Chapter 8
How to Move? Control, Navigation and Path Planning for Mobile Robots

Jiefei Wang and Damith Herath

8.1 Learning Objectives

You will learn about:

- *Controllers and control techniques used in robotics, including the PID controller*
- *Mobile robot locomotion types*
- *Robot path planning and obstacle avoidance.*

8.2 Introduction

When we think of robots, we think of them as manipulators, such as in manufacturing facilities where they are fixed to a location or robots that are moving about (Fig. 8.1). Robots that move around in the environment are called *mobile robots*. This chapter looks at mobile robots, how to control them, different locomotion types and algorithms used for planning paths, and obstacle avoidance while navigating.

J. Wang (✉)
The School of Engineering and Information Technology, University of New South Wales, Canberra, Australia
e-mail: Jiefei.wang@adfa.edu.au

D. Herath
Collaborative Robotics Lab, University of Canberra, Canberra, Australia
e-mail: Damith.Herath@Canberra.edu.au

D. Herath and D. St-Onge (eds.), *Foundations of Robotics*,
https://doi.org/10.1007/978-981-19-1983-1_8

Fig. 8.1 A Kinova Gen3 lite robot arm mounted on a Clearpath Dingo Indoor mobile robotic platform (left) alongside a Jackal Unmanned Ground Vehicle used for outdoor navigation. (right) (Credits Clearpath/Kinova)

An Industry Perspective

Dana Leslie
Former Clearpath Robotics' Employee

Like many young engineers, I have my parents to thank for enabling me to explore the world through robotics. The enjoyment of playing with lego, electronics kits, and computer programming at a young age, was undoubtedly the catalyst that resulted in my career trajectory.

After studying electrical engineering at the University of Victoria, I was fortunate to get a start in the industry by landing my first job at Cellula Robotics, a subsea robotics company. It was here that our team designed, manufactured, and deployed robots to the darkest depths of the ocean, studying and learning about the undersea world!

From water onto land, the robots I've helped design continued to evolve; developing wheeled terrestrial systems at Clearpath Robotics in Ontario, and most recently legged humanoids at Agility Robotics in Oregon.

During the design of a mobile robot, diodes were incorporated into the power system to enable battery hot-swapping. Consequently, the energy generated by back-EMF from the motors (while braking or being pushed) could not be absorbed by the battery. The result was an uncontrolled increase in voltage, causing various subsystems to glitch, with the robot lifelessly rolling to a halt…

This type of challenge is trivial to conceptualise, but much harder to quantify. It's only apparent in a fully integrated system, is correlated to things outside of your control, and is intensified when carrying heavy payloads or traveling down ramps. (Increased mechanical to electrical energy conversion.)

In the end, through comprehensive and iterative testing, the solution was a combination of reducing deceleration rates, varying system capacitance, and utilising transient-voltage-suppression diodes.

It's nice to be able to power your robot by giving it a push, but it's critical that your robot behaves when it's in a hurry to stop.

Innovation in embedded sensing, processing, power electronics, and battery chemistries have collectively advanced the robotics industry throughout my career.

Precise and energy-dense servo-actuators have recently enabled cutting-edge humanoid robotic development that is poised to redefine the workforce; automating the dullest and dangerous of human tasks.

These same actuators have advanced robotic manipulation, enabling the technology to emerge from the factory line and onto the front lines. Robotic arms are no longer just being used to assemble cars, they're being used to flip hamburgers and pack your groceries!

8.3 Mobile Robots

Mobile robots have received much attention in the last few decades due to their ability to explore complex environments such as space, rescue operations, and accomplish tasks autonomously without human effort. Mobile robots can be broadly categorised as wheeled, legged, and flying robots.

8.3.1 Wheeled Robots

Wheeled robots traverse around the ground using motorised wheels to propel themselves and a comparatively easier to design, build, and operate for movement in flat or rocky terrain than robots that use legs or wings. They are also better controlled as they have fewer degrees of freedom than flying robots. One of the challenges of wheeled robots is that they cannot operate well over certain ground surfaces, such as sharp declines, rugged terrain, or areas with low friction. Nevertheless, wheeled robots are the most popular in the consumer market due to the low cost and simplicity of differential steering mechanisms they employ. Although wheeled robots can have any number of wheels, the mechanisms need to be modified to keep dynamic balance based on the number of wheels. Three or four wheels are the most popular and sufficient for static and dynamic balance among all wheeled robots, which are widely used in research projects.

8.3.1.1 Kinematic Modelling

This book primarily discusses two types of robots and their motions, mobile robots and arm type robots. In either type, we need to understand how the movements generated by the actuators translate into complex body movements. To design a robot to act in the environment, we need to understand these geometric relationships of motion.

Kinematics is the study of motions of points, bodies, and systems of bodies (such as robots) without considering the forces acting on these systems. In this chapter, we will discuss some common wheel configurations and their respective kinematic models used in mobile robots that use motors to drive them around. Then, Chap. 10 will delve into modelling kinematics of arm-type robots.

8.3.1.2 Holonomic Drive

Holonomic refers to the relationship between controllable and total degrees of freedom of a robot. If the controllable degree of freedom is equal to the total degrees of freedom, then the robot is said to be Holonomic. A robot built on castor wheels or omniwheels is a good example of a holonomic drive. It can freely move in any direction, and the controllable degrees of freedom is equal to total degrees of freedom.

If the controllable degree of freedom is less than the total degrees of freedom, it is known as *non-holonomic* drive. For example, a car has three degrees of freedom: its position in two axes and orientation. However, there are only two controllable degrees of freedom: acceleration (or braking) and the turning angle of the steering wheel. This makes it difficult for the driver to turn the car in any direction (unless it skids or slides).

For a typical differential drive robot (see Fig. 8.4), the non-holonomic constraint could be written as:

$$\dot{x} \sin \phi - \dot{y} \cos \phi = 0$$

8.3.1.3 Three-Wheeled Robots

One of the most common actuator configurations to drive a mobile robot is the three-wheeled configuration (also known as the tricycle model).

There are two types of three-wheeled robots:

- Differentially steered—two separately powered wheels with an extra free rotating wheel. The robot direction can be changed by varying the relative rate of rotation of the two separately driven wheels. If both the wheels are driven in the same direction and speed, the robot will go straight. Otherwise, depending on the speed of rotation and its direction (Fig. 8.2).
- Two wheels powered by a single actuator and a powered steering wheel.

The centre of gravity in this type of robot has to lay inside the triangle formed by the wheels. If too much weight is allocated to the side of the free rotating wheel, it will cause an imbalance that could make the robot tip over.

Let us now explore how a differentially steered three-wheeled robot could be modelled kinematically.

The model presented in Fig. 8.3 introduces a virtual wheel for the front set of differential drive wheels. The two wheels along the centreline of the robot essentially represent the whole system. With the said constraints, the robot can only exercise two degrees of freedom. Thus, the derivation of the kinematic model refers to the robot's simplified model. The *instantaneous centre of rotation* (also known as the instantaneous velocity centre) in this model refers to an imaginary point attached to the robot where at a given point in time has zero velocity while the rest of the robot body is in planar motion. You could imagine the robot to be rotating around this point at the time instance being considered.

It can be shown that the continuous time form of the vehicle model (with respect to the centre of the front wheel) can be derived as follows:

$$\begin{aligned}
\dot{x}(t) &= V(t) \cos(\phi(t) + \gamma(t)) \\
\dot{y}(t) &= V(t) \sin(\phi(t) + \gamma(t)) \\
\dot{\varphi}(t) &= \frac{V(t) \sin(\gamma(t))}{B}
\end{aligned}$$

where $x(t)$ and $y(t)$ denote the position of the vehicle, the angle $\phi(t)$ is the orientation of the robot with respect to the x-axis, and $V(t)$ represents the linear velocity of the

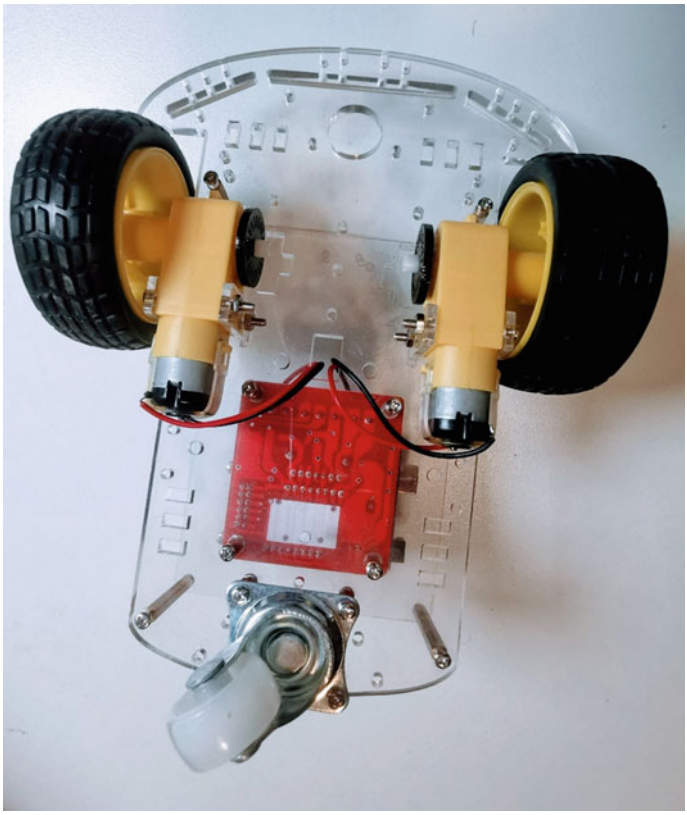

Fig. 8.2 Differentially steered three-wheeled robot. The front two wheels (top) are powered by two DC motors. A back castor wheel is free to rotate around and is not powered

front wheel. The angle γ is defined as the steer angle of the vehicle. B is the base length between the two sets of wheels.

