“Gheorghe Asachi” Technical University of Iași
Faculty of Automatic Control and Systems Engineering
Master programme: Systems and Control

# Integration Project Systems & Control

*Alexandru Cohal*

**May 2016**

## 1. Introduction

Two-link arm robots are already very popular in industrial environments due to their advantages in completing various tasks very fast and very precise. Due to the development of the computational possibilities, the planning of their motion can be done autonomous, without the need of an operator.

The purpose of this project is to develop a program which can generate the motion steps of a two-link arm robots from a start point to a goal point without colliding into obstacles.

## 2. Problem definition

A two-link arm robot is considered (Fig. 1). The configuration of this robot is given by the two angles $\theta_1$ and $\theta_2$ (each angle has values in the interval $[0, 2\pi)$).

A kinematic model is assumed, where the control inputs for the system are the angular velocities $\omega_1$ and $\omega_2$ ($\omega_1 = \dot{\theta}_1$ and $\omega_2 = \dot{\theta}_2$), restricted such that $\omega_{1,2} \in \{-1, 0, 1\} \frac{rad}{s}$. Thus, each joint can be stopped or it can rotate clockwise or counter-clockwise with constant angular velocity.

The joints are not constrained by limiting angles (each joint can move from angle $2\pi_-$ to $0_+$ by crossing through 0). The robot is non-intersecting ($link$ 2 crosses above $link$ 1, when $\theta_2$ crosses through $\pi$).

The robot links have a negligible width.

The robot evolves in a plane cluttered with polygonal and convex obstacles.

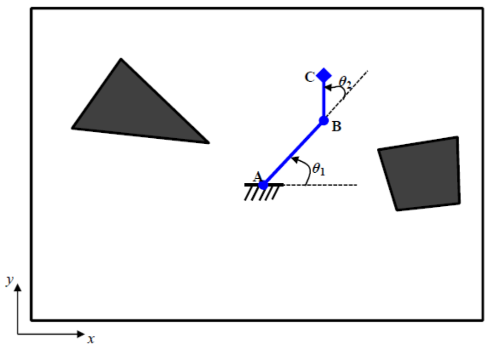


*Fig. 1. Two-link arm robot (blue) evolving in a 2D environment cluttered with obstacles.*

**Inputs** of the problem (information given by the user):

- the environment bounds
- the set of vertices of each obstacle
- the position of the base $A$ of the robot
- the link lengths $L_1$ and $L_2$
- the initial configuration of the robot ($\theta_{10}$ and $\theta_{20}$) chosen such that the robot does not collide with any obstacle
- the final (desired) position of the effector $C$, denoted by ($x_{C\_final}$ and $y_{C\_final}$).

**Outputs** of the problem (information to be determined):

- a control sequence for inputs ($\omega_1$ and $\omega_2$) such that the effector C reaches the desired position and the robot body does not collide with any obstacles while moving.

## 3. Solution

Stages of solving the problem:

a) Read and Store the input information

- Read the environment bounds
- Display an empty figure having the given bounds
- Read the number of obstacles
- For each obstacle, read the points (provided by mouse clicks inside the figure) which define it, construct the convex hull and then display the convex polygon which represents that obstacle in the figure
- Read the points defining the initial position of the two-link arm robot (points A, B and C from Fig. 1) and then display it on the figure; Compute the lengths and the angles ($\theta_{10}$ , $\theta_{20}$) of the links
- Read the goal point of the end-effector (point C) and then display it on the figure.

b) Make an initial verification: Verify if the distance from the base-point of the robot to the goal point of the end effector is smaller or at least equal with the sum of the links' lengths. If this condition is not fulfilled then a solution for the given problem cannot be found and an error message is displayed.

c) Generate the Configuration Space (initially, a resolution step has to be set). The solution of the problem will be determined by searching in the space of all possible configurations of the two-link arm robot (which is a 2D space due to the fact that the robot has 2 Degrees of Freedom). By using the C-space, the solution of the problem is easier to be determined because it will only require to find a path (the shortest or the least expensive) in a graph.

d) Determine the two positions in the Configuration Space of the goal point (by using Inverse Kinematics). Verify if at least one of these positions corresponds to a valid point in the configuration space. If not, the goal point cannot be reached and an error message is displayed.

e) On the generated Configuration Space, apply the Grassfire Algorithm (using neighbours in 4 directions only) and determine the shortest path from the start point to one of the goal points by passing only through valid configurations (those which do not imply a collision with an obstacle). Display the path found and then move the two-link arm robot (displayed in the first figure) from the initial position to the goal position using the sequence found.

f) Construct the equivalent Graph of the generated Configuration Space (using neighbours in 8 directions and associating costs to edges (1 for horizontal or vertical neighbouring and $\sqrt{2}$ for diagonal neighbouring)). The nodes of the graph will be only the points representing valid configurations.

g) Using the generated Graph, apply the Dijkstra's algorithm in order to determine the shortest path from the start point to one of the goal points. Display the path found and then move the two-link arm robot (displayed in the first figure) from the initial position to the goal position using the sequence found.

h) Display the statistics (number of visited nodes, length and cost of shortest path) for both algorithms applied.

