Università degli studi di Napoli Federico II

Laurea magistrale in ingegneria dell'automazione e robotica

Field and service robotics

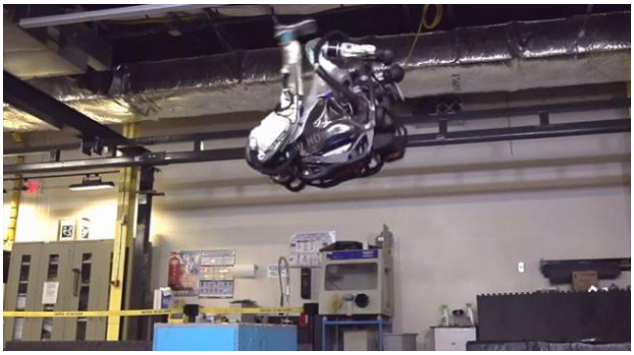
Homework 1

a.y. 2023/24

Professor: Fabio Ruggiero

Salvatore Del Peschio

(a) Atlas standing



(b) Atlas backflipping

1. Consider the ATLAS robot from Boston Dynamics in the pictures above. On the left, ATLAS is standing. On the right, ATLAS is performing a backflip. If ATLAS actuators can produce unbounded torques, establish whether each of the following statements is true or not, and briefly justify your answer.

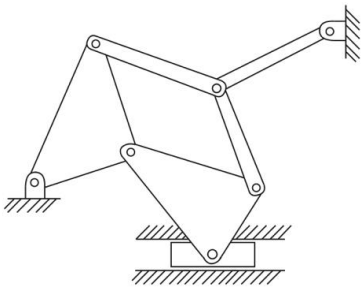
a) While standing, ATLAS is fully actuated.

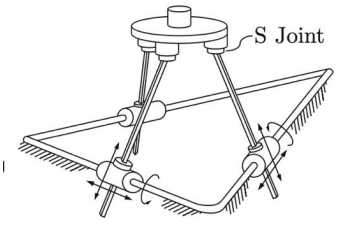
b) While doing backflip, ATLAS is fully actuated

ATLAS is a humanoid robot, as such it is similar to a human body and therefore has a considerable number of actuators, even multiple ones per joint, these properties ensure it with a remarkable agility.

a) TRUE: ATLAS is fully actuated while standing because when it is in contact with the ground, due to friction with it, it is able to accelerate its center of mass in all directions controlling the actuation of its legs.

b) FALSE: ATLAS is underactuated when it is not in contact with the ground, such as during the backflipping motion, there is no combination of inputs to the actuators that can change the ballistic trajectory of its center of mass, thus making it underactuated.





2. Consider the planar mechanism on the above-left and the spatial mechanism on the above-right. Determine the number of the degrees of freedom for each mechanism, comparing the result with your intuition about the possible motions of these mechanisms. For each mechanism, write its configuration space topology.

1. The planar mechanism consists of a quadrilateral connected to the grounds by two revolute joints and one prismatic joint; the structure of the triangular links and the large number of constraints suggest that this can be regarded as a rigid structure. Using Grübler's formula considering $N = 7$ links and $J = 9$ joints we obtain

$$DoFs = m(N - 1 - J) + \sum_{i=1}^{J} f_i = 3(7 - 1 - 9) + 9 = 0$$

   It makes no sense to talk about configuration space topology for a structure without degrees of freedom.

2. The structure of the space mechanism is that of a parallel robot, this suggests the possibility of being able to achieve any position and orientation with the upper platform in a limited working space. The mechanism is composed by three prismatic joints ($f_i = 1$), three cylindrical joints ($f_i = 2$) and three spherical joints ($f_i = 3$). Using Grübler's formula considering $N = 8$ links and $J = 9$ joints we obtain

$$DoFs = m(N - 1 - J) + \sum_{i=1}^{J} f_i = 6(8 - 1 - 9) + 18 = 6$$

   The configuration space topology is

$$\mathbb{R}^3 \times \mathbb{T}^3$$

   It has been obtained from the cylindrical joints at the base, it could also be derived intuitively by considering the coordinates needed to describe the upper platform in three-dimensional space.