A simpler kinematic model can be derived from the model discussed earlier in many simple robot configurations where the system makes the velocity of the robot $V(t)$ and the angular velocity of the robot $\dot{\phi}(t)$ directly available (e.g. via wheel encoders). Then the process model for the corresponding system can be represented as follows (Fig. 8.4):

Following simpler equations can be derived then:

$$\begin{aligned}\dot{x}(t) &= V(t)\cos(\phi(t)) \\ \dot{y}(t) &= V(t)\sin(\phi(t)) \\ \dot{\varphi}(t) &= \omega(t)\end{aligned}$$

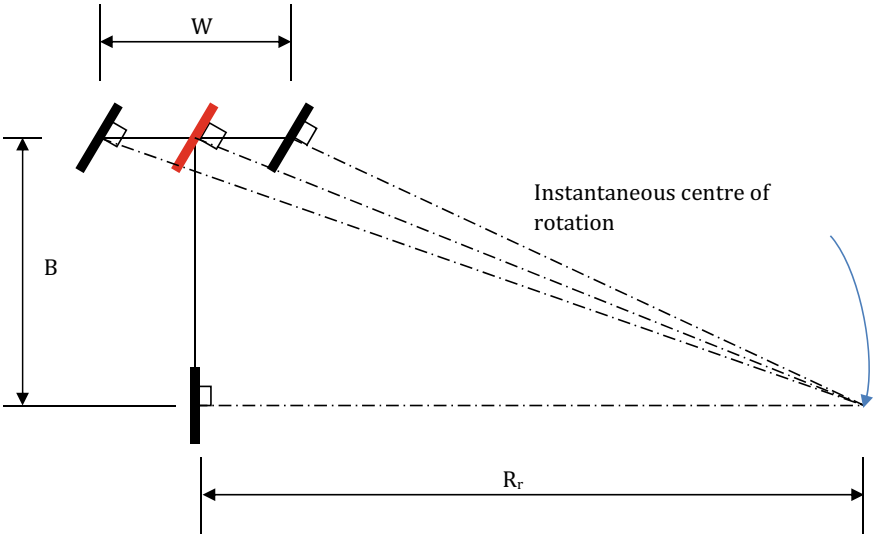

Fig. 8.3 Vehicle geometry of a typical three-wheeled robot

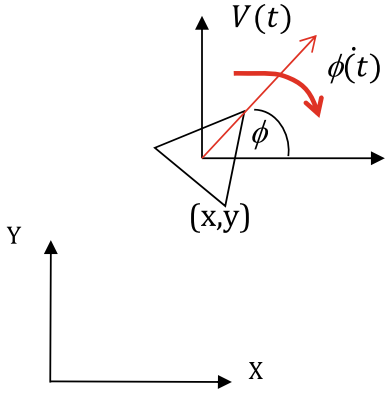

Fig. 8.4 Simplified robot model

8.3.1.4 Two-Wheeled Robots

Two-wheeled robots are harder to balance than other types because they must keep moving to maintain upright. The centre of gravity of the robot body is kept below the axle. Usually, this is accomplished by mounting the batteries below the body. They can have their wheels parallel to each other, and these vehicles are called dicycles, or one wheel in front of the other, tandemly placed wheels (bicycle). Two-wheeled robots must keep moving to remain upright, and they can do this by driving in the direction the robot is falling. To balance, the base of the robot must stay under its centre of gravity. For a robot that has left and right wheels, it needs at least two

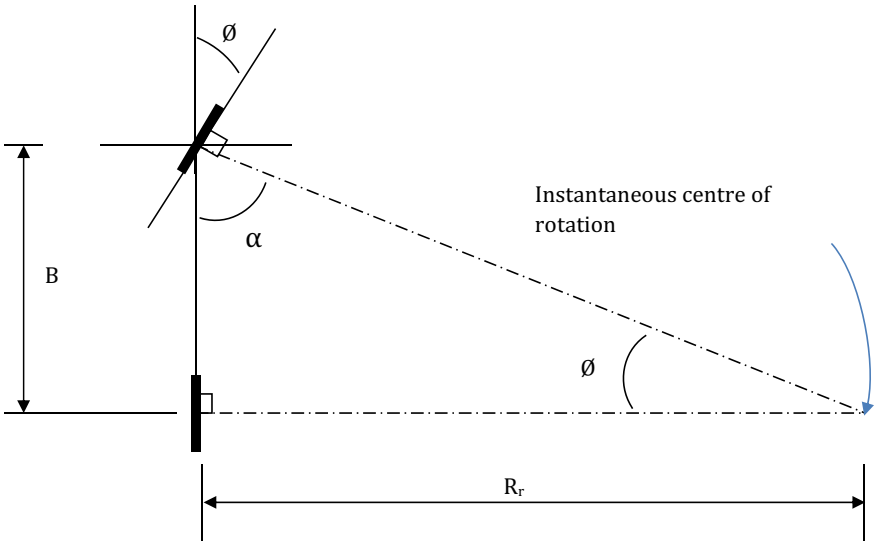

Fig. 8.5 Wheel configuraiton of a two-wheeled bicycle robot

sensors. A tilt sensor is used to determine tilt angle and wheel encoders that keep track of the position of the robot's platform (Fig. 8.5).

where $R_{rr} = \frac{B}{\tan \phi}$, $\alpha + \phi + 90^\circ = 180^\circ$.

8.3.1.5 Four-Wheeled Robots

There are several configurations possible with four wheels.

- *Two powered and two free rotating wheels*

Same as the differentially steered ones mentioned previously but with two free rotating wheels for extra balance.

Four-wheeled robots are more stable than three-wheeled ones as the centre of gravity has to remain inside the rectangle formed by the four wheels instead of a triangle. Still, it is advisable to keep the centre of gravity to the middle of the rectangle as this is the most stable configuration, especially when taking sharp turns or moving over a non-even surface.

- *Two-by-two powered wheels for tank-like movement*

This type of robot uses two pairs of powered wheels, and each pair turns in the same direction. The tricky part of this kind of propulsion is getting all the wheels to turn with the same speed. If the wheels in a pair are not running at the same speed, the

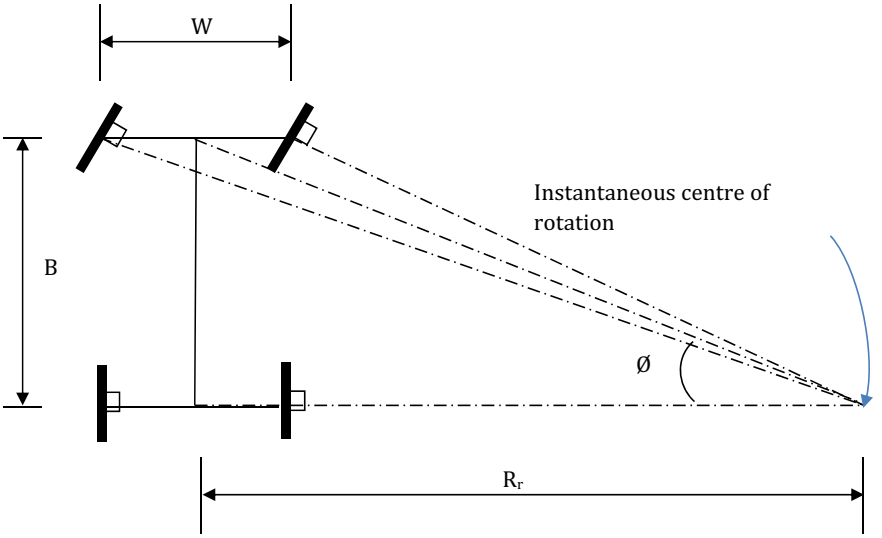

Fig. 8.6 Ackerman drive

slower one will slip. If the pairs do not run at the same speed, the robot is not able to drive straight. A good design has to incorporate some form of car-like steering.

- *Car-like steering (Ackerman drive)*

This method allows the robot to turn the same way a car does (Fig. 8.6). However, this system does have an advantage over previous methods where it only needs one motor to drive the rear wheels and a servo for steering. The previous methods would require either two motors or a highly complex gearbox since they require two output axles with independent speed and direction of rotation.

where $R_{rr} = \frac{B}{\tan \phi}$.

8.3.1.6 Omnidirectional Wheels

Omnidirectional (Omni) wheeled robots fall under a class of unconventional mobile robots (Fig. 8.7).

