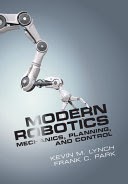
Modern Robotics - Mechanics, Planning, And Control

*Cambridge University Press – 2017*

[*https://modernrobotics.northwestern.edu/nu-gm-book-resource/foundations-of-robot-motion/*](https://modernrobotics.northwestern.edu/nu-gm-book-resource/foundations-of-robot-motion/)

<https://www.youtube.com/watch?v=jVu-Hijns70&list=PLggLP4f-rq02vX0OQQ5vrCxbJrzamYDfx&index=1>

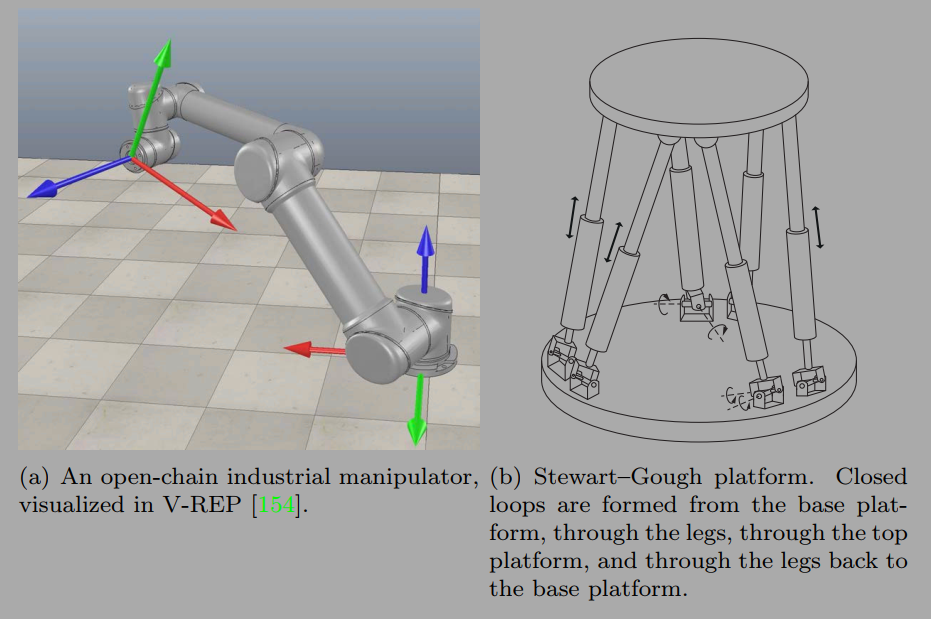


More information on the book, including software, videos, and a feedback form can be found at <http://modernrobotics.org>.

# 1 - Preview

Basically, a mechanism is constructed by connecting rigid bodies, called ***links***, together by means of ***joints***, so that relative motion between adjacent links becomes possible. ***Actuation*** of the *joints*, typically by electric motors, then causes the robot to move and exert forces in desired ways.

The links of a robot mechanism can be arranged in serial fashion, like the familiar *open-chain* arm shown in Figure 1.1(a). Robot mechanisms can also have links that form *closed loops*, such as the *Stewart–Gough* platform shown in Figure 1.1(b). In the case of an open chain, all the joints are actuated, while in the case of mechanisms with closed loops, only a subset of the joints may be actuated.



*Figure 1.1: Open-chain and closed-chain robot mechanisms.*

# 2 - Configuration Space

A robot is mechanically constructed by connecting a set of bodies, called links, to each other using various types of **joints**. **Actuators**, such as electric motors, deliver forces or torques that cause the robot’s links to move. Usually an ***end-effector***, such as a gripper or hand for grasping and manipulating objects, is attached to a specific link. All the robots considered in this book have links that can be modeled as rigid bodies.

Perhaps the most fundamental question one can ask about a robot is, where is it? The answer is given by the robot’s ***configuration***: a specification of the positions of all points of the robot. Since the robot’s links are rigid and of a known shape,1 only a few numbers are needed to represent its configuration.

The number of ***degrees of freedom (dof)*** of a robot is the smallest number of real-valued coordinates needed to represent its configuration. In the example above, the door has one degree of freedom.

The n-dimensional space containing all possible configurations of the robot is called the **configuration space** (**C-space**). The configuration of a robot is represented by a point in its C-space.