3. State whether the following sentences regarding underactuation or fully actuation are true or false. Briefly justify your answers.

a) A car with inputs the steering angle and the throttle is underactuated.

b) The KUKA youBot system on the slides is fully actuated.

c) The hexarotor system with co-planar propellers is fully actuated.

d) 7-DoF KUKA iiwa robot is redundant.

a) TRUE: A machine is definitely underactuated because it has two inputs and three degrees of freedom in the plane; moreover, it is clear that it cannot generate accelerations directed perpendicular to the direction in which it is facing.

b) TRUE: We know that a system is underactuated if it is underactuated in all states and times. The KUKA youBot is fully actuated because, unless it is at the limits of its workspace or in a singular arm configuration, through the moving base or the actuation of the arm it is capable of generating accelerations of its center of mass or end effector in all directions.

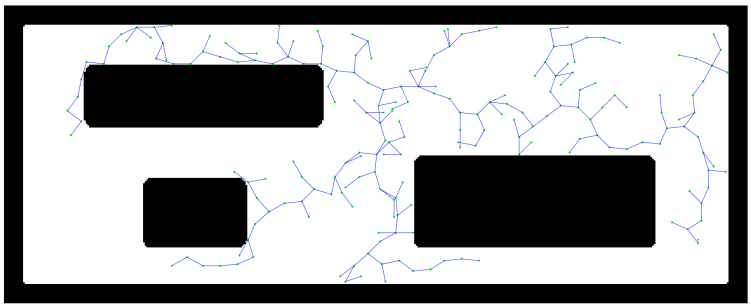
c) FALSE: Despite having a sufficient number of actuators the hexarotor is underactuated since it is unable to generate accelerations along the plane on which its propellers are arranged.

d) TRUE: A 7-DoF KUKA iiwa robot is intrinsically redundant for any task since it always possesses at least one redundant DoF.

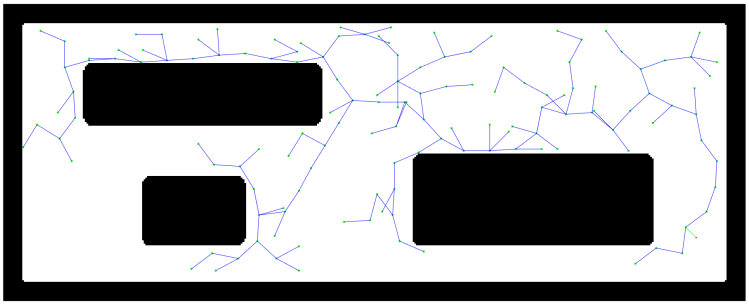
4. Consider the map attached as *image_map.mat*. The 0 value represents an obstacle, the 1 value is a free point in the partitioned map. To show the map, use the command *imshow(image_map)*. Implement yourself in a software program the RRT method for a point robot moving in the above map from $q_i = [30\ 125]$ to $q_f = [135\ 400]$. Select a suitable maximum number of iterations and show the obtained graph if a solution has been found. Otherwise, report a failure and increase the maximum number of iterations. Repeat the procedure for a finite number of times at your choice. [Hint: the robot is a point; therefore, the collision check algorithm is very simple: for a given point, you must just check whether the associated value of the map is 0 or 1.]

The RRT algorithm was implemented in the following way:

- The map was imported and all variables useful for program execution were created, among them also the delta parameter, the number of points to be generated for each iteration and the maximum number of iterations.
- A certain amount of q_{new} per iteration are generated in the following way:
  - A q_{rand} is generated at a random location.
  - The q_{near} is extracted with the *get_closest_index* function.
  - The q_{new} is generated, if it is within the map and there are no collisions (*check_collisions* function) between it and q_{near} then q_{new} is added to the list of nodes and drawn on the map with the corresponding arc connecting it to q_{near}.
- Starting from the node closest to q_f we check if the path that connects them is collision-free, if it is the algorithm signals that a solution has been found, otherwise, it checks the second closest node and so on, if no node provides a solution the program reports a failure and if the number of attempts is less than the pre-established maximum, others q_{new} nodes are generated.