An omniwheel could be thought of as having many smaller wheels making up a large one, and the smaller ones are mounted at an angle to the axis of the core wheel. This allows the wheels to move in two directions and move holonomically, which means it can instantaneously move in any direction, unlike a car, which moves non-holonomicallly and has to be in motion to change heading. In addition, omniwheeled robots can move in at any angle in any direction without rotating beforehand. Some omniwheel robots use a triangular platform, with the three wheels spaced at 60-degree angles. The advantage of using omniwheels is that they make it easier for

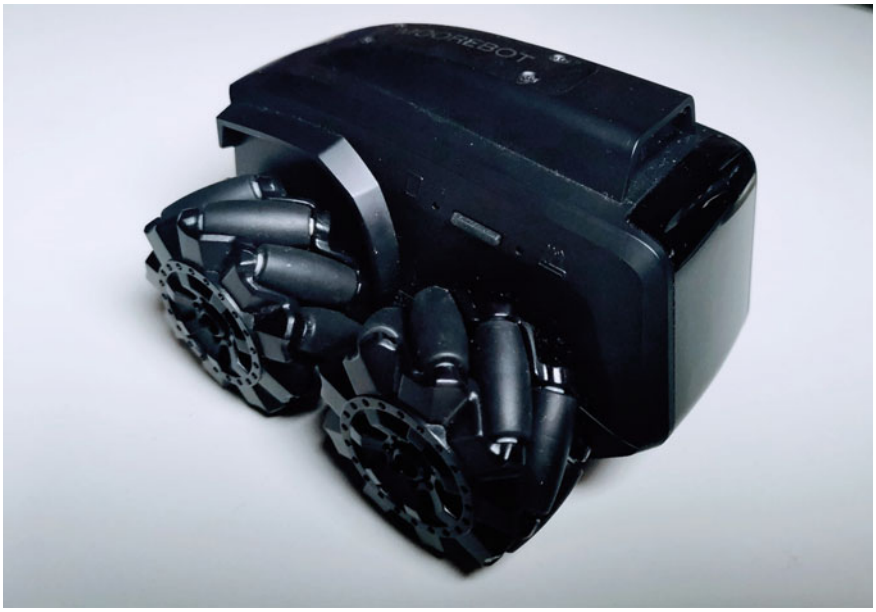

Fig. 8.7 A set of Mecanum wheels (a type of omniwheel) on a home robot

robots to be designed with wheels mounted on an unaligned axis. The disadvantage of using omniwheels is that they have poor efficiency due to not all the wheels rotating in the direction of movement, which also causes loss from friction, and are more computationally complex because of the angle calculations of movement.

8.3.2 Walking Robots

Legged robots are inspired by human beings, legged animals or insects which use leg mechanisms to provide locomotion. Compared with wheeled robots, they are more versatile. They can traverse extreme environments such as unstructured, uneven, unstable, rugged terrain and complex confined spaces such as underground environments and industrial structures.

Legged robots can be categorised by the number of limbs they use. Robots with more legs tend to be more stable, while fewer legs lend themselves to greater manoeuvrability. For a legged robot to keep its balance, it requires maintaining its centre of gravity within its polygon of stability. The polygon of stability is the horizontal surface defined by the leg-ground contact points made by the robot. These multidegrees of freedom legs are usually modelled as kinematics chains which is covered in Chap. 10.

8.3.2.1 Robot Gait

The periodic contact of the robot's legs with the ground is called the gait of the walker. The specific gait depends on the leg configuration of the robot and parameters such as the speed, terrain the robot is moving, intended task and power limitations of the robot. Milton Hildebrand was one of the earliest zoologists to study animal gaits. Various researchers have since adopted his method for gait-pattern specification in robotics, providing a formal method for studying and improving robot gait.

8.3.2.2 Two-Legged Robots

Two-legged robots are also called bipedal robots. The fundamental challenges for two-legged robots are stability and motion control, which refers to balance and movement control. In advanced systems, accelerometers or gyroscopes provide dynamic feedback to control the balance. Such sensors are also used for motion control, walking, jumping, and even running, combined with technologies such as machine learning. On the other hand, the *passive walker* is a bipedal mechanism that "walks" without actuation, simply using gravity as its energy source (Fig. 8.8).

Fig. 8.8 A bipedal robot

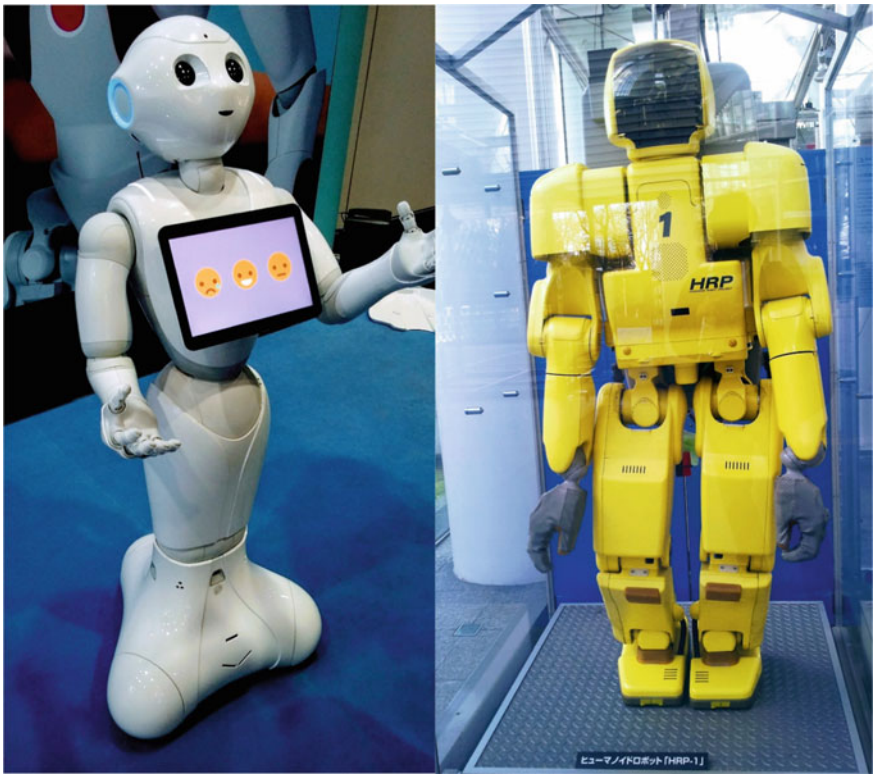

Fig. 8.9 Pepper robot (left)—a wheeled semi-humanoid robot used in retail marketing. The HRP-1 (right)—an early Humanoid Robot Prototype developed by the National Institute of Advanced Industrial Science and Technology (AIST), Japan, on public display at its premises

8.3.2.3 Humanoid Robots

If you close your eyes and think about a robot, what would you picture in your mind? Most likely a fictional creature like Arnold Schwarzenegger in the Terminator series movies or C-3PO from Star Wars. It is likely a *humanoid*—a humanlike robot with a head and body with arms and legs, probably painted metallic silver. Humanoid robots are expected to imitate human motion and interaction (Fig. 8.9) and have their roots in longing and mythmaking, as discussed in our first chapter. With years of research, they are becoming commercially available in several application domains, including in competitive game-playing (such as in the RoboCup humanoid league[1]) and social and interactive robots such as the Pepper (Fig. 8.9) by Softbank Robotics. Strictly speaking, Pepper is a semi-humanoid robot with a wheeled robot base and not a bipedal robot. As mentioned earlier, a wheeled robot is much simpler, stable

[1] https://humanoid.robocup.org/.

and economical to produce. How these robots are deployed are constantly expanding, and with the development of new technology, the market will follow suit.

8.3.2.4 Four-Legged Robots

Four-legged robots are also called quadruped robots. They have better stability compared to two-legged robots during movement. Also, the lower centre of gravity and four legs keep them well balanced when they are not moving. They can move either by moving one leg at a time or by moving the alternate pair of legs (Fig. 8.10).

Types of Gait for Four-Legged Robots

Four-legged robots can walk with statically and dynamically stable gaits. In the *statically stable gait*, each leg of the robot is lifted up and down sequentially, and there are three stance legs at least at any moment. This type of gait is called creeping gait (Zhao et al., 2012). *Dynamically stable* gaits are often used in four-legged robots to walk and run due to their efficiencies, such as trotting, pace, bounce, and gallop gait (Fukuoka & Kimura, 2009). In trotting gait, two of the legs are in the same diagonal lift, and the two legs are in contact with the ground until the other two legs lift off, and then repeat the motion two by two in order.

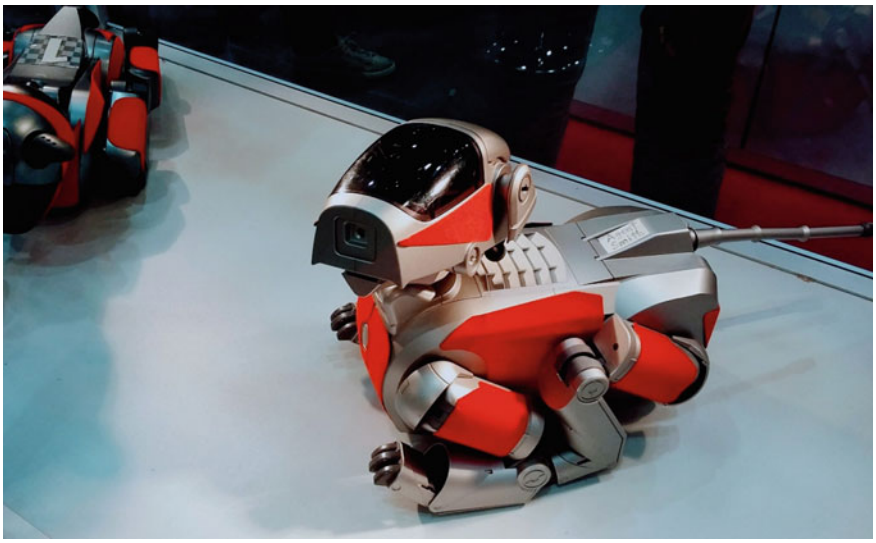

Fig. 8.10 Sony Aibo robot dog—One of the early versions of Sony's four-legged robot dog series