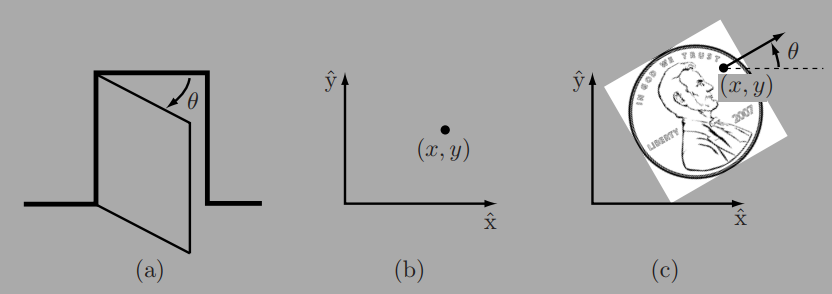
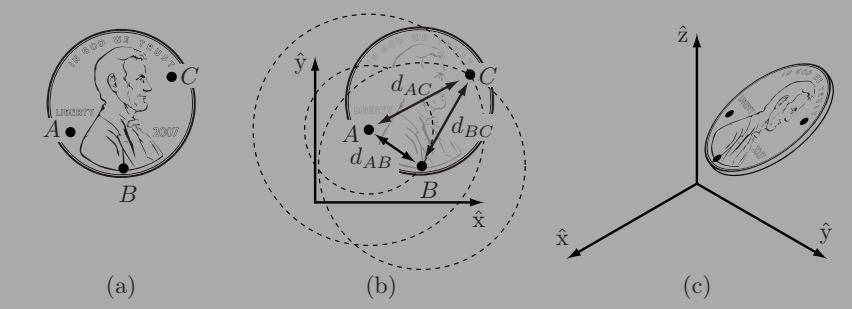


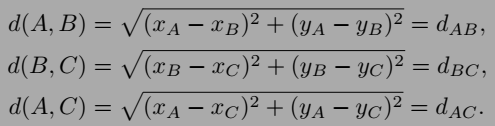
Figure 2.1: (a) The configuration of a door is described by the angle ✓. (b) The configuration of a point in a plane is described by coordinates (x, y). (c) The configuration of a coin on a table is described by (x, y, ✓), where ✓ defines the direction in which Abraham Lincoln is looking.

## 2.1 Degrees of Freedom of a Rigid Body

******

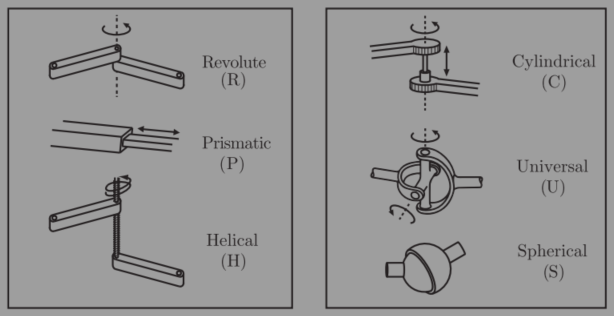
according to the definition of a rigid body, the distance between point A and point B, denoted *d(A, B)*, is always constant regardless of where the coin is. Similarly, the distances *d(B, C)* and *d(A, C)* must be constant.

The following equality constraints on the coordinates *(xA, yA)*, *(xB, yB)*, and *(xC, yC)* must therefore always be satisfied:



We have just established that a rigid body moving in three-dimensional space, which we call a **spatial rigid body**, has six degrees of freedom. Similarly, a rigid body moving in a two-dimensional plane, which we henceforth call a **planar rigid body**, has three degrees of freedom. This latter result can also be obtained by considering the planar rigid body to be a spatial rigid body with six degrees of freedom but with the three independent constraints *zA* = *zB* = zC = 0.

## 2.2 Degrees of Freedom of a Robot



*Figure 2.3: Typical robot joints.*

**2.2.1 Robot Joints**

Figure 2.3 illustrates the basic joints found in typical robots. Every joint connects exactly two links; joints that simultaneously connect three or more links are not allowed. The ***revolute joint*** (R), also called a hinge joint, allows rotational motion about the joint axis. The ***prismatic joint*** (P), also called a sliding or linear joint, allows translational (or rectilinear) motion along the direction of the joint axis. The ***helical joint*** (H), also called a screw joint, allows simultaneous rotation and translation about a screw axis. Revolute, prismatic, and helical joints all have one degree of freedom.

Joints can also have multiple degrees of freedom. The **cylindrical joint** (C) has two degrees of freedom and allows independent translations and rotations about a single fixed joint axis. The **universal joint** (U) is another two-degreeof-freedom joint that consists of a pair of revolute joints arranged so that their joint axes are orthogonal. The **spherical joint** (S), also called a ball-and-socket joint, has three degrees of freedom and functions much like our shoulder joint.

A joint can be viewed as providing freedoms to allow one rigid body to move relative to another. It can also be viewed as providing constraints on the possible motions of the two rigid bodies it connects. For example, a revolutejoint can be viewed as allowing one freedom of motion between two rigid bodies in space, or it can be viewed as providing five constraints on the motion of one rigid body relative to the other.