(a) RRT with $\delta = 10$

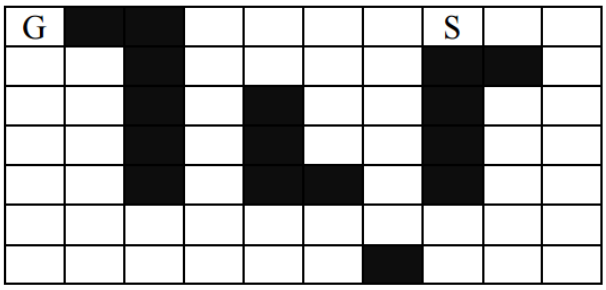


(b) RRT with $\delta = 15$

5. Given the map below (select the one based on your matriculation number), implement a software program for the motion planning method based on the numerical navigation function. The black cells are obstacles, the cell with S is the starting one, the goal cell is denoted by G. If the algorithm is successful, provide the sequence of cells from the start to the goal. First choose 4 adjacent cells, then 8 and comment on the comparison.

The algorithm was implemented through the use of recursive functions, the use of these is based on the need to use the outputs of the function as its inputs until a termination condition is satisfied. The developed functions are described below:

- *get_adj_orth_cells*: returns the four-adjacent cells of all cells contained in a list passed as input.
- *get_adj_cells*: returns the eight-adjacent cells of all cells contained in a list passed as input.
- *compute_cost_map*: take as input the starting cell and fills all the cells of the *nav_map* with the cost to reach that cell. Starting from a cell it assigns the costs to the adjacent cells by



nav_map

assigning the correct value to the free cells and -1 to those occupied by obstacles. Subsequently, it find all the cells adjacent to the cell passed as parameter and call itself passing as input the adjacent cells that have just been filled with their cost, this operation is repeated until every cell of the *nav_map* has been filled.

- *steepest_descent*: take as input a cell and find the free adjacent cell with the minimum cost value, it call itself until we reach the starting position (0 cost value).
- *main*: take as input the exploration function to use (*get_adj_orth_cells* or *get_adj_cells*) and executes the algorithm printing the *nav_map* and the best path. In addiction, it measures the time needed to compute the cost map and to find the best path.

Example of execution:

```
Computed cost map:
[[ 0. -1. -1. 13. 14. 15. 16. 17. 18. 19.]
 [ 1.  2. -1. 12. 13. 14. 15. -1. -1. 18.]
 [ 2.  3. -1. 11. -1. 15. 14. -1. 16. 17.]
 [ 3.  4. -1. 10. -1. 14. 13. -1. 15. 16.]
 [ 4.  5. -1.  9. -1. -1. 12. -1. 14. 15.]
 [ 5.  6.  7.  8.  9. 10. 11. 12. 13. 14.]
 [ 6.  7.  8.  9. 10. 11. -1. 13. 14. 15.]]
Execution time: 0.000541s
Best path:[0, 7]->[0, 6]->[0, 5]->[0, 4]->[0, 3]->[1, 3]->[2, 3]->[3, 3]->[4, 3]->
[5, 3]->[5, 2]->[5, 1]->[5, 0]->[4, 0]->[3, 0]->[2, 0]->[1, 0]->[0, 0]
Path length: 18
Execution time: 0.000076s

Computed cost map:
[[ 0. -1. -1. 10. 10. 10. 11. 12. 13. 14.]
 [ 1.  1. -1.  9.  9. 10. 11. -1. -1. 14.]
 [ 2.  2. -1.  8. -1. 10. 11. -1. 13. 13.]
 [ 3.  3. -1.  7. -1. 10. 10. -1. 12. 12.]
 [ 4.  4. -1.  6. -1. -1.  9. -1. 11. 12.]
 [ 5.  5.  5.  6.  7.  8.  9. 10. 11. 12.]
 [ 6.  6.  6.  6.  7.  8. -1. 10. 11. 12.]]
Execution time: 0.000573s
Best path:[0, 7]->[0, 6]->[0, 5]->[1, 4]->[2, 3]->[3, 3]->[4, 3]->[5, 2]->[4, 1]->
[3, 1]->[2, 1]->[1, 1]->[0, 0]
Path length: 13
Execution time: 0.000214s
```

It is possible to notice the better performance of the algorithm with the four-adjacent cells due to the smaller number of cells to check during the cost assignment and search for the optimal path, despite this it is clear that the freedom of movement of the algorithm using eight-adjacent cells allows you to find shortest paths.