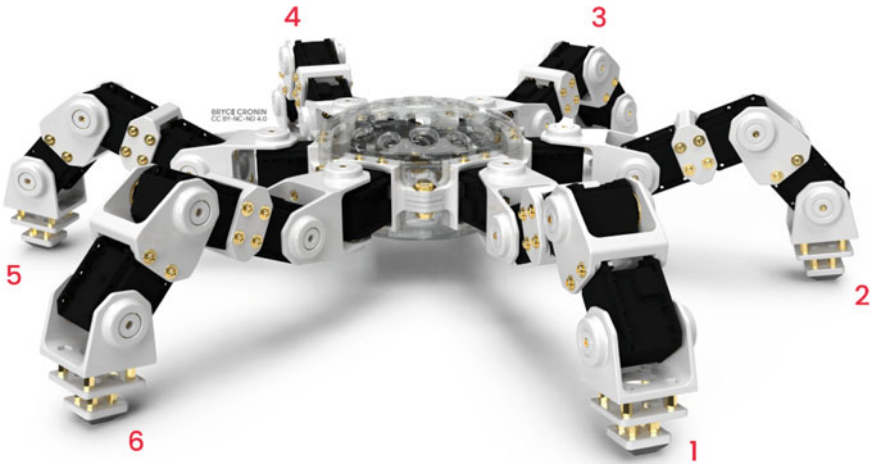

Fig. 8.11 A robot hexapod (Credit Bryce Cronin)

8.3.2.5 Six-Legged Robots

Six-legged robots are also called hexapods. They are designed to mimic the mechanics of insects. Their legs move in a "wave" form from the back to the front. As a result, six-legged robots offer greater stability while moving and standing, they can operate just on three legs, and the remaining legs provide flexibility and increase their capabilities. In Chaps. 12 and 17, you will explore the design and implementation of a hexapod robot (Fig. 8.11).

Types of Gait for Six-Legged Robots

One by one is the simplest gait, which moves each leg forward one after the other in a clockwise or anticlockwise direction while the remaining five legs are in the *stance phase*—not moving. For *a quadruped gait* (Fig. 8.11), the robot moves the front two legs (1 and 2) forward, and the rest (3, 4, 5, 6) support the body, then the robot moves the middle two legs (3, 6) to push the body forward while the rest of the legs (1, 2, 4, 5) support, then swing the last two legs (4 and 5) forward while the other legs support (1, 2, 3, 6) the robot. The pattern is then repeated. The *tripod gait* uses two legs on one side and another on the other side (e.g. 1, 5, and 3), as in a tripod, to hold the robot steady while moving the three remaining legs forward (2, 4, and 6) together.

8.3.2.6 Eight-Legged Robot

Spiders and other arachnids inspire eight-legged robots. Compared with other legged robots, eight-legged robots offer the greatest stability with potential use in more

challenging environments such as in hazardous areas to perform reconnaissance, identify structural damages, and perform maintenance tasks.

8.3.3 Flying Robots

Much effort has been devoted to improving the flight endurance and payload of Unmanned Aerial Vehicles (UAVs), commonly known as drones, which has resulted in various configurations in different sizes, capabilities, and endurance. Unlike legged and wheeled robots, flying robots are free to utilise the full six degrees of freedom, allowing for different types of flight for a drone. These are known as Yaw, Pitch, and Roll (Fig. 8.12).

Yaw (ψ) – This is the rotation of the drone's head to either right or left. It is the basic movement to spin the drone. In a remotely piloted drone, this is usually achieved using the left throttle stick by moving to either the left or right.

Pitch (θ) – This is the drone's movement, either forward or backward. The forward pitch is generally achieved in a remotely piloted drone by pushing the throttle stick forward, making the drone tilt and move forward, away from you. Backward pitch is achieved by moving the throttle stick backwards.

Roll (Ø) – Roll makes the drone fly sideways to either left or right. The right throttle stick controls the roll in a remotely piloted drone.

8.3.3.1 Multicopters

A multicopter is a type of flying vehicle with propellers driven by motors (Fig. 8.13). The main rotor blade(s) produces a forceful thrust used for both lifting and propelling the vehicle. Multirotor uncrewed aerial vehicles are capable of vertical take-off and landing (VTOL) and may hover at a place, unlike fixed-wing aircraft. Their hovering

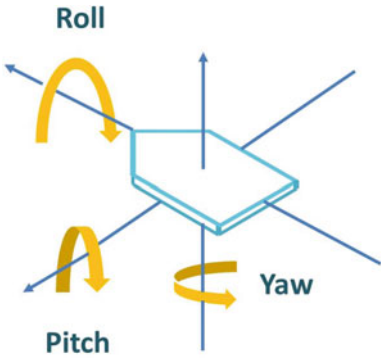

Fig. 8.12 Roll, pitch, and yaw

Fig. 8.13 Different types of multicopper (clockwise from top left—A quadrotor—DJI MAVIC PRO, A hexacopter—Custom built model, DJI Phantom Model and An octocopter—Custom built model)

capability and ability to maintain speed make them ideal for civilian fields, monitoring, surveillance, and aerial photography work. One of the challenges with multicopters is that they consume more power, leading to limited endurance. Also, multicopters, unlike fixed-winged counterparts, are inherently aerodynamically unstable and requires an on-board flight controller (an autopilot) to maintain stability.

Multicopters can be divided into specific categories based on the number and positioning of motors, and each category has its own mission (Fig. 8.14). And based on the mission requirements, they are classified in various configurations such as *Monocopter* (1 rotor), Tricopter (3 rotors), quadcopter (4 rotors), *hexacopter* (6 rotors) (X/ + configurations), *Octacopter* (8 rotors) (X/ + configurations), X8-rotor, and Y6-rotor. A quadrotor is a multirotor helicopter lifted and propelled by four rotors. It is a useful tool for university researchers to test and evaluate new ideas in several fields, including flight control theory, navigation, real-time systems.

8.3.3.2 A Quadrotor Example

A quadrotor (drone) is able to perform three manoeuvres in the vertical plane: hover, climb, or descend.

Hover—To hover, the net thrust of the four rotors push the drone up and must be exactly equal to the gravitational force pulling it down.

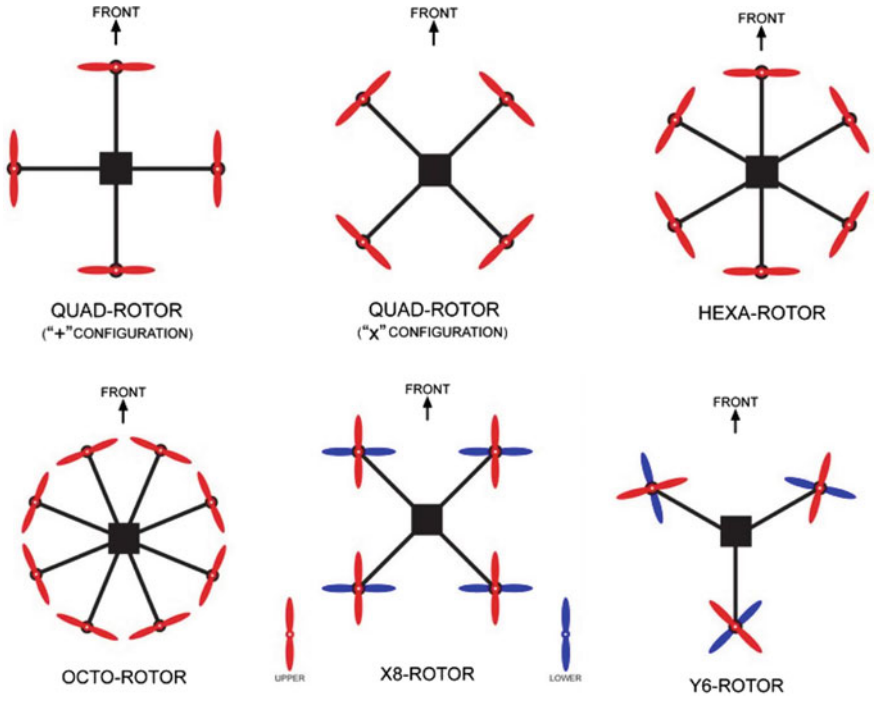

Fig. 8.14 Various configurations possible with the hoverfly multirotor control board (Ed Darack, 2014)

Climb (Ascend)—Increasing the thrust (speed) of the four rotors so that the upward force is greater than the weight and pull of gravity.

Descend—Dropping back down requires doing the exact opposite of the climb, decreasing the rotor thrust (speed) so the net force is downward.

To fly forward, an increase in the quadcopter motor rpm (rotation rate) of rotors 3 and 4 (rear motors) and a decrease in the rate of rotors 1 and 2 (front motors) is required. The total thrust force will remain equal to the weight so that the drone will stay at the same vertical level. To rotate the drone without creating imbalances, a decrease in the spin of motors 1 and 3 with an increase in the spin of rotors 2 and 4 is required (Fig. 8.15).

Mathematical Model of a Quadcopter

The structure of the quadcopter is presented in the below figure, including the corresponding angular velocities, torques and forces created by the rotors (Fig. 8.16).

The absolute linear position $\boldsymbol{\xi}$ of the quadcopter is defined in the inertial frame. Angular position is defined with three Euler angles $\boldsymbol{\eta}$. Vector $\boldsymbol{q}$ contains the linear and angular position vectors.