**2.2.2 Grubler’s Formula**

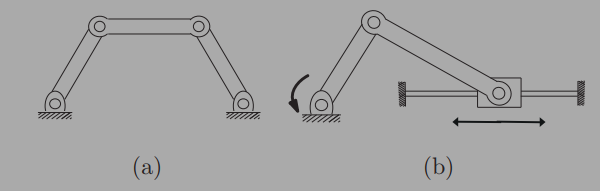
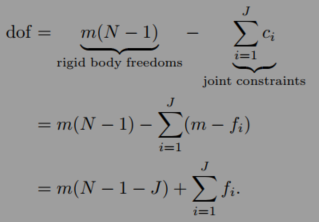
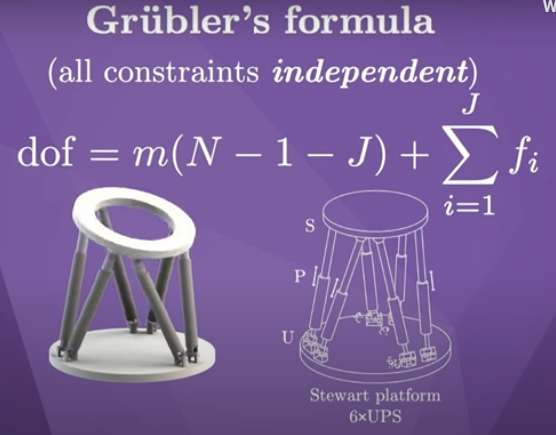


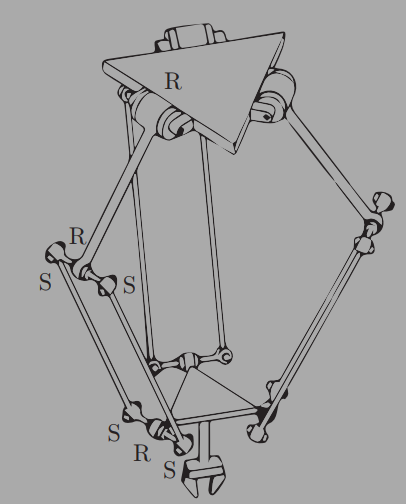
Figure 2.4: (a) Four-bar linkage. (b) Slider–crank mechanism.

*Grubler’s formula for the number of degrees of freedom of the robot is*



This formula holds only if all joint constraints are independent.





Example 2.7 (Delta robot). The Delta robot of Figure 2.8 consists of two platforms – the lower one mobile, the upper one stationary – connected by three legs. Each leg contains a parallelogram closed chain and consists of three revolute joints, four spherical joints, and five links. Adding the two platforms, there are N = 17 links and J = 21 joints (nine revolute and 12 spherical). By Grubler’s formula,

dof = 6(17 - 1 - 21) + 9(1) + 12(3) = 15.

Of these 15 degrees of freedom, however, only three are visible at the end-efector on the moving platform. In fact, the parallelogram leg design ensures that the moving platform always remains parallel to the fixed platform, so that the Delta robot acts as an x–y–z Cartesian positioning device. The other 12 internal degrees of freedom are accounted for by torsion of the 12 links in the parallelograms

## 2.3 Configuration Space: Topology and Representation

### 2.3.1 Configuration Space Topology

Until now we have been focusing on one important aspect of a robot’s C-space – its dimension, or the number of degrees of freedom. However, the shape of the space is also important.

Consider a point moving on the surface of a sphere. The point’s C-space is two dimensional, as the configuration can be described by two coordinates, latitude and longitude. As another example, a point moving on a plane also has a two-dimensional C-space, with coordinates (x, y). While both a plane and the surface of a sphere are two dimensional, clearly they do not have the same shape – the plane extends infinitely while the sphere wraps around

we say that two spaces are **topologically equivalent** if one can be continuously deformed into the other without cutting or gluing.

Topologically distinct one-dimensional spaces include the circle, the line, and a closed interval of the line. The circle is written mathematically as ***S*** or ***S1***, a one-dimensional “*sphere*.” The line can be written as **E** or **E1**, indicating a one-dimensional *Euclidean (or “flat”) space*. Since a ***point*** in E1 is usually represented by a real number (after choosing an origin and a length scale), it is often written as ***R*** or ***R1*** instead. A closed interval of the line, which contains its endpoints, can be written **[a, b] ⇢ R1**.

S – Sphere

E – Euclidean/flat space

R – point (Real)

In higher dimensions, Rn is the n-dimensional Euclidean space and Sn is the n-dimensional surface of a sphere in (n + 1)-dimensional space. For example, S2 is the two-dimensional surface of a sphere in three-dimensional space.

Some C-spaces can be expressed as the Cartesian product of two or more spaces of lower dimension; that is, points in such a C-space can be represented as the union of the representations of points in the lower-dimensional spaces.

For example:

* The C-space of a rigid body in the plane can be written as R2 x S1, since the configuration can be represented as the concatenation of the coordinates (x, y) representing R2 and an angle ✓ representing S1.
* The C-space of a PR robot arm can be written R1 ⇥ S1. (We will occasionally ignore joint limits, i.e., bounds on the travel of the joints, when expressing the topology of the C-space; with joint limits, the C-space is the Cartesian product of two closed intervals of the line.)
* The C-space of a 2R robot arm can be written S1 x S1 = T2, where Tn is the n-dimensional surface of a torus in an (n + 1)-dimensional space.
* As we saw in Section 2.1 when we counted the degrees of freedom of a rigid body in three dimensions, the configuration of a rigid body can be described by a point in R3, plus a point on a two-dimensional sphere S2, plus a point on a one-dimensional circle S1, giving a total C-space of R3 x S2 x S1.

