In Robotics, **configuration space** (also called C-Space) refers to the set of positions reachable by a robot's end-effector considered to be a rigid body in three-dimensional space.

- **C-Space**: Gives one unified way to talk about any robot.

---

**Fig 1: Mapping of a robot and an obstacle from World Space to C-Space**

The image shown above contains the following elements:

- **World ‘W’** where
  \[ W = \mathbb{R}^2 \text{ or } \mathbb{R}^3 \]

- **Obstacle ‘O’** is a subset of ‘W’
  \[ O \subset W \]

- **Configuration ‘q’** and configuration space ‘C’
  \[ q \in C \]

Such that:

\[ A(q) \rightarrow W \]

I.e. ‘A’ is a function that implies the geometry of the object is known and hence, it is the only thing that needs to be transmitted for the configuration of the robot. It enables rendering from C-Space to the world ‘W’.

- **C_{obs} = C-obstacle** i.e. all configurations in ‘C’ such that
  \[ C_{obs} = \{ q \in C \mid A(q) \cap O \neq \phi \} \]
  where \( \phi \) implies ‘null’ or empty.

- **Free C-Space**:
  \[ C_{free} = C \setminus C_{obs} \]

- **C-space** can also be defined as the number of numbers needed to be transmitted such that anyone plotting these numbers ends up with the same final result.
For a 2-link arm:

C-space reduces the complexity of the arm/robot to a single point but, due to this, the complexity of the obstacles increases in C-Space.

Creating C-Spaces for full-dimensional spaces is difficult – the obstacle can never be fully constructed; only the parts that can be reconfigured to fit the C-Space can be constructed. It is always a trade-off between “need to know more about the obstacle” vs. “computational difficulty of finding the information about the obstacle” i.e. for a 26 dimensional space this would be $10^{26}$. Thus, it is always a trade-off between “exploration” and “exploitation”.

$$C_{obs} = o \ominus A(0) = \{(o - a) | o \in O \text{ and } a \in A(0)\}$$

where, $\ominus$ is the ‘Minkowski difference’.

Example:

According to this diagram, the implication is that it is not possible to go from 0 to $2\pi$ when the obstacle has been added into the C-space. However, this is incorrect as the manipulator can rotate anti-clockwise and reach the goal ($2\pi$) from the initial state (0). This implies that instead of a line, the robot would actually look more like a circle in C-space i.e. 0 and $2\pi$ are IDENTIFIED which implies that if you leave the line at $2\pi$ you will emerge again at 0.
Fig 4: ‘0’ and ‘2π’ are identified due to which the C-space can be represented in this manner

Some more examples of identified points/edges:

Fig 5: Edges ‘0’ and ‘2π’ are identified for this 2-link arm which maps the C-Space into a torus

In the above figure in world space the last configuration of the arm is such that θ₁ is constrained by the obstacle but θ₂ can spin. It is easier to handle the torus as compared to sphere-like topologies as spheres have singularities at the poles whereas the torus does not.

Fig 6: In the image on the left the two marked edges are identified which maps the C-Space into a cylinder and the right image maps to a ‘two-sphere’ in C-Space
C_{obs} is a good concept but it is very difficult to use practically for path planning. As seen in the figure below the robot maps to a point but the obstacles that are relatively simple in the world-space map to very complex obstacles in C-Space.

**Piano Movers’ Problem**

This was a motion planning problem by Schwartz and Sharir. It states the following:

1. A world \( W' = \mathbb{R}^2 \) or \( \mathbb{R}^3 \)
2. Obstacle region \( O' \subset W \)
3. A robot ‘A’ and its C-Space ‘C’
4. \( C_{free} = C \setminus C_{obs} \)
5. \( q_i \in C_{free} \) and \( q_g \in C_{free} \)
6. Compute a path 

   \[ \tau : [0,1] \rightarrow C_{free} \]

   such that –

   \[ q_i = \tau(0) \text{ and } q_g = \tau(1) \]

   (Note: They aim to find one feasible path which may not be optimal)
Considering the 2-link robot mentioned earlier,
\[
\begin{pmatrix} x \\ y \end{pmatrix} = X_g \text{ and } q = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} \in C
\]

I.e. initial state ‘\(q_i\)’ is known and the target ‘\(X_g\)’ is known.

\[Q_g\] is a disjoint set because you can only state the final position of the hand to be a point; the final position of the arm (depending on the redundancy of the robot) will be a set of points as shown in the figure.

\(Q_g\) is a set of all solutions of IK\(X_g\) which may be non-linear and also discontinuous due to joint limits, obstacles(self collisions and external obstacles) etc.

\(Q_g\) can be generated using various methods like:

- Symbolic solvers like [IKFAST] – these do not work for all classes of problems, and they are also only suited to some particular robot-arm types
- Numeric solvers like Newton’s method

Newton’s Method

We want a \(q^*\) such that

\[FK(q^*) = X_g\]

In other words, our goal is to find the roots that satisfy the following equation \(f(q)\)

\[FK(q) - X_g = 0\]

**Newton’s Method: 2-dimension**

1. Initialize (guess) at some \(q_0\)
2. Linearize \(f(q)\) about \(q_0\)
   \[
   \frac{f(q_0) - f(q)}{q_0 - q} = M
   \]
Solve for \( f(q) = 0 \) leads to

\[ q = q_0 - \frac{f(q)}{M} \]

This will be our \( q_1 \). If \(|f(q_1)| < \varepsilon\), which means a sufficient approximation of \( q^* \) is reached. If not, go back to step 1 and repeat the iterative process.

* Newton’s method works well on convex functions due to quadratic convergence.
* Newton’s method may not work on functions with local minima, since \( M = 0 \) at these points.

**Newton’s Method: N-dimension**

Here we have

\[ FK(q) = X_g \]

\( q \) is \( N \times 1 \) and \( X_g \) is \( M \times 1 \). We will look at a 3-link arm moving on a plane \((x, y)\), so \( N = 3 \) and \( M = 2 \). In this case, after expanding \( f(q) = 0 \) we will get

\[
\begin{align*}
\cos \theta_1 + \cos(\theta_1 + \theta_2) + \cos(\theta_1 + \theta_2 + \theta_3) &= X_g \theta_1 \\
\sin \theta_1 + \sin(\theta_1 + \theta_2) + \sin(\theta_1 + \theta_2 + \theta_3) &= Y_g \theta_2
\end{align*}
\]

Linearizing about \( q_0 \),

\[ f(q) \approx f(q_0) + \frac{df}{dq}(q - q_0) \]

In this equation, \( \frac{df}{dq} \) is an \( M \times N \) Jacobian Matrix. In our case,

\[
\frac{df}{dq} = \begin{bmatrix}
\frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} \\
\frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3}
\end{bmatrix} = J
\]

Since we want \( f(q) = 0 \), we can arrive at

\[ q = q_0 - J^+ f(q_0) \]

where \( J^+ \) is called the pseudo-inverse of \( J \).