

ROBOTICS

Robotics Fundamentals

Mechanics, motion, sensors, control, arms and your first robot build.

10 Chapters

Philip Osztromok

Generated with Claude

ROBOTICS FUNDAMENTALS

Contents

Chapter 1	What Is a Robot?	2
Chapter 2	Mechanics and Structure	15
Chapter 3	Motion: Motors, Gears and Wheels	29
Chapter 4	Drive Systems and Kinematics	43
Chapter 5	Sensors: Seeing the World	60
Chapter 6	Position, Orientation and Feedback	80
Chapter 7	Control: Making It Do What You Want	99
Chapter 8	Behaviour and Decision-Making	121
Chapter 9	Arms and Manipulators	141
Chapter 10	Power, Safety and Building Your First Robot	161

Robotics Fundamentals

Course 1 · Chapter 1 · What Is a Robot?

Basic Electronics ended with a promise: the parts you learned (sensors, motor drivers, boards, batteries) are the building blocks of a robot. This course explains the robot itself. Each chapter takes one subsystem (the frame, the drive train, the sensors, the controller, the arm, the power supply), says what it does and why it is designed that way, and puts numbers on it. This first chapter gives the overall picture: what makes a machine a robot, the loop at the heart of every robot, the main kinds of robot, how to count the ways a mechanism can move, and how a small robot is laid out, with each part pointing to the chapter of this course that explains it.

You are assumed to have finished Basic Electronics, or to know the same ground (Ohm's law, motors and H-bridges, ADC and PWM, a microcontroller board). No mechanics or control theory is assumed. The history dates and the figures for typical robots below are **from memory and approximate**: they are there to give a sense of scale and order, so check a source before quoting one.

1. What makes a machine a robot?

There is no single agreed definition, but most share three ideas: a robot is a machine that **senses** its surroundings, **decides** what to do using a program, and **acts** physically on the world, and it does so as a repeating cycle, not as a single fixed sequence. Leave out any one and you have something else.

MACHINE	SENSES?	DECIDES?	ACTS PHYSICALLY?	A ROBOT?
Radio-controlled car	No (the human watches)	No (the human decides)	Yes	No: a remote-controlled machine. The thinking is in the driver's head.
Washing machine	A little (water level, door, temperature)	A fixed programme with a few branches	Yes (drum, pump, valve)	Borderline: an automatic machine. It does not adapt to the world beyond a few switches.
Thermostat and boiler	Yes (temperature)	Yes (on or off)	Yes (heat)	Usually not called one, but it is the same loop at its

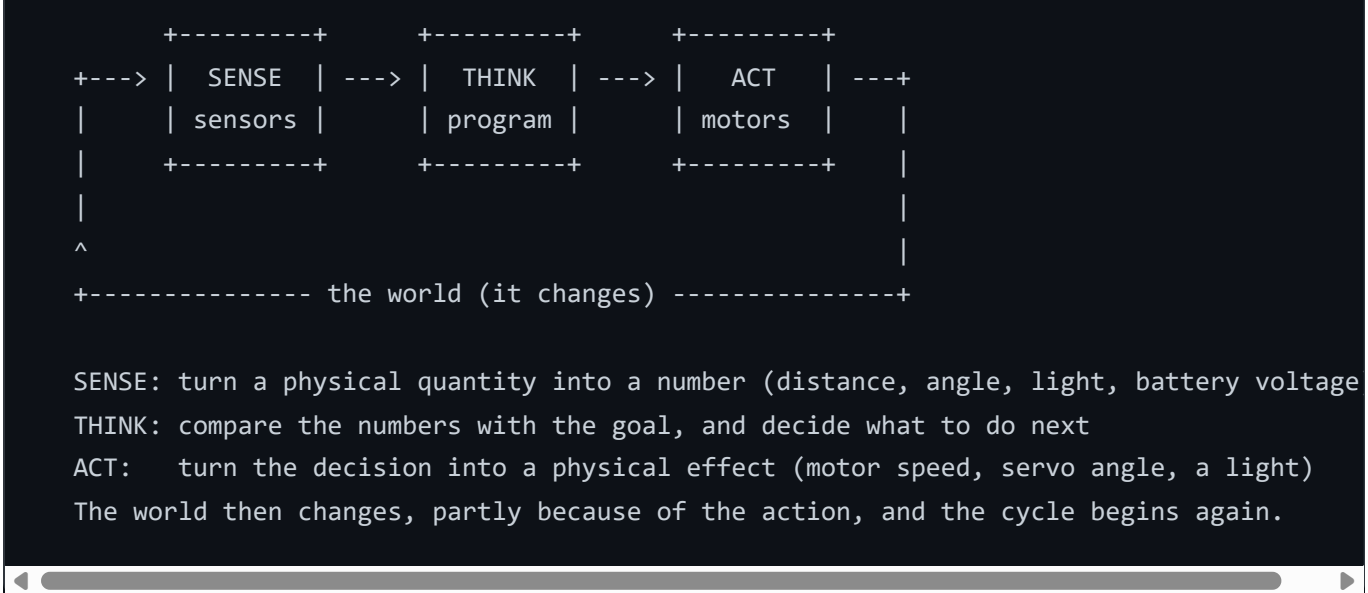
MACHINE	SENSES?	DECIDES?	ACTS PHYSICALLY?	A ROBOT?
				smallest: sense, compare, act.
Robot vacuum cleaner	Yes (bumpers, cliff, wall and dirt sensors)	Yes (cleaning pattern, obstacle behaviour)	Yes (wheels, brushes, fan)	Yes.
Industrial arm on an assembly line	Some (encoders, safety scanners)	A stored, repeated path	Yes	Yes: a programmable manipulator, though it adapts little.
Self-driving car	Yes (cameras, radar, lidar)	Yes	Yes (steering, throttle, brake)	Yes.

A USEFUL TEST

Ask: if the world changed during the task (a chair moved, a wheel slipped, a load was heavier than expected), would the machine notice and behave differently? The more it does, the more robot-like it is. Robots sit on a scale, not on one side of a line.

2. The sense-think-act loop

Every robot, from a toy buggy to a Mars rover, runs the same cycle many times a second:



The loop is **closed** through the world: what the robot does changes what it next senses. That is why a robot can correct its own mistakes (the line curved, so turn more) and also why it can go wrong in ways a fixed machine cannot (it turns too much, senses the line on the other side, turns back, and oscillates). Chapter 7 of this course, Robotics Fundamentals 7, is about making that loop behave.

The loop in pseudocode

This is deliberately hardware-neutral. The function names stand for whatever your board's library calls them (Basic Electronics 10 covered the reading and driving side). Real board code belongs to the planned course Programming for Robotics.

```
# One robot, one loop. All names are placeholders.
LOOP_PERIOD = 0.05          # seconds: 20 times a second (20 Hz)
STOP_DISTANCE = 0.30        # metres

while True:
    t_start = now()

    # SENSE: read the world into numbers
    distance = read_distance_sensor()
    line     = read_line_sensors()

    # THINK: decide
    if distance < STOP_DISTANCE:
        left, right = 0.0, 0.0                # something is close: stop
    else:
        left, right = follow_line(line)   # otherwise steer along the line

    # ACT: send the decision to the motors
    set_motor_speeds(left, right)

    # keep a steady rate: wait out the rest of this period
    wait_until(t_start + LOOP_PERIOD)
```

Three things are worth noticing already. The program is just three steps in a loop. The *quality* of the robot is in the details of each step (how good the sensors are, how smart the decision is, how well the motors respond). And the loop has a **rate**, which the next section shows is a safety number, not a detail.

Loop rate and reaction distance

A robot does not react instantly. Between something happening in the world and the robot's response taking effect, time passes in four places: the sensor takes a while to produce a reading, the reading waits for the next pass of the loop (up to one whole period), the program takes time to compute, and the motors and driver take time to respond. In the worst case the delays add:

```
Loop rate and period:        f = 1 / T            T = 1 / f          20 Hz  ->  T = 50

Worst-case latency:           t(lat) = t(sensor) + T + t(compute) + t(actuator)

Reaction distance:            d(react) = v x t(lat)       (how far it travels before th
Braking distance:             d(brake) = v^2 / (2 a)      (a = the deceleration the rob
Stopping distance:            d(stop)  = d(react) + d(brake)

Worked example: a robot at 0.5 m/s, 20 Hz loop, sensor delay 30 ms, compute 5 ms, motor
       t(lat)   = 30 + 50 + 5 + 20 = 105 ms = 0.105 s
       d(react) = 0.5 x 0.105 = 0.0525 m = 52.5 mm
       d(brake) = 0.5^2 / (2 x 2) = 0.25 / 4 = 0.0625 m = 62.5 mm
       d(stop)  = 52.5 + 62.5 = 115 mm       (it must see the wall more than 115 mm ahe

The same robot at 4 times the speed (2 m/s):
       d(react) = 2 x 0.105 = 0.21 m
       d(brake) = 2^2 / (2 x 2) = 4 / 4 = 1.0 m
       d(stop)  = 1.21 m                     (the speed is 4 times larger, the stopping
```

The lesson is the square in the braking distance: **doubling the speed quadruples the braking distance**, while the reaction part only doubles. A faster loop shortens the reaction distance, but at high speed the braking dominates, and the real fix is to slow down, brake harder, or see further. The same arithmetic is behind the stopping distances in the Highway Code.

Levels of autonomy

LEVEL	WHO SENSES AND DECIDES?	THE HUMAN'S JOB	EXAMPLE
Teleoperated	The human, through the robot's cameras and controls	Everything except the physical doing	A bomb-disposal robot; a surgical robot arm (the surgeon drives it)
Supervised or shared	The robot handles the details, the human the goals	Choose the target, watch, step in	A drone flying to a waypoint; a robot vacuum with a phone app

LEVEL	WHO SENSES AND DECIDES?	THE HUMAN'S JOB	EXAMPLE
Autonomous	The robot	Set the task, maintain it	A warehouse robot moving shelves; a Mars rover between commands

3. The main kinds of robot

Robots are named by how they move or what they do. Each type is a different answer to the same question: how do you get a controllable body to move through (or work on) the world?

TYPE	HOW IT MOVES	STRENGTHS	WEAKNESSES	EXAMPLE
Arm (manipulator)	A fixed base; a chain of joints and links moves a hand or tool	Fast, precise, strong; mature technology	Cannot go anywhere; limited reach	Factory welding or pick-and-place arm (Robotics Fundamentals 9)
Wheeled	Wheels or tracks on flat or rough ground	Simple, efficient, cheap, stable; the easiest to build	Stairs, steps and soft ground defeat it	Robot vacuum, line follower, warehouse rover (Robotics Fundamentals 4)
Legged	Legs that lift and place the feet	Rough terrain, stairs, steps	Complex, costly in energy, harder to control	Four-legged inspection robots, hexapod kits
Aerial (drone)	Propellers that hold the body in the air	Fast, unaffected by ground obstacles, wide view	Short endurance, constantly fighting gravity, small payload	Quadcopters (multirotors)
Humanoid	Two legs, two arms, a head: the human shape	Fits spaces and tools designed for people	The hardest balance, the highest cost and complexity	Research and developing commercial humanoids
Others	Swimming, crawling, climbing, rolling	Suited to one environment	Narrow	Underwater vehicles, pipe crawlers, snake robots

WHY THIS COURSE CONCENTRATES ON WHEELS AND ARMS

> A wheeled robot and a small arm teach almost every idea the other types use (drives, sensors, feedback, control, behaviour, power) with the least cost and the fewest things to go wrong. Legs, rotors and the humanoid shape are variations on those ideas with harder mechanics and control. You build the skill on the simple types first.

4. Degrees of freedom

The **degrees of freedom** (DOF) of a body or mechanism is the number of independent numbers needed to say where every part of it is. It is the first number to ask about any robot, because it says how many things the robot can do, how many motors it needs, and how hard the control will be.

A free body

- A rigid body on a flat floor (a plane) has **3** DOF: two for its position (x and y) and one for its heading (θ).
- A rigid body in space has **6** DOF: three positions (x, y, z) and three rotations (roll, pitch, yaw).

A drone is a 6-DOF body. A robot arm's hand needs 6 DOF to be put at any position *and* any orientation within reach, which is why industrial arms usually have six joints. An arm with fewer joints cannot reach every pose; one with seven or more is *redundant* (it has spare freedom, useful for avoiding obstacles).

Counting a mechanism: Gruebler's formula

A mechanism is a set of rigid **links** connected by **joints**, with one link fixed (the ground). Each link added is a free body; each joint removes freedom. For a mechanism that lies in a plane, **Gruebler's formula** (also called the Chebyshev-Gruebler-Kutzbach criterion) counts the result:

```
Planar mechanism:        M = 3 (n - 1) - 2 j1 - j2

  n  = number of links, INCLUDING the fixed ground link
  j1 = number of joints that allow one motion (revolute: a hinge; prismatic: a slider)
  j2 = number of joints that allow two motions (a cam, a rolling-and-sliding contact; r
  M  = degrees of freedom (the number of independent motors needed to drive the mechani

Why:  each moving link is a planar body with 3 DOF, so (n - 1) links give 3 (n - 1); ea

Spatial mechanism (all joints revolute or prismatic):        M = 6 (n - 1) - 5 j1
```

```
Worked examples:
  Two-link planar arm (ground, upper arm, forearm; 2 hinges):          n = 3, j1 = 2:
  Three-link planar arm (ground + 3 links; 3 hinges):                  n = 4, j1 = 3:
  Four-bar linkage (ground, crank, coupler, rocker; 4 hinges):         n = 4, j1 = 4:
  Triangle of three links (3 hinges):                                  n = 3, j1 = 3:
  Six-joint spatial arm (ground + 6 links; 6 hinges):                  n = 7, j1 = 6:
```

GRUEBLER'S FORMULA COUNTS, IT DOES NOT CHECK

The formula assumes the constraints are independent. Some special arrangements (for example a parallelogram linkage whose links happen to be equal and parallel) can move more than the formula says, so treat the result as a first count and confirm with a sketch or a model. It is also not the way to count a wheeled robot, whose wheels roll on the ground: the table below covers that case.

Degrees of freedom of robots

For a robot we separate two numbers: the DOF of the whole body in the world (how it can be placed), and the number of **actuators** (motors) it has. When actuators are fewer than the body's DOF, the robot is **underactuated**.

ROBOT	BODY DOF	ACTUATORS	COMMENT
Two-wheeled differential drive on the floor	3 (x, y, θ)	2 (one per wheel)	It can reach any pose but cannot slide sideways: a *non-holonomic* constraint. Parking takes a manoeuvre (Robotics Fundamentals 4).
Omni-wheel or mecanum platform	3	3 or 4	Can move in any direction at once, at the cost of complex wheels (*holonomic*).
Six-joint industrial arm	6 (the hand)	6	Fully actuated: one motor per DOF. A gripper adds one more.
Quadcopter	6	4 (rotors)	Underactuated: it cannot move sideways without first tilting, and the controller handles that for you.
Quadruped with 3 joints per leg	6 for the body + 12 joints = 18	12	The body's own 6 DOF are not directly driven: they are controlled through the feet's push on the ground.

ROBOT	BODY DOF	ACTUATORS	COMMENT
Humanoid (a typical research design)	6 for the trunk + the joints (often 20 to 30)	20 to 30	Many actuators, and balance on two feet is the hardest part.

More DOF means more capability, and more cost, weight, wiring, power and control effort. A good robot has just the freedom the task needs: a line follower has two driven wheels, not six.

5. Autonomy, endurance and availability

Autonomy means little if the robot is flat. The energy budget of Basic Electronics 11 (E = V × C, usable energy 0.8 × E, runtime = energy / power) applies to every robot, and two more numbers follow from it: how long a robot works on a charge, and what fraction of the time it is ready.

```
Runtime:           t(run) = E(usable) / P            E(usable) = 0.8 x V x C
Availability:      A = t(run) / ( t(run) + t(recharge or battery swap) )        the fr

Illustrative figures (typical orders of magnitude, from memory; real robots vary widely
       Wheeled rover      average 20 W      t(run) = 40 / 20  = 2.0 h   (120 min)
       Legged robot       average 80 W      t(run) = 40 / 80  = 0.5 h   (30 min)
       Quadcopter hover   average 200 W     t(run) = 40 / 200 = 0.2 h   (12 min)

Availability:
       Rover, 1 h to recharge:                    A = 2 / (2 + 1)        = 0.667   (66
       Quadcopter, 45 min to recharge the pack:   A = 12 / (12 + 45)     = 0.211   (21
       Quadcopter, swapping to a charged pack in 2 min:    A = 12 / (12 + 2)  = 0.857

The same battery gives the rover ten times the working time of the drone, because stayi
```

6. A short history

The dates below are from memory and are approximate to the year; they show the order in which the ideas appeared, not exact records.

WHEN	WHAT	WHY IT MATTERED
1921	Karel Capek's play *R.U.R.* introduces the word "robot" (from the Czech *robota*, forced labour; the word is credited to his brother Josef)	The name and the long-lasting idea of artificial workers
Late 1940s	W. Grey Walter's "tortoises" in Bristol: small wheeled robots with a light sensor, a bump sensor and a few vacuum-tube circuits	Showed that a few simple reactions could produce lifelike behaviour (seeking light, returning to a charger)
1950s to 1961	George Devol patents a programmable arm; the Unimate, built with Joseph Engelberger, starts work in a General Motors plant in 1961	The first industrial robot: a repeatable, programmable arm
Late 1960s to early 1970s	"Shakey" at the Stanford Research Institute: a wheeled robot with a camera and a range finder that planned its own moves	An early robot that sensed, planned and acted, and a source of path-planning ideas still in use
1970s	Electric arms such as the Stanford Arm and the PUMA; the first humanoid-style research robots in Japan	Smaller, electrically driven, computer-controlled arms became the industrial standard
1997	NASA's Sojourner rover explores Mars (Mars Pathfinder mission)	A robot working millions of kilometres from any human; semi-autonomous because radio signals take minutes
2000 to 2005	Honda's ASIMO humanoid walks in public; the iRobot Roomba (2002) brings robots into homes; a Stanford vehicle wins the DARPA Grand Challenge (2005)	Walking, mass-market cleaning robots and long-distance driverless vehicles all became practical
2010s to now	Large warehouse fleets (Amazon bought Kiva Systems in 2012), commercial legged robots, cheap drones, low-cost kits and open-source software	Robots in logistics and inspection, and robotics reachable by hobbyists and students

7. Anatomy of a small robot

Here is a small two-wheeled robot of the kind you will design on paper through this course and build in Robotics Fundamentals 10: a chassis carrying two geared motors, a controller board, an ultrasonic distance sensor, three line sensors, two wheel encoders and a battery. Look at its parts as the three steps of the loop plus the power that feeds them.

```
SENSE                  THINK                  ACT
distance sensor ---+                        +--> left motor driver --> left motor
line sensors (3) --+                        |
wheel encoders (2) +--> microcontroller ------+
battery monitor ---+    (Pico-class board)    |
                                              +--> right motor driver --> right motor
                                              |
                                              +--> status LED and buzzer

POWER
battery --> fuse --> switch --+--> 5 V regulator ------> controller board and sensors
                              |
                              +--> motor drivers (battery voltage) --> motors
```

Each subsystem and where it is covered

SUBSYSTEM	WHAT IT DOES	PARTS IN THE SMALL ROBOT	COVERED IN
Structure	Holds everything in place, takes the loads, sets the stability	Chassis plate, motor mounts, standoffs, fasteners, a caster	Robotics Fundamentals 2
Actuation and transmission	Turns electrical energy into motion with the right torque and speed	Two geared DC motors, wheels	Robotics Fundamentals 3, with the motor model from Basic Electronics 9
Drive system	Arranges the wheels so that the robot can steer and its motion can be calculated	Differential drive (two powered wheels, one free caster)	Robotics Fundamentals 4
Sensors	Measure the world and the robot itself	Ultrasonic distance, line sensors, bumper switches	Robotics Fundamentals 5, with ADC and analogue signals from Basic Electronics 7
Position and feedback	Tell the robot how far and how fast it moved, and which way it faces	Wheel encoders; an optional inertial sensor	Robotics Fundamentals 6
Control	Makes a motor or the robot follow a target by	A speed loop on each wheel, a steering loop for	Robotics Fundamentals 7

SUBSYSTEM	WHAT IT DOES	PARTS IN THE SMALL ROBOT	COVERED IN
	correcting the error	the line	
Behaviour	Chooses what to do now: follow the line, avoid an obstacle, stop	A small state machine in the program	Robotics Fundamentals 8
Manipulator (if fitted)	Reaches out and handles objects	Not on this robot; a servo arm could be added	Robotics Fundamentals 9
Power and safety	Supplies energy, protects people and parts	Battery, fuse, switch, regulator, battery monitor, an emergency stop	Robotics Fundamentals 10, with Basic Electronics 9 and Basic Electronics 11
The controller	Runs the loop	A Pico-class board (Basic Electronics 10 and Basic Electronics 11 explain the choice)	Programmed in the planned course Programming for Robotics

Notice how the subsystems depend on each other: the motors chosen in Robotics Fundamentals 3 decide how heavy the battery must be (Robotics Fundamentals 10), which changes the weight the structure carries (Robotics Fundamentals 2). Robot design is mostly a loop of its own: choose, calculate, discover a conflict, change. This course keeps one small robot in mind through every chapter so that those links are visible.

8. Challenges

CHALLENGE 1: COUNT THE DEGREES OF FREEDOM

Use Gruebler's formula to find the DOF of each mechanism, and say how many motors fully drive it. (a) A planar five-bar linkage (ground and four moving links, joined in a closed loop by five hinges). (b) A slider-crank (ground, crank, connecting rod and a slider that moves along a straight guide: three hinges and one prismatic joint). (c) A planar arm with ground plus three links and three hinges. (d) A spatial arm with ground plus four links and four hinges. (e) A planar triangle (ground plus two links, joined by three hinges) and a planar square frame (ground plus three links, joined by four hinges): which is a rigid structure and which can move? Then state the body DOF, the number of actuators and whether the robot is underactuated for: (f) a quadruped with three joints per leg; (g) a differential-drive robot on the floor.

View solution

CHALLENGE 2: LOOP RATE AND STOPPING DISTANCE

A small robot drives at 0.8 m/s. Its loop runs at 25 Hz, its distance sensor adds 25 ms, the program takes 5 ms and the motors and driver need 30 ms to respond. It can brake at 1.5 m/s^2. (a) Find the worst-case latency, the reaction distance, the braking distance and the stopping distance. (b) The sensor reliably sees 0.5 m ahead. Find the highest speed at which the robot can still stop within that range (solve $v \times t + v^2/(2a) = 0.5$), and then the speed if you allow only half the range (0.25 m) as a safety margin. (c) The loop is rewritten to run at 100 Hz. Find the new latency and the new stopping distance at 0.8 m/s, and say how much distance the faster loop saves compared with the braking distance. (d) In one sentence, say what you would change first to make the robot safer at high speed.