### 2.3.2 Configuration Space Representation

To perform computations, we must have a numerical **representation** of the space, consisting of a set of real numbers. We are familiar with this idea from linear algebra – a vector is a natural way to represent a point in a Euclidean space.

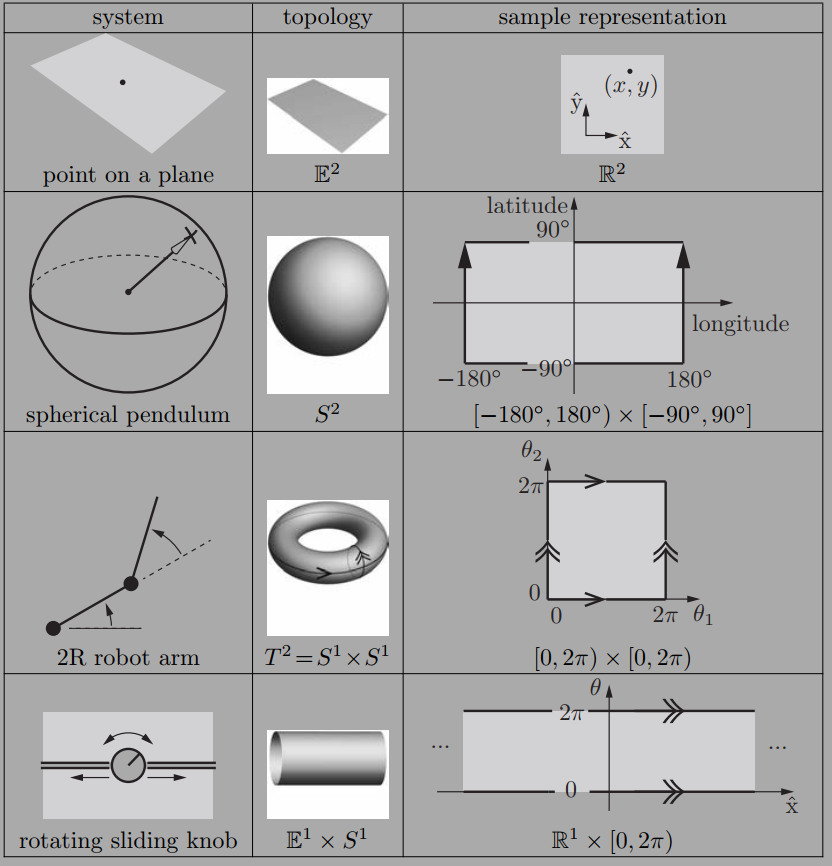


Table 2.2: Four topologically different two-dimensional C-spaces and example coordinate representations. In the latitude-longitude representation of the sphere, the latitudes -90 ° and 90 ° each correspond to a single point (the South Pole and the North Pole, respectively), and the longitude parameter wraps around at 180 ° and -180 °; the edges with the arrows are glued together. Similarly, the coordinate representations of the torus and cylinder wrap around at the edges marked with corresponding arrows.

While it is natural to choose a reference frame and length scale and to use a vector to represent points in a Euclidean space, representing a point on a curved space, such as a sphere, is less obvious. One solution for a sphere is to use latitude and longitude coordinates. A choice of n coordinates, or parameters, to represent an n-dimensional space is called an **explicit parametrization** of the space. Such an explicit parametrization is valid for a particular range of the parameters (e.g., [-90 °, 90 °] for latitude and [-180 °, 180**"**) for longitude for a sphere, where, on Earth, negative values correspond to “south” and “west,” respectively).

The latitude–longitude representation of a sphere is unsatisfactory if you are walking near the North Pole (where the latitude equals 90 °) or South Pole (where the latitude equals -90°), where taking a very small step can result in a large change in the coordinates. The North and South Poles are singularities of the representation,

If you can assume that the configuration never approaches a singularity of the representation, you can ignore this issue. If you cannot make this assumption, there are two ways to overcome the problem.

* Use more than one coordinate chart on the space,
* Use an ***implicit representation*** instead of an explicit parametrization. An implicit representation views the n-dimensional space as embedded in a Euclidean space of more than n dimensions, just as a two-dimensional unit sphere can be viewed as a surface embedded in a three-dimensional Euclidean space. An implicit representation uses the coordinates of the higher-dimensional space (e.g., (x, y, z) in the three-dimensional space), but subjects these coordinates to constraints that reduce the number of degrees of freedom (e.g., x2 + y2 + z2 = 1 for the unit sphere).