Fig. 8.15 A quadcopter rotor configuration

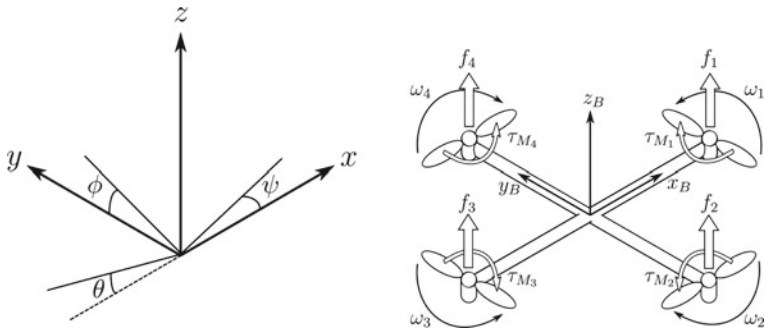

Fig. 8.16 Inertial and body frames of a quadcopter

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \boldsymbol{\eta} = \begin{bmatrix} \emptyset \\ \theta \\ \psi \end{bmatrix}, \boldsymbol{q} = \begin{bmatrix} \xi \\ \eta \end{bmatrix}$$

The origin of the body frame is in the centre of mass of the quadcopter. In the body frame, the linear velocities are determined by V_B and the angular velocities by ν

$$V_B = \begin{bmatrix} v_x, B \\ v_y, B \\ v_z, B \end{bmatrix}, \quad \boldsymbol{v} = \begin{bmatrix} \mathrm{p} \\ q \\ \mathrm{r} \end{bmatrix}$$

The rotation matrix from the body frame to the inertial frame is

$$R = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\emptyset - S_\psi C_\theta & C_\psi S_\theta C_\emptyset + S_\psi S_\theta \\ S_\psi C_\theta & S_\psi S_\theta S_\emptyset + C_\psi C_\theta & S_\psi S_\theta C_\emptyset - C_\psi S_\theta \\ -S_\theta & C_\theta S_\emptyset & C_\theta C_\psi \end{bmatrix}$$

where $S_x = \sin(x)$ and $C_x = \cos(S)$. The rotation matric R is orthogonal thus $R^{-1} = R^T$ which is the rotation matrix from the inertial frame to the body frame. The transformation matric for angular velocities from the inertial frame to the body frame is W_η, and from the body frame to the inertial frame is W_η^{-1}:

$$\dot{\eta} = W_\eta^{-1} v \quad \text{then} \quad v = W_\eta \dot{\eta},$$

The quadcopter is assumed to have a symmetric structure with the four arms aligned with the body x- and y-axes. Thus, the inertia matrix is diagonal matrix I in which $I_{xx} = I_{yy}$

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

The inverse of the following equation could be used to solve for the required rotor speeds to achieve the desired thrust (T_Σ) and moments $\tau = (\tau_1, \tau_2, \tau_3)$ of the quadcopter (Mahony et al., 2012);

$$\begin{pmatrix} T_\Sigma \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{pmatrix} = \begin{bmatrix} C_T & C_T & C_T & C_T \\ 0 & dc_T & 0 & -dc_T \\ -dc_T & 0 & dc_T & 0 \\ -C_q & C_q & -C_q & C_q \end{bmatrix} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix}$$

where C_T (>0) and C_q are two coefficients that can be experimentally determined for the considered quadcopter using thrust tests.

8.3.3.3 Fixed Wings

Fixed-wing UAVs require a runway for take-off and landing and also, unlike multicopters, cannot hover and maintain flight at low speeds. However, they have longer endurance and can fly at high cruising speeds because of the successful generalisation of larger fixed-wing planes with slight modifications and improvements.

Fixed wings are the main lift generating elements in response to forward accelerating speed. The velocity and steeper angle of air flowing over the fixed wings controls the lift produced. Fixed-wing drones require a higher initial speed and a thrust to load ratio of less than 1 to initiate a flight. If fixed-wing and Multirotor are compared for the same amount of payload, fixed-wing drones are more comfortable with less power requirement and thrust loading of less than 1. Rudder, ailerons, and elevators control aircraft orientation in yaw, roll, and pitch angles.

8.3.3.4 Other Flying Robots

There are also some non-conventional configurations of UAVs used for scientific research. They include hybrid, convertible and flapping wing drones that can take off vertically or act as an insect for spying missions. Flapping wing drones inspired by insects such as small dragonflies[2] and birds[3] have regularly appeared in the research literature and at times as commercial prototypes. Due to the lightweight and flexible wings, the flapping drones can contribute well to stable flight in a windy environment. A large amount of research work on flapping wing drones has been carried out by researchers and biologists because of their exclusive manoeuvrability benefits. Blimps and airships are other categories of flying robots that utilise a lifting gas that is less dense than the environment it is operating.

8.4 Controlling Robots

Using the Sense, Think, Act framework, the robot's *controller* can be thought of as the component within the Think element responsible for the robot's movements. It is usually a microcontroller or an onboard computer or a mix of these used to store information about the robot and its surrounding environment and execute designated programmes that operate the robot. The *control system* includes data processing, control algorithms, logic analysis, and other processing activities which enable the robot to perform as designed. Based on the different requirements, more sophisticated robots have more sophisticated control systems.

The control system involves all three aspects of the sense, think, and act loop during execution. First, the perception system provides information about the environment, the robot itself, and the relationship between the robot and the environment. Based on the information from the sensors and the robot's objectives, the cognition and control system must then decide on how to act and what to do to achieve its objectives. The appropriate commands are then sent to the actuators, which move the mechanical structure. The control system coordinates all the input data and plans the robot's motion towards the desired goal.

Various control techniques have been proposed and are being researched. The control strategies of mobile robots can be divided into open-loop and closed-loop feedback strategies. When it comes to *open-loop* control, human operators are involved in sending instructions. The robot relays information to the operator only to perform as instructed. An example of such a system is piloting a drone using a drone controller. The robot's success in achieving its mission is essentially dependent on your piloting skills—the controller simply relays your "intent" to the drone. Most of the time, control commands such as velocities or torques are calculated beforehand, based on the knowledge of the initial and end position ("Goal pose") of the

[2] https://spectrum.ieee.org/somehow-an-incredible-robotic-dragonfly-is-now-on-indiegogo.

[3] https://spectrum.ieee.org/festo-bioinspired-robots-bionicswift.

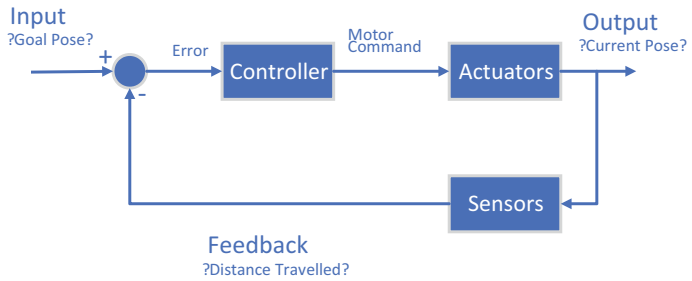

Fig. 8.17 A typical closed-loop feedback controller

robot. However, this strategy cannot compensate for disturbances and model errors ("Error").

On the other hand, closed-loop control strategies could provide the required compensation since the inputs are functions of the actual state of the system and not only of the initial and endpoints. Therefore, disturbances and errors causing deviations from the predicted state are compensated by real-time sensor data ("Feedback") (Fig. 8.17). Formally, we could define a feedback controller as enabling a robot to reach and maintain the desired state (called a *set point*) by repeatedly comparing its current state with the desired goal state. Here, *feedback* refers to the information that is literally "fed back" into the system's controller. When a system is operating at the desired state, it is said to be operating at the *steady state*.

8.4.1 PID Controllers

A *PID controller* is a control loop feedback mechanism that calculates the difference between a desired value (*setpoint*) and the actual output from a process and use that result to apply a correction to the process. The term PID stands for **P**roportional–**I**ntegral–**D**erivative feedback control, and it is one of the most commonly used controllers in the industry. It is the best starting point when designing an autonomous control system and is very popular in commercial autopilot systems and open-source developments.

The main goal of this process is to maintain a specified setpoint value. For example, you may want a DC motor to maintain a setpoint value $r(t)$ of 600 encoder pulses per second. The actual motor speed $y(t)$, called the process variable, is subtracted from the setpoint value 600 to find the error value $e(t)$. The PID controller then computes the new control value $u(t)$ to apply to the motor based on the computed error value. In the case of a DC motor, the control value would be a pulse-width-modulated (PWM) signal. The (t) represents a time parameter being passed into the process (Fig. 8.18).

Let us now look at how each of the three elements, P, I, D, contributes to the overall controller.

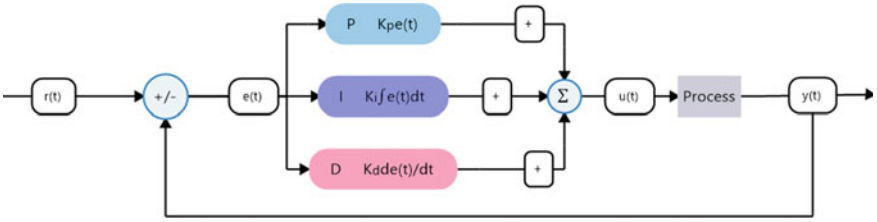

Fig. 8.18 A PID controller—$r(t)$ is the reference setpoint, $e(t)$ is the difference between the process output and the desired setpoint, $u(t)$ is the process input control value, $y(t)$ is the process output

8.4.1.1 Proportional Control (P)

This element takes some proportion of the current error value. The proportion is specified by a constant called the gain value, and a proportional response is represented by the letters Kp. As an example, Kp may be set to 0.25, which will compute a value of 25% of the error value. This is used to compute the corrective response to the process. Since it requires an error to generate the proportional response, there is no proportional part of the corrective response if there is no error. For example, when controlling a drone autonomously, increasing the P gain Kp typically leads to shorter rise time (i.e. the drone reaches the required altitude quickly) and larger overshoots. Although it can decrease the system's settling time, it can also lead the drone to display highly oscillatory or unstable behaviour (Fig. 8.19).