View solution

CHALLENGE 3: ANATOMY, ENDURANCE AND AVAILABILITY

(a) A robot vacuum cleaner: list four things it senses, what its thinking step must decide, and two things it acts with, then name which chapter of this course covers each subsystem. (b) A 36 Wh battery pack gives 0.8 × 36 usable energy. Find the runtime of a 15 W robot vacuum, a 25 W wheeled rover and a 90 W legged robot. (c) The vacuum takes 3 h to recharge; the legged robot has a 1-minute battery swap. Find the availability of each. (d) The legged robot walks at 0.7 m/s and the rover drives at 1.2 m/s. Find the energy each uses per metre and how far each travels on one charge. (e) In one sentence, say what the results show about choosing a legged robot over a wheeled one.

View solution

QUICK REFERENCE

- **A robot:** senses, decides with a program, and acts physically, in a repeating loop closed through the world
- **The loop:** sense, think, act, then wait for the next period; the loop rate is $f = 1/T$
- **Latency:** $t(lat) = t(sensor) + T + t(compute) + t(actuator)$; reaction distance = $v \times t(lat)$; braking distance = $v^2/(2a)$; doubling the speed quadruples the braking distance
- **Types:** arm, wheeled, legged, aerial, humanoid, others; a wheeled robot and a small arm teach most of the ideas

- **DOF:** a rigid body has 3 (plane) or 6 (space); planar mechanisms M = 3(n − 1) − 2 j1 − j2; spatial M = 6(n − 1) − 5 j1; underactuated means fewer actuators than DOF
- **Endurance:** t(run) = 0.8 × V × C / P; availability A = t(run) / (t(run) + t(recharge or swap))
- **Anatomy:** structure (2), motors and gears (3), drive (4), sensors (5), feedback (6), control (7), behaviour (8), arms (9), power, safety and the build (10)

Next, Robotics Fundamentals 2 turns to the body of a robot: frames and materials, levers, linkages and joints, why centre of gravity decides whether a robot tips over, and how 3D-printed and laser-cut parts are designed and joined with fasteners.

Robotics Fundamentals

Course 1 · Chapter 2 · Mechanics and Structure

Robotics Fundamentals 1 gave the overall picture and put *structure* first in the list of subsystems. This chapter looks at it properly. The structure of a robot holds the motors, sensors, board and battery in the right places, carries their loads without bending too much, and decides whether the robot stays upright. Most first robots that fail do so for mechanical reasons: a frame that flexes so the wheels rub, a battery that puts the weight in the wrong place so the robot tips backwards on a ramp, a screw that strips out of a printed part. All of it can be checked with arithmetic before anything is cut or printed.

The material values below (stiffness, density, yield strength) and the fastener figures are **typical, approximate values from memory**. Real materials vary by grade, supplier and, for printed parts, by print settings, so use the numbers to compare options and to check the order of size, and read the datasheet before relying on one. The formulas themselves are standard.

1. Frames and materials

A frame has three jobs: **locate** parts (motors in line, sensors facing the right way), **carry loads** (weight, motor thrust, a knock against a wall) and **stay stiff enough** that parts do not move relative to each other. Weight matters too: every gram has to be carried by the motors and paid for by the battery (Basic Electronics 11 gave the energy side of that). The design question is always how much stiffness and strength you get for each gram.

Typical materials

MATERIAL	STIFFNESS E (GPA)	DENSITY (KG/M³)	YIELD OR STRENGTH (MPA)	E / DENSITY (MN·M/KG)	IN ROBOTS
Aluminium alloy (6061-T6)	69	2700	about 275 (yield)	25.6	Plates, angle, tube, arms and brackets. Light, easy to cut and drill, does not rust.
Mild steel	200	7850	about 250 (yield)	25.5	Shafts, big frames, anything that must be cheap and stiff and

MATERIAL	STIFFNESS E (GPA)	DENSITY (KG/M³)	YIELD OR STRENGTH (MPA)	E / DENSITY (MN·M/KG)	IN ROBOTS
					weight does not matter. Rusts unless coated.
PLA (3D printed)	about 3.5	about 1250	about 50, lower between layers	2.8	The easy plastic to print. Stiff but brittle, and softens at around 55 to 60 °C, so keep it away from hot motors and cars in summer.
ABS or PETG (printed)	about 2 to 2.5	about 1050 to 1270	about 40 to 50	about 2 (ABS)	Tougher and more heat-resistant than PLA; ABS is harder to print, PETG easier.
Acrylic (cast PMMA sheet)	about 3	about 1190	about 70, brittle	2.5	Laser-cut chassis plates. Cracks from sharp corners and over-tight screws.
Birch plywood	about 9 along the face grain	about 680	varies with grade and direction	13.2	Laser-cut or hand-cut plates; cheap, forgiving, good stiffness for its weight, best kept dry.
Carbon-fibre laminate	about 60 (plate with fibres in several directions); about 130 along the fibres of a unidirectional layer	about 1600	several hundred, brittle	about 37.5 (plate)	Tubes and plates where low weight is worth the cost (arms, drones). Conducts electricity: keep it away from bare wiring.

The last numeric column, E / density, is the stiffness you get per kilogram of material. Notice that aluminium and steel are almost identical on it, even though steel is three times heavier: for a given mass you can have a steel part or an aluminium part about three times as thick, and thickness is what stiffness depends on most (next section). Plastics are about ten times worse

than metals per kilogram, so a plastic part must be made thicker or ribbed to match. Carbon fibre is the one that beats the metals.

> **CHANGE THE SHAPE BEFORE YOU CHANGE THE MATERIAL**
>
> How a part is shaped usually matters more than what it is made of. A beam's bending stiffness depends on E and on the shape through I, and I grows with the cube of the depth. Making a flat strip stand on edge, adding a rib or folding a flange gives a gain that switching material rarely can. The example below shows a factor of 44 from turning a bar on its side.

Beam bending: the cantilever

Most robot structure is some form of **cantilever**: a beam fixed at one end and loaded at the other (an arm held out from a motor, a sensor mast, a bracket holding a motor). Two things go wrong with it. It can **deflect** too much (the sensor points at the floor, the arm droops), or it can **yield** (bend permanently or break). The formulas give both:

```
Cantilever, length L, load F at the free end, material stiffness E, second moment of ar

  Tip deflection:            delta = F L^3 / (3 E I)
  Rectangular section:       I = b h^3 / 12          (b = width, h = depth measured IN
  Bending moment at the wall:  M = F L
  Greatest stress (outer face at the wall):   sigma = M c / I = (F L)(h / 2) / I = 6 F
  Safety factor:             SF = yield strength / sigma

  Cross-section orientation (the same bar, 20 mm x 3 mm; load acts downwards):

       FLAT  (load across the 3 mm)                ON EDGE  (load along the 20 mm)

       +--------------------+                          +---+
       |                    |  h = 3 mm                |   |
       +--------------------+                          |   |
             b = 20 mm                                 |   |   h = 20 mm
                                                       |   |
       I = 20 x 3^3 / 12 = 45 mm^4                     |   |
                                                       +---+
                                                        b = 3 mm
                                                  I = 3 x 20^3 / 12 = 2000 mm^4

  Beam sketch:       ||============================+
```

```
                    ||  fixed        length L       |
                    ||                              v  F                    (the free
```

Two scalings are worth remembering. Deflection grows with the **cube of the length** (twice as long: eight times the droop) and falls with the **cube of the depth** (twice as deep: one eighth). Stress grows with the length but falls with the *square* of the depth.

```
Worked example: a sensor arm made from 6061 aluminium flat bar, b = 20 mm, h = 3 mm, L =
Use E = 69 GPa and yield = 275 MPa (typical values).

  Flat (h = 3 mm):
       I     = 0.020 x 0.003^3 / 12 = 4.5e-11 m^4
       delta = F L^3 / (3 E I) = 5 x 0.003375 / (3 x 69e9 x 4.5e-11) = 0.016875 / 9.315
       sigma = 6 F L / (b h^2) = 6 x 5 x 0.15 / (0.020 x 0.003^2) = 4.5 / 1.8e-7 = 25 M
       SF    = 275 / 25 = 11            (very safe against yielding, yet the tip still

  The same bar turned on edge (b = 3 mm, h = 20 mm):
       I     = 0.003 x 0.020^3 / 12 = 2.0e-9 m^4        (44.4 times larger, because (20/
       delta = 0.016875 / (3 x 69e9 x 2.0e-9) = 0.0000408 m = 0.041 mm
       sigma = 6 x 5 x 0.15 / (0.003 x 0.020^2) = 4.5 / 1.2e-6 = 3.75 MPa       SF = 73

  The same flat bar twice as long (L = 0.30 m):  delta = 1.81 x 8 = 14.5 mm,   sigma = 

  The same flat bar, printed in PLA instead (E = 3.5 GPa):  delta = 1.81 x (69 / 3.5) =
       which against PLA's roughly 50 MPa leaves SF of only about 2, and less across th

  Mass of the flat bar:  aluminium 0.020 x 0.003 x 0.15 x 2700 = 24.3 g;     PLA  0.020
```

SAFETY FACTORS ARE NOT DECORATION

The formulas give the stress in an ideal beam under a known, steady load. Real robots get dropped, driven into walls and loaded at the corner of a bracket where stress concentrates. A common rule of thumb is a safety factor of about 2 to 3 for steady loads on ductile metal, and 5 or more for brittle materials (acrylic, carbon fibre, PLA) or for loads that arrive as knocks. Check also what gives way first: it is usually a joint, a bolt hole or a thin printed neck, not the long bar.

2. Levers, linkages and joints

Moment (torque) balance

The turning effect of a force is its **moment**, also called torque: the force multiplied by the *perpendicular* distance from the pivot to the line of the force. A body that is not turning has its moments balanced, and one that is not moving has its forces balanced. These two conditions solve most mechanics problems on a robot.

```
Moment of a force about a pivot:          M = F x d               (d = perpendicular di
Balance (equilibrium):                   sum of moments about any point = 0         sum
Lever mechanical advantage:              MA = d(effort) / d(load)                   F(e

  Simple lever (effort and load on opposite sides of the pivot):

       d(effort) = 0.30 m                              d(load) = 0.05 m
  effort  |                                                 | load (a weight W)
      v   |                                                 v
  ========O=================================================^==   (O is the pivot)

  The trade: a lever gives a larger force at the cost of a smaller movement. The effort

Worked example 1: a 2 kg load hangs 0.05 m from the pivot of a lever, and you push 0.30
       W = m g = 2 x 9.81 = 19.62 N          MA = 0.30 / 0.05 = 6          F(effort) = 

Worked example 2: a hobby servo (stall torque about 1.2 N m, taken as a typical figure)
       with its weight acting at its middle (0.10 m).
       Torque from the arm itself:        0.05 x 9.81 x 0.10 = 0.049 N m
       Torque left for a load at the tip:  1.2 - 0.049 = 1.151 N m
       Load force at the tip:             F = 1.151 / 0.20 = 5.76 N          m = 5.76 /
       A servo should not run at its stall torque for long, so a design should keep wel

Worked example 3: play in a joint grows with the length of what is attached to it.
       0.5 degrees of free play in a joint, measured at the end of a 0.20 m arm:     0.
```

Linkages

A **linkage** is a set of rigid links joined by joints so that one input motion produces a controlled output motion. Gruebler's formula from Robotics Fundamentals 1 counts the freedom: the four-bar linkage has one, so one motor drives it. The four-bar is the workhorse: it appears in grippers (a servo opens two jaws), in parallelogram arms that keep a platform level while it lifts, and in the legs of walking robots.

```
Four-bar linkage:   ground link (fixed), crank (driven by the motor), coupler, rocker (

                  coupler
             B o===============o C
              /                 \
      crank  /                   \  rocker
            /                     \
         A o=======================o D       A and D are pivots on the ground link
                    ground

Grashof's rule: let s = the shortest link, l = the longest, p and q = the other two.
       If  s + l  <=  p + q,  then the shortest link can turn through a full circle rel
       If the shortest link is next to the ground (a driven crank), the result is a cra
Worked example: ground 90 mm, crank 40 mm, coupler 100 mm, rocker 80 mm.   s = 40, l =
       The 40 mm crank is the shortest link and is next to the ground, so it can turn c

Mechanical advantage of a linkage changes through the motion.  Near the end of a stroke
gives a very large output force (a toggle). A gripper uses this to clamp firmly without
```

Joints

JOINT	ALLOWS	TYPICAL CONSTRUCTION	WATCH FOR
Revolute (hinge)	Rotation about one axis	A shoulder screw or pin through the links, a bushing, a ball bearing, or the output shaft of a servo or motor	Play (slop) and friction: a loose pin gives wobble at the tip, an over-tight one binds
Prismatic (slider)	Sliding along one axis	Linear rail and carriage, round rods in bushings, a lead screw	Binding if the two rods are not parallel; dirt on the rails
Ball joint	Rotation about three axes	Ball-and-socket, used on pushrods and link ends	Slack, and it adds freedom you may not want
Flexure (living hinge)	A small bend, from a thin section	A thin band left in a printed or cut part	Fatigue: it cracks after many bends in a brittle plastic
Fixed (rigid)	Nothing	Screws, glue, a clamp	A joint you meant to be rigid but that flexes is the usual cause of a floppy frame

Wherever a joint turns, **friction** costs torque and **play** costs accuracy. A ball bearing is a bit more costly than a plain bushing, but it turns with very little friction and little play. Where a robot has to repeat a motion accurately (an arm picking up a block), put the effort into rigid links and tight, well-supported joints before a better motor.

3. Centre of gravity and stability

All the weight of a robot can be treated as acting at a single point, its **centre of gravity** (CoG). Where that point is decides whether the robot stays on its wheels. For a robot made of several parts, find it by taking moments: it is the **weighted average** of the positions of the parts, with mass as the weight.

```
Centre of gravity of parts i = 1, 2, 3 ... with masses m(i) at positions x(i) (along th

       M = sum of m(i)            x(cg) = sum of m(i) x(i) / M            z(cg) = sum o
       (the same idea works for y, across the robot)

Worked example: the small two-wheeled robot, x measured forward from the drive axle, z
The caster (a free-swivelling ball or wheel at the front) touches the ground at x = +0.

  Part                        m (kg)     x (m)     z (m)     m x         m z
  Chassis plate                0.20      0.000     0.030     0.00000     0.00600
  Two motors and gearboxes     0.12      0.000     0.032     0.00000     0.00384
  Two wheels                   0.06      0.000     0.032     0.00000     0.00192
  Battery pack                 0.25     -0.040     0.045    -0.01000     0.01125
  Controller board             0.05      0.020     0.060     0.00100     0.00300
  Caster                       0.03      0.090     0.012     0.00270     0.00036
  Front sensor board           0.02      0.090     0.020     0.00180     0.00040
  Total                        0.73                         -0.00450     0.02677

       x(cg) = -0.00450 / 0.73 = -0.00616 m     (6 mm BEHIND the drive axle)
       z(cg) =  0.02677 / 0.73 =  0.03667 m     (36.7 mm above the floor)
```

The support polygon

A robot is stable while the CoG, projected straight down onto the floor, lies **inside the support polygon**: the shape formed by joining the points where the robot touches the ground. Two wheels and a caster make a triangle; four wheels make a rectangle. The result for the example is bad news:

```
Top view (the floor). Forward is up the page.

                      C          caster contact          (x = +0.090 m)
                     /|\
                    / | \
                   /  |  \
                  /   |   \
                 /    |    \
        x = 0    L-----+-----R    wheel contacts          (x = 0, y = -0.060 and +0.0
                       *          CoG, as first calculated  (x = -0.006 m: just OUTSI

The CoG is behind the axle line. The robot will sit on its tail and the caster will han

The fix: move the battery forward. What battery position puts the CoG at x = +0.020 m (
       Required total moment: 0.020 x 0.73 = 0.0146.     Moment of everything but the b
       Battery moment needed: 0.0146 - 0.0055 = 0.0091     so   x(battery) = 0.0091 / 0

Check:  x(cg) = (0.0055 + 0.25 x 0.0364) / 0.73 = 0.0146 / 0.73 = +0.0200 m.     Weight
Where the weight is carried (moments about the axle):   caster share = x(cg) / 0.090 = 
(Too little on the caster and it loses contact over bumps; too much and the drive wheel
```

Tipping on a slope, and when it tips instead of slipping

A robot tips over an edge of its support polygon when the line of its CoG, which stays vertical, crosses that edge. On a slope, the line of weight stays vertical while the robot tilts, so it moves relative to the contact points. At the tipping angle, the CoG is directly above the tipping edge. With h the height of the CoG above the ground and d its horizontal distance from the edge it will tip over:

```
Tipping angle on a slope:             tan(theta) = d / h         d = horizontal distanc
       Sideways, across a slope:    d = track / 2  (CoG centred between the wheels):
       Wheels slip instead when:    tan(theta) = mu              (mu = friction coeffi
       The robot tips FIRST if  d / h  <  mu     and slips first if mu  <  d / h.

Tipping or skidding in a straight line or a turn (acceleration a on level ground, the s

Worked example 1: the corrected small robot (track 0.12 m, h = 0.0367 m, x(cg) = +0.020
       Sideways:            tan(theta) = 0.060 / 0.0367 = 1.636      theta = 58.6 deg
       Facing uphill:       it tips backwards about the wheel axle:   d = 0.020       t
       Facing downhill:     it tips forwards about the caster:        d = 0.090 - 0.020
       Wheels slip at:      tan(theta) = 0.8      theta = 38.7 deg
```

```
        So uphill the robot tips backwards at 28.6 deg, BEFORE it would slip (38.7 deg).
        CoG moved forward, or lowered, to climb steeply. Moving the CoG 10 mm forward ad

 Worked example 2: a tall robot (track 0.15 m, CoG height 0.15 m).
        Sideways tipping:     tan(theta) = 0.075 / 0.15 = 0.5            theta = 26.6 de
        Tips in a turn at     a(lateral) = g x (track / 2) / h = 9.81 x 0.5 = 4.9 m/s^2.
        Braking: if the CoG is 0.10 m behind the front contact line, a robot tips forwar
```

FOUR WAYS TO MAKE A ROBOT MORE STABLE

Lower the CoG (heavy parts, mainly the battery, low in the frame), **widen** the track and wheelbase (a bigger polygon), **centre** the CoG in the polygon, and **slow down** or smooth the acceleration (tipping in a turn depends on speed squared). Put the heaviest parts low and between the wheels. A tall mast with a camera or sensor on top raises the CoG a great deal: weigh it as part of the sum above, not as an afterthought.

4. 3D-printed and laser-cut parts

Making your own parts is the great strength of hobby robotics: a plate with exactly the right motor holes, drawn in an afternoon and made in an hour. The two common methods have different strengths. The design rules below are typical values from memory; the limits of your own machine, and of the material you use, are what count.

	3D PRINTING (FILAMENT)	LASER CUTTING (SHEET)
Shape	Almost any 3D shape: brackets, enclosures, wheels, gears	Flat parts, cut out of 3 to 6 mm sheet; 3D shapes by slotting parts together
Materials	PLA, PETG, ABS, nylon, flexible TPU	Acrylic, plywood, MDF, card. Never PVC (it gives off chlorine)
Strength	Directional: weakest between the layers	The sheet is the same in all directions across its plane
Accuracy	About 0.2 mm; holes print undersized	About 0.1 mm, with the kerf (width of the cut) to allow for
Speed and cost	Slow (hours for a big part), cheap in material	Fast (minutes), cheap for flat parts

	3D PRINTING (FILAMENT)	LASER CUTTING (SHEET)
Good for	One-off complex shapes, mounts, covers	Chassis plates, decks, a quick first frame

Design rules for printed parts

- **Print so the load runs along the layers, not across them.** A printed part is much weaker where one layer meets the next, often only about half to two thirds of the strength within a layer. A bracket lying flat is stronger in bending than the same one standing on its edge when the load pulls the layers apart.
- **Thickness is cheap stiffness.** Doubling the thickness of a plate makes it 8 times stiffer (the h^3 rule above) for twice the plastic. Ribs and fillets do the job for less. Round the inside corners of cut-outs to avoid cracks starting there.
- **Walls and infill:** a few perimeters (2 to 4) and 20 to 40 % infill is a common default; the walls carry most of the load, not the infill.
- **Holes print small, by about 0.2 mm.** Print a hole for an M3 screw at 3.4 to 3.6 mm, not 3.0 mm, and check with a test piece. Slots and press fits need testing the same way.
- **Overhangs steeper than about 45 degrees from the vertical need support** or they sag; design a part so that it can print without support where it can.
- **Heat:** PLA softens at around 55 to 60 °C. Motors under load, drivers and sunlit dashboards can reach it; use PETG or ABS there.

Design rules for laser-cut parts

- **Kerf.** The laser removes a strip about 0.1 to 0.2 mm wide. A slot cut to exactly the sheet thickness is a loose fit; draw it a hair narrower, or adjust the drawing by the machine's kerf. Test on scrap.
- **Tab and slot joints.** Parts slot together at right angles and are held with a screw and nut in a captured slot (a nut trap). That gives a stiff box from flat sheet.
- **Keep holes clear of edges** (at least the hole's diameter, better twice) and round the corners, particularly in acrylic.
- **Acrylic cracks.** Do not over-tighten screws, and use washers.

5. Fasteners

The robot's parts are held together mostly by **metric screws**. Their sizes are written M2, M3, M4 and so on: the M means metric and the number is the nominal diameter in millimetres. M3 (3 mm) is the commonest size on small robots. A screw is also described by its **length** (under the head for most types), its **head** (cap head with a hex socket, button head, countersunk) and its **property class** (8.8 is a common steel grade for general use; stainless steel A2-70 is common too).