## 4. Implementation

The solution was implemented in *Matlab* environment. For each stage presented before was implemented one or more functions for a better structuring of the program and for easier future developments. The resolution of the generated Configuration Space was set to $0.1\ rad$.

Some particularities of the implementation realized are:

- The functions which were using the generated Configuration Space, also needed the positions of the start point and the goal points in this space. In order to not compute these positions every time, the Configuration Space was stored in a structure (`Cspace`) together with these positions and with the value of the resolution used.
- The Grassfire algorithm was applied directly on the matrix storing the Configuration Space, not on the equivalent graph (because the neighbours were easy to find out). Thus, the graph was not needed to be generated, saving time (1.11 seconds) and memory (496 Kbytes).
- In order to display the paths resulted from the Grassfire and Dijkstra algorithms, the figure containing the Configuration Space had to be generated multiple times. A function was implemented for this, but the time needed for each execution of this function is very high (7.4 seconds) compared with the execution time needed by the other functions (e.g. 2.69 seconds for applying Grassfire algorithm and displaying the visited nodes and the shortest path). This high execution time is caused by the high number of points which have to be plotted (3969 points for a resolution of the Configuration Space of $0.1\ rad$).
- The Dijkstra's algorithm was optimized by not visiting the nodes which are placed at a distance greater than the current distance to a goal node. Thus, the number of visited nodes decreases (also, the execution time decreases).
- Due to the fact that the robot's joints are not constrained by limiting angles, the Configuration Space had to be circular (e.g. the neighbours from the left side of the points from the first column are the points from the right column of the Configuration Space). A function which was assuring this circularity was implemented.
- The simulation of motion of the displayed robot from the start position to the goal position implies the modification of the positions of the links and joints in the figure. This was done by storing the handlers for each component in a structure and then modifying their properties (e.g. `XData` and `YData` for changing the position).
- Due to the resolution of the Configuration Space, the goal points not always have a perfect correspondent (the angles $\theta_{1\_goal}$ and $\theta_{2\_goal}$ are not multiples of the configuration space resolution). This made that the simulation of the movement of the robot to not end in the desired goal point. A supplementary step was added at the end of the simulation process such that the robot to be in the end in the goal position.

## 5. Results

The results obtained for a certain input set is shown in the following figures.

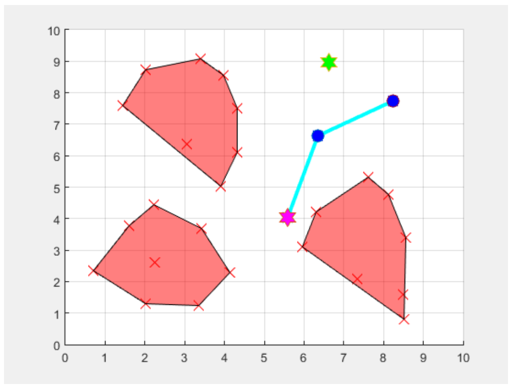


*Fig. 2. The initial environment*
*(red convex polygons – obstacles, cyan lines – robot's links, blue discs – robot's joints, magenta hexagram – initial position of the end-effector, green hexagram – goal position of the end-effector)*

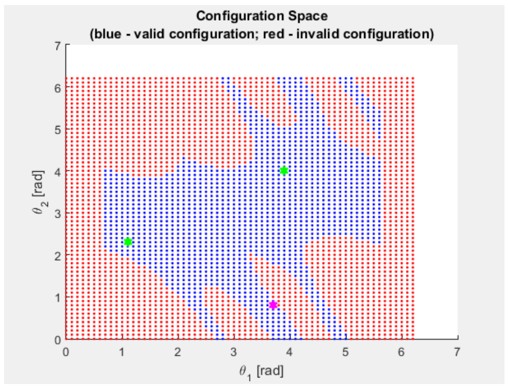


*Fig. 3. The generated Configuration Space*
*(blue – valid configurations, red – invalid configurations, magenta – initial point, green – goal points)*

