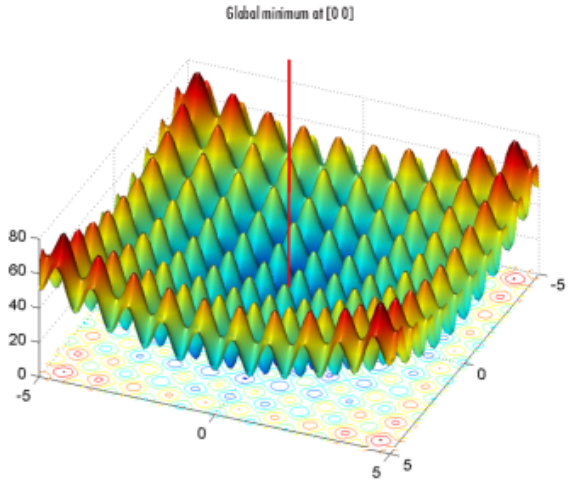
Algorithm 4: Genetic Algorithm

4.6.1 Minimizing a function on $\mathbb{R}^2$

A classical function to test GA is the Rastrigin's function:

$$f(x, y) = 20 + x^2 + y^2 - 10\left(\cos(2\pi x) + \cos(2\pi y)\right)$$

The graph looks like this, with the global minimum at (0,0):



For this kind of problem:

- An individual is of course a (x,y) position
- A random individual consists in random (x,y) values in [-5,5]
- The evaluation function is $f(x, y)$
- Crossing two individuals consists in computing intermediary values: $x_{\text{new}} = \alpha x_1 + (1-\alpha)x_2$ where α is a random number in $[0, 1]$
- Mutating an individual is slightly changing its (x,y) values with a random number

4.6.2 The traveling salesman problem

Like the Tower of Hanoi, the TSP is a classical mathematical problem. We assume a salesman has to go in various cities to sell his stuff. Plane tickets are of course at a different price to go from one city to another (or maybe there is not always a direct plane from A to B). The goal is to find the sequence of cities that will have the salesman go only once in each city, come back at his initial city, and all that with the minimum price.

In our case, the ECN administration forgot to give you the diplomas at the end of the master. I thus have to go to all students's capital cities to deliver it and of course I do not want to spend too much on the plane tickets. The file `tsp.yaml` gives the list of all cities and their distances[3].

In this instance of Genetic algorithm:

- An individual is a sequence of numbers from 1 to N giving the travel ordering
- A random individual consists in a random permutation of [1..N]
- The evaluation function of an individual is the price required to travel through the corresponding city ordering

[3]Plane tickets price are not proportional to distances but it was easy to generate automatically all distances between the cities

- Crossing two travels consists in keeping the beginning of the first parent up to a random city, and finish with the remaining cities in the ordering given by the second parent
- Mutating an individual can be switching two cities randomly
- **C++ skills:** Yaml file input