ITEM	WHAT IT DOES	NOTE
Machine screw and nut	Clamps parts together through clearance holes	The normal choice for metal and acrylic; a washer under the head and under the nut spreads the load on plastic
Nyloc nut	A nut with a nylon ring that grips the thread	Resists vibration without a threadlocker; do not reuse many times
Standoff (spacer)	Holds two plates apart at a fixed distance, or mounts a board off a plate	Threaded hex standoffs in M3 are common; a Pico-class board uses M2 screws through holes of about 2.1 mm
Heat-set insert	A brass threaded insert pushed into a printed hole with a hot iron	Gives a strong metal thread in a plastic part that can be unscrewed many times
Self-tapping screw	Cuts its own thread in plastic or thin sheet	Fine for light, rarely removed parts; strips out when overused
Threadlocker	Liquid that hardens in the thread and stops it vibrating loose	Blue is medium-strength (can be undone with tools); red is permanent. Use on metal threads only: it can attack some plastics

Clearance holes and tolerances

A **tolerance** is the allowed variation in a size. A screw has to pass through a hole that is a little bigger than it is, so that parts can be assembled even if holes are slightly out of position. The standard clearance diameters for metric screws (from memory of the ISO 273 table) are:

```
Screw    Close (fine) hole    Normal (medium) hole      Tap-drill (for cutting a thread
M2       2.2 mm               2.4 mm                    1.6 mm
M3       3.2 mm               3.4 mm                    2.5 mm
M4       4.3 mm               4.5 mm                    3.3 mm

Use the normal hole by default. Use the close hole where two parts must locate accurate
```

```
Stack-up: tolerances ADD. Two parts joined by a screw, each with a hole position out by
much bigger than the screw. Slotted holes let you adjust a motor mount's position.
```

Screws: shear, preload and tightening (approximate)

Three simple estimates tell you whether a bolted joint is sensible. These are rough guides: the real values depend on the grade, the thread, lubrication and the materials joined, and the margin you leave should reflect that.

```
Tensile stress area (the effective area of the threaded part):        A(t) = (pi / 4)(d
Shear strength (approximate):     about 0.6 x ultimate tensile strength.    Class 8.8: u
Single-shear capacity:            F(shear) = 0.6 x 800 x A(t)
Preload (the clamping force from tightening), aimed at about 70 % of the proof load:
Tightening torque (approximate):  T = K F(pre) d        K about 0.2 for a dry thread (th
Bearing stress on a plastic wall:  F / (d t)            d = hole diameter, t = wall thic

  Screw    pitch p     A(t)         F(shear), 8.8        F(pre)         T
  M2       0.4 mm      2.07 mm^2     995 N               842 N         0.34 N m
  M3       0.5 mm      5.03 mm^2    2415 N              2043 N         1.23 N m
  M4       0.7 mm      8.78 mm^2    4214 N              3564 N         2.85 N m

Worked example: A(t) for M3 = (pi / 4)(3 - 0.9382 x 0.5)^2 = (pi / 4)(2.5309)^2 = 5.03 m
       F(pre) = 0.7 x 580 x 5.03 = 2043 N;     T = 0.2 x 2043 x 0.003 = 1.23 N m.

Now a motor bracket in 3 mm printed PLA, held by M3 screws through 3 mm holes. Say the p
       F(bearing) = 20 MPa x (3 mm x 3 mm) = 20 x 9 = 180 N per screw
       The screw itself takes 2415 N in shear: 13 times more. THE PLASTIC FAILS FIRST, n
So in plastic you do not tighten to 1.2 N m: the plastic would crush or the thread strip
undone, use a heat-set insert. The 1.23 N m figure is for steel screws in metal.
```

THREAD ENGAGEMENT

A screw that is screwed only a turn or two into a threaded hole can strip. As a rule of thumb the engagement should be at least 1 to 1.5 times the screw diameter in steel and about 1.5 to 2 times in aluminium; plastic threads are weak, so use an insert. Leave the screw ends slightly short of the other side of the hole, or check that they do not hit a wire.

6. Challenges

CHALLENGE 1: DEFLECTION, STRESS AND SHAPE

A sensor arm is a 6061 aluminium flat bar, 25 mm wide and 4 mm thick, 0.20 m long, fixed at one end and carrying a load of F = 8 N at its tip (E = 69 GPa, yield 275 MPa). (a) Mounted flat (4 mm in the direction of the load), find I, the tip deflection, the greatest stress and the safety factor. (b) Turned on edge (25 mm in the direction of the load), find the same quantities, and the factor by which the deflection falls. (c) The flat bar is lengthened to 0.30 m: find the new deflection, stress and safety factor. (d) The flat 0.20 m bar is replaced by the same section in ABS (E = 2.2 GPa): by what factor does the deflection grow? (e) For the flat 0.20 m bar, find the largest tip load that keeps the deflection under 1 mm.

View solution

CHALLENGE 2: CENTRE OF GRAVITY AND TIPPING

A small robot has these parts (x forward from the drive axle, z above the floor): chassis 0.25 kg at x = 0, z = 0.030 m; motors 0.14 kg at 0, 0.032; wheels 0.08 kg at 0, 0.033; battery 0.30 kg at x = −0.020, z = 0.040; controller 0.06 kg at 0.030, 0.065; sensor mast 0.05 kg at 0.120, 0.090; caster 0.03 kg at 0.110, 0.010. The caster touches the ground at x = 0.110 m and the track (distance between the wheels) is 0.14 m. (a) Find the total mass and the CoG (x and z). Is the CoG inside the support polygon, and what share of the weight does the caster carry? (b) Find the tipping angle sideways, and the tipping angle facing uphill and downhill. The tyres have mu = 0.8: which happens first facing uphill, tipping or slipping? (c) Find the acceleration at which the robot lifts its front (a = g x(cg) / h) when accelerating forward. (d) Where must the battery go (same height) for the CoG to be at x = +0.030 m? Find the new uphill tipping angle and the new caster share. (e) Name two other ways to improve the uphill stability.

View solution

CHALLENGE 3: A SERVO ARM AND ITS SCREWS

A servo with a stall torque of 1.5 N m holds out an arm 0.25 m long. The arm weighs 0.08 kg (weight acting at its middle) and carries a gripper of 0.05 kg at the tip. To avoid running near stall, the design may use only 60 % of the stall torque. (a) Find the torque the servo needs just to hold the arm and gripper horizontal. (b) Find the heaviest payload (in kg) the gripper may carry at the tip. (c) The arm joint has 0.7 degrees of free play. Find the wobble at the tip. (d) An M3 class 8.8 screw (pitch 0.5 mm) holds the arm bracket. Find its tensile stress area, its single-shear capacity (0.6 x 800 MPa), its preload (70 % of 580 MPa) and the approximate tightening torque (K = 0.2). (e) The bracket is 3 mm printed

PLA with a 3 mm hole, and the plastic can bear about 20 MPa. Find the bearing capacity of one screw hole, compare it with the screw's shear capacity, and say which fails first and how the bracket should be assembled.

View solution

QUICK REFERENCE

- **Materials:** aluminium and steel have about the same E/density; carbon fibre is better, plastics about ten times worse; change the shape before the material
- **Cantilever:** deflection = $F L^3 / (3 E I)$; rectangle $I = b h^3 / 12$ (h along the load); stress = $6 F L / (b h^2)$; safety factor = yield / stress
- **Scaling:** deflection $\propto L^3$ and $1/h^3$; stress $\propto L$ and $1/h^2$; a bar on edge is $(h/b)^2$ times stiffer than flat
- **Moments:** $M = F d$; balance: the sum of moments = 0; lever MA = d(effort) / d(load); play at the tip = L tan(angle)
- **Linkages:** four-bar has 1 DOF; Grashof: $s + l \leq p + q$ lets the shortest link turn fully
- **CoG:** $x(cg) = \Sigma m x / \Sigma m$, $z(cg) = \Sigma m z / \Sigma m$; stable while the CoG is over the support polygon
- **Tipping:** $\tan(\theta) = d / h$ (sideways d = track/2); slips first if $\mu < d/h$; tips at $a = g d / h$
- **Parts:** load along the printed layers; add about 0.2 mm to printed holes; allow for laser kerf; PLA softens near 55 to 60 °C
- **Screws:** $A(t) = (\pi/4)(d - 0.9382\ p)^2$; shear about 0.6 × UTS × A(t); preload about 0.7 × proof × A(t); $T = 0.2 F d$; in plastic the plastic fails first, so use washers and inserts

Next, Robotics Fundamentals 3 turns to motion: how a motor's torque, speed and power are chosen, how gears and gearboxes trade speed for torque, how belts, chains and wheels deliver it to the ground, and how to size a motor for the robot you have just designed the structure of.

Robotics Fundamentals

Course 1 · Chapter 3 · Motion: Motors, Gears and Wheels

Robotics Fundamentals 2 designed the structure and moved the battery until the robot stayed on its wheels. This chapter makes it move. A robot's motion system has a chain of parts: a motor that turns electrical energy into rotation, a gearbox (or belt) that trades speed for torque, and a wheel that turns torque into a push against the ground. Each link has a number attached to it, and the design task is to choose them so that the robot reaches the speed you want, climbs the slope you want, and does not overheat the motor while doing it. Basic Electronics 9 covered the motor from the electrical side (the model, PWM, drivers). Here we meet the same motor from the *mechanical* side, and then size one for a small robot, with all the arithmetic shown.

The efficiencies, friction coefficients, rolling-resistance coefficients and motor constants in this chapter are **typical, approximate values from memory**. Real gearboxes, tyres and motors vary by make, wear, temperature and load, so use the figures to compare options and to check the order of size, and then read the datasheet (or measure) before relying on one. The formulas themselves are standard.

1. Torque, speed and power

Torque (τ) is a turning force: a force multiplied by the perpendicular distance from the axis (Robotics Fundamentals 2 called it a moment). **Angular speed** (ω) is how fast something turns. **Power** is the rate at which energy is delivered, and for a rotating shaft it is the product of the two. Everything else in this chapter follows from three relations and the fact that a gearbox cannot create power.

```
Torque from a force at a radius:       tau = F r                (N m = newton x metre)
Angular speed from rpm:                omega = 2 pi n / 60      (rad/s; n in revolution
Rotational power:                      P = tau omega            (W = N m x rad/s)
Linear power (the same power):         P = F v                  with  v = omega r

Units you will meet on datasheets:
       1 kg cm  = 0.0981 N m          1 N m = 10.2 kg cm
       1 oz in  = 0.00706 N m         1 N m = 141.6 oz in
       Datasheets often quote "torque" in kg cm, which means "kilogram-force x centimet

Worked examples:
```

```
        6000 rpm   -> omega = 2 pi x 6000 / 60 = 628.3 rad/s
        120 rpm    -> omega = 2 pi x 120 / 60  = 12.57 rad/s
        A motor delivering 0.01 N m at 6000 rpm:   P = 0.01 x 628.3 = 6.28 W
        A wheel of radius 0.04 m pushing the robot with F = 5 N:   tau = 5 x 0.04 = 0.20
        The same wheel at 120 rpm:   v = omega r = 12.57 x 0.04 = 0.503 m/s,   P = F v =
        A servo rated 6 kg cm at its horn:   6 x 0.0981 = 0.589 N m
```

SPEED AND TORQUE TRADE; POWER DOES NOT APPEAR FROM NOWHERE

A motor of a given power can give high speed and low torque, or low speed and high torque, depending on the gearing. A gearbox only *exchanges* one for the other (and loses a little as heat). This is why a small fast motor can drive a robot: gear it down. It is also why a robot that "has plenty of motor" but the wrong gearing still cannot climb a ramp or reach the speed you want.

2. DC motor characteristics (the mechanical view)

The torque-speed line

A brushed DC motor (Basic Electronics 9) has two extreme operating points. With nothing on the shaft it runs at its **no-load speed** ω_{nl} and gives no useful torque. Held still (stalled) it gives its **stall torque** τ_s and draws its **stall current** $I_s = V / R$. In between, to a good approximation, torque falls in a **straight line** as speed rises. The motor's behaviour is described by two constants, k_t (torque per ampere, in N m/A) and k_e (back-EMF per rad/s, in V s/rad), which have the same value in SI units.

```
Electrical model (Basic Electronics 9):     V = k(e) omega + I R                tau = k(

Stall (omega = 0):      I(s) = V / R            tau(s) = k V / R
No load (tau = 0):      omega(nl) = V / k       (in a real motor a little current I(0)
Torque-speed line:      tau = tau(s) (1 - omega / omega(nl))
Mechanical power:       P = tau omega = tau(s) omega (1 - omega / omega(nl))        a p
Peak power:             at omega = omega(nl) / 2 and tau = tau(s) / 2:              P(m
Efficiency:             eta = P(mech) / P(elec) = tau omega / (V I)

Torque (N m)
0.018 |*                                  tau(s) = k V / R = 0.006 x 6 / 2.0
      | *
      |  *
      |   *
```

```
        |    *
  0.009 |     *          peak power here: P(max) = 0.018 x 1000 / 4 = 4.5 W
        |      *
        |       *
        |        *
        |         *
      0 +----------*---------------------- Speed (rad/s)
        0        500                1000   omega(nl) = V / k = 6 / 0.006 = 1000 rad/s  (9
```

Here is an illustrative small hobby motor, with the kind of constants a 6 V gear-motor's bare motor typically has (typical values from memory): V = 6 V, R = 2.0 Ω, k = 0.006 N m/A, and a no-load current I_0 = 0.15 A that stands for friction. The table computes, for each current, the speed from ω = (V − I R) / k, the shaft torque from τ = k (I − I_0) and the efficiency.

Current I	Speed (rpm)	Shaft torque (N m)	P(mech) (W)	P(elec) = V I (W)	Efficiency
0.150 A	9072	0.0000	0.00	0.90	0 %
0.300 A	8594	0.0009	0.81	1.80	45.0 %
0.500 A	7958	0.0021	1.75	3.00	58.3 %
0.671 A	7413	0.0031	2.43	4.03	60.3 %
1.000 A	6366	0.0051	3.40	6.00	56.7 %
1.500 A	4775	0.0081	4.05	9.00	45.0 %
2.000 A	3183	0.0111	3.70	12.00	30.8 %
3.000 A	0	0.0171	0.00	18.00	0 %

```
Peak efficiency:       I = sqrt(I(0) V / R) = sqrt(0.15 x 6 / 2) = 0.671 A;      eta(ma
Peak power:            I = (I(0) + V / R) / 2 = (0.15 + 3.0) / 2 = 1.575 A;       P =
```

Three lessons come out of that table, and they decide how a motor is used:

- **Peak efficiency and peak power are at different places.** A small brushed motor is most efficient at a fairly light load, about 10 to 20 % of its stall torque, and at a fairly high speed. At peak power the efficiency has dropped to about 45 %, and half the electrical power is heating the windings.
- **Run it in the efficient region, not at the peak-power point.** A robot that spends its life at low load and high speed wastes the least battery and keeps the motor cool. The usual design rule is to choose the gearing so that the motor's torque at the worst expected load is no more than about a quarter to a third of its stall torque (a typical guide, and the datasheet's rated continuous torque takes priority).

- **Stall is dangerous.** At stall the motor produces no power but dissipates $I^2R = 3^2 \times 2.0 = 18$ W, four times the 4.5 W it could ever deliver. Small motors overheat within seconds. A robot jammed against a wall with the motors driving is a stalled robot: the driver should limit current (Basic Electronics 9).

> **THE DATASHEET'S "STALL TORQUE" IS NOT A WORKING TORQUE**
>
> Stall torque is the number the motor gives when it cannot turn; it can be held for only a short time. It is useful as a scale (the line above starts there) and for checking that the motor can start the load, but design to the continuous rating. Also, stall torque and no-load speed both scale with the supply voltage: at half the voltage, both halve, which is why a sagging battery makes a robot both slower and weaker.

3. Gear ratios and gearboxes

A hobby motor is typically fast and weak, 5000 to 15000 rpm and a few hundredths of a newton metre, while a wheel wants about 50 to 300 rpm and much more torque. A **gearbox** bridges the gap. If the driven gear has N times as many teeth as the driving gear (a **reduction** of N:1), the output turns N times slower and, ignoring losses, delivers N times the torque. Gears lose a little power in friction, so the real output torque is multiplied by an efficiency η.

```
Gear ratio (reduction):     N = teeth(driven) / teeth(driving)
Output speed:               omega(out) = omega(in) / N
Output torque:              tau(out) = tau(in) N eta                    eta = efficienc
Output power:               P(out) = tau(out) omega(out) = tau(in) omega(in) eta = P(in
Several stages in series:   N = N1 x N2 x N3 ...        eta = eta1 x eta2 x eta3 ...

A two-stage spur-gear train (each pair of gears turns the next shaft the opposite way):

    motor ---[ 12T ]=====[ 48T ]--+                        Stage 1:  48 / 12 = 4:1
                                  | (same shaft)
                                  +--[ 15T ]=====[ 45T ]--- output  Stage 2:  45 / 15 =

Worked example:  motor at 6000 rpm and 0.010 N m through that train, eta = 0.92 per sta
       Ratio:         N = 4 x 3 = 12:1              Efficiency:  eta = 0.92 x 0.92 = 0
       Output speed:  6000 / 12 = 500 rpm           Output torque:  0.010 x 12 x 0.846
       Power in:      0.010 x 628.3 = 6.28 W         Power out:  0.1016 x 52.36 = 5.32
```

Note the direction of the trade. Using a bigger ratio multiplies torque and divides speed by the same factor, but every extra stage costs efficiency and adds weight, noise and backlash. Common hobby gear motors come as ready-made packages with a fixed ratio (for example 30:1, 50:1, 100:1 or 298:1, among many others), which is how most small robots obtain their drivetrain.

Types of gearbox

TYPE	HOW IT WORKS	TYPICAL EFFICIENCY	STRENGTHS	WATCH FOR
Spur gears	Parallel shafts, straight teeth, stages stacked side by side	about 90 to 98 % per stage (hobby plastic and sintered gear motors nearer the low end)	Simple, cheap, easy to build or print	Noisy at speed; the output is offset from the motor shaft; backlash adds up through the stages
Planetary	A central sun gear, several planet gears on a carrier, an outer ring gear; the output is the carrier	about 90 to 97 % per stage	Compact and in line with the motor; the load is shared by several teeth, so it carries more torque for its size; ratios of about 3:1 to 10:1 per stage	More parts and cost; stages are stacked for big ratios; cheap ones have noticeable backlash
Worm	A screw-like worm turns a worm wheel at a right angle	about 40 to 90 %, depending on the lead angle (see below)	Large single-stage ratios (10:1 to 100:1 or more); right-angle drive; can be self-locking	Often inefficient and hot; a self-locking worm cannot be back-driven, so a pushed robot cannot coast
Helical and bevel	Slanted teeth (quieter); bevel gears turn the axis through an angle	about 90 to 98 %	Quiet running; direction change	Helical gears push sideways along the shaft and need thrust bearings

```
Planetary stage (sun = input, ring gear held fixed, carrier = output):        N = 1 + R
       Geometry:  R = S + 2 P  (P = planet teeth).   Example: S = 12, P = 12, so R = 36

Worm drive:   ratio N = (teeth on the worm wheel) / (number of starts on the worm).
       Efficiency (approximately):   eta = tan(lambda) / tan(lambda + phi)        lamb
       Self-locking when:             lambda < phi     (the wheel cannot drive the worm
       lambda = 5 deg,  mu = 0.10 (phi = 5.71 deg):    eta = tan 5 / tan 10.71 = 0.0875
```

```
        lambda = 15 deg, mu = 0.05 (phi = 2.86 deg):    eta = tan 15 / tan 17.86 = 0.267
        The pattern holds for every worm: the more efficient it is, the less it holds a

  Backlash:  the small free movement between gear teeth when the direction reverses. At t
        1.5 degrees of backlash at the output of a gearbox driving a wheel of radius 0.0
        Backlash does not matter for a wheel that mostly turns one way, but it matters f
```

BACK-DRIVING AND HOLDING

Whether a gearbox can be turned backwards from the output also decides what happens when power is off. A high-ratio, low-efficiency gearbox (a worm, or a 298:1 spur train) tends to hold its position and resists being pushed, which suits an arm but means the robot will not freewheel when you push it by hand. A high-efficiency gearbox lets the load drive the motor, which can be useful (the robot rolls freely, and the motor can brake it and even recover energy) and can be a nuisance (an arm falls when the power drops).

4. Belts and chains

Where the motor cannot sit directly in line with the axle, a **belt** or a **chain** carries the motion over a distance, still with a ratio set by size: for toothed pulleys or sprockets the ratio is the ratio of tooth counts. Neither needs the precision of a gear mesh, so they tolerate small misalignment.

	TOOTHED BELT (GT2, HTD)	ROLLER CHAIN	ROUND OR V BELT
Efficiency (typical)	about 95 to 98 %	about 95 to 98 %	about 90 to 95 %, and depends on tension
Slip	None, if it fits the teeth correctly	None	Can slip, so it is not for timing
Noise and cost	Quiet, light, cheap; 3D-printable pulleys	Noisier, heavier, needs lubrication	Quiet, cheapest
Strength	Good for small robots; rated by belt width	Carries high loads; used in larger robots	Light loads only
Upkeep	Tension it correctly, with an idler or by sliding the motor mount	Chain stretches; needs adjustment and oil	Re-tension from time to time

```
Belt or chain ratio:        N = teeth(driven) / teeth(driving)          (the same as for
Pitch diameter of a toothed pulley:      d = (teeth x pitch) / pi
Open-belt length (approximate):          L = 2 C + (pi / 2)(d1 + d2) + (d2 - d1)^2 / (4

Worked example: GT2 belt (pitch 2 mm), a 16-tooth pulley on the motor and a 48-tooth pu
       Ratio:          N = 48 / 16 = 3:1
       d1 = 16 x 2 / pi = 10.19 mm          d2 = 48 x 2 / pi = 30.56 mm
       L  = 2 x 60 + (pi / 2)(10.19 + 30.56) + (30.56 - 10.19)^2 / (4 x 60) = 120 + 64.
       Belt teeth:     185.7 / 2 = 92.9 teeth, so choose the nearest stocked belt (abou
Efficiency of this stage: about 0.97 (typical), so it is a cheap place to take a ratio
```

A belt stage is a handy way to add a little ratio and move the motor away from the wheel, while a chain (or a belt) joining two wheels on the same side makes one motor drive both of them, which is a form of four-wheel drive from two motors. Direct drive (a wheel bolted to a geared motor's output shaft) is simplest, cheapest and the right answer for most first robots.