A disadvantage of this approach is that the representation has more numbers than the number of degrees of freedom. An advantage is that there are no singularities in the representation – a point moving smoothly around the sphere is represented by a smoothly changing (x, y, z), even at the North and South poles.

We will use implicit representations throughout the book, beginning in the next chapter. In particular, we use nine numbers, subject to six constraints, to represent the three orientation freedoms of a rigid body in space. This is called a ***rotation matrix***. In addition to being singularity-free, the rotation matrix representation allows us to use linear algebra to perform computations.

## 2.4 Configuration and Velocity Constraints

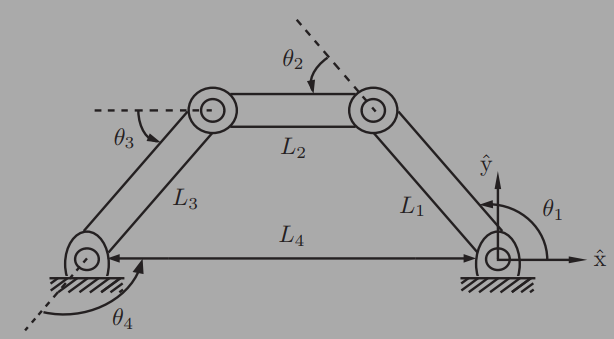
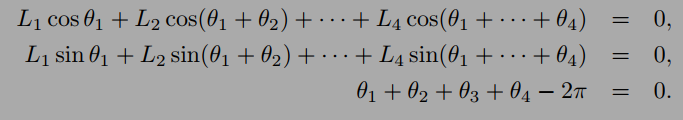


Figure 2.10: The four-bar linkage.



These equations are sometimes referred to as **loop-closure equations**. For the four-bar linkage they are given by a set of three equations in four unknowns.

### Explanation from AI:

The formulas represent a closed-loop robotic mechanism, where the sum of the position vectors of the links must equal zero, ensuring the loop closes.

**Why Cosine and Sine?**

The use of cosine and sine stems from the need to resolve the position of each link in the robot arm into its horizontal (x) and vertical (y) components.

Imagine a single link of length Lᵢ at an angle θᵢ with respect to the horizontal axis.

The horizontal component of this link's position is given by Lᵢcos θᵢ.

The vertical component is given by Lᵢsin θᵢ.

When you have multiple links connected in a chain (as in a robot arm), the position of the end of the arm is the sum of the horizontal and vertical components of each link. This is why you see the summation in the equations.

In essence, these equations are performing a coordinate transformation from the joint space (angles θᵢ) to the Cartesian space (x, y position of the end-effector).

These equations are used in both forward kinematics and inverse kinematics:

* **Forward Kinematics**: Given the joint angles (θᵢ), calculate the position of the end-effector. This is a relatively straightforward calculation using the equations above.
* **Inverse Kinematics**: Given the desired position of the end-effector, calculate the required joint angles (θᵢ) to reach that position. This is generally more complex, often involving iterative numerical methods or analytical solutions for simpler robots.

**Understanding the Formulas**

The equations you've provided are fundamental in robotics, specifically in the realm of kinematics, which deals with the motion of objects without considering the forces causing that motion. In robotics, kinematics is crucial for understanding how a robot's joints and links move to achieve a desired position and orientation of its end-effector (the "hand" or tool at the end of the robot arm).

**L1 cos θ1**

L1 cos θ1 represents the x-coordinate contribution of the first robot arm segment, where:

* L1 is the length of the first arm segment
* θ1 is the angle it makes with the horizontal
* cos θ1 projects this length onto the x-axis

For example, if L1 = 10 units and θ1 = 60°:

L1 cos θ1 = 10 × cos(60°) = 5 units

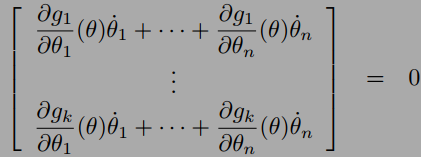
This means the first segment contributes 5 units in the x-direction.

### End of AI Explanation

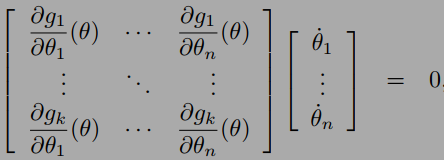
Suppose that a closed-chain robot with loop-closure equations g(θ) = 0, g : Rn -> Rk, is in motion, following the time trajectory θ(t). Differentiating both sides of g(θ(t)) = 0 with respect to t, we obtain



thus



This can be expressed as a matrix multiplying a column vector



### Explanation from AI:

This set of equations describes the velocity-level kinematics of a closed-chain robot. It's derived from the position-level kinematics (like the ones we discussed earlier with sines and cosines) by taking the time derivative. This gives us a relationship between the joint velocities and the end-effector velocities.