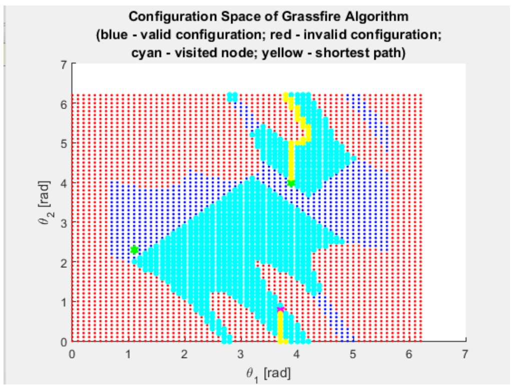


*Fig. 4. The Configuration Space after applying Grassfire algorithm*
*(blue – valid configurations, red – invalid configurations, magenta – initial point, green – goal points, cyan – visited points, yellow – shortest path)*

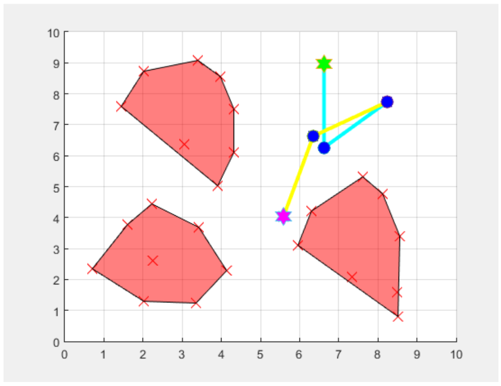


*Fig. 5. The environment containing the initial position of the robot (yellow) and the final position (cyan) using the path generated by the Grassfire algorithm (red convex polygons – obstacles, cyan and yellow lines – robot's links, blue discs – robot's joints, magenta hexagram – initial position of the end-effector, green hexagram – goal position of the end-effector)*

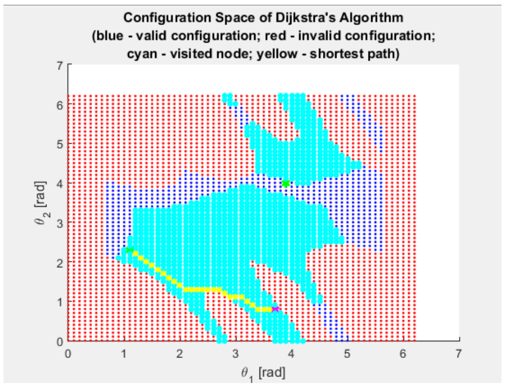


*Fig. 6. The Configuration Space after applying Dijkstra's Algorithm (blue – valid configurations, red – invalid configurations, magenta – initial point, green – goal points, cyan – visited points, yellow – shortest path)*

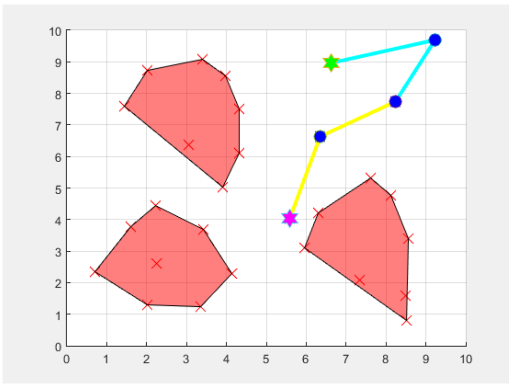
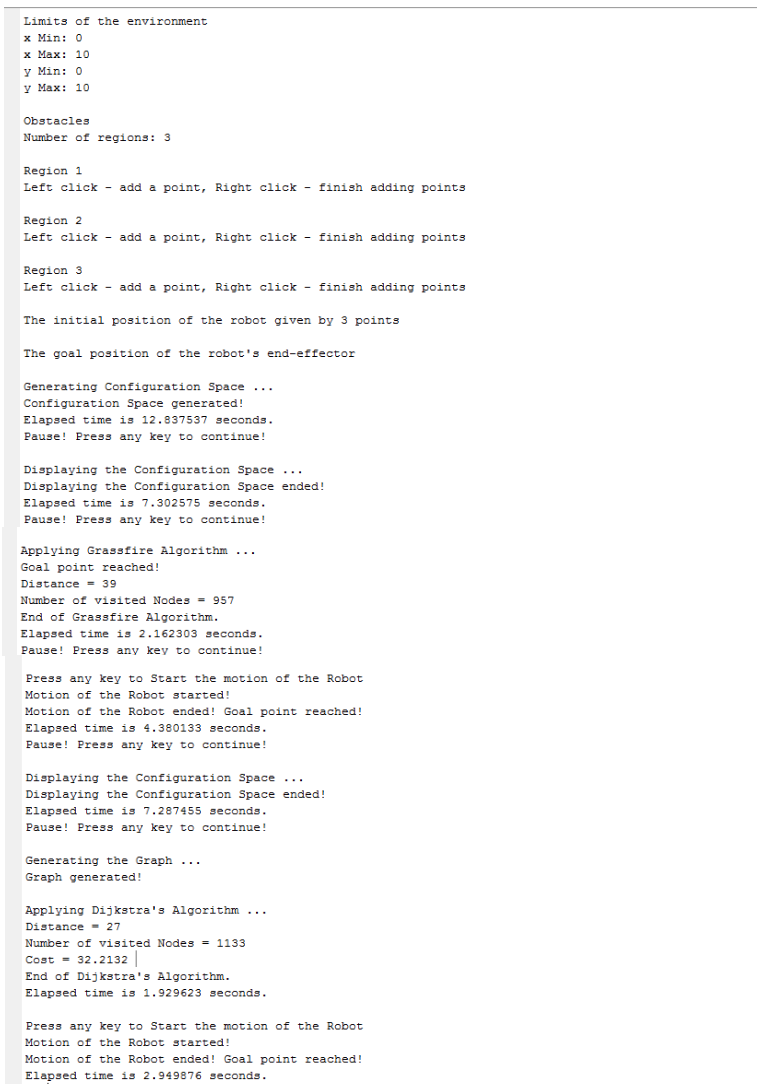