5. Wheels and traction

A wheel does two things: it converts rotation into forward speed, and it converts torque into a push on the ground. The first depends on the **diameter**, the second on **grip**.

```
Speed:            v = omega r = pi D n / 60                    (D = wheel diameter i
Drive force:      F = tau(wheel) / r                           so a larger wheel giv
Traction limit:   F(max) = mu N                                N = load carried by t
Rolling drag:     F(roll) = Crr m g cos(theta)                 Crr = rolling-resista
A robot can climb a slope without the wheels slipping only if:     tan(theta) <= s mu -

Typical values (approximate; surfaces and tyres vary a great deal):

  Contact                                    mu (friction, dry)       Crr (rolling
  Soft rubber or silicone tyre on clean tile     0.7 to 1.0               0.02 to 0.04
  Rubber tyre on carpet                          0.6 to 0.9               0.04 to 0.10
  Hard plastic wheel on a smooth floor           0.2 to 0.4               0.01 to 0.02
  Rubber on a wet or dusty floor                 0.3 to 0.5 (and falling) about the sa

Worked example: wheel diameter 0.080 m (r = 0.040 m).
       To drive at 0.5 m/s:     omega = v / r = 0.5 / 0.04 = 12.5 rad/s      n = 12.5 x
       Pushing with F = 5 N:     tau = 5 x 0.04 = 0.20 N m   (shared between the driven
```

```
Wheel diameter 0.060 m:   119.4 x (0.080 / 0.060) = 159.2 rpm for the same speed
Wheel diameter 0.100 m:   95.5 rpm and 0.050 N m per newton.
```

Choosing a wheel

- **Diameter.** A bigger wheel gives more speed from the same motor and gearbox, rolls over bumps and gaps more easily (a rough guide is that a wheel will climb a step of up to a fraction of its radius, depending on grip) and raises the robot higher. It also needs more torque for the same push ($\tau = F\,r$) and may push the centre of gravity up. Small robots often use 40 to 100 mm wheels.
- **Grip.** Soft rubber or silicone gives high μ and wears and picks up dust; hard plastic is fast and slides. Rubber bands over a printed wheel are a cheap way to add grip. A rubber tyre that works on the lab bench may skid on a dusty floor, so test where it will be used.
- **Width.** A wider tyre does not increase the friction coefficient but spreads wear and improves stability; a narrow tyre turns more easily on the spot.
- **Where the weight sits.** Traction depends on the load on the *driven* wheels. Put the weight over them (Robotics Fundamentals 2 gave 22 % on the caster and the rest on the drive

WHEEL TYPE	WHAT IT IS	GOOD FOR	DRAWBACKS
Standard fixed wheel	A round wheel on a fixed axle	Most robots; efficient and simple	Cannot move sideways; needs steering by differential speed or a steering mechanism
Caster (ball or swivel)	A free-turning support	Third point of contact on a two-wheel robot	Drag and bounce; swivel casters can flip the robot's heading when it reverses
Omni wheel	A wheel ringed with small rollers that let it slide sideways	Robots that must move in any direction (Robotics Fundamentals 4)	Less grip and a bumpier ride
Mecanum wheel	Angled rollers; four of them let the robot drive in any direction	Sideways motion in tight spaces (Robotics Fundamentals 4)	Low efficiency and grip, harder control; in practice mecanum drivetrains tend to be heavier and more costly
Foam or pneumatic	A soft, deformable tyre	Rough ground and shock absorption	Squashes (the radius changes with load) and has higher rolling

WHEEL TYPE	WHAT IT IS	GOOD FOR	DRAWBACKS
wheel			resistance on hard floors
Tracks	A belt of tread round two or more wheels	Soft ground and obstacles (Robotics Fundamentals 4)	Skid steering, heavy, lossy, and friction in turning

6. Sizing a motor and gearbox for a robot

Now the pieces fit together. We work from the wheels back to the motor, in this order: the **force** the robot needs, the **torque** that force needs at each wheel, the **motor torque** after the gearbox, the **speed** (which fixes the gear ratio) and the **current** (which checks the electrical side).

```
The motion requirement, in six steps (air drag is ignored; at robot speeds it is small)

1. Force needed at the ground:     F = m (a + g sin(theta) + Crr g cos(theta))
       a = acceleration wanted (m/s^2),   theta = slope,   Crr = rolling-resistance coe
       three terms: to speed up, to climb, and to overcome rolling drag
2. Torque at each driven wheel:    tau(wheel) = F r / n(wheels)               r = wheel
3. Motor torque after the gearbox: tau(motor) = margin x tau(wheel) / (N eta)         N
4. Speed:                          omega(wheel) = v / r        omega(motor) = N omega(wh
5. Current:                        I = tau(motor) / k + I(0)        (and the motor's te
6. Checks:                         tau(motor) is below about 25 to 30 % of stall torque

Choosing N:  run the motor at about 60 to 70 % of its no-load speed when the robot is a
       N  =  (0.6 to 0.7) x omega(nl) / omega(wheel)
```

Worked example: a 2 kg two-motor robot

A two-motor robot of mass 2 kg (the heavier cousin of Robotics Fundamentals 2's example, with the same layout so that 22 % of its weight is on the caster) must drive at 0.5 m/s, reach that speed in 1 second ($a = 0.5$ m/s^2), and be able to do that while climbing a 10 degree slope. Wheels: 80 mm diameter rubber tyres on a hard floor, $C_{rr} = 0.03$ and $\mu = 0.8$ (typical). The motor is the illustrative one from section 2 (6 V, 2.0 Ω, $k = 0.006$, $I_0 = 0.15$ A, stall torque 0.018 N m, no-load speed 9549 rpm), in a three-stage spur gearbox with 0.90 per stage.

```
Step 1  Force.   sin 10 deg = 0.17365,  cos 10 deg = 0.98481
       accelerate:  m a             = 2 x 0.5                        = 1.000 N
```

```
        climb:       m g sin(theta)  = 2 x 9.81 x 0.17365           = 3.407 N
        rolling:     Crr m g cos(theta) = 0.03 x 2 x 9.81 x 0.98481  = 0.580 N
        F = 1.000 + 3.407 + 0.580 = 4.987 N               (steady climb with no acceler

  Step 2  Wheel torque (n = 2, r = 0.040 m):    tau(wheel) = 4.987 x 0.040 / 2 = 0.0997 N

  Step 4  Speed and the gear ratio first:
        wheel:   omega = 0.5 / 0.04 = 12.5 rad/s = 119.4 rpm
        motor no-load speed 9549 rpm,  so  N = (0.6 to 0.7) x 9549 / 119.4 = 48 to 56:
        At 50:1 the motor turns at 12.5 x 50 = 625 rad/s = 5968 rpm at 0.5 m/s, which is
        Gearbox efficiency (three spur stages):  eta = 0.9^3 = 0.729

  Step 3  Motor torque with the 1.5 margin:     tau(motor) = 1.5 x 0.0997 / (50 x 0.729) =
        Percentage of stall torque:            0.00410 / 0.018 = 22.8 %        (inside th

  Step 5  Current:    with margin:  I = 0.00410 / 0.006 + 0.15 = 0.834 A          without
        Voltage needed at 0.5 m/s (no margin):   V(need) = 0.006 x 625 + 0.606 x 2.0 = 3
        With the margin load:                    V(need) = 3.75 + 0.834 x 2.0 = 5.42 V
        Electrical power, two motors (no margin):  2 x 4.96 x 0.606 = 6.01 W
        Mechanical power at the wheels:            F v = 4.987 x 0.5 = 2.49 W      overa
        (motor about 57 % at this point x gearbox 73 % = about 41 %. Real robots are ine

  Step 6  Checks.
        Stall wheel torque:       0.018 x 50 x 0.729 = 0.656 N m each,  so the pair coul
        Traction limit:           s mu m g cos(theta) = 0.78 x 0.8 x 2 x 9.81 x 0.98481 =
        Steepest slope without slipping:   tan(theta) = s mu - Crr = 0.78 x 0.8 - 0.03 =
        Speed at the worst load:   with I = 0.606 A the motor can still turn at (6 - 0.6
                                   so the wheel speed could reach 798 / 50 x 0.04 = 0.63
```

How sensitive is the answer to the ratio? Repeating the margin-load calculation (acceleration on the slope, torque margin 1.5) for neighbouring ratios shows why 50:1 is a reasonable pick:

```
  Ratio N   Motor torque (N m)   Current I   % of stall torque   Voltage needed at 0.5 m/s
  30:1      0.00684              1.290 A     38.0 %              4.83 V
  40:1      0.00513              1.005 A     28.5 %              5.01 V
  50:1      0.00410              0.834 A     22.8 %              5.42 V
  60:1      0.00342              0.720 A     19.0 %              5.94 V

  The trade: a lower ratio asks more of the torque (heat), a higher ratio asks more of th
```

Steady-state force on a slope scales strongly with the slope angle, which is why a robot that is fine on the flat can fail on a ramp:

```
Slope     Steady force F = m g (sin + Crr cos)      Torque per wheel (r = 0.04, 2 wheel
0 deg     0.589 N                                   0.0118 N m
5 deg     2.296 N                                   0.0459 N m
10 deg    3.987 N                                   0.0797 N m
15 deg    5.647 N                                   0.1129 N m
20 deg    7.264 N                                   0.1453 N m

From flat to 10 degrees the force needed rises almost seven times.
```

A small sizing calculator

Because the same six steps are repeated every time a design changes, it pays to put them in a few lines. This is hardware-neutral pseudocode in the style of Python (real board code is left to the course Programming for Robotics):

```
def size_drivetrain(mass, n_driven, wheel_d, v_target, accel, slope_deg, crr,
                    ratio, gear_eta, volts, resistance, kt, i_noload, margin=1.5):
    g = 9.81
    theta = radians(slope_deg)
    r = wheel_d / 2

    force = mass * (accel + g * sin(theta) + crr * g * cos(theta))     # step 1
    wheel_torque = force * r / n_driven                                # step 2
    motor_torque = margin * wheel_torque / (ratio * gear_eta)          # step 3
    motor_speed = ratio * v_target / r                                 # step 4 (rad/s)
    current = motor_torque / kt + i_noload                             # step 5
    volts_needed = kt * motor_speed + current * resistance
    duty = volts_needed / volts                                        # must stay belo
    stall_fraction = motor_torque / (kt * volts / resistance)          # should be belo

    return force, wheel_torque, motor_torque, current, duty, stall_fraction

# the 2 kg example:
# size_drivetrain(2, 2, 0.08, 0.5, 0.5, 10, 0.03, 50, 0.729, 6, 2.0, 0.006, 0.15)
#   -> force 4.99 N, wheel torque 0.0997 N m, motor torque 0.0041 N m, current 0.834 A,
```

WHAT THIS MODEL LEAVES OUT

It treats the motor as an ideal line, assumes steady friction and a constant battery voltage, and ignores air drag, gearbox losses that vary with load and temperature, and the extra force when turning (a skid or a heavy caster resists the turn). It also assumes the weight is shared as designed; accelerating shifts weight to the rear wheels. Build the first robot with a motor and ratio that give a margin, then measure the actual current on the bench (Basic Electronics 9) and adjust. The calculation tells you the neighbourhood, and the tests tell you the street.

7. Challenges

CHALLENGE 1: READING A MOTOR'S TORQUE-SPEED LINE

A brushed DC motor is rated at 12 V with a stall current of 4 A and a stall torque of 0.12 N m. Use the ideal model (ignore friction). (a) Find the winding resistance R, the torque constant k (which also equals the back-EMF constant), and the no-load speed in rad/s and rpm. (b) Express the stall torque in kg cm. (c) At a load torque of 0.04 N m, find the speed (rad/s and rpm), the mechanical power, the current, the electrical power and the efficiency. (d) Find the peak mechanical power, and the speed, torque, current and efficiency at which it occurs. (e) How much heat is produced in the windings at stall, and how does it compare with the peak output power? (f) At half the supply voltage (6 V), what are the stall torque and the no-load speed?

View solution

CHALLENGE 2: GEARBOX, BELT AND WORM

A motor delivers 0.020 N m at 4000 rpm. It drives a two-stage spur gearbox: stage 1 is a 12-tooth pinion driving a 48-tooth gear, stage 2 is a 15-tooth pinion driving a 45-tooth gear. Each stage is 92 % efficient. (a) Find the total ratio and the total efficiency. (b) Find the output speed (rpm) and torque. (c) Find the input power, the output power and the power lost as heat. (d) The output drives a wheel of diameter 0.10 m: find the speed in m/s and the push at the rim. (e) A belt stage with a 16-tooth and a 32-tooth pulley (97 % efficient) is added after the gearbox: find the new speed, torque, wheel speed and push. (f) Instead of the spur gearbox, the motor drives a 2-start worm and a 30-tooth worm wheel; the worm has a lead angle of 8 degrees and a friction coefficient of 0.08. Find the ratio, the efficiency, the output torque and speed, and say whether it is self-locking. (g) 1.5 degrees of backlash at the output of the spur gearbox drives a wheel of radius 0.05 m: how much free motion at the rim?

View solution

CHALLENGE 3: SIZE THE DRIVE FOR A FOUR-WHEEL ROBOT

A 3 kg robot has four driven wheels of diameter 0.10 m, one motor each. It must drive at 0.8 m/s, accelerate at 0.4 m/s^2, and climb a 15 degree slope with rolling resistance Crr = 0.04. The motors are 12 V, with winding resistance 4 Ω and k = 0.02 N m/A (ignore friction), and the gearboxes are 75 % efficient. (a) Find the force needed and the torque each wheel must supply. (b) Find the motor's stall torque, stall current and no-load speed (rpm), and choose a gear ratio that puts the motor at about 75 % of no-load speed at 0.8 m/s (choose a whole-number ratio near that). (c) With a gear ratio of 30:1 and a torque margin of 1.5, find the motor torque, current and the percentage of stall torque. (d) Using the current without the margin, find the PWM duty needed at 0.8 m/s, and say whether the headroom is adequate. (e) With a friction coefficient of 0.6, check the traction limit when all four wheels are driven. Then repeat it if only two wheels are driven and they carry half the weight. What happens?

View solution

QUICK REFERENCE

- **Torque, speed, power:** $\tau = F\ r$; $\omega = 2\ \pi\ n / 60$; $P = \tau\ \omega = F\ v$; 1 kg cm = 0.0981 N m; 1 oz in = 0.00706 N m
- **DC motor:** $V = k\ \omega + I\ R$; $\tau = k\ I$; $\tau(s) = k\ V / R$; $\omega(nl) = V / k$; $\tau = \tau(s)(1 - \omega / \omega(nl))$; $P(max) = \tau(s)\ \omega(nl) / 4$ at half speed
- **Efficiency:** peak efficiency (about 60 % in a small brushed motor) at about 10 to 20 % of stall torque, peak power at about half stall torque with efficiency about 45 %; stall wastes I^2R as heat
- **Gears:** output speed = input / N; output torque = input × N × η; ratios multiply and efficiencies multiply; planetary N = 1 + R / S (ring fixed)
- **Gearbox types:** spur about 90 to 98 % per stage; planetary about 90 to 97 %; worm about 40 to 90 % and self-locking if the lead angle is less than the friction angle; backlash at the rim = r × angle (radians)
- **Belts and chains:** ratio = teeth ratio; pulley d = teeth × pitch / π; efficiency about 95 to 98 %
- **Wheels:** $v = \pi\ D\ n / 60$; $F(max) = \mu\ N$; climbable if $\tan(\theta) \leq s\ \mu - Crr$; larger wheel means more speed per rpm and more torque per newton
- **Sizing:** $F = m\ (a + g \sin \theta + Crr\ g \cos \theta)$; wheel torque = F r / n; motor torque = margin × wheel torque / (N η); N about (0.6 to 0.7) ω(nl) / ω(wheel)
- **Checks:** motor torque below about 25 to 30 % of stall; PWM duty below about 90 %; F below the traction limit; $I = \tau / k + I(0)$

Next, Robotics Fundamentals 4 looks at drive systems and kinematics: how two wheels driven at different speeds steer a robot (differential drive), how omni and mecanum wheels move it

sideways, how to calculate where the robot has got to from its wheel speeds (odometry), what turning radius it has, how tracks compare with wheels, and a first look at legs.

Robotics Fundamentals

Course 1 · Chapter 4 · Drive Systems and Kinematics

Robotics Fundamentals 3 chose a motor, a gearbox and a wheel, and showed how much force and torque it takes to move a robot. This chapter is about *how those wheels are arranged* and what the arrangement lets the robot do. Two wheels driven at different speeds steer a robot; a car steers by turning its front wheels; a wheel with rollers round its rim lets a robot slide sideways. Each layout has a short piece of geometry attached, called its **kinematics**: the relationship between wheel speeds and the motion of the robot, with no forces in it. Kinematics works in both directions. **Forward kinematics** turns wheel speeds into robot motion, and when you add up that motion over time it becomes **odometry**, the robot's own running estimate of where it is. **Inverse kinematics** turns a wanted motion into the wheel speeds that produce it. We then look at why odometry drifts, compare tracks with wheels, and take a first look at legs.

Dimensions, slip factors, encoder counts and pressures in this chapter are **typical, approximate or illustrative values from memory**, chosen to make the arithmetic clear. Real robots vary, so measure yours (the chapter ends with how). The formulas themselves are standard geometry and are exact for the idealised models described.

1. Holonomic and non-holonomic motion

A robot on a flat floor has a **pose** of three numbers: its position (x, y) and its heading θ. Robotics Fundamentals 1 counted degrees of freedom, and a ground robot has three. What matters for a drive system is how many of those three the robot can *change independently, at any moment*.

- A **holonomic** drive (in the robotics sense) can move in any direction, and rotate, independently and instantly. It controls all three: forward speed, sideways speed and turning rate. Omni-wheel and mecanum robots are holonomic.
- A **non-holonomic** drive cannot move sideways. A differential-drive robot and a car each control only two velocities (forward speed and turning rate, or forward speed and steering angle). The constraint is that the wheels will not slide sideways, which in symbols says the velocity along the axle direction is zero.

A non-holonomic robot can still *reach* any pose, but it cannot go there directly. A car cannot slide into a parking space beside it: it must go forward and back with the wheels turned (parallel

parking), which is the classic non-holonomic manoeuvre. A holonomic robot would simply slide across. The price of that freedom is extra complexity, lower grip and more motors.

```
Robot frame:   x forward,  y to the LEFT,  theta counter-clockwise from the world x axi

Velocities the robot body can have:      vx (forward)      vy (sideways, left positive)

Differential drive and car:   two controllable velocities, and the constraint   vy = 0
Omni and mecanum:             all three controllable
```

2. The drive types

DRIVE	HOW IT STEERS	HOLONOMIC?	ACTUATORS	STRENGTHS	DRAWBACKS
Differential drive	Two driven wheels at different speeds, plus a caster or two for balance	No	2 motors	Simplest; can spin on the spot (zero turning radius); easy to model and to build	Cannot go sideways; the caster adds drag; a straight line needs matched wheels or feedback (Robotics Fundamentals 7)
Car-style (Ackermann)	Front wheels turn; rear (or front) wheels drive	No	1 or 2 drive motors, 1 steering servo	Stable at speed; good for outdoor and larger robots; familiar	Cannot turn on the spot; a minimum turning radius; reversing out of a tight spot takes planning
Skid-steer and tracks	Left side and right side driven at different speeds; the wheels or tracks slide sideways in a turn	No	2 motors (or 4 with one per wheel)	Robust; good on loose ground; no caster; can spin on the spot	Turning needs a lot of force and wears the tyres; odometry from wheel speed is unreliable (section 9)

DRIVE	HOW IT STEERS	HOLONOMIC?	ACTUATORS	STRENGTHS	DRAWBACKS
Omni wheels (3 or 4)	Each wheel drives in one direction and rolls freely across it, so the robot can go any way	Yes	3 or 4 motors	Moves in any direction at once; compact	Low grip; bumpy ride from the rollers; vibration; each motor needs its own speed control
Mecanum wheels (4)	Angled rollers; the four wheel speeds combine into any direction	Yes	4 motors	Sideways and diagonal motion; the robot keeps its heading while moving	Lower efficiency and grip than ordinary wheels; flat floor only; needs matched left and right wheel types (section 8)
Swerve and synchro	Every wheel has its own drive motor and its own steering motor	Yes	8 for four wheels	Holonomic motion with full-size wheel grip	Complex, heavy and costly; mostly seen in larger competition robots
Legs	Legs lifted and placed in a repeating pattern (gait)	Depends on the design	about 3 per leg	Cope with steps, gaps and rough ground	Many actuators, complex control, slow and energy-hungry on flat floors (section 10)

WHICH ONE FOR A FIRST ROBOT?

Differential drive, nearly every time: two motors, one or two casters, and the simplest mathematics. It is also where the formulas below are easiest to learn. The other layouts then become variations: the same idea of "wheel speeds from body motion, and back again", with more terms.