8.4.1.2 Derivative Control (D)

The derivative term is used to estimate the future trends of the error based on its current rate of change. It is used to add a dampening effect to the system such that the quicker the change rate, the greater the controlling or dampening effect. In that sense, increasing the D gain Kd typically leads to smaller overshoot and a better-damped behaviour. However, increasing Kd could lead to larger steady-state errors (Fig. 8.20).

8.4.1.3 Integral Control (I)

Element I takes all past error values and integrates them over time. The term integrates simply means to accumulate or add up. This results in the integral term growing until the error goes to zero. When the error is eliminated, the integral term will stop growing. If an error still exists after the application of proportional control, the integral term tries to eliminate the error by adding in its accumulated error value. This will result in the proportional effect diminishing as the error decreases, and the growing integral effect compensates for this. Increasing the I gain Ki leads to a

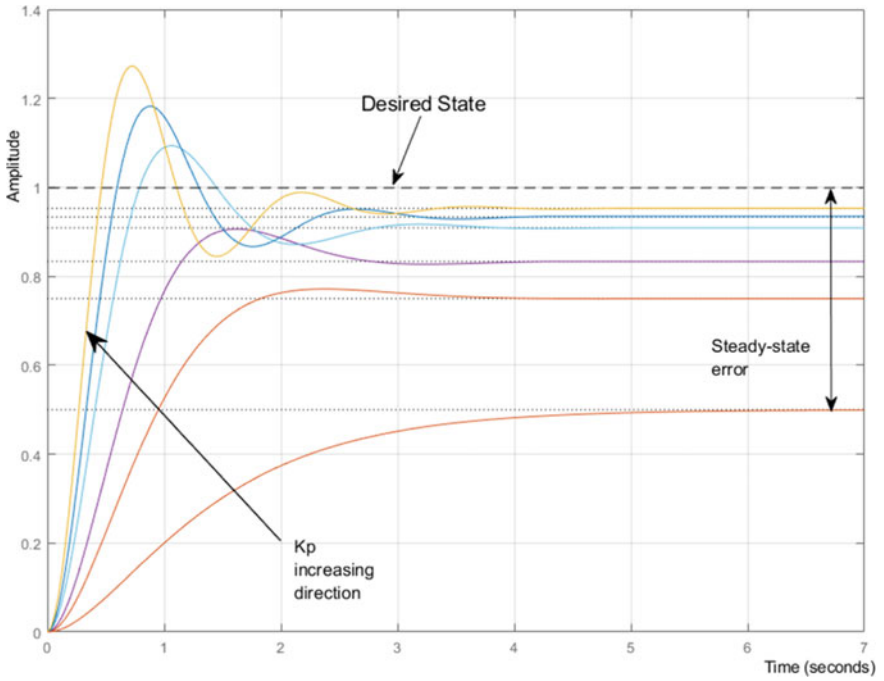

Fig. 8.19 An example showing the effects of increasing Kp—shorter rise time but oscillatory behaviour increasing. (No Integral and Derivative control)

reduction in the steady-state error (often elimination) but also could lead to larger oscillations (Fig. 8.21).

Another issue to be mindful of when using the integral term in a controller refers to *Integral windup.* This is common in most physical systems (nonlinear systems), where a significant change in the setpoint (either positive or negative) results in the integral term accumulating significant errors that cannot be offset by errors in the opposite direction leading to a loss of control. Researchers have developed several anti-windup techniques over the years to counter the phenomenon. One common technique is setting boundaries for the integral term depending on the known system limitations, such as actuator operational range.

8.4.1.4 Tuning a PID Controller

As understood from this brief overview of the role of each element of the PID controller, it is not possible to independently tune the three different gains. Each of them aims to offer the desired response characteristic (e.g. faster response, damped and smooth oscillations, near-zero steady-state error) but has a negative effect that must be compensated by re-tuning another gain. Therefore, PID tuning is a highly

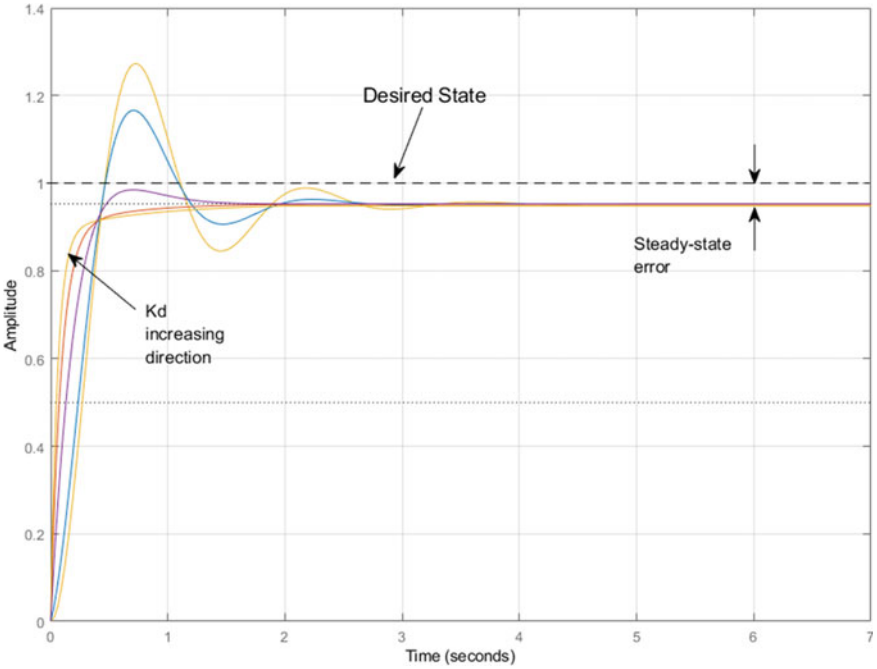

Fig. 8.20 An example showing the effects of increasing Kd with a constant Kp (No Integral control)

coupled and iterative procedure. The PID controller consists of the additive action of the Proportional, the Integral, and the Derivative component. Not all of them have to be present; therefore, we often employ P controllers, PI controllers or PD controllers when a simpler controller yields the desired result.

8.4.2 Fuzzy Logic Controllers

The fuzzy logic theory was developed in the mid-1960s as a way to deal with the imprecision and uncertainty inherent to perception systems. Since then, it has been used in many engineering applications. Designers consider it one of the simpler solutions available for many nonlinear control problems, including most robotics navigation and control problems. Fuzzy logic is more advantageous than traditional solutions because it allows computers to act more like humans, responding effectively to complex inputs to deal with linguistic notions such as “too hot”, “too cold” or “just right”. Furthermore, fuzzy logic is well suited to low-cost implementations based on cheap sensors, low-resolution analog-to-digital converters, and 4-bit or 8-bit microcontroller chips. Such systems can be easily upgraded by adding new rules to improve performance or by adding new features. In many cases, fuzzy control can

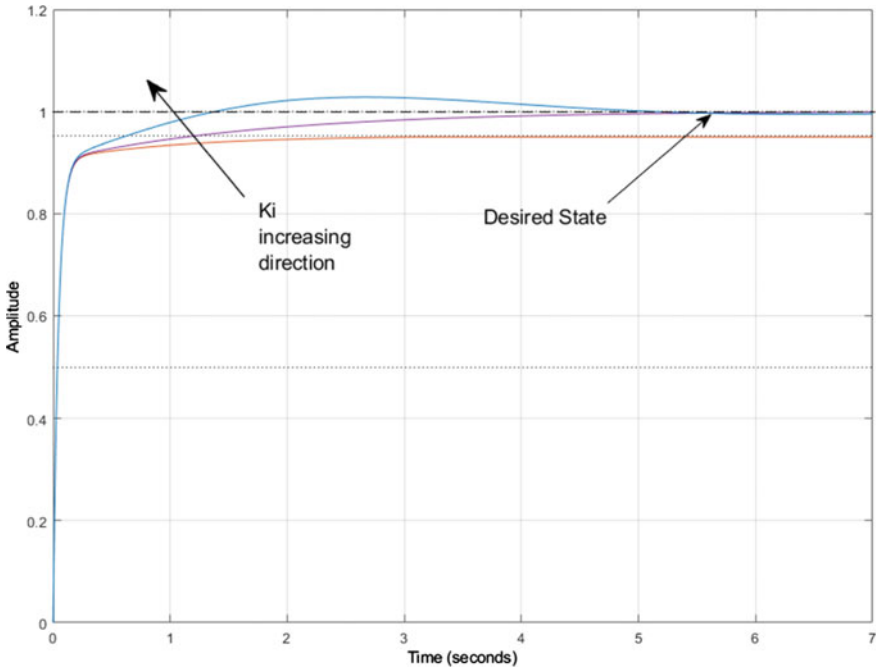

Fig. 8.21 An example showing the effects of increasing Ki with a constant Kp and Kd

improve existing traditional control systems by adding an extra layer of intelligence to the current control method.

8.4.2.1 A Simple Example

Consider a ground robot moving towards a target.