**Significance and Use:**

* **Velocity Analysis**: This equation is used to analyze the velocities of the robot's links and joints. Given a set of joint velocities (θ̇), you can use the Jacobian to determine if these velocities are consistent with the loop constraints.
* **Singularity Analysis**: The Jacobian matrix is also essential for identifying singularities in the robot's configuration. A singularity occurs when the Jacobian loses rank, meaning that certain directions of motion become impossible.
* **Control**: In robot control, this equation can be used to determine the necessary joint velocities to achieve a desired end-effector velocity while respecting the loop constraints.
* **In summary**: These equations provide a way to analyze and control the motion of closed-chain robots at the velocity level. The Jacobian matrix plays a central role in relating joint velocities to the constraints imposed by the closed loops in the robot's

**Why do derivatives  allow us to analyze and control the motion?**

This is a great question that gets at the heart of why calculus is so important in fields like robotics, physics, and engineering. Here's a breakdown of how derivatives empower us to analyze and control motion:

*1. Capturing Instantaneous Change*

* **Average vs. Instantaneous**: Imagine a car traveling 100 miles in 2 hours. Its average speed is 50 mph. However, this doesn't tell us the car's speed at any specific moment. It might have been going faster or slower at different times.
* **The Power of Derivatives**: Derivatives give us the instantaneous rate of change. In the car example, the derivative of the car's position with respect to time would give us its exact speed at any given instant. This is crucial for understanding motion because motion is inherently about change at every moment.

*2. Linking Position, Velocity, and Acceleration*

* A Chain of Derivatives: Derivatives create a direct link between fundamental motion quantities:
  + The derivative of an object's position with respect to time is its velocity.
  + The derivative of an object's velocity with respect to time is its acceleration.
* This chain is essential for both analysis and control. If we know the position of a robot at all times, we can use derivatives to determine its velocity and acceleration. Conversely, if we want to control the robot's acceleration (by applying forces), we can use integration (the opposite of differentiation) to figure out how its velocity and position will change.

*3. Predicting Future Motion*

* **Extrapolation**: Because derivatives tell us how something is changing right now, we can use them to predict how it will change in the near future. This is the basis of many control algorithms.
* **Example**: If we know a robot's current position, velocity, and acceleration, we can use this information (along with physical laws) to predict where it will be a short time later. This allows us to make control adjustments to keep it on the desired path.

*4. Analyzing Complex Motion*

* **Multivariable Systems**: In robotics, motion often involves multiple variables (e.g., the angles of many joints). Partial derivatives (which are part of the Jacobian matrix we discussed earlier) allow us to analyze how changes in each variable affect the overall motion.
* **Constraints**: As we've seen, derivatives help us analyze how motion is affected by constraints (like the closed loops in a robot arm). By taking derivatives of the constraint equations, we get relationships that velocities and accelerations must satisfy.

### End of AI Explanation

The constraints (2.7) can be written



where A(θ) ϵ Rkn. Velocity constraints of this form are called **Pfaffian constraints**.

holonomic constraints of the form g(θ) = 0 are also **called integrable constraints** – the velocity constraints that they imply can be integrated to give equivalent configuration constraints.

## 2.5 Task Space and Workspace

The **task space** is a space in which the robot’s task can be naturally expressed. For example, if the robot’s task is to plot with a pen on a piece of paper, the task space would be R2.

The **workspace** is a specification of the configurations that the end-effector of the robot can reach. The definition of the workspace is primarily driven by the robot’s structure, independently of the task.

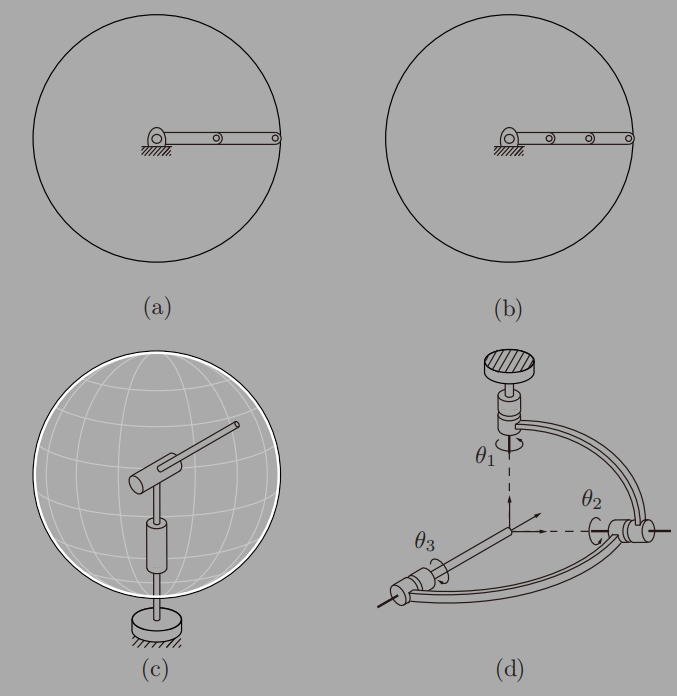


Figure 2.12: Examples of workspaces for various robots: (a) a planar 2R open chain; (b) a planar 3R open chain; (c) a spherical 2R open chain; (d) a 3R orienting mechanism.