3. Differential drive: forward kinematics

Let the two wheels be a distance L apart (the **track width**, measured between the contact points) and have ground speeds v_R and v_L (metres per second at the rim, positive when driving the robot forward). The robot body moves at the *average* of the two, and it turns at the *difference* divided by the wheel spacing.

```
Forward speed of the robot centre:     v = (vR + vL) / 2
Turning rate (counter-clockwise +):    omega = (vR - vL) / L
Wheel ground speed from wheel rpm:     v(wheel) = pi D n / 60 = omega(wheel) r

Turning radius (centre to the point the robot circles about):
       R = v / omega = (L / 2) (vR + vL) / (vR - vL)
       R is positive when the centre of the circle (the ICC) is on the left (vR greater
       vR = vL:  R is infinite (a straight line).      vR = -vL:  R = 0 (spinning on th

Why:  both wheels travel round circles about the same point, the instantaneous centre o
       The wheel nearer the ICC has the smaller radius, so it goes slower, in the ratio

Worked example:  L = 0.15 m, vR = 0.60 m/s, vL = 0.40 m/s.
       v     = (0.60 + 0.40) / 2 = 0.500 m/s
       omega = (0.60 - 0.40) / 0.15 = 1.333 rad/s = 76.4 deg/s           (one full cir
       R     = (0.15 / 2)(1.00 / 0.20) = 0.375 m                         (check: v / o
       The left wheel's circle has radius 0.375 - 0.075 = 0.300 m and the right wheel's

The axle line, seen from above (left turn, one character is 0.0075 m; the robot faces o

*-------------------------------------L---------C---------R
|<--------------0.300 m---------------->|
|<-----------------R = 0.375 m------------------>|
|<------------------------0.450 m--------------------------->|
ICC                                 left wheel  centre  right wheel
```

The table below fixes v_R at 0.50 m/s and slows the left wheel, with L = 0.15 m. Notice how quickly the radius shrinks once the wheel speeds differ a lot, and that a stopped wheel gives a pivot, not a spin.

vL (m/s)	vL/vR	omega (rad/s)	omega (deg/s)	Turning radius R (m)	Motion
+0.50	1.00	0.000	0.0	infinite	straight lin
+0.45	0.90	0.333	19.1	1.425	gentle curve
+0.40	0.80	0.667	38.2	0.675	curve
+0.25	0.50	1.667	95.5	0.225	tight curve
+0.00	0.00	3.333	191.0	0.075	pivot about

```
-0.25      -0.50    5.000          286.5          0.025                ICC between
-0.50      -1.00    6.667          382.0          0.000                spin on the
```

4. Differential drive: inverse kinematics

Planners and controllers think in v and ω ("go at 0.4 m/s and turn at 2 rad/s"), but the motors

```
vR = v + omega L / 2
vL = v - omega L / 2                          (check: their average is v, and (vR - vL

Wheel angular speed:   omega(wheel) = v(wheel) / r          rpm = 60 v(wheel) / (2 pi r
Encoder rate wanted:   ticks per second = v(wheel) / d(tick)         (d(tick) from sect

Worked example (L = 0.15 m, wheel radius r = 0.0325 m):   v = 0.40 m/s, omega = 2.0 rad/
       vR = 0.40 + 2.0 x 0.075 = 0.55 m/s          rpm(R) = 60 x 0.55 / (2 pi x 0.0325)
       vL = 0.40 - 2.0 x 0.075 = 0.25 m/s          rpm(L) = 60 x 0.25 / (2 pi x 0.0325)

Spin on the spot at omega = 3.0 rad/s (v = 0):   vR = +0.225 m/s, vL = -0.225 m/s

When a wheel runs out of speed.  Suppose v = 0.5 and omega = 3.0:   vR = 0.5 + 0.225 = (
       If the wheels can do at most 0.60 m/s, do NOT just clip vR (that changes the turn
       Scale BOTH commands by 0.60 / 0.725 = 0.828:   vR = 0.600,  vL = 0.228,   so v =
       The robot follows the same path (the same R = v / omega = 0.167 m), only more sl
```

5. Encoders, distance and odometry

How does the robot know the wheel speeds? A wheel **encoder** produces a number of pulses (or **ticks**) for each turn of its shaft, and counting them measures how far the wheel has turned. Robotics Fundamentals 6 treats encoders properly (how they work, direction sensing, resolution); here we only need the arithmetic. An encoder is usually on the *motor* shaft, before the gearbox, so each wheel revolution corresponds to the gear ratio times as many ticks.

```
Distance per tick:          d(tick) = 2 pi r / (CPR x gear ratio)          CPR = count
Distance from a count:      d = ticks x d(tick) = 2 pi r ticks / (CPR x gear ratio)
Ticks per wheel revolution: CPR x gear ratio
```

```
Worked example:  wheel diameter 65 mm (r = 0.0325 m), encoder 12 counts per motor revol
                 gearbox 50:1.
       Ticks per wheel revolution:      12 x 50 = 600
       Wheel circumference:             2 pi x 0.0325 = 0.2042 m
       Distance per tick:               0.2042 / 600 = 0.000340 m = 0.340 mm
       1200 ticks:   2.0 wheel revolutions = 0.408 m          1500 ticks: 2.5 revolutio
       Resolution of heading:  one tick of difference between the wheels turns the robo
```

Resolution is therefore rarely the limit with a geared encoder: a tick is a third of a millimetre. The errors that matter come from the model (the wheel diameter and track width you assumed) and

The pose update

Odometry keeps a running pose (x, y, θ). At each update (typically 20 to 100 times a second) it reads how many ticks each wheel has counted since the last update, converts them to distances Δd_L and Δd_R, and moves the pose. Over one short step the wheel speeds are constant, so the robot moves along a **circular arc** (section 3), and the exact update follows from the arc geometry.

```
Step distances:           dL = ticksL x d(tick)        dR = ticksR x d(tick)
Centre travel, turn:      ds = (dR + dL) / 2           dtheta = (dR - dL) / L

Exact (arc) update:       R = ds / dtheta
                          x(new)     = x + R ( sin(theta + dtheta) - sin(theta) )
                          y(new)     = y - R ( cos(theta + dtheta) - cos(theta) )
                          theta(new) = theta + dtheta
If dtheta is nearly 0:    x(new) = x + ds cos(theta)        y(new) = y + ds sin(theta)

Common shortcut (midpoint):   x(new) = x + ds cos(theta + dtheta/2)       y(new) = y + (
       It is simple and very close for short steps. The crudest version, x + ds cos(the

Worked example, starting at (0, 0) facing along x (theta = 0), with the robot of sectio
       ticksL = 1200, ticksR = 1500   (a deliberately long step, so that the difference
       dL = 1200 x 0.0003403 = 0.4084 m          dR = 1500 x 0.0003403 = 0.5105 m
       ds = 0.4595 m                             dtheta = (0.5105 - 0.4084) / 0.15 = 0
       R  = ds / dtheta = 0.675 m
       Arc:       x = 0.675 x sin(0.6807) = 0.4248 m          y = -0.675 x (cos(0.680
       Midpoint:  x = 0.4595 cos(19.5 deg) = 0.4331 m         y = 0.4595 sin(19.5 deg
```

```
        Old-heading shortcut:  x = 0.4595, y = 0   (15 cm from the arc answer: far too c
        A second step, both wheels 0.300 m:  straight, so x = 0.4248 + 0.300 cos(39 deg)
```

Here is the update as short, hardware-neutral Python-style pseudocode (real board code, with its timers and interrupts for the encoders, is left to the course Programming for Robotics):

```
def update_pose(x, y, theta, ticks_l, ticks_r, d_tick, track):
    d_l = ticks_l * d_tick                    # distance each wheel rolled since the l
    d_r = ticks_r * d_tick
    ds = (d_r + d_l) / 2                      # distance moved by the robot centre
    dtheta = (d_r - d_l) / track              # change of heading (radians, counter-cl

    if abs(dtheta) < 1e-6:                    # driving straight
        x += ds * cos(theta)
        y += ds * sin(theta)
    else:                                     # moving along an arc of radius R
        R = ds / dtheta
        x += R * (sin(theta + dtheta) - sin(theta))
        y -= R * (cos(theta + dtheta) - cos(theta))

    theta = wrap_to_pi(theta + dtheta)        # keep the heading between -pi and +pi
    return x, y, theta

# call it every loop with the ticks counted SINCE THE LAST CALL (read, then reset or di
```

ODOMETRY IS A RUNNING TOTAL, SO ITS MISTAKES ONLY GROW

Every update adds a small error and nothing ever subtracts one. The pose is accurate over a short distance and becomes unreliable over a long one, which is the subject of the next section. Robotics Fundamentals 6 adds sensors (an inertial unit for heading, for example) that correct it, and Robotics Fundamentals 5 covers sensors that look at the world.

6. Odometry drift: where the errors come from

Odometry assumes that the wheel diameter, the track width and the wheel-on-floor contact are exactly what the software believes. Each can be a little wrong, and each gives a different kind of error. The errors that come from the robot's own dimensions are **systematic**: the same every

time, so they can be measured and removed. The errors from slip and bumps are **random**: different every time, so they can only be reduced (by feedback from sensors), not calibrated away.

ERROR SOURCE	KIND	EFFECT	REMEDY
Wheel diameters not equal (tyre wear, tolerance, load)	Systematic	A straight command curves; heading and position drift	Calibrate each wheel's effective diameter; use heading feedback (Robotics Fundamentals 6)
Wheel diameter different from the assumed value	Systematic	All distances scaled wrongly (both wheels the same way)	Drive a measured straight line and rescale
Track width different from the assumed value (the contact points are not where you measured)	Systematic	All turns scaled wrongly, so every heading is off	Spin a measured number of turns and rescale
Wheel slip (acceleration, wet floor, collisions, skid-steer turning)	Random	The wheels turned more than the robot moved	Gentle acceleration; better tyres; extra sensors
Uneven floor, bumps, a lifted wheel	Random	Wheel travel differs from ground travel	Sensors that look at the world
Encoder resolution and missed counts, and update rate	Small, random	Rounding, and an arc approximated by a straight step	Higher resolution; use the arc update; read counters fast

Worked example 1: one tyre 1 % too big

A robot with L = 0.15 m is told to drive 3.0 m straight. The software believes both wheels are the same size, so it runs both until each encoder has counted a distance of 3.0 m. The right tyre is in fact 1 % bigger than the left one, so its wheel rolls 1 % further for the same number of ticks.

```
What odometry thinks:   both wheels 3.000 m,  heading change 0,  pose (3.000, 0.000, 0 (

What really happens:    left wheel 3.000 m,    right wheel 3.000 x 1.01 = 3.030 m
       centre travel  ds     = 3.015 m
       heading change dtheta = (3.030 - 3.000) / 0.15 = 0.200 rad = 11.46 deg
       circle radius  R      = ds / dtheta = 15.075 m    (check: (L/2)(vR + vL)/(vR - vL
       true position:  x = R sin(0.2) = 2.995 m           y = R (1 - cos(0.2)) = 0.300
```

```
Error:  the robot is 0.30 m to one side of where odometry says, and pointing 11.5 degre
        Rule of thumb for this robot:   heading error per metre = 0.01 / 0.15 = 0.0667
        and the sideways error grows with the SQUARE of the distance (about 0.0667 x 3.
```

Worked example 2: track width 3 % out

The software uses L = 0.150 m but the true effective track width is 0.155 m (the tyre contact patches are further apart than the hubs). The robot is told to rotate on the spot through 360 degrees, which, by the software's belief, needs each wheel to roll once round a circle of radius L / 2, in opposite directions.

```
Each wheel must roll (in opposite directions):   (L(assumed) / 2) x 2 pi = 0.075 x 6.28
True rotation achieved:   dtheta = (dR - dL) / L(true) = (0.4712 + 0.4712) / 0.155 = 6.
Error:   11.6 degrees short, every full turn. The ratio is  L(assumed) / L(true) = 0.15
Four commanded quarter-turns (a square path) add up to the same 11.6 degrees, so after
```

Calibration

Because these errors are systematic, two measured tests remove most of them. Do each several times on the real floor and average.

1. **Straight line, for wheel diameters.** Drive a long, measured straight line and measure how far each wheel's distance was wrong and which way the robot curved. Scale each wheel's effective diameter by (true distance / odometry distance); the curve vanishes once the two wheels agree.
2. **On-the-spot turn, for track width.** Command a rotation of several full turns and measure the real angle (a marked floor works). Then L(effective) = L(assumed) × (angle by odometry) / (true angle). In worked example 2: 0.150 × 360 / 348.4 = 0.155 m.

A well-known refinement, the **UMBmark** test, drives a square clockwise and then anticlockwise and uses the two sets of end-point errors to separate the diameter error from the track error (the details are from memory, so look the method up before relying on it). Even after calibration the random errors remain, so a long-running robot needs another source of position information.

> **ODOMETRY IS FOR THE SHORT TERM**

Dead reckoning is an excellent estimate over a metre or a few seconds, and an increasingly poor one over minutes. Good designs use it as the *short-term* source and correct it with something that does not drift: a heading sensor, a wall or a landmark, or a map. Robotics Fundamentals 6 develops this idea, and Robotics Fundamentals 8 returns to mapping and localisation.

7. Car-style (Ackermann) steering

When a car turns, all four wheels must travel round circles about the *same* point, or some wheel would have to slide sideways. The rear wheels are fixed, so the centre of the circle lies on the line through the rear axle, and each front wheel must point at right angles to the line from that centre to it. That means the two front wheels turn by *different* angles: the inner wheel more than the outer. A linkage that does this is called **Ackermann steering**, and many hobby robots use a simple version of it, or a single steering servo with the two front wheels turned equally (which is only approximately right and scrubs the tyres a little).

```
The "bicycle model": merge the two front wheels into one wheel at the middle of the fro

                                    F  front wheel, steered by the angle delta
    R_f = W / sin(delta)       /    |
                          /          |  W = wheelbase
                    /                |
              /                      |
        /                            |
 *-----------------------------------+  rear axle centre
 ICC     R = W / tan(delta)   (distance from the ICC to the rear axle centre)

Turn radius of the rear axle centre:     R = W / tan(delta)
Turn radius of the front axle centre:    R_f = W / sin(delta)          (always larger
Turning rate:                            omega = v tan(delta) / W      (v = speed of t
Pose update for a short step ds:         dtheta = ds tan(delta) / W

Minimum turning radius:  R(min) = W / tan(delta(max))     A car cannot turn tighter tha

Two front wheels (track T between the steered wheels), turning about a centre at distan
       inner:   delta(i) = atan( W / (R - T/2) )          outer:   delta(o) = atan( W /
       Ackermann condition:   cot(delta(o)) - cot(delta(i)) = T / W          (the exact

Worked example:  W = 0.25 m,  T = 0.15 m,  bicycle-model steering angle delta = 20 deg,
       R  = 0.25 / tan(20 deg) = 0.25 / 0.3640 = 0.687 m
       R_f = 0.25 / sin(20 deg) = 0.731 m
```

```
        omega = 0.5 x 0.3640 / 0.25 = 0.728 rad/s = 41.7 deg/s
        inner wheel: atan(0.25 / (0.687 - 0.075)) = atan(0.4085) = 22.2 deg          out
        Check:  cot(18.2 deg) - cot(22.2 deg) = 3.048 - 2.448 = 0.600 = T / W = 0.15 / 0
        At a maximum steering angle of 30 deg:  R(min) = 0.25 / tan(30 deg) = 0.433 m
```

8. Omni and mecanum wheels: holonomic drives

Omni wheels

An **omni wheel** has small rollers around its rim, with their axes at right angles to the wheel's axle. The wheel drives the robot along the direction it points, but the rollers let it slide freely across that direction. With three omni wheels at 120 degrees to each other (a "kiwi" drive) or four at 90 degrees, the sum of the wheels' pushes can point anywhere. The standard inverse kinematics for a robot with wheels at angles φ_i (the angle of each wheel's position, measured from the robot's x axis; each wheel drives tangentially, counter-clockwise) and distance R_b from the centre is:

```
Wheel i angular speed:       omega(i) = ( -sin(phi_i) vx + cos(phi_i) vy + R_b omega_rob

Three wheels at 0, 120, 240 deg, wheel radius r = 0.03 m, moving straight forward at vx
       wheel at   0 deg:   -sin(0)   x 0.30 / 0.03 =  0.00 rad/s       (it points along
       wheel at 120 deg:   -sin(120) x 0.30 / 0.03 = -8.66 rad/s
       wheel at 240 deg:   -sin(240) x 0.30 / 0.03 = +8.66 rad/s
       Every wheel's speed has the robot's turning term in it, so spinning is just  R_b
```

Mecanum wheels

A **mecanum wheel** has rollers set at 45 degrees to its axle, so the wheel's push on the floor points diagonally. Four of them, fitted so the roller pattern is a mirror image front to back and left to right, give the robot a push in any direction by choosing the four speeds: if the diagonal components cancel, the robot goes forward; if the forward components cancel, it goes sideways.

```
Four mecanum wheels seen from above (the line in each wheel is the roller touching the

           FL  [ / ]         [ \ ]  FR
               [ / ]         [ \ ]
           RL  [ \ ]         [ / ]  RR
```

```
               [ \ ]         [ / ]

  Each wheel pushes along its roller line, so FL and RR push forward-right, FR and RL pus

  Inverse kinematics.   vx forward, vy to the left, omega counter-clockwise;  k = lx + ly
        FL = ( vx - vy - k omega ) / r              FR = ( vx + vy + k omega ) / r
        RL = ( vx + vy - k omega ) / r              RR = ( vx - vy + k omega ) / r

  Forward kinematics (the inverse of the above, with k = lx + ly):
        vx    = (r / 4) (  FL + FR + RL + RR )
        vy    = (r / 4) ( -FL + FR + RL - RR )
        omega = (r / (4 k)) ( -FL + FR - RL + RR )

  What the four wheels do (+ forward, - backward, 0 stopped):

  Robot motion              FL   FR   RL   RR
  Forward                   +    +    +    +
  Backward                  -    -    -    -
  Slide left (vy +)         -    +    +    -
  Slide right (vy -)        +    -    -    +
  Rotate counter-clockwise  -    +    -    +
  Diagonal forward-left     0    +    +    0     (vx = vy: two wheels do all the work)

  Worked example:  r = 0.04 m, lx = 0.10 m, ly = 0.09 m (k = 0.19 m);  vx = 0.30 m/s forw
        k omega = 0.19 x 0.50 = 0.095 m/s
        FL = (0.30 - 0.20 - 0.095) / 0.04 =  0.125 rad/s            FR = (0.30 + 0.20 + (
        RL = (0.30 + 0.20 - 0.095) / 0.04 = 10.125 rad/s            RR = (0.30 - 0.20 + (
        Check with the forward equations:   sum = 30.0   so vx = 0.04 / 4 x 30.0 = 0.30 r
                                            -FL+FR+RL-RR = 20.0   so vy = 0.01 x 20.0 = (
                                            -FL+FR-RL+RR = 9.5    so omega = 0.04 x 9.5 /
```

CHECK THE SIGN CONVENTION ON YOUR OWN WHEELS

Mecanum wheels come in left-hand and right-hand versions, and different libraries and books define the axes and the signs differently. The equations above are the commonly used set for the roller layout the opposite of what the equations assume. Test each motion with the robot on blocks before putting it on the floor. As with differential drive, if one wheel needs more than its maximum speed, scale *all four* commands down together so the direction of travel is kept.

Holonomic drives are attractive but have costs. Mecanum and omni wheels are less efficient and grip less than ordinary tyres (the rollers slip a little in the very act of moving sideways), so their

odometry is more error-prone and they push less hard (Robotics Fundamentals 3 gave traction as $F_{max} = \mu N$); mecanum wheels work well only on flat, smooth floors; and each wheel needs its own motor and speed control. They suit a robot that must move sideways in a tight space, such as a competition robot lining up on a target. A differential drive remains the better choice for most others.

9. Skid-steer, tracks and wheels

A **skid-steer** robot (four or six wheels, or two tracks, with the left side driven separately from the right side) has no caster and no steering. To turn, the two sides run at different speeds, and since the wheels or tracks cannot turn to follow the curve, they **slide sideways** over the ground. The simple differential formulas from section 3 then misreport the turn, because the effective turning geometry is wider than the geometry of the wheels.

```
Effective track width:     L(eff) = alpha x L(geometric)      alpha about 1.3 to 2 on h
Turning rate:              omega = (vR - vL) / L(eff)         (use L(eff) in the odomet

Worked example:  tracks with a geometric track width of 0.20 m, alpha = 1.6 (illustrati
       Spin command, vR = +0.30 and vL = -0.30 m/s:
       the simple differential model says omega = 0.60 / 0.20 = 3.00 rad/s,   in reality
       Odometry that trusts the geometric width would over-estimate every turn by 1.6 t

The force needed to turn.  When the robot pivots, the whole footprint slides sideways.

       M(turn) = mu m g Lc / 4              (a standard approximation for a rectangular
       The drive must supply it as a couple: opposite pushes F on the two sides:   F x

Worked example:  m = 2 kg, mu = 0.8, Lc = 0.20 m, L(eff) = 0.32 m:
       M(turn) = 0.8 x 2 x 9.81 x 0.20 / 4 = 0.785 N m             F = 0.785 / 0.32 = 2
       Compare the whole rolling resistance of this robot in a straight line (Crr = 0.0
       So turning on the spot takes about four times the force of rolling straight, per
```

	WHEELS (DIFFERENTIAL, WITH A CASTER)	TRACKS OR SKID-STEER
Ground pressure	High: the weight is on a few small contact patches (illustratively, a 2 kg robot on about 4.5 cm^2 of contact is about 44 kPa)	Low: the weight is spread (the same robot on two tracks, each 20 cm long and 3 cm wide, 120 cm^2 of contact in all is about 1.6 kPa, roughly 27 times less)
Soft ground, sand, mud, snow	Sinks and slips	Much better: floats and grips
Obstacles and steps	A wheel climbs a step of up to a fraction of its radius	A long track climbs steps and gaps larger than a small wheel can, especially with a sloping front
Efficiency on a hard floor	Best: low rolling resistance	Lower: track friction and scrub, plus the turning force above
Turning	Small force; the caster needs little	Needs a lot of force and slides; harsh on floors and on the drive train
Odometry quality	Good (calibrate it)	Poor: slip on every turn, so lean on other sensors
Weight, noise, cost, maintenance	Light, quiet, cheap	Heavier, noisier, dearer, with tracks that can come off

The choice follows the terrain: wheels indoors and on hard flat ground, tracks (or large, grippy wheels, or a skid-steer with a lot of grip) outdoors on soft or loose ground. Four-wheel skid-steer robots are a common middle way: easy to build and tough, with the same odometry caveat.

10. Legs, briefly

Legs deal with ground that wheels cannot: stairs, rubble, gaps, a stepping stone. A leg only needs isolated footholds, where a wheel needs a continuous surface. The cost is complexity. A walking robot typically needs about **three actuators per leg** (a hip swing, a hip lift and a knee), so the actuator count and the control problem grow quickly.

ROBOT	LEGS OR WHEELS	ACTUATORS (APPROX.)	NOTES
Differential-drive robot	2 drive wheels + caster	2	Chapter 3 and this chapter
Mecanum robot	4 wheels	4	Holonomic
Biped	2 legs	10 to 12 (about 5 or 6 per leg)	Balance is an active control problem
Quadruped	4 legs	12 (3 per leg)	Can stand still stably on three legs while one steps
Hexapod	6 legs	18 (3 per leg)	Tripod gait: three legs always on the ground

- **Gait.** A gait is the repeating pattern in which legs lift and land. A hexapod's usual *tripod gait* moves three legs at once (the front and back legs on one side and the middle leg on the other) while the other three support the body, then swaps. The body is always supported by a triangle of feet.
- **Static stability.** A robot is statically stable when its centre of gravity (Robotics Fundamentals 2) lies over the polygon formed by the feet on the ground. Hexapods and slow quadrupeds can stay inside this polygon all the time; faster walking and running rely on *dynamic* balance, in which the robot is only stable because it keeps moving.
- **Kinematics.** A leg is a small arm: its foot position follows from its joint angles by the same forward and inverse kinematics as a robot arm, and Robotics Fundamentals 9 works through them for a two-link arm, which is the basis of a leg.
- **Cost.** Legs are slower and use more energy per metre than wheels on flat ground, because they lift and lower the body at every step. They are worth it only when the terrain demands them.

11. Challenges

CHALLENGE 1: DIFFERENTIAL-DRIVE KINEMATICS AND ODOMETRY

A differential-drive robot has track width L = 0.20 m and wheels of diameter 0.08 m. Each wheel's encoder gives 600 ticks per wheel revolution (20 counts per motor revolution through a 30:1 gearbox). (a) Find the distance per tick. (b) In one update the left encoder counts 450 ticks and the right counts 700. Find the left and right distances, the centre travel ds, the heading change (radians and degrees)

and the turning radius. (c) The robot starts the update at pose (0.50 m, 0.20 m) with heading 90 degrees. Find the new pose using the exact arc update, and then using the midpoint shortcut, and say how far apart they are. (d) Using inverse kinematics, find vR and vL for v = 0.30 m/s and omega = -1.50 rad/s (which way does it turn?), the wheel speeds in rpm and the encoder rates in ticks per second. (e) To spin on the spot at 90 degrees per second, what ground speed, rpm and tick rate does each wheel need, and how long does a full turn take? (f) For vR = 0.40 and vL = 0.30 m/s, find v, omega and the turning radius, and say on which side the centre of the circle lies.