The fuzzy logic controller (FLC) used has two inputs: error in distance (Δe_d) and error in the angle of orientation (Δe_a) of the robot. The controller's output (that is, the control signals) would be pulse-width-modulated signals to control the angular velocity of the two servo wheels. Therefore, the fuzzy logic controller is a two-input, two-output system. The block diagram of the robotic system is shown in Fig. 8.22.

8.5 Path Planning

Path planning is the means of finding a suitable (optimal) path for a moving platform to travel from its starting point to the goal point in a given environment. Early work on path planning focused on planning paths for robotic manipulators, where a perfect

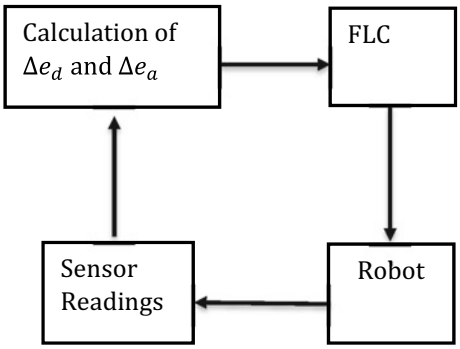

Fig. 8.22 Fuzzy logic control system

world model and precise knowledge of the joint angles were assumed. However, these assumptions cannot be made for mobile robots operating in partly known or unknown environments and with localisation uncertainties.

Classical algorithms, such as Dijkstra's algorithm (Dijkstra, 1959), A and A* algorithms (Hart et al., 1968), apply a global graph search to find the least-cost path from the starting point to the target point. There are also other methods for sampling the local environment to determine the least-cost path (Kuffner & LaValle, 2000). The main purpose of obtaining the best path is to find the shortest path with minimal energy usage and maximum coverage of an area or optimised predicted perception quality. In some situations, it is beneficial to choose from a given set of trajectories that can be followed by the robot's controller rather than planning a specific and maybe impossible path (Dey et al., 2011). Therefore, different path planning algorithms are used for different situations, with most algorithms relying on heuristic and probabilistic techniques.

8.5.1 Heuristic Path Planning Algorithms

Heuristic methods use an estimated cost function for target-oriented path searching which considerably reduces the computational time. These algorithms calculate the path based on the fewest number of grid cells in the queue by assigning a cost to each node with respect to the difference of its distance from that of the minimal distance between the starting and goal nodes.

8.5.1.1 A* Algorithm

The most well-known path planning algorithm is the A* algorithm (Hart et al., 1968) which uses a best-first search method to find the least-cost path from the starting to the goal node. Unlike other path planning techniques, we can consider that the

A* algorithm has a "brain" that can do the calculations. It is widely used for games and web-based maps to find the shortest path in a very efficient way. The vehicle traverses towards the goal node until it either reaches it or determines that there is no available path with a heuristic function used to evaluate the goodness of each node.

Considering a graph map with multiple nodes, what the A* algorithm does is that at each step, it picks the node according to the value "*f*", which is equal to the sum of "*g*" and "*h*". At each step, it picks the node having the lowest "*f*" value and proceeds to the next until it finds the goal point.

$$f(\text{node}) = g(\text{node}) + h(\text{node})$$

where:

*g(*node*)* is the travelling cost from the initial point to the current point; *h(*node*)* is the heuristic function that includes the cost from the starting node to the current location, $c(n, n')$ and estimated cost from the current location to goal $h(n')$.

A* (star) Pathfinding Pseudocode

```
// Initialise both open and closed list
let the openList and closedList equal empty list of nodes

// Add the start node
put the startNode on the openList (leave it's f at zero)

// loop until find the end
while the openList is not empty

// Get the current node
let the currentNode equal the node with the least f value
remove the currentNode from the openList
add the currentNode to the closedList

// Found the goal
if currentNode is the goal
Goal found! Backtrack to get path

// Generate children
let the children of the currentNode equal the adjacent nodes
for each child in the children
```

```
// Child is on the closedList
if child is in the closedList
continue to beginning of for loop

// Create the f, g, and h values
child.g = currentNode.g + distance between child and current
child.h = distance from child to end
child.f = child.g + child.h

// Child is already in openList
if child.position is in the openList's nodes positions
if the child.g is higher than the openList node's g
continue to beginning of for loop

// Add the child to the openList
add the child to the openList
```

For example:
We would like to find the shortest path between A to K in the following map. The number written with red is the distance between the nodes, and the number in the blue circle written in black is the heuristics value. A* uses $f(n) = g(n) + h(n)$ to find the shortest path.

Let's start with start point A. A has three nodes: B, E, and F, then we can start calculate $f(B)$, $f(E)$, and $f(F)$:

$$f(B) = 3 + 8 = 11$$
$$f(E) = 1 + 1 = 2$$
$$f(F) = 5 + 4 = 9$$

$f(E) < f(F) < f(B)$, so we will choose E as the new start node.

For node E, it two nodes F and H, $f(F) = 7\ (1 + 6) + 4 = 11, f(H) = 3\ (1 + 2) + 4 = 7, f(H) < f(F)$, so we will choose H as the new start node.

For node H, it has two nodes J and I, $f(J) = 5\ (1 + 2 + 2) + 3 = 8, f(I) = 4\ (1 + 2 + 1) + 2 = 6, f(I) < f(J)$, so we will choose I as the new start node.

For node I, it has two nodes D and K, $f(D) = 10\ (1 + 2 + 1 + 6) + 5 = 15, f(K) = 6(1 + 2 + 1 + 2) + 0 = 6, f(K) < f(D)$, so we will choose K as the next node, as K is the goal point, the algorithm stop here.

The shortest path from A to K is A—E—H—I—K (Fig. 8.23).

The A* algorithm is similar to *Dijkstra's algorithm* (Dijkstra, 1959), except that it guides its search towards the most promising states, which can save a significant amount of computational effort. The limitation of the above approaches is that they need a complete map of the area under exploration. However, when operating in real-world scenarios, as new information might be added to the map, replanning is

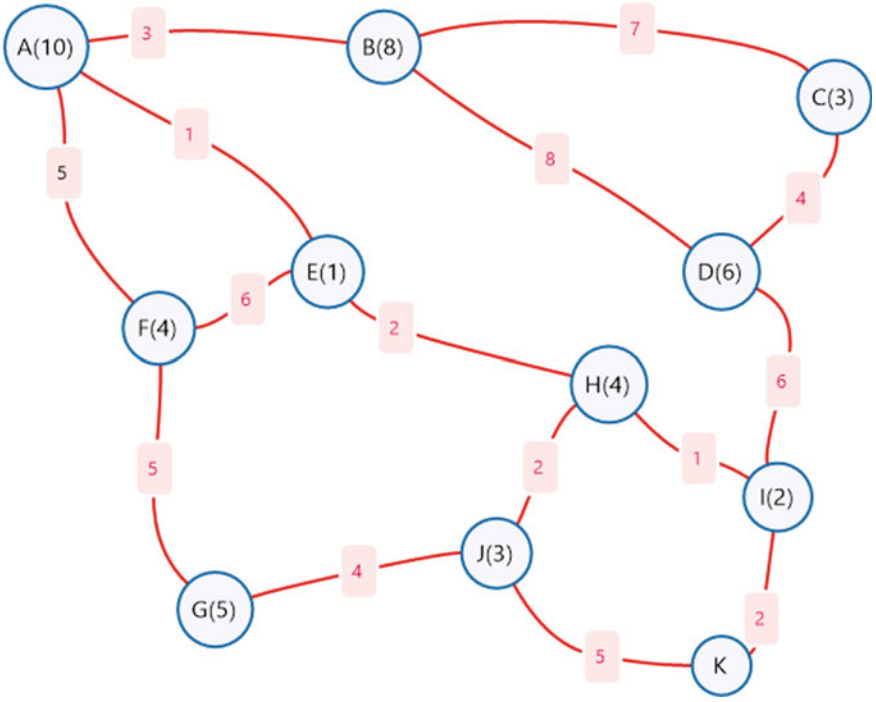

Fig. 8.23 A* algorithm example

essential. While A ∗ could be used to plan from scratch for every update, this is computationally expensive.

Instead, the D* Lite (Koenig & Likhachev, 2005) and Focussed Dynamic A* (D*) (Stentz et al., 1995) algorithms search for a path from the goal towards the start and update nodes only when changes occur. An updated path is calculated based on the previous path, which is much more effective than the A ∗ algorithm and Dijkstra's algorithm. *D* Lite algorithm* is one of the most popular goal-directed navigation algorithms that is widely used for mobile robot navigation in unknown environments. It is a reverse searching method and can replan from the current position when new obstacles are blocking the path.

Finally, *Field D** is an interpolation based path planning and replanning algorithm (Ferguson & Stentz, 2006). In contrast to other methods in which nodes are defined as the centres of grids, it defines nodes on the corners of grids. Then linear interpolation is used to create waypoints along the edges of grids which allows the planning of direct, low-cost, smooth paths in non-uniform environments. D* and its variants are widely used for autonomous robots, including Mars rovers and autonomous cars (Stentz & Hebert, 1995; Urmson et al., 2008).

8.5.2 Probabilistic Path Planning Algorithms

Probabilistic approaches sample the configuration space randomly, which helps to decrease the path planning time and memory usage. However, their main disadvantage is that they cannot always be guaranteed to find the optimal path.

Much work has been conducted based on probabilistic path planning methods. One of the most popular approaches is the *probabilistic roadmap* (PRM) algorithm (Kavraki & Latombe, 1998; Kavraki et al., 1996) which generally consists of two phases: firstly, it randomly samples points in the configuration space to build a roadmap graph and then connects the sampled configurations to their neighbours; and secondly, in the query phase, the starting and goal nodes are connected to their neighbours in the graph and the path calculated using a heuristic method. Although any existing path can be found if there is a sufficiently increasing number of samples, as situations such as narrow corridors in large environments can rapidly increase the path planning time, deliberate sampling strategies are necessary. While multiple queries can be executed on the same graph-based on PRMs, some pre-processing is needed during which, in some cases, obstacles are defined.