# 3 - Rigid-Body Motions

## 3.1 Rigid-Body Motions in the Plane

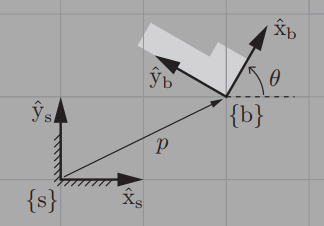


Figure 3.3: The body frame {b} is expressed in the fixed-frame coordinates {s} by the vector p and the directions of the unit axes x^b and y^b. In this example, p = (2, 1) and θ = 60 °, so x^b = (cos θ, sin θ) = (0.5, 1/ Ö2) and y^b = (- sin θ, cos θ) = (-1/ Ö2, 0.5)

To describe the configuration of the planar body, only the position and orientation of the body frame with respect to the fixed frame need to be specified. The body-frame origin p can be expressed in terms of the coordinate axes of {s} as



## 3.2 Rotations and Angular Velocities

### 3.2.1 Rotation Matrices

### 3.2.2 Angular Velocities

### 3.2.3 Exponential Coordinate Representation of Rotation

## 3.3 Rigid-Body Motions and Twists

3.3.1 Homogeneous Transformation Matrices

3.3.2 Twists

3.3.3 Exponential Coordinate Representation of Rigid-Body Motions

## 3.4 Wrenches

## 3.5 Summary

## 3.6 Software

## 3.7 Notes and References

# 4 - Forward Kinematics

## 4.1 Product of Exponentials Formula

4.1.1 First Formulation: Screw Axes in the Base Frame

4.1.2 Examples

4.1.3 Second Formulation: Screw Axes in the End-E↵ector Frame

## 4.2 The Universal Robot Description Format

## 4.3 Summary

## 4.4 Software

## 4.5 Notes and References

## 4.6 Exercises

# 5 - Velocity Kinematics and Statics

## 5.1 Manipulator Jacobian

5.1.1 Space Jacobian

5.1.2 Body Jacobian

5.1.3 Visualizing the Space and Body Jacobian

5.1.4 Relationship between the Space and Body Jacobian

5.1.5 Alternative Notions of the Jacobian

5.1.6 Looking Ahead to Inverse Velocity Kinematics

## 5.2 Statics of Open Chains

## 5.3 Singularity Analysis

## 5.4 Manipulability

## 5.5 Summary

## 5.6 Software

## 5.7 Notes and References

## 5.8 Exercises

# 6 - Inverse Kinematics

## 6.1 Analytic Inverse Kinematics

6.1.1 6R PUMA-Type Arm

6.1.2 Stanford-Type Arms

## 6.2 Numerical Inverse Kinematics

6.2.1 Newton–Raphson Method

6.2.2 Numerical Inverse Kinematics Algorithm

## 6.3 Inverse Velocity Kinematics

## 6.4 A Note on Closed Loops

## 6.5 Summary

## 6.6 Software

## 6.7 Notes and References

## 6.8 Exercises

# 7 - Kinematics of Closed Chains

## 7.1 Inverse and Forward Kinematics

7.1.1 3xRPR Planar Parallel Mechanism

7.1.2 Stewart–Gough Platform

7.1.3 General Parallel Mechanisms

## 7.2 Diferential Kinematics

7.2.1 Stewart–Gough Platform

7.2.2 General Parallel Mechanisms

## 7.3 Singularities

## 7.4 Summary

## 7.5 Notes and References

## 7.6 Exercises

# 8 - Dynamics of Open Chains

## 8.1 Lagrangian Formulation

8.1.1 Basic Concepts and Motivating Examples

8.1.2 General Formulation

8.1.3 Understanding the Mass Matrix

8.1.4 Lagrangian Dynamics vs. Newton–Euler Dynamics

## 8.2 Dynamics of a Single Rigid Body

8.2.1 Classical Formulation

8.2.2 Twist–Wrench Formulation

8.2.3 Dynamics in Other Frames

## 8.3 Newton–Euler Inverse Dynamics

8.3.1 Derivation

8.3.2 Newton-Euler Inverse Dynamics Algorithm

## 8.4 Dynamic Equations in Closed Form

## 8.5 Forward Dynamics of Open Chains

## 8.6 Dynamics in the Task Space

## 8.7 Constrained Dynamics

## 8.8 Robot Dynamics in the URDF

## 8.9 Actuation, Gearing, and Friction

8.9.1 DC Motors and Gearing

8.9.2 Apparent Inertia

8.9.3 Newton–Euler Inverse Dynamics Algorithm Accounting for Motor Inertias and Gearing

8.9.4 Friction

8.9.5 Joint and Link Flexibility

## 8.10 Summary

## 8.11 Software

## 8.12 Notes and References

## 8.13 Exercises

# 9 - Trajectory Generation

## 9.1 Definitions

## 9.2 Point-to-Point Trajectories

9.2.1 Straight-Line Paths