*Fig. 7. The environment containing the initial position of the robot (yellow) and the final position (cyan) using the path generated by the Dijkstra's algorithm (red convex polygons – obstacles, cyan and yellow lines – robot's links, blue discs – robot's joints, magenta hexagram – initial position of the end-effector, green hexagram – goal position of the end-effector)*



```
Limits of the environment
x Min: 0
x Max: 10
y Min: 0
y Max: 10

Obstacles
Number of regions: 3

Region 1
Left click - add a point, Right click - finish adding points

Region 2
Left click - add a point, Right click - finish adding points

Region 3
Left click - add a point, Right click - finish adding points

The initial position of the robot given by 3 points

The goal position of the robot's end-effector

Generating Configuration Space ...
Configuration Space generated!
Elapsed time is 12.837537 seconds.
Pause! Press any key to continue!

Displaying the Configuration Space ...
Displaying the Configuration Space ended!
Elapsed time is 7.302575 seconds.
Pause! Press any key to continue!

Applying Grassfire Algorithm ...
Goal point reached!
Distance = 39
Number of visited Nodes = 957
End of Grassfire Algorithm.
Elapsed time is 2.162303 seconds.
Pause! Press any key to continue!

Press any key to Start the motion of the Robot
Motion of the Robot started!
Motion of the Robot ended! Goal point reached!
Elapsed time is 4.380133 seconds.
Pause! Press any key to continue!

Displaying the Configuration Space ...
Displaying the Configuration Space ended!
Elapsed time is 7.287455 seconds.
Pause! Press any key to continue!

Generating the Graph ...
Graph generated!

Applying Dijkstra's Algorithm ...
Distance = 27
Number of visited Nodes = 1133
Cost = 32.2132
End of Dijkstra's Algorithm.
Elapsed time is 1.929623 seconds.

Press any key to Start the motion of the Robot
Motion of the Robot started!
Motion of the Robot ended! Goal point reached!
Elapsed time is 2.949876 seconds.
```

*Fig. 8. The messages displayed by the program (the most important are: the execution times, number of visited nodes, length of paths, cost of Dijkstra's path)*

## 6. Conclusion

The purpose of this program is to find a path from an initial point to a desired point of a two-link arm robot. The solution was found using the equivalent Configuration Space of the given environment. Two methods were used to find the solution: Grassfire Algorithm which considers the neighbours of a point from the Configuration Space only from 4 directions (horizontal and vertical) and Dijkstra's Algorithm which considers the neighbours of a point from all the 8 directions (horizontal, vertical and diagonal), associating in the same time different costs. The solution found by the Dijkstra's algorithm is better (due to the fact that considers 8 directions) (shorter length of the path, smaller execution time) but the memory used is higher (32 Kbytes instead of 528 Kbytes for a resolution of the Configuration Space of 0.1 $rad$). The same results as the ones provided by Dijkstra's algorithm can be obtained in a smaller execution time by using the A* algorithm.