View solution

CHALLENGE 2: ACKERMANN AND MECANUM GEOMETRY

A car-style robot has wheelbase W = 0.30 m and a front track T = 0.18 m. (a) For a bicycle-model steering angle of 15 degrees, find the turn radius of the rear axle centre, the radius of the front axle centre, and the turning rate at 0.80 m/s. (b) With a maximum steering angle of 28 degrees, find the minimum turning radius. (c) For a turn of radius 1.00 m about the rear axle centre, find the bicycle-model angle and the inner and outer front wheel angles, and verify the Ackermann condition cot(outer) - cot(inner) = T / W. A mecanum robot has wheel radius 0.05 m, lx = 0.12 m and ly = 0.10 m. (d) Find the four wheel speeds (rad/s and rpm) for vx = 0.40 m/s, vy = -0.20 m/s (to the right) and omega = 0.80 rad/s. (e) Check them with the forward equations. (f) The wheels can do at most 12 rad/s: scale the command and give the speed, sideways speed and turning rate that actually result.

View solution

CHALLENGE 3: ODOMETRY DRIFT, CALIBRATION, AND TRACKS

A differential-drive robot assumes wheels of diameter 0.070 m and a track width of 0.180 m. In fact its right tyre has diameter 0.0707 m (the left is 0.0700 m) and the true effective track width is 0.186 m. (a) The robot is commanded to drive 4.00 m straight by odometry (each wheel's encoder says 4.00 m). Find the distance each wheel really rolls, the real heading change (radians and degrees), the circle radius, and the robot's real position relative to where odometry says it is (taking odometry's answer as (4.00, 0)). (b) It is commanded to spin through 360 degrees by odometry. How far does it really turn, and by how many degrees is it short? (c) Give the correction factors that calibration should apply to the right-wheel diameter and to the track width. (d) The robot is rebuilt with tracks with a geometric track width of 0.16 m and an effective width factor alpha = 1.7. If the software trusts the geometric width, how many degrees does the robot actually turn when odometry says it turned 90 degrees? (e) For a 3 kg tracked robot (friction coefficient 0.9, contact length 0.22 m), find the turning moment and the force each side must supply to spin on the spot, and compare it with the force needed to roll the whole robot straight (rolling-resistance coefficient 0.03).

View solution

QUICK REFERENCE

- **Holonomic:** can move in any direction independently (omni, mecanum, swerve). **Non-holonomic:** no sideways motion (differential drive, car): two controllable velocities, three pose variables.
- **Differential drive:** $v = (v_R + v_L) / 2$; $\omega = (v_R - v_L) / L$; $R = (L / 2)(v_R + v_L) / (v_R - v_L)$; inverse: $v_R = v + \omega L / 2$, $v_L = v - \omega L / 2$; if a wheel saturates, scale both.
- **Encoders:** $d(\text{tick}) = 2 \pi r / (\text{CPR} \times \text{gear ratio})$; ticks per wheel revolution = CPR × gear ratio; heading resolution per tick = $d(\text{tick}) / L$.
- **Odometry update:** $ds = (dR + dL) / 2$; $d\theta = (dR - dL) / L$; arc: $x \mathrel{+}= R(\sin(\theta + d\theta) - \sin\theta)$, $y \mathrel{-}= R(\cos(\theta + d\theta) - \cos\theta)$ with $R = ds / d\theta$; straight if $d\theta$ is about 0.
- **Drift:** diameter and track-width errors are systematic (calibrate with a straight run and an on-the-spot turn); slip and bumps are random; a 1 % wheel mismatch on a 0.15 m track gives about 3.8 degrees of heading error per metre.
- **Ackermann:** $R = W / \tan(\delta)$; $R_{front} = W / \sin(\delta)$; $\omega = v \tan(\delta) / W$; $R_{min} = W / \tan(\delta_{max})$; inner and outer wheels: $\operatorname{atan}(W / (R - T/2))$ (inner) and $\operatorname{atan}(W / (R + T/2))$ (outer); $\cot(\text{outer}) - \cot(\text{inner}) = T / W$.
- **Mecanum:** $FL = (v_x - v_y - k\omega) / r$, $FR = (v_x + v_y + k\omega) / r$, $RL = (v_x + v_y - k\omega) / r$, $RR = (v_x - v_y + k\omega) / r$ with $k = l_x + l_y$; check the signs on your own wheels.
- **Skid-steer and tracks:** $L(\text{eff}) = \alpha L$ (about 1.3 to 2, measure it); turning moment about $\mu m g L_c / 4$; low ground pressure and good on soft ground, but poor odometry and costly turning.
- **Legs:** about 3 actuators per leg; tripod gait; statically stable if the centre of gravity is over the support polygon; a leg is a small arm.

Next, Robotics Fundamentals 5 turns to sensors, the robot's way of seeing the world: distance sensors (ultrasonic, infrared, time-of-flight and lidar), line, bump and light sensors, a first look at cameras, and the ideas of accuracy, noise and range that decide which sensor to choose. Odometry told the robot where it thinks it is; sensors begin to tell it what is actually around it.

Robotics Fundamentals

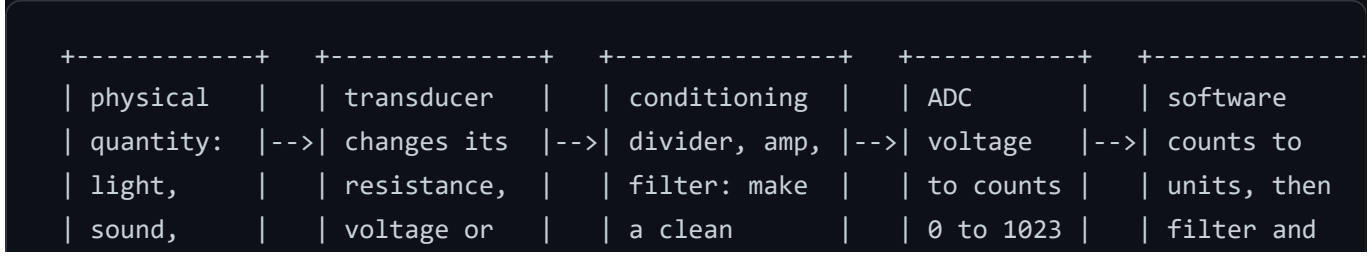
Course 1 · Chapter 5 · Sensors: Seeing the World

Robotics Fundamentals 1 described the loop at the heart of every robot: sense, think, act. Robotics Fundamentals 3 and 4 were the “act” half, motors, wheels and the geometry that turns wheel speeds into motion, and they ended with a robot that keeps a running guess of where it is (odometry) but has no idea what is around it. This chapter is the “sense” half. A **sensor** is the robot's only window on the world, and it is a narrow and smeared one: every sensor measures one thing, over a limited range, with some noise, some delay and some bias. A robot designer's skill is knowing what each sensor really tells you, what it cannot tell you, and how to clean up what it says. We follow a measurement from the physics to the number in the program, look at the numbers that describe a sensor (its specifications), meet the common robot sensors (distance, line, bump, light, current, camera), and then cover the three things that make sensor data usable: filtering noise, calibration, and working out how large the remaining error is.

Ranges, beam angles, resolutions, response times and prices in this chapter are **typical or approximate values from memory**, for the kind of low-cost parts used on small robots; they vary a lot between models, so always read the datasheet of the part you actually buy. The formulas (time of flight, similar triangles, the statistics of averaging, error propagation) are standard and exact for the idealised models described.

1. From physical quantity to number

A computer can only work with numbers, and the world is made of light, sound, force and heat. A sensor is a chain of conversions, and knowing the chain tells you where each error can creep in. The first link is the **transducer**, a part whose electrical behaviour changes with the physical quantity: a light-dependent resistor's resistance falls as light rises, a microphone-like ultrasonic element produces a voltage when sound arrives, a photodiode passes a current in proportion to light, a switch opens or closes. The rest of the chain turns that change into a clean voltage and then into a number.



```
| force...   |   | current      |   | voltage       |   | (10 bit)  |   | calibrate
+------------+   +--------------+   +---------------+   +-----------+   +--------------
Where each link is covered:   divider: Basic Electronics 3     conditioning: Basic Elec
```

The electronics of the middle links is the subject of Basic Electronics 3 (voltage dividers, which turn a resistance change into a voltage), Basic Electronics 7 (signal conditioning: amplifying a small signal and removing high-frequency noise) and Basic Electronics 10 (analogue-to-digital converter (ADC) with reference voltage V_{ref} turns the input into a whole number, and its smallest step is the **least significant bit** (LSB).

```
LSB = Vref / 2^n                          counts = floor( V / Vref x 2^n ),  limited to

Worked example: a 10-bit ADC with Vref = 5.0 V.   LSB = 5.0 / 1024 = 4.883 mV.
       A sensor that gives 20 mV per centimetre therefore resolves 4.883 / 20 = 0.244 c

Worked example: a light-dependent resistor (LDR) on the top of a divider, with 10 kohm
       Bright light,  R(LDR) =    2 kohm:    Vout = 5.0 x 10 / 12   = 4.167 V     count
       Dim room,      R(LDR) =   20 kohm:    Vout = 5.0 x 10 / 30   = 1.667 V     count
       Nearly dark,   R(LDR) = 1000 kohm:    Vout = 5.0 x 10 / 1010 = 0.0495 V    count

Notice that the answer is not proportional to the light, and the resolution is uneven.
cover a huge change of resistance. That is typical: the raw number is a measure of the
```

THREE KINDS OF OUTPUT

Analogue sensors give a voltage that you read with an ADC (the LDR above). **Digital** sensors report an on or off level (a switch, a threshold comparator), or a number already converted inside the sensor and sent over a bus (many distance, inertial and camera modules). **Timing** sensors give a pulse whose width or position carries the measurement (the ultrasonic sensor below). The processing differs, but the questions are the same: what range, what resolution, how noisy, how fast?

2. Sensor specifications

A datasheet describes a sensor with a handful of terms. Learning to read them is the quickest way to avoid buying the wrong part.

SPECIFICATION	MEANING	EXAMPLE (TYPICAL, APPROXIMATE)
Range	Smallest and largest value it can measure. Many sensors also have a *minimum* range, below which they fail	Ultrasonic 2 cm to 4 m
Resolution	The smallest change in the quantity that changes the output. Set by the ADC or by the sensor's own steps. Not the same as accuracy	0.3 cm for an ultrasonic module; 4.88 mV for a 10-bit ADC at 5 V
Accuracy	How close a reading is to the *true* value, including any consistent offset (bias) and the random noise	±3 mm plus 1 % of the reading
Precision (repeatability)	How tightly repeated readings of the *same* true value cluster: the random spread. A precise sensor can still be consistently wrong	Standard deviation 0.15 cm
Noise	Random variation in the output that does not come from the quantity. Quoted as a standard deviation (σ) or a peak-to-peak value	σ = 0.5 cm at 1 m
Response time	How long the output takes to follow a change: the time to reach 63 % (one time constant) or 90 % of a step	An LDR may take tens of milliseconds to respond (typical); a photodiode microseconds
Sample (update) rate	How many new readings per second the sensor can supply	About 15 to 25 Hz for an ultrasonic sensor; 10 Hz for a typical 2D lidar rotation
Field of view (beam angle)	The cone or width the sensor sees. A wide beam gives coverage but poor direction; a narrow beam the reverse	Ultrasonic about 15 degrees each side of the axis; infrared a few degrees
Linearity and sensitivity	Whether the output is proportional to the input, and how much output you get per unit of input	Infrared triangulation is strongly nonlinear (section 3)

Accuracy versus precision

The two words are often swapped, but they describe two different faults. **Bias** (a systematic error) makes a sensor inaccurate; **noise** (a random error) makes it imprecise. They need different cures: calibration removes bias, averaging reduces noise (sections 9 and 10). Here are ten readings from each of two imaginary sensors measuring a true value of 100.0.

```
                        94          97          100         103         106
                        |           |           |           |           |
true value                                      ^
A: accurate, not precise                  ** * *  ** *   * **
B: precise, not accurate                                      262

(one column is 0.25 units; * is one reading, a digit is that many readings in the same

Sensor A:  mean 99.95,  sigma 1.60       bias 0 (almost),  noise large          Single
Sensor B:  mean 103.00, sigma 0.15       bias +3.0,        noise tiny           Every r
Averaging 100 readings:  A's mean has sigma 1.60 / 10 = 0.16 (good).    B's mean is sti
```

RESOLUTION IS NOT ACCURACY

A sensor that displays 0.1 mm steps but is only accurate to ±5 mm is no better than a ruler with 5 mm marks. Decimals in a number show the sensor's resolution (and the code's formatting), not its truth. The accuracy line of the datasheet, with its conditions (temperature, target colour, distance), is the one to trust.

Sample rate and the robot's speed

A robot moves between readings. The distance it travels between two readings is v / f (speed over update rate). At 0.5 m/s with a 15 Hz sensor, that is 0.5 / 15 = 3.3 cm. Add the delay of any filter (section 9) and the time the program takes, and the robot may be 10 cm closer to a wall than its last number says. When you choose a sensor, ask whether it updates fast enough for the speed you want, and whether the robot can stop within the distance at which it first sees an obstacle (stopping distances come from the motor and wheel figures of Robotics Fundamentals 3, and Robotics Fundamentals 7 deals with the control loop's own delay).

3. Distance sensors

Ultrasonic: time of flight of sound

An ultrasonic sensor sends a short burst of sound at about 40 kHz (too high to hear), waits for the echo, and measures the time. Sound travels out to the object and back, so the distance is half of the speed times the time. The speed of sound in air depends on temperature T (in degrees Celsius), by a good approximation:

```
sensor                                          object
[ S ] ))) ---------------- d ----------------->   |
[ S ] <--------------------- d ------------------   |   echo
round trip time  t = 2 d / v          so     d = v t / 2          (v = speed of sound)

v(T) = 331.3 + 0.606 T   m/s          (dry air; T in degrees Celsius; humidity adds a s

T (deg C):    -10       0        20       30       40
v (m/s):     325.24   331.30   343.42   349.48   355.54          (the speed rises by 0.

Worked example (T = 20 deg C):  the echo takes t = 5.83 ms.
       v = 331.3 + 0.606 x 20 = 343.42 m/s           d = 343.42 x 0.00583 / 2 = 1.001 m
Time per centimetre of range:      2 x 0.01 / 343.42 = 58.2 microseconds      (so a tim
Longest wait for a 4 m range:      2 x 4 / 343.42 = 23.3 ms          Allowing the echoe
The beam spreads:  about 15 degrees each side of the axis (approximate), so at 2 m the
```

The wide beam is both the strength and the weakness of the ultrasonic sensor. It is a good "is something in front?" detector, since a thin pole will still reflect some sound, but it cannot tell you *where* across the beam the object is. Know the usual troubles before you trust one:

- **Soft and angled surfaces.** Fabric and foam absorb the sound; a smooth flat wall at a steep angle (more than about 20 or 30 degrees from square-on, approximate) reflects the sound away like a mirror, and the sensor sees nothing, or a longer echo that went off a second wall.
- **Crosstalk.** If several sensors fire at once, each hears the others' echoes. Fire them in turn.
- **Minimum range.** The same element transmits and listens, so for a short time after the burst it cannot hear: a blind zone of a few centimetres.
- **Temperature.** Section 11 shows how much error a wrong assumed temperature makes, and how to remove it.

Infrared triangulation

An infrared (IR) distance sensor of the triangulation type has an infrared emitter and a position-sensitive detector side by side, behind a lens, a fixed distance b (the **baseline**) apart. The emitter's narrow beam hits the object; the reflected light comes back to the lens at an angle that depends on the distance, and lands on a spot on the detector that moves as the object moves. Similar triangles turn the spot's position x into the distance.

```
        O  object
        |\
        | \
        |  \   reflected ray
      d |   \
        |    \
        E--b--L   emitter E, lens L, baseline b
              |\   f = lens to detector
        ------+-*------   detector
              |x|        spot position x from the axis (schematic, not to scale)

   Similar triangles:   x / f = b / d      so      d = b f / x       and     x = b f / d

   Worked example (illustrative, b = 20 mm, f = 10 mm, so b f = 200 mm squared):
          d (mm):                          100     150     200     300     500
          spot position x (mm):            2.00    1.33    1.00    0.67    0.40
          error in d for a spot error of 0.01 mm,   d error = d^2 x 0.01 / 200:     0.5 mm

   Close up, a tiny spot error means a tiny distance error; far away the same spot error m
```

The output voltage therefore follows roughly V ≈ k / d (typical sensors: a curve like 12 / d with d in centimetres and V in volts, rising to a peak at a minimum distance and then falling again closer in, so readings below the minimum range are *wrong*, not just inaccurate). With a 10-bit ADC at 5 V this nonlinearity decides the resolution:

```
   Model V = 12 / d (d in cm, typical shape, approximate).   Counts = floor(V / 5.0 x 1024

   d (cm)     V (V)     counts     size of one count (cm)
    10        1.200     245        0.04
    20        0.600     122        0.16
    40        0.300      61        0.65
    80        0.150      30        2.6

   One ADC count is worth 2.6 cm at 80 cm but only 0.04 cm at 10 cm: the same sensor and A
   The software linearises the reading, either by the formula d = k / V (fit k and an offs
```

Triangulation sensors have a narrow beam (they can find a thin pole or a table edge) and a faster update than ultrasound (tens of milliseconds, typical), and they fail in direct sunlight and on very dark, shiny or transparent surfaces. A simpler relative, the **IR reflective** sensor (an emitter and a

phototransistor pointing the same way), measures how much light bounces back, which depends on the surface as much as on the distance; use it to detect the presence of something (a line, a table edge) rather than to measure a distance.

Laser time of flight

Light travels about 3×10^8 m/s, a million times faster than sound, so timing a pulse of light to centimetre accuracy needs a stopwatch of picosecond resolution: a distance of 1 cm is a round trip of $2 \times 0.01 / (3 \times 10^8) = 66.7$ ps. Small laser sensors do this with specialised detectors that build up a histogram of arrival times from many pulses, or with the **phase-shift** method: modulate the light at frequency f and measure the phase delay φ of the returning light.

```
Time of flight:   d = c t / 2                c = 3 x 10^8 m/s (approx)          5 mm r
Phase shift:      d = c phi / (4 pi f)       unambiguous range = c / (2 f)       (phi

Worked example: f = 20 MHz.   Unambiguous range = 3e8 / (2 x 20e6) = 7.5 m.   A measure
       d = 3e8 x (pi / 2) / (4 pi x 20e6) = 1.875 m          One degree of phase is wor
```

Typical small laser time-of-flight modules reach about 2 m indoors (approximate, less in sunlight and on dark targets) with millimetre-level resolution, a narrow beam (a few degrees to a few tens of degrees depending on the model) and a digital interface. Glass and mirrors confuse them, as they do the infrared sensors.

Lidar

Lidar (light detection and ranging) is a laser distance sensor on a rotating or scanning mount, so it measures distance in many directions. A common low-cost 2D unit spins a single ranging element and reports a ring of readings, each an angle θ and a range r, about 5 to 10 times a second. The points convert to a map in the robot's frame by the usual polar-to-Cartesian formulas, and the spacing between neighbouring points grows with range.

```
Point in the robot frame:    x = r cos(theta)       y = r sin(theta)          Spacing o

Worked example:  a reading r = 2.0 m at theta = 30 degrees:   x = 2.0 x 0.8660 = 1.732 m
With a 1 degree step (0.01745 rad):   360 points per turn, and at 10 turns a second 360
       Spacing at  4 m:  4 x 0.01745 = 0.070 m = 7.0 cm           Spacing at 10 m:  10
A thin pole 5 cm wide can fall between two points at 4 m and be missed; a wall cannot.
```

A 2D lidar sees a single horizontal slice, so it misses a table top above it and a step below it; 3D lidar and depth cameras add the missing dimension at higher cost and data rate. Lidar is the sensor of choice for mapping a room (Robotics Fundamentals 8 returns to mapping and localisation).

DISTANCE SENSOR	RANGE (TYPICAL)	BEAM	RESOLUTION	STRENGTHS	WEAKNESSES
Ultrasonic	2 cm to 4 m	Wide, about 15 degrees each side	About 0.3 cm	Cheap; works on most surfaces including glass; unaffected by light	Wide beam; soft and angled surfaces missed; slow; depends on temperature
IR triangulation	About 10 cm to 80 cm (varies by model)	Narrow	Fine close, coarse far (1 over d)	Cheap; narrow beam; fast	Nonlinear; sunlight and surface colour matter; short range
IR reflective	Under 5 cm	Narrow	Not for distance	Very cheap; ideal for lines and edges	Depends on surface reflectance
Laser time of flight (single point)	About 3 cm to 2 m (small modules)	Narrow	About 1 mm	Accurate; compact; digital output	Sunlight and dark or transparent targets; costs more
2D lidar	About 0.15 m to 12 m (low-cost)	360 degree scan, narrow per point	Millimetres to centimetres, points spaced by r dtheta	Maps a whole plane; long range	Costly; single slice; moving parts; data rate

4. Line sensors

A line-following robot needs to know where a dark line (typically black tape on a pale floor, about 19 mm wide, approximate) lies under it. The standard device is a **reflectance array**: a row of IR reflective sensors (an emitter and a phototransistor each), a few millimetres above the floor, spaced across the robot's width. A pale floor reflects a lot of light (a low or a high reading depending on the circuit), and a black line absorbs it. Each sensor yields an analogue value; the program turns the whole row into a single number: the position of the line.

Threshold and normalisation

Raw values differ between sensors (their gains and mounting heights differ) and between floors, so first **calibrate each sensor**: sweep the array over both the line and the floor and record each sensor's lowest value lo_i (white) and highest hi_i (black). Then rescale every reading to a common 0 to 1 scale, $w_i = (raw_i - lo_i) / (hi_i - lo_i)$ (limited to 0 and 1). A simple "line or not" decision compares w_i with a threshold, usually 0.5, halfway between the floor and the line.

