COM S 476/576 Homework 2: Geometric Representations and Transformations

Task 1 [5 points]: Solve problem problem 3.1 in the textbook, S. M. LaValle, Planning Algorithm, 2006.

Task 2 [5 points]: Solve problem problem 3.7 in the textbook, S. M. LaValle, Planning Algorithm, 2006.

Task 3 [10 points]: Consider a 2D kinematic chain $A_1, \ldots, A_m$ for some integer $m \geq 0$. Each link has width W and length L. The distance between the two points of attachment is D. Link A_1 is attached to the origin. For each i such that $1 < i \leq m$, link A_i is attached to link A_{i-1} by a revolute joint. The configuration of the chain is expressed with m angles $(\theta_1, \ldots, \theta_m)$, where θ_1 represents the angle between A_1 and the x-axis, and for each i such that $1 < i \leq m$, θ_i represents the angle between A_i and A_{i-1}.

Your task is to implement the following components.

1. Define a new ROS msg called `Chain2D.msg` that includes the following fields
   - `config`: A variable-length array of type float32 that represents the configuration $(\theta_1, \ldots, \theta_m)$ of the chain for any integer $m \geq 0$.
   - `W`: A float32 variable that represents the width W of each link.
   - `L`: A float32 variable that represents the length L of each link.
   - `D`: A float32 variable that represents the distance D between the two points of attachment.

2. Implement the `publish(config, W, L, D)` function in `hw2_chain_configurator.py`, which can be found in the course repo. This function should keep publishing a message of type `Chain2D` defined in the previous task at the rate of 1 Hz to a channel called `chain_config` based on the given parameters of `config`, `W`, `L`, `D`. Once you implement this function, you should be able to run

   ```
   rosrun cs476 hw2_chain_configurator.py \
     0.7853981633974483 1.5707963267948966 -0.7853981633974483 -W 2 -L 12 -D 10
   ```

   which should publish at the rate of 1 Hz a `Chain2D` message with

   $$\begin{aligned} \texttt{config} &= [0.7853981633974483, 1.5707963267948966, -0.7853981633974483] \\ \texttt{W} &= 2 \\ \texttt{L} &= 12 \\ \texttt{D} &= 10 \end{aligned}$$

3. Implement the `get_chain_msg()` in `hw2_chain_plotter.py`, which can be found in the course repo. This function should receive a message from the `chain_config` channel (the same channel to which `hw2_chain_configurator.py` publishes). It should wait until a message is received and then return a `Chain2D` object from the received message.

4. Implement the `get_link_positions(config, W, L, D)` in `hw2_chain_plotter.py`, which can be found in the course repo. Note that this function should work for any number $m \geq 0$ of links. The input `config`, `W`, `L`, `D` are defined as in those in the `Chain2D` ROS msg. The function should return a tuple `(joint_positions, link_vertices)` where

   - `joint_positions` is a list of length $m+1$ such that `joint_positions[i]` is the position `[x,y]` of the joint between link A_i and A_{i+1} for any `i` $\in \{1, \ldots, m-1\}$. For any $m > 0$, `joint_positions[0] = [0, 0]`. Finally, `joint_positions[m]` is the position of the joint between A_m and a non-existing joint A_{m+1}, with the same geometry as the other links. Note that if $m = 0$, `joint_positions` should be an empty list.
   - `link_vertices` is a list of length m such that `link_vertices[i]` is the list of `[x,y]` positions of vertices of link A_{i+1}.

   Once this function and the `get_chain_msg()` in the previous task is implemented, you should be able to run

   ```
   rosrun cs476 hw2_chain_plotter.py
   ```

   which should generate a plot as shown in the figure below.

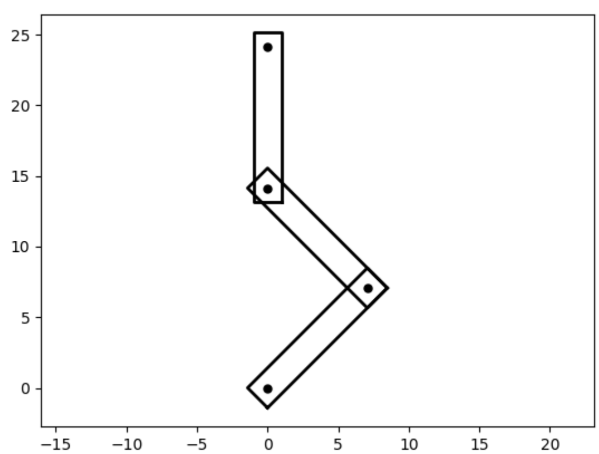

Figure 1: The plot generated after running `hw2_chain_plotter.py`, together with `hw2_chain_configurator.py` with the parameters in Task 3.2.

Submission: For Task 1 and 2, please submit a pdf with your solution. For Task 3, please submit a single zip file on Canvas containing the followings:

- your code (with comments, explaining clearly what each function/class is doing), and
- a text file explaining clearly how to compile and run your code.