8.6 Obstacle Avoidance

In mobile robotics, the goal of obstacle avoidance is generally to navigate from one location to the goal location while avoiding collisions with obstacles during the robot motion in a known or unknown environment. Therefore, obstacle avoidance is almost always is combined with path planning. The process requires an understanding of the environment, such as a full map or partial map, a target location and robot's location (localisation) (discussed in the next chapter), and sensors such as cameras or laser sensors to provide obstacle information.

Obstacle avoidance is always comprised of obstacle detection and collision avoidance. There are varieties of algorithms that use different kinds of sensors and techniques to achieve the goal of obstacle detection. The processed data received from sensors are then sent to the controller to operate the robot to avoid obstacles. There are some widely used obstacle avoidance algorithms such as bug algorithms, VFH, and other proximity-based techniques (e.g. sonar, bumper sensors).

8.6.1 Bug Algorithm

The bug algorithms are the simplest obstacle avoidance method among all obstacle avoidance methods. In the bug algorithm, the main idea is to track the contour of the obstacles found in the robot's path and make the robot circumnavigate it (Lumelsky,

2005; Lumelsky & Stepanov, 1987). There are several modified versions of the bug algorithm, such as Bug 1, Bug 2, DistBug, and Tangential Bug algorithm.

Bug 1 algorithm is the simplest of all Bug algorithm variations. It reaches the goal almost all the time with high reliability. But the matter of concern with this method is efficiency. The robot moves on the shortest path joining the robot's position X and goal location until it encounters a hurdle in the path. When an obstacle confronts it, it starts revolving around its surface and calculates the distance from the destination point. After one complete revolution, it figures out the point of departure closest to the goal. Then, it maintains or changes the direction of motion depending on the distance of leaving point from the hit point. This method can be illustrated in the following steps:

- Head towards the goal
- If an obstacle is encountered, circumnavigate it and remember how close you get to the goal
- Return to that closest point and continue

Robot revolves around every obstacle on the way towards the goal, increasing the computational efforts. But ease of implementation makes it worth it when only completion of the task is required irrespective of time.

Generally speaking, the bug algorithms work well with single obstacle avoidance. However, these bug algorithms are not very reliable in a more complex and cluttered environment, and in some tricky conditions, one version works better than the other version.

8.6.2 The Vector Field Histogram (VFH)

Vector field histogram is a real-time obstacle avoidance method for mobile robots developed by Borenstein and Koren (1991). This method contains three major components that help to achieve obstacle avoidance. Firstly, the robot generates a two-dimensional sensory histogram around its body or within a limited angle and starts updating the histogram data at every stage. Secondly, the two-dimensional histogram data are converted into a one-dimensional polar histogram. Finally, it selects the lower polar dense area and moves the vehicle, calculating the direction.

This approach overcomes the issue of sensor noise. A histogram is a graph between probabilities of the presence of obstacles to the angle associated with the sensor reading. The probabilities are obtained by creating a local occupancy grid map (see Chap. 9) of the environment of the robot's surroundings. The histogram is used to discover all the passages large enough to allow the robot to pass through. The selection of path is based on a cost function which is a function of the alignment of the robot's path with the goal and on the difference between the current wheel orientation and the new direction. A minimum cost function is desirable. One of the advantages of using VFH is that it conquers the problem of sensor noise by making a polar histogram that represents the probability of obstacle of a particular

angular direction. Some demerits need to be taken into consideration when using this technique, such as VFH does not guarantee the completeness, which can lead to an unfinished task. It can be problematic to pass through a narrow passage using this method. Moreover, it does not consider the robot's dynamics and its environment, making it not ideal for use in a complex dynamic environment.

8.7 Chapter Summary

Robots that move around in the environment instead of being fixed to a single location are called mobile robots. These can be categorised according to the type of locomotion they utilise, such as wheeled, legged, or flying.

A robot controller essentially provides the controlling commands to its actuators to drive the robot towards the desired goal. A common control loop is the PID (proportional–integral–derivative) controller, which uses sensor feedback to update the control signal in a repeated manner. Essentially the controller applies a correction to a control function where the correction could be proportional to the error (P) or reflective of the cumulative error (I) or the change in the error rate (D). A PID controller requires tuning of its parameters, which usually requires an iterative trial and error approach or sophisticated tuning algorithms to realise optimal performance.

For a robot to move from a given point to the desired goal point, it needs to plan a path between the two points using some optimal criteria, for example, shortest distance, the lowest energy consumption, or the largest area coverage. Many techniques have evolved over the years, including heuristic and probabilistic techniques, each having its own merits and concerns. Additionally, a complimentary problem in path planning is the obstacle avoidance problem. Again, researchers have come up with various strategies and techniques to solve the problem.

As a roboticist developing a mobile robot, your task is to select, develop, and implement techniques, algorithms, and platforms based on the ideas discussed in this chapter to suit the requirements of the job at hand.

8.8 Review Questions

- If using a PID controller for a drone, increasing the P gain Kp typically leads to shorter or longer rise times?
- If using a PID controller for a drone, increasing the I gain KI, would it result in smaller or larger oscillations?
- Comparing two-wheeled, three-wheeled, four-wheeled robots, which one is the most unstable type?
- What does pitch, yaw and roll mean in a drone?
- What is the difference between classic and heuristic path planning algorithms?

8.9 Further Reading

The chapter covered introductory material on several related topics. Once the basic concepts are well understood, you can explore these topics in more depth and expand onto advanced topics. Following titles, *Introduction to Robotics: Mechanics and Control* (3rd Edition) by John Craig, *Modern Robotics Mechanics, Planning, and Control* by Kevin M. Lynch and *Robotics Modelling, Planning and Control* by Bruno Siciliano provide some excellent reading. Another highly recommended book on mobile robots is the book by Roland Siegwart, *Introduction to autonomous mobile robots.*

References

Borenstein, J., & Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation, 7*(3), 278–288.

da Silva, L. R., Flesch, R. C. C., & Normey-Rico, J. E. (2018). Analysis of anti-windup techniques in PID control of processes with measurement noise. *IFAC-PapersOnLine 51*(4), 948–953.

Dey, D., Liu, T. Y., Sofman, B., & Bagnell, D. (2011). Efficient optimisation of control libraries. Technical report, DTIC Document.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269–271.

Ed, Darack, https://www.airspacemag.com/flight-today/build-your-own-drone-180951417, 2014.

Ferguson, D., & Stentz, A. (2006). Using interpolation to improve path planning: The field D* algorithm. *Journal of Field Robotics, 23*(2), 79–101.

Fukuoka, Y., & Kimura, H. (2009). Dynamic locomotion of a biomorphic quadruped "Tekken" robot using various gaits: Walk, trot, free-gait and bound. *Applied Bionics & Biomechanics, 6*(1), 63–71.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

https://en.wikibooks.org/wiki/Robotics/Types_of_Robots/Wheeled, 2021.

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), 566–580.

Kavraki, L. E., & Latombe, J. -C. (1998). Probabilistic roadmaps for robot path planning.

Koenig, S., & Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, *21*(3), 354–363.

Kuffner, J. J., & LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings. ICRA'00 IEEE international conference on robotics and automation, 2000*, (vol 2, pp. 995–1001). IEEE.

Lumelsky, V. J. (2005). *Sensing, Intelligence, Motion: How Robots and Humans Move in an Unstructured World*. John Wiley & Sons.

Lumelsky, V. J., & Stepanov, A. A. (1987). Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica, 2*, 403–430.

Mahony, R., Kumar, V., & Corke, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics & Automation Magazine, 19*(3), 20–32. https://doi.org/10.1109/MRA.2012.2206474

Stentz, A., et al. (1995). The focussed D* algorithm for real-time replanning. *In IJCAI, 95*, 1652–1659.

Stentz, A., & Hebert, M. (1995). A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots, 2*(2), 127–145.
Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., et al. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics, 25*(8), 425–466.
Zhao, D., Jing, X., Dan, W., et al. (2012). Gait Definition and successive gait-transition method based on energy consumption for a quadruped. *Chinese Journal of Mechanical Engineering, 25*(1), 29–37.

Jiefei Wang 's research focuses on sensing, guidance, and control for autonomous systems. He received the master's degree in electrical engineering from Australian National University in 2011, and the Ph.D. degree in electrical engineering from the University of New South Wales in 2016. His research interests include sensing and image processing, scene understanding for obstacle avoidance, control of autonomous systems, and aerial robotics.

Damith Herath is an Associate Professor in Robotics and Art at the University of Canberra. Damith is a multi-award winning entrepreneur and a roboticist with extensive experience leading multidisciplinary research teams on complex robotic integration, industrial and research projects for over two decades. He founded Australia's first collaborative robotics startup in 2011 and was named one of the most innovative young tech companies in Australia in 2014. Teams he led in 2015 and 2016 consecutively became finalists and, in 2016, a top-ten category winner in the coveted Amazon Robotics Challenge—an industry-focused competition amongst the robotics research elite. In addition, Damith has chaired several international workshops on Robots and Art and is the lead editor of the book *Robots and Art: Exploring an Unlikely Symbiosis*—the first significant work to feature leading roboticists and artists together in the field of Robotic Art.