The weighted average line position

Let sensor i sit at position x_i across the bar (zero at the centre, positive to the right). If you only use the on or off decisions, the position can only take values at the sensors, a resolution equal to the spacing. Using the full analogue weights gives a much finer, continuous answer, the **weighted average** (a centre of mass of the dark pixels):

```
Line position:    x(line) = sum( w_i x_i ) / sum( w_i )             Line lost if   sum

Sensor positions for n sensors with pitch p:   x_i = ( i - (n - 1) / 2 ) p        (i = (

Worked example: 8 sensors, pitch 8 mm (typical), so the positions are -28, -20, -12, -4,

sensor i:        0       1       2       3       4       5       6       7
x_i (mm):      -28     -20     -12      -4      +4     +12     +20     +28
w_i:          0.00    0.00    0.00    0.30    1.00    0.45    0.00    0.00
w_i x_i:      0.00    0.00    0.00   -1.20   +4.00   +5.40    0.00    0.00

       sum(w) = 1.75              sum(w x) = -1.20 + 4.00 + 5.40 = 8.20 mm
       x(line) = 8.20 / 1.75 = 4.69 mm to the right of the centre of the bar.
       With only on or off (threshold 0.5): only sensor 4 is on, so x = +4 mm, a resolu

A noisy white-floor reading of 0.05 on the far sensor 7 (x = +28 mm) shifts the answer
Ignore weights below a small noise floor (about 0.05 to 0.1) so that far-away noise can
```

The robot steers to drive that position to zero. With the sign convention of Robotics Fundamentals 4, where counter-clockwise turning is positive, a line to the *right* (positive x) needs a clockwise turn. How hard to steer for a given error is the job of the control loop in Robotics Fundamentals 7, and Robotics Fundamentals 8 builds the full line-following behaviour. The array is best mounted ahead of the wheels: the further ahead it sits, the earlier it sees the line curve away, although a long lever arm also makes a hunting (oscillating) robot swing more.

Here is the calculation as short, hardware-neutral Python-style pseudocode. The sensor reading and the steering are left to Programming for Robotics.

```
def line_position(raw, lo, hi, pitch, floor=0.05, lost=0.2):
    n = len(raw)
    total = 0.0
    moment = 0.0
    for i in range(n):
        w = (raw[i] - lo[i]) / (hi[i] - lo[i])       # 0 on the floor, 1 on the line
        w = min(1.0, max(0.0, w))
        if w < floor:                                 # ignore small noise readings
            w = 0.0
        x = (i - (n - 1) / 2) * pitch                 # position across the bar, mm
        total += w
        moment += w * x
    if total < lost:
        return None                                   # line lost: the caller decides wh
    return moment / total                             # millimetres, positive to the rig
```

5. Bump and limit switches

The simplest sensor is a switch. A **bump switch** (a micro-switch with a lever, or a whisker) tells a robot that it has touched something; a **limit switch** tells a mechanism that it has reached the end of its travel (an arm joint, a gantry axis, a gripper closed). Both are digital and cheap, and both are the last line of defence, which is why they are worth fitting even on a robot full of cleverer sensors.

- **Wiring.** A switch needs a pull-up (or pull-down) resistor so the input has a defined level when the switch is open (a pull-up is a resistor from the input to the supply). The common arrangement is a pull-up with the switch to ground: open reads high, pressed reads low.
- **Fail-safe wiring.** A switch that is *normally closed* (pressed means open) makes a broken wire look the same as a pressed switch, so a fault stops the robot rather than leaving it blind. For safety-related limits, this is the better choice.
- **Bounce.** The metal contacts rattle for a few milliseconds when they close (typically 1 to 10 ms, approximate), making a burst of on and off readings. Debounce in hardware (a small capacitor) or in software: accept a change only if the new level has been stable for about 10 to 20 ms.

- **Placement.** A bump switch only reports after contact, so it tells you about an obstacle *after* the robot has hit it; at speed it is a collision detector, not an avoidance sensor. A limit switch should be set so that the mechanism stops before it hits its hard stop, with some travel to spare for the time it takes to brake.

6. Light sensors

Light sensors turn brightness into a voltage. A **light-dependent resistor** (LDR) is cheap and slow, with a response of tens of milliseconds and a strongly nonlinear resistance; a **photodiode** or **phototransistor** is fast and more linear, giving a current in proportion to the light. Section 1 worked an LDR in a divider: the output is higher in bright light (when the LDR is on the top of the divider), and the divider turns resistance into a voltage that the ADC reads.

- **Uses.** Detecting day or night, seeking a light source (two sensors pointing slightly apart; steer toward the brighter), reading a surface's brightness (the line and edge sensors above), and as the receiver in infrared beam-break and remote-control links.
- **Ambient light.** Room lights, sunlight and shadows change the reading. Sensors that detect the robot's own infrared *modulated* at a known frequency (and ignore everything else) are much more robust; a simple software version takes one reading with the emitter on and one with it off, and uses the difference.
- **Colour.** A colour sensor is a small group of photodiodes behind red, green and blue filters, read as three numbers; the same lighting caveats apply, so calibrate against white and black.

7. Current sensing, briefly

A robot can *feel* its own motors by measuring the current they draw. A motor under load draws more current (Robotics Fundamentals 3), and a stalled motor draws the maximum, so the current tells you about load, stall, a jammed gripper and the state of the battery. The usual sensor is a **shunt resistor**: a small resistor, a fraction of an ohm, in series with the motor. The current I causes a voltage V = I R across it, which is tiny, so an amplifier (Basic Electronics 7) raises it to a range the ADC can read.

```
Shunt voltage:  V = I R             Heat in the shunt:  P = I^2 R             Amplifi

Worked example: R = 0.1 ohm, G = 10 (illustrative).
```

```
        Running, I = 0.4 A:   V = 0.04 V,   amplified 0.40 V,   counts = floor(0.40 / 5 
        Stalled, I = 2.5 A:   V = 0.25 V,   amplified 2.50 V,   counts = 512
A program that sees counts rise towards 512 can cut the power before the motor and wiri
The compromise: a bigger R gives a bigger signal but wastes more voltage and power; a s
```

8. Cameras, briefly

A camera is a sensor with hundreds of thousands of “pixels” instead of one, so it can see what the other sensors cannot: colour, shape, texture, text, the difference between a ball and a wall, and many objects at once. Cameras are now cheap, and are why vision-based robots can do things that were impossible with distance sensors alone. But they are hard to use, for four reasons.

- **Data volume.** A 640 by 480 colour image at 30 frames a second is 640 × 480 × 3 bytes × 30 = 27,648,000 bytes a second (about 27.6 MB/s) before any processing. A small controller cannot cope; it needs a more powerful processor, or a camera module that does some of the work itself.
- **Depth is lost.** A single camera records a flat projection; a distant large object and a near small one can look the same. If the size of an object is known, distance follows from similar triangles, d = f W / w (focal length f in pixels, real width W, width in the image w in pixels). For f = 600 pixels and a 0.10 m wide object seen 60 pixels wide, d = 600 × 0.10 / 60 = 1.00 m; but a one-pixel error in w (59 or 61 pixels) gives 1.017 m or 0.984 m, so the error is about 1.7 % per pixel and grows as the object shrinks. Two cameras (stereo) or a depth sensor measure depth directly.
- **Lighting.** The same object looks different in sunlight, shade and lamp light; algorithms that work in the lab can fail outdoors.
- **Interpretation.** A distance sensor gives you a distance. A camera gives you an array of numbers, and turning it into “there is a red ball two metres to the left” takes software (computer vision), which is outside the scope of this course.

9. Noise and filtering

Real readings jump about. Part of it is **noise**: small random variation, often well described by a bell-shaped (Gaussian) spread with a standard deviation σ. Part is **outliers**: occasional wild

values (an ultrasonic echo off the wrong wall, a missed pulse reading zero, a spike from a motor). Filters reduce both, and different filters suit each kind. Every filter makes the same trade: **less noise, but more delay**.

Averaging many independent readings

If you take N readings with independent random errors each of standard deviation σ, their mean has a smaller spread, the **standard error**:

```
Mean:   x(mean) = (x1 + x2 + ... + xN) / N            Spread of the mean:   sigma(mea

Worked example: sigma = 0.5 cm for a single reading.
       N:            1       4       16      25      100
       sigma(mean):  0.50    0.25    0.125   0.100   0.050 cm

Four times the readings only halves the noise, and 100 times the readings cut it by 10:
It holds only for independent random noise. A bias (section 2), or a slow drift that ch
```

Filters for a stream of readings

FILTER	FORMULA	GOOD FOR	COST
Moving average of N	Mean of the last N readings	Smoothing random noise; simple	Delay of (N − 1) / 2 samples; smears an outlier over N readings; stores N values
Median of N (N odd)	The middle value of the last N sorted	Removing outliers and spikes, while keeping sharp edges	Less smoothing of ordinary noise; a little more computation
Exponential filter	$y_k = y_{k-1} + \alpha\,(x_k - y_{k-1})$, $0 < \alpha \le 1$	Smoothing with almost no memory or computation: one stored number	Delay grows as α falls; an outlier decays away gradually, never vanishes at once

The exponential filter (also called an exponential moving average or EMA) is the digital version of the simple RC low-pass filter of Basic Electronics 4. The factor α says how much of each new reading to trust: α = 1 passes the readings unchanged, and a small α smooths a lot and reacts slowly. After a step change in the input, the output covers a fraction $1 - (1 - \alpha)^n$ of the step in n samples; it reaches 63 % after about $1 / (-\ln(1 - \alpha))$ samples, roughly $(1 - \alpha) / \alpha$ as a rule of thumb, which with a sample interval dt gives a time constant of about $dt\,(1 - \alpha) / \alpha$.

Worked example: one outlier

An ultrasonic sensor measures a wall 100 cm away. Twelve consecutive readings contain one wild echo (130.0). The table compares a mean of the last three readings (MA3), a median of the last three (Med3) and an exponential filter with α = 0.25 (started from the first reading).

```
k     raw      MA3      Med3     EMA(0.25)
 1    100.4      -        -       100.40
 2     99.7      -        -       100.23
 3    100.2    100.10    100.2    100.22
 4     99.5     99.80     99.7    100.04
 5    130.0    109.90    100.2    107.53        <-- the outlier arrives
 6    100.1    109.87    100.1    105.67
 7     99.8    109.97    100.1    104.20
 8    100.6    100.17    100.1    103.30
 9     99.9    100.10     99.9    102.45
10    100.3    100.27    100.3    101.91
11     99.6     99.93     99.9    101.34
12    100.0     99.97    100.0    101.00

Mean of all 12 readings:  102.51 (the outlier drags it 2.5 cm off)         Median of al
MA3:     jumps by 9.9 and stays wrong for three readings (the outlier is in the window
Med3:    never worse than 0.3 off: a single wild value never becomes the middle one.

Step response of the EMA (alpha = 0.25) to a jump from 100 to 120:     after 1, 2, 3, 4

The delay is the price: a moving average of N readings lags the truth by (N - 1) / 2 sa
```

The three filters as short, hardware-neutral Python-style pseudocode:

```
def moving_average(window, x, n):
    window.append(x)                        # window is a list of the latest readings
    if len(window) > n:
        window.pop(0)                       # drop the oldest reading
    return sum(window) / len(window)

def median_filter(window, x, n):            # n should be odd
    window.append(x)
    if len(window) > n:
        window.pop(0)
    return sorted(window)[len(window) // 2]   # the middle value
```

```
def exponential_filter(y_prev, x, alpha):     # keep y_prev between calls; start it at
    return y_prev + alpha * (x - y_prev)      # alpha = 1 passes x unchanged, small alp
```

FILTER THE RIGHT THING, AT THE RIGHT RATE

Hardware does part of this for free: a capacitor across a sensor output is an RC filter before the ADC (Basic Electronics 4). Filter *before* you take a decision from the number, but keep the delay small in a control loop (Robotics Fundamentals 7): a robot steering by a heavily smoothed reading reacts late and can oscillate. For a quick "obstacle ahead?" decision, a median of 3 or 5 removes false alarms with little delay.

10. Calibration

Calibration removes the *systematic* part of a sensor's error by measuring it against known true values. If a sensor reads in a straight-line way, reading = gain × true + offset, only two numbers need finding, and the inverse turns a reading back into the true value.

```
Two-point calibration:   take two known true values t1 and t2 (at the two ends of the w
       g = (t2 - t1) / (r2 - r1)              true = g (r - r1) + t1              (th

Worked example: a distance sensor reads 12.4 cm at a true 10.0 cm and 102.9 cm at a tru
       g = (100.0 - 10.0) / (102.9 - 12.4) = 90.0 / 90.5 = 0.99448          true = 0.994
       Reading 77.3 cm:   true = 0.99448 x 77.3 - 2.331 = 74.54 cm        (without cali

Check points (true, reading):   (30, 32.1)   (50, 52.6)   (75, 77.3)      errors of the

Least-squares straight line through ALL five points (10, 12.4), (30, 32.1), (50, 52.6),
       so true = (r - 2.172) / 1.00543      errors at the five points:   +0.17   -0.23
```

- **Average the calibration readings.** A two-point calibration puts the noise of two readings straight into the gain and the offset, so average 50 or 100 readings at each point (section 9: the noise falls as 1 over the square root of N) and place the points near the two ends of the range you use.
- **More points, and a check point.** A line through several points, fitted by least squares (the line that minimises the sum of squared errors), averages their noise; and a point *not* used in

the fit shows whether the straight-line model is any good. If it is not, use a curve or a lookup table.

- **Calibrate in a linear variable.** The infrared sensor of section 3 is nonlinear in distance but close to linear in 1 / V (since V ≈ k / d), so fit d against 1 / V as a straight line.
- **Line and light sensors** are calibrated in the same way, with the floor and the line as the two known "true" points: section 4's lo_i and hi_i are a two-point calibration of each sensor.
- **Calibrations expire.** Temperature, supply voltage, lighting and wear change a sensor, so recalibrate when conditions change, or compensate (section 11).

11. Error propagation

A measured distance is computed from other measured quantities, each with its own error. **Error propagation** estimates the error in the result. For a product or quotient, the *relative* errors of independent inputs add in quadrature (square, add, square root); for a sum or difference, the *absolute* errors do.

```
For q = a b / c  (independent errors):     dq / q = sqrt( (da/a)^2 + (db/b)^2 + (dc/c)^

The ultrasonic distance  d = v t / 2,  with  v = 331.3 + 0.606 T:      dd / d = sqrt( (

Worked example 1: a 3 deg C error in the assumed temperature (T = 20 deg C), at d = 1 m
       dv / v = 0.606 x 3 / 343.42 = 0.529 %    ->   an error of 5.3 mm at 1 m  (and 21

Worked example 2: a timing jitter of 20 microseconds in the echo measurement.
       dt / t = 20e-6 / 5.824e-3 = 0.343 % at 1 m   ->   3.4 mm.      The distance erro

Combined at 1 m:   sqrt( 5.3^2 + 3.4^2 ) = 6.3 mm  (0.63 %).         Combined at 4 m:

The same error as a bias.  If the program assumes 20 deg C when the air is really 5 deg
       the true speed is 331.3 + 0.606 x 5 = 334.33 m/s;   the echo takes 2 / 334.33 = 
The fix: measure the temperature (a cheap sensor next to the transducer) and use v(T) =
```

FIND THE DOMINANT ERROR FIRST

In the example the temperature term was worth more than the timing term from about 1 m onwards. Improving the timer further would have gained nothing. Error propagation is how you find out which input to improve, and which to leave alone.

12. Choosing a sensor

Start from the *job*, not the sensor. Ask what the robot needs to know, how accurately, how fast, over what range, and in what environment (light, dust, floor, other robots), and what the data will cost in wiring, processing and money.

JOB	GOOD CHOICES	TRADE-OFFS
Is something in front of me? (obstacle detection)	Ultrasonic; laser time-of-flight; IR triangulation	Ultrasonic is wide and cheap but misses angled and soft surfaces; laser and IR are narrow and may miss a thin object between beams; none sees a glass door reliably
Follow a line	IR reflectance array (5 to 8 sensors); a camera for complex tracks	The array is cheap and fast but sensitive to ambient light and floor; a camera is flexible but needs processing
Detect a table edge or step (cliff)	Downward IR reflective sensor	Very cheap; short range; fooled by dark or shiny floors, so test on the real floor
Measure distance to a wall to a few millimetres	Laser time-of-flight; IR triangulation at close range	Higher cost; sunlight and surface colour affect them; ultrasonic is cheaper but only good to a few millimetres at best (typical)
Map a room	2D lidar (a camera plus software is an alternative)	Costly, with a high data rate; sees one plane only
Know that I have hit something	Bump switch or whisker	Very cheap and certain, but reports only after contact
Know a mechanism has reached its end	Limit switch (normally closed wiring); hall or optical end-stop	Switches wear; optical sensors need clean surroundings
Know it is day or night, or find a light	LDR; photodiode; phototransistor	An LDR is cheap but slow and nonlinear; a photodiode is faster and more linear but needs a better circuit
Detect a stalled motor or a jammed gripper	Current sensing (shunt and amplifier)	Costs a resistor, an amplifier and an ADC channel; the shunt wastes some power and the signal is noisy
Recognise objects, colours or markers	Camera (with software)	Richest data but heavy processing, lighting dependence and cost

Two more rules of thumb. First, **combine sensors** so that each covers the other's weakness: an ultrasonic sensor for wide coverage with an infrared for a narrow check, a bump switch under everything as a last resort. Second, **measure on your own robot, floor and lighting** before designing the behaviour around a number. Robotics Fundamentals 6 adds the sensors that measure the robot itself (encoders, gyroscopes, accelerometers) and how to combine their data with those from this chapter.

13. Challenges

CHALLENGE 1: TIME OF FLIGHT AND ITS ERRORS

(a) An ultrasonic sensor in air at 25 degrees C measures an echo time of 8.40 ms. Find the speed of sound and the distance. (b) The program wrongly assumes 20 degrees C. What distance does it report for the same echo, and what is the error in millimetres and in per cent? (c) The air is really 35 degrees C and an object is truly 2.00 m away, but the program assumes 20 degrees C. Find the echo time and the distance reported, and the error. (d) At 1.5 m and 20 degrees C, the timing jitter is 30 microseconds and the temperature is known to within 2 degrees C. Find the error from each source and the combined error in millimetres. (e) A laser time-of-flight sensor must resolve 5 mm: what timing resolution does it need? A phase-shift sensor modulated at 10 MHz measures a phase of 60 degrees: find the distance, the unambiguous range, and the distance that one degree of phase is worth. (f) How wide is the ultrasonic beam at 3 m, taking 15 degrees each side of the axis, and why can it not tell you which side a thin pole is on?

View solution

CHALLENGE 2: FILTERING AND AVERAGING

A distance sensor watches a target 150 cm away and produces ten readings (cm): 150.2, 149.6, 150.5, 149.9, 168.0, 150.1, 149.7, 150.4, 150.0, 149.8. (a) Find the mean, the median, and the mean without the outlier. (b) Find the moving average of 3 and the median of 3 for readings 3 to 10, and say how long each is wrong because of the outlier and by how much at worst. (c) Find the exponential filter output with alpha = 0.4, starting from the first reading, for all ten readings, and the error at each step from the outlier onwards (take the true value as 150.0). (d) A single reading has sigma = 0.4 cm. How many independent readings give a mean with sigma at most 0.1 cm, and how long does that take at 25 readings a second? How far does a robot at 0.3 m/s move in that time? How many for 0.05 cm? (e) With alpha = 0.4 and dt = 0.04 s, find the approximate time constant, and the fraction of a step the filter has covered after 1 to 5 samples. (f) What delay in seconds does the moving average of 3 add?

View solution

CHALLENGE 3: LINE POSITION, CALIBRATION, CURRENT AND CHOICE

(a) A six-sensor line array has pitch 10 mm. Each sensor's calibration gave lo = 60, 70, 80, 75, 65, 90 and hi = 860, 880, 900, 870, 850, 900 (ADC counts). The raw readings are 62, 150, 700, 820, 280, 95. Find the sensor positions, the six normalised weights, and the line position by the weighted average. What would an on or off decision with a threshold of 0.5 report? Which way should the robot turn (use the convention of Robotics Fundamentals 4)? (b) A distance sensor reads 8.2 cm at a true 10.0 cm and 196.5 cm at a true 200.0 cm. Find the two-point calibration (gain and offset), and the true value for a reading of 120.0 cm. A check point reads 101.4 cm when the true distance is 100.0 cm: what does the calibration give there, and what does that tell you? (c) A motor current sensor uses a shunt of 0.05 ohm and an amplifier of gain 20, into a 10-bit ADC at 5 V. Find the counts at a running current of 0.5 A and a stall current of 3.2 A, the heat in the shunt at stall, and the count at which a 2.0 A trip should operate. (d) Suggest a sensor, with one reason and one drawback, for: stopping a small robot at a table edge; keeping a robot 20 cm from a wall to within 5 mm; stopping a robot arm joint at the end of its travel safely.

View solution

QUICK REFERENCE

- **Chain:** physical quantity, transducer, conditioning, ADC, software. LSB = $V_{ref} / 2^n$; counts = floor($V / V_{ref} \times 2^n$).
- **Specifications:** range, resolution (not accuracy), accuracy (bias plus noise), precision (spread), noise, response time, field of view, sample rate. Bias is cured by calibration, noise by averaging.
- **Ultrasonic:** $d = v\,t / 2$; $v = 331.3 + 0.606\,T$ m/s; 58.2 µs per cm at 20 °C; wide beam; soft and angled surfaces fail.
- **IR triangulation:** $d = b\,f / x$; output about k / d; error grows as d^2; resolution is coarse far away.
- **Laser time of flight and lidar:** $d = c\,t / 2$ (1 cm is 66.7 ps); phase shift $d = c\,\varphi / (4\,\pi\,f)$, range $c / (2f)$; lidar $x = r \cos\theta$, $y = r \sin\theta$, point spacing $r\,d\theta$.
- **Line sensors:** normalise $(raw - lo) / (hi - lo)$; position = $\Sigma\,w\,x / \Sigma\,w$; lost if $\Sigma\,w$ is small.
- **Switches:** pull-up resistor, debounce 10 to 20 ms, normally closed wiring for fail-safe limits. **Current sensing:** $V = I\,R$, $P = I^2R$, amplify then read.
- **Camera:** rich but heavy data (27.6 MB/s for 640 × 480 colour at 30 frames a second); depth needs two views or a known size, $d = f\,W / w$.

- **Filters:** mean, moving average (delay (N − 1) / 2 samples), median (kills outliers), exponential y += α (x − y); σ(mean) = σ / √N.
- **Calibration:** g = (t2 − t1) / (r2 − r1), true = g (r − r1) + t1; average the readings, use more points and a check point; fit a linear variable.
- **Error propagation:** relative errors of a product add in quadrature; temperature error is proportional to range, timing error is constant; fix the dominant error first.

Next, Robotics Fundamentals 6 turns the sensors on the robot itself: position, orientation and feedback. Wheel encoders, which count the turns the odometry of Robotics Fundamentals 4 depends on; inertial units with gyroscopes, accelerometers and compasses that measure how the robot is turning and tilting; why those readings drift, and how to combine several imperfect sensors (sensor fusion) in plain words; dead reckoning; and why GPS is not the answer indoors. The filters, calibration and error estimates from this chapter will be used all the way through.