9.2.2 Time Scaling a Straight-Line Path

## 9.3 Polynomial Via Point Trajectories

## 9.4 Time-Optimal Time Scaling

9.4.1 The (s, s˙) Phase Plane

9.4.2 The Time-Scaling Algorithm

9.4.3 A Variation on the Time-Scaling Algorithm

9.4.4 Assumptions and Caveats

## 9.5 Summary

## 9.6 Software

## 9.7 Notes and References

## 9.8 Exercises

# 10 - Motion Planning

## 10.1 Overview of Motion Planning

10.1.1 Types of Motion Planning Problems

10.1.2 Properties of Motion Planners

10.1.3 Motion Planning Methods

## 10.2 Foundations

10.2.1 Configuration Space Obstacles

10.2.2 Distance to Obstacles and Collision Detection

10.2.3 Graphs and Trees

10.2.4 Graph Search

## 10.3 Complete Path Planners

## 10.4 Grid Methods

10.4.1 Multi-Resolution Grid Representation

10.4.2 Grid Methods with Motion Constraints

## 10.5 Sampling Methods

10.5.1 The RRT Algorithm

10.5.2 The PRM Algorithm

## 10.6 Virtual Potential Fields

10.6.1 A Point in C-space

10.6.2 Navigation Functions

10.6.3 Workspace Potential

10.6.4 Wheeled Mobile Robots

10.6.5 Use of Potential Fields in Planners

## 10.7 Nonlinear Optimization

## 10.8 Smoothing

## 10.9 Summary

## 10.10 Notes and References

## 10.11 Exercises

# 11 - Robot Control

## 11.1 Control System Overview

## 11.2 Error Dynamics

11.2.1 Error Response

11.2.2 Linear Error Dynamics

## 11.3 Motion Control with Velocity Inputs

11.3.1 Motion Control of a Single Joint

11.3.2 Motion Control of a Multi-joint Robot

11.3.3 Task-Space Motion Control

## 11.4 Motion Control with Torque or Force Inputs

11.4.1 Motion Control of a Single Joint

11.4.2 Motion Control of a Multi-joint Robot

11.4.3 Task-Space Motion Control

## 11.5 Force Control

## 11.6 Hybrid Motion–Force Control

11.6.1 Natural and Artificial Constraints

11.6.2 A Hybrid Motion–Force Controller

## 11.7 Impedance Control

11.7.1 Impedance-Control Algorithm

11.7.2 Admittance-Control Algorithm

## 11.8 Low-Level Joint Force/Torque Control

## 11.9 Other Topics

## 11.10 Summary

## 11.11 Software

## 11.12 Notes and References

## 11.13 Exercises

# 12 - Grasping and Manipulation

## 12.1 Contact Kinematics

12.1.1 First-Order Analysis of a Single Contact

12.1.2 Contact Types: Rolling, Sliding, and Breaking Free

12.1.3 Multiple Contacts

12.1.4 Collections of Bodies

12.1.5 Other Types of Contacts

12.1.6 Planar Graphical Methods

12.1.7 Form Closure

## 12.2 Contact Forces and Friction

12.2.1 Friction

12.2.2 Planar Graphical Methods

12.2.3 Force Closure

12.2.4 Duality of Force and Motion Freedoms

## 12.3 Manipulation

## 12.4 Summary

## 12.5 Notes and References

## 12.6 Exercises

# 13 - Wheeled Mobile Robots

## 13.1 Types of Wheeled Mobile Robots

## 13.2 Omnidirectional Wheeled Mobile Robots

13.2.1 Modeling

13.2.2 Motion Planning

13.2.3 Feedback Control

## 13.3 Nonholonomic Wheeled Mobile Robots

13.3.1 Modeling

13.3.2 Controllability

13.3.3 Motion Planning

13.3.4 Feedback Control

## 13.4 Odometry

## 13.5 Mobile Manipulation

## 13.6 Summary

## 13.7 Notes and References

## 13.8 Exercises

# A - Summary of Useful Formulas

# B - Other Representations of Rotations

## B.1 Euler Angles

B.1.1 Algorithm for Computing the ZYX Euler Angles

B.1.2 Other Euler Angle Representations

## B.2 Roll–Pitch–Yaw Angles

## B.3 Unit Quaternions

## B.4 Cayley–Rodrigues Parameters

# C - Denavit–Hartenberg Parameters

## C.1 Assigning Link Frames

## C.2 Why Four Parameters are Suficient

## C.3 Manipulator Forward Kinematics

## C.4 Examples

## C.5 Relation Between the PoE and D–H Representations

## C.6 A Final Comparison

# D - Optimization and Lagrange Multipliers