Robotics Fundamentals

Course 1 · Chapter 6 · Position, Orientation and Feedback

Robotics Fundamentals 5 was about the sensors that look *outward*: distance, line, light and camera. This chapter is about the sensors that look *inward* and answer three questions every moving robot has to ask. How far have my wheels turned, and how fast? Which way am I facing, and am I tilted? And so, where am I? The answers are the robot's **pose** (its position and heading), and they are the raw material for everything in Robotics Fundamentals 7 onwards: a robot cannot correct an error it cannot measure. We start with encoders, which count wheel turns (the odometry of Robotics Fundamentals 4 depends on them), then the inertial measurement unit (gyroscope, accelerometer and compass), then the central trick of the chapter: no single sensor is trustworthy, but two sensors with *opposite* weaknesses can be combined into one good one (sensor fusion). We then use that to improve dead reckoning, see why GPS is not the answer indoors, tidy up reference frames and angle wrapping, and finish with why all of this matters: **feedback**.

Sensor specifications in this chapter (counts per revolution, scale factors, noise densities, GPS accuracy, update rates, magnetic field strengths) are **typical or approximate values from memory**, for the low-cost parts used on small robots. They vary widely between models, so read the datasheet of the part you buy. The formulas (resolution, quantisation, integration, atan2 tilt, the complementary filter and its time constant, the frame rotation, angle wrapping) are standard and exact for the idealised models described.

1. Pose, and two ways of knowing it

On a flat floor a robot's pose has three numbers: (x, y, θ), a position and a heading, as in Robotics Fundamentals 4. A flying or tilting robot adds height, pitch and roll. There are two families of sensor for finding it, and the difference between them runs through the whole chapter.

	RELATIVE (INCREMENTAL) SENSING	ABSOLUTE SENSING
What it tells you	How much the pose has *changed* since the last reading	Where you are, or which way you face, now, against something fixed

	RELATIVE (INCREMENTAL) SENSING	ABSOLUTE SENSING
Examples	Incremental encoders, gyroscope, wheel odometry	Compass, GPS, absolute encoder, potentiometer, a wall or beacon at a known place
Error behaviour	Small, smooth readings, but the small errors are **added up** and the total drifts without limit	Each reading has its own error, which does **not** accumulate, but readings are noisier, slower or sometimes unavailable
Best at	Fast changes, short times	Slow changes, long times

The two columns are complementary. Relative sensors are good in the short term and bad in the long term; absolute sensors are the reverse. Nearly every technique in this chapter is a way of using an absolute reference to rein in the drift of a relative one.

2. Incremental (quadrature) encoders

A wheel encoder is attached to a motor or wheel shaft and produces a train of pulses as the shaft turns. A **magnetic** encoder has a small ring magnet on the shaft and two Hall sensors beside it; an **optical** encoder has a slotted disc passing between a light source and a detector. A single pulse train tells you how far the shaft has turned but not *which way*. The fix is to use two sensors placed so that their signals are a quarter of a cycle (90 degrees) apart. These are the **A** and **B** channels, and the arrangement is called **quadrature**.

```
Forward (A leads B). Each state below lasts 5 characters.  State is written AB.

state  00   10   11   01   00   10   11   01
A      _____----------__________----------_____
B      __________----------__________----------
count       +1   +1   +1   +1   +1   +1   +1       (each change of A or B is one count)

Reverse (B leads A): the same four states in the opposite order, 00 01 11 10 00 ...  an

Four states per cycle, always in this order, so:   forward 00 -> 10 -> 11 -> 01 -> 00
A step to the same state is 0;  a jump of two (00 to 11, or 01 to 10) means a state was
```

Counting every edge: 1x, 2x and 4x decoding

One full cycle of A has two edges, and B adds two more, so a cycle has four transitions. The **CPR** (cycles, or lines, per revolution) printed in a datasheet counts *cycles of one channel*. You choose how much of the signal to use: counting only the rising edge of A gives 1× (one count per cycle); counting both edges of A gives 2×; counting every edge of both A and B gives **4×**, which is four times the resolution from the same encoder and is the usual choice. Watch the vocabulary: some datasheets quote "pulses per revolution" (PPR, one channel), others quote counts after 4× decoding. Always check which, because it changes every number below by a factor of four. In this chapter CPR means cycles per revolution of one channel, and the counts per revolution are 4 × CPR.

```
Counts per revolution (shaft where the encoder sits):    4 x CPR
If the encoder is on the motor, per revolution of the WHEEL:    counts = 4 x CPR x gear

Angular resolution of the wheel:    360 / (CPR x 4 x gear)  degrees per count

Worked example: a small geared motor with a magnetic encoder, CPR = 11 (typical), gear
       Counts per motor revolution:   4 x 11 = 44                    Counts per wheel
       Angular resolution:            360 / 1320 = 0.273 degrees per count
       Wheel circumference:           pi x 65 = 204.2 mm             Distance per coun

Fast, but how fast?   Wheel at 0.75 m/s (near the top speed of this motor, illustrative
       The motor shaft turns 30 times as fast as the wheel:   0.75 / 0.2042 = 3.67 whee
```

Because the encoder sits on the fast motor shaft, the gearbox multiplies the resolution for free: a fraction of a millimetre per count is typical, much finer than the sloppiness of the wheel and the floor. The price is the **count rate**: 4848 changes a second, each of which must be seen. A program that checks the pins in a loop (polling) will miss some, and a missed state breaks the count. The remedy is an **interrupt**: the hardware stops the main program the instant a pin changes, runs a very short routine, and returns. Interrupts are covered in Basic Electronics 10, together with the rules for writing a short interrupt routine; many microcontrollers also have a hardware counter that decodes quadrature with no software at all, and a dedicated decoder chip is another option.

Decoding in pseudocode

The decoder keeps the last state and looks up the move from the old state to the new one in a table. This is hardware-neutral Python-style pseudocode; the interrupt wiring is left to Programming for Robotics.

```
# state = (A << 1) | B, so 00 = 0, 01 = 1, 10 = 2, 11 = 3.   Row = old state, column = n
STEP = [  0, -1, +1,  0,       # old 00:  to 01 is -1 (reverse), to 10 is +1 (forward),
         +1,  0,  0, -1,       # old 01
         -1,  0,  0, +1,       # old 10
          0, +1, -1,  0 ]      # old 11

count = 0
state = 0

def on_edge(a, b):             # called by the interrupt whenever A or B changes
    global count, state
    new = (a << 1) | b
    count += STEP[(state << 2) | new]
    state = new

def wheel_speed(count_now, count_before, dt, d_count):     # d_count = distance per cou
    return (count_now - count_before) * d_count / dt        # metres per second, negati
```

Speed from counts, and the quantisation error

Speed is the change in count over a time interval, v = Δcount × d_{count} / Δt, and it is the number the control loop of Robotics Fundamentals 7 will use. But counts are whole numbers. In one interval the encoder can only report 38 or 39 counts, never 38.8. Over a window of N counts the reading can be wrong by about one count, so the relative error is about 1 / N. This is **quantisation error**, and it is worst at low speed and over short intervals.

```
Counts in one interval:   N = v dt / d_count          Speed resolution (one count):   v

Same encoder as above (d_count = 0.1547 mm = 0.0001547 m).
       v = 0.30 m/s,  interval 20 ms:     N = 0.30 x 0.020 / 0.0001547 = 38.8 counts
       v = 0.30 m/s,  interval 100 ms:    N = 193.9 counts
       v = 0.02 m/s,  interval 20 ms:     N = 2.6 counts

The resolution of the speed reading, d_count / dt = 0.0001547 / 0.020 = 0.0077 m/s, doe
The trade is the same as for every filter in Robotics Fundamentals 5:  a longer interva
At very low speeds, time the gap BETWEEN counts instead (speed = d_count / gap): the ga
```

TWO WAYS TO GET SPEED

> **Count over a fixed time** (above) is simple and is best at high speed. **Time between counts** gives a good answer at low speed but is noisy at high speed, where the gaps are tiny, and gives nothing when the wheel is stopped. Some robots use the first at high speed and switch to the second at low speed, or smooth either with the exponential filter of Robotics Fundamentals 5.

What encoders cannot tell you

- **Where the *wheel* is, not where the *robot* is.** The encoder counts shaft turns. If the wheel slips on the floor, or lifts, or the gearbox has backlash, the count says one thing and the ground says another (Robotics Fundamentals 4 section 6).
- **The starting position.** An incremental encoder starts at zero whenever the power comes on. It counts *changes*, so it cannot know where the shaft was. That is the job of the absolute sensors below.
- **Missed counts.** A noisy line, a slow interrupt or a long delay with interrupts switched off drops counts, and the count is wrong from then on, silently. A dropped state shows up as a jump of two in the table above.
- **Index pulse.** Many encoders add a third channel, **Z** (index), that pulses once per revolution, which gives an absolute reference once per turn.

3. Absolute encoders and potentiometers

An **absolute encoder** reports the shaft angle directly, and still knows it after the power is cycled. A disc carries several concentric tracks of coded patterns, read together, so each position has a unique code; n tracks give 2^n positions in one turn. Codes are usually **Gray code**, in which only one bit changes between neighbouring positions, so a reading taken exactly at a boundary is at worst one position out, not wildly wrong. Many low-cost absolute encoders are magnetic and send their number over a digital bus. They are used at joints (Robotics Fundamentals 9), where the arm must know its angle at switch-on without searching for a home position.

The simplest absolute angle sensor is the **potentiometer** (Basic Electronics 3): a resistive track and a wiper on the shaft form a voltage divider whose output is proportional to angle, read by an ADC (Basic Electronics 7 and 10). Its limits are the limited travel (typically 270 or 300 degrees, approximate), the dead zone at the end of the track, wear, and the noise of a sliding contact. Hobby servos use one internally to close their own position loop.

```
Absolute encoder:   resolution = 360 / 2^n degrees          12 bit:  360 / 4096 = 0.087
Potentiometer on an n-bit ADC over a travel of T degrees:   resolution = T / 2^n

Incremental versus absolute at a joint:   the incremental encoder on a geared motor abo
       the 12-bit absolute encoder is a little finer than the potentiometer, needs no h
       starting angle and the fine incremental count for the changes.
```

4. The inertial measurement unit (IMU)

An **IMU** is a small chip (usually mounted on a breakout board and read over a serial bus such as I2C or SPI, mentioned in Basic Electronics 10) that combines several motion sensors, normally a three-axis **gyroscope**, a three-axis **accelerometer**, and sometimes a three-axis **magnetometer** (compass). Each measures a different thing, each is flawed in a different way, and the flaws are what the next section exploits. The sensors are tiny vibrating or flexing structures (MEMS) whose motion is converted to a voltage, then digitised inside the chip, so you receive *integers* and a **scale factor** that converts them to units.

Gyroscope: rate of turn

A gyroscope measures **angular rate**: how fast the robot is turning, in degrees per second (or radians per second), about each axis. It does not measure angle. To get the heading you **integrate**: add up the rate multiplied by the time step.

```
Rate from the raw integer:    rate = raw / sensitivity          (sensitivity in LSB per
Typical: a 16-bit gyro on the +-250 degree/s range:   sensitivity = 32768 / 250 = 131 L
A raw reading of 262 means 262 / 131 = 2.0 degrees per second.

Integration:   angle(new) = angle(old) + rate x dt          (dt = the true time between

Worked example 1, perfect gyro:  turning at 45 degrees/s, 100 samples a second (dt = 0.
Worked example 2, wrong dt:  the program assumes dt = 0.0100 s but the loop really runs
       so the true turn is 94.5 degrees while the program reports 90.0: a 5 % scale err
```

Drift from bias

A real gyro never reads exactly zero when still. It has a **bias** (zero-rate offset): a small constant rate that it reports even when stationary. It also has random **noise** on every sample. Integrating

noise gives a slow random wander, but integrating a bias gives a steady run-away, because a constant added each step makes a straight-line growth in the angle.

```
Heading error from a bias b after a time t:    error = b t

Worked example: a gyro with a bias of 0.5 degrees/s (typical for a low-cost part straigh
       After  1 minute:   0.5 x 60 = 30 degrees          After 1 hour:   0.5 x 3600 = 1
A robot that thinks it has turned 30 degrees after sitting still for a minute is useles

Cure 1: measure the bias while the robot is stationary at start-up (say 200 samples, 2
       The noise averages down as 1 over the square root of N (Robotics Fundamentals 5)
       The residual 0.0035 degrees/s then drifts only 0.0035 x 60 = 0.2 degrees in a mi
Cure 2: the bias is not constant. It moves with temperature and over minutes (the “bias

The random-noise part is small by comparison:   per-sample noise 0.05 degrees/s at 100
Bias dominates; the noise only adds a little uncertainty. After calibration the fault t
```

Accelerometer: tilt from gravity

An accelerometer measures **acceleration along each axis**, in units of g (one g is 9.81 m/s^2) or m/s^2. Importantly, it cannot tell gravity from motion: a robot standing still reads *one g, straight up*, and that reading is exactly what lets us find the tilt. Tipping the robot moves part of gravity from one axis to another. With the sensor flat, a_z reads 1 g and the others read 0; tilted, the ratio of the axes gives the angle. Use **atan2**, not a plain arctangent, because atan2 takes both numbers separately, gives the correct quadrant (a full range of angle instead of just −90 to +90 degrees) and does not divide by zero.

```
Tilt (roll about the x axis, pitch about the y axis; signs depend on how the chip is mo
       roll  = atan2( ay, az )           pitch = atan2( ax, sqrt( ay^2 + az^2 ) )
Magnitude check:   |a| = sqrt( ax^2 + ay^2 + az^2 )   should be 1.00 g if the robot is

Worked example (robot standing still, nose tilted 15 degrees):    ax = sin 15 = 0.2588
       pitch = atan2( 0.2588, 0.9659 ) = 15.0 degrees        |a| = sqrt( 0.0670 + 0.933

Now the robot accelerates forward at 0.3 g (a hard start), still tilted 15 degrees:
       pitch = atan2( 0.5588, 0.9659 ) = 30.1 degrees          error 15.1 degrees

What the accelerometer cannot do:  it cannot see rotation about the vertical axis (turn
```

The accelerometer is therefore **accurate on average and over the long term** (gravity does not drift) but **wrong during motion**, whenever the robot accelerates, brakes, turns a corner or bumps over a floor joint, and noisy from motor vibration. A simple safeguard is to trust the tilt only when the magnitude is close to 1 g, and ignore it otherwise; but as the example shows, a push below that threshold still slips through, so a better answer is needed. It is the exact opposite of the gyroscope: the gyro is smooth and fast but drifts; the accelerometer is noisy and fooled by motion but does not drift.

Magnetometer: the compass

A magnetometer measures the magnetic field along each axis, in microtesla (μT) or gauss (1 gauss = 100 μT). The Earth's field is roughly 25 to 65 μT in total (approximate, depending on where you are), with the horizontal part, the part that points toward magnetic north, typically about 15 to 30 μT in mid-latitudes. With the sensor flat, the heading follows from the two horizontal axes:

```
Heading from the two horizontal axes:    heading = atan2( my, mx )      (this is an ang
       vary between chips, so check with a test. A compass bearing is measured clockwis
Declination:  magnetic north is not true north; the difference is a few degrees in many
Tilt:  if the robot is not level, the heading is wrong unless the tilt is measured (acc

Disturbances.  Hard iron: a fixed magnet or a steel part rotating with the robot adds a
       Both are usually calibrated by turning the robot through a full circle and recor

Worked example (illustrative): a horizontal field of 18 microtesla; the robot faces so
       A motor magnet adds a hard-iron offset of (+12, +5):   the sensor reads (30, 5),
       A full turn gives mx from -6 to 30 and my from -13 to 23:   offset = ( (30 + -6)
```

A compass is an absolute heading with no drift, which is its great value, but a robot is a hostile place for one. Motor wires carry currents that make fields of their own (and the field changes with the motor current, so it cannot be calibrated away entirely), steel frames, batteries and the reinforced concrete of a building all bend the field, and the reading can jump near a large metal object. Mount the sensor away from the motors and wires, and treat it as a slow, rough correction rather than a precise one.

Units, scale factors and sample rate

SENSOR	MEASURES	TYPICAL UNITS AND SCALE (APPROXIMATE)	TYPICAL SAMPLE RATE	STRONG AT	WEAK AT
Gyroscope	Rate of turn	Degrees per second; 131 LSB per degree/s on the ±250 range	100 to 1000 Hz or more	Fast, smooth, short-term heading and tilt change	Bias: drifts without limit
Accelerometer	Acceleration (gravity plus motion)	g; 16384 LSB per g on the ±2 g range	100 to 1000 Hz or more	Long-term tilt (roll and pitch); no drift	Wrong when accelerating; noisy; no heading
Magnetometer	Magnetic field	µT; a fraction of a µT per LSB	10 to 100 Hz	Absolute heading; no drift	Disturbed by motors, steel, wiring; needs calibration

Always read the sample rate and the scale you have set. A rate of 100 Hz means dt = 0.01 s, and the integration above depends on dt being right. Choosing a larger measurement range (±2000 degrees/s instead of ±250) stops a fast spin from saturating the sensor, but it makes each LSB coarser, so pick the smallest range that the robot's real motion will not exceed.

5. Sensor fusion in plain words

Sensor fusion means combining several imperfect measurements into one better estimate. The idea is not about averaging. It is about each sensor covering the other's weakness. The gyro is good at *changes* (the high-frequency part of the motion) but cannot hold an angle for long; the accelerometer is poor at changes (noisy, fooled by motion) but right about the angle on average (the low-frequency part). The obvious strategy: **trust the gyro in the short term and the accelerometer in the long term**.

The complementary filter

The simplest way to do that is one line. At each step, integrate the gyro as usual, then pull the result a small fraction of the way toward the angle the accelerometer gives. The gyro term supplies the fast changes; the accelerometer term slowly drags the angle back to the truth and cancels the drift.

```
angle(new) = alpha x ( angle(old) + gyro x dt ) + ( 1 - alpha ) x accel_angle
       gyro = the rate in degrees/s, bias already removed        accel_angle = the tilt

alpha is the trust in the gyro each step, and ( 1 - alpha ) is the trust in the acceler

Time constant:    tau = alpha x dt / ( 1 - alpha )     (approximate; the exact value is
       Meaning:   changes faster than about tau are taken from the gyro;   errors that
A bias b left in the gyro is held to a fixed angle error of  tau x b  instead of growin

Worked example: dt = 0.01 s (100 Hz), alpha = 0.98.      tau = 0.98 x 0.01 / 0.02 = 0.4
       The robot is standing still at a true tilt of 10 degrees and the accelerometer r
       it is turning at 0.5 degrees/s all the time).
       step k:          1        2        5        10       49       100      200
       angle (deg):   0.205    0.406    0.984    1.874    6.438    8.886    10.065
       The angle climbs toward 10 with a time constant near tau: after 49 steps (0.49 s
       Final value:   accel_angle + tau x b = 10 + 0.49 x 0.5 = 10.245 degrees.   The g

Compare with each sensor alone (true angle 10 degrees, still robot):   gyro only: error
Now add a 0.2 s bump during which the accelerometer is wrong by 15 degrees (the hard-st
       after 20 steps (0.2 s)  15 x ( 1 - 0.98^20 ) = 15 x 0.332 = 5.0 degrees of error
```

Here is the filter as hardware-neutral Python-style pseudocode.

```
def complementary(angle, gyro_rate, accel_angle, dt, alpha=0.98):
    # gyro_rate: degrees per second, bias already subtracted
    # accel_angle: degrees, from atan2 of the accelerometer axes
    return alpha * (angle + gyro_rate * dt) + (1 - alpha) * accel_angle

# each loop:  rate = (gyro_raw / 131.0) - gyro_bias
#             tilt = degrees(atan2(ax, az))
#             angle = complementary(angle, rate, tilt, dt)
```

CHOOSING ALPHA

A value close to 1 (a long tau) trusts the gyro more: very smooth, but slow to correct drift and slow to recover from a bad start. A smaller alpha (a short tau) corrects faster but lets the noise and the motion errors of the accelerometer through. Pick tau to be a few times longer than the disturbances you want to ignore (a bump lasts a fraction of a second) and a few times shorter than the time the gyro takes to drift by an unacceptable amount. Values of tau between about 0.3 s and 1 s are common for a small robot (a rule of thumb, not a law).

The Kalman filter, conceptually

The **Kalman filter** is the more powerful relative of the complementary filter. It does the same job, and no matrix algebra is needed to understand what it does. It keeps two things: a best estimate (here, the angle, and also the gyro's bias) and a measure of *how uncertain* that estimate is. Each step has two halves. In **predict**, it moves the estimate forward using a model (the gyro rate), and the uncertainty grows, because every prediction adds error. In **update**, it takes a measurement (the accelerometer angle), compares it with the prediction, and corrects the estimate by a fraction of the difference, where the fraction is the **Kalman gain**. The gain is worked out each step from the two uncertainties: a trustworthy measurement gets a large gain; a noisy one, or a confident prediction, a small one. That is the improvement over the complementary filter, whose alpha is one fixed number chosen by hand. The Kalman filter changes its trust as conditions change (for example, it can trust the accelerometer less when the readings are noisy), and it can estimate the gyro's bias as a by-product. Its cost is complexity: a model, noise figures to be chosen, and more computation. For a small robot that only needs tilt, the complementary filter is usually enough; the Kalman filter earns its place when you fuse many sensors with different rates and accuracies (wheel odometry, IMU and GPS, for example). It is the foundation of most professional navigation software.

6. Dead reckoning

Dead reckoning means estimating where you are by starting from a known point and adding up your own motion, with no outside reference. Robotics Fundamentals 4 built it from wheel encoders: from the ticks of each wheel, $ds = (d_R + d_L) / 2$ gives the distance moved and $d\theta = (d_R - d_L) / L$ gives the heading change, where L is the track width. That second line is the weak point. It divides a *small difference* of two large numbers by a small number, so a one-per-cent error in either wheel, or in L, or a slip on one side, becomes a heading error.

The improvement is natural once we have a gyroscope: **take the distance from the wheels and the heading from the gyro**. The distance is a well-behaved measurement (equal slip on both wheels affects it only moderately), but the heading is the part that suffers from slip and an inexact track width, and a gyro does not care about the floor. The pose update of Robotics Fundamentals 4 is unchanged except that $d\theta$ comes from the gyro (rate × dt), or from a fusion of the gyro with the wheels.

Why the position error grows so fast

A heading error is not just an error in the direction; every metre travelled after it carries the robot sideways. If the heading is off by e radians, a straight run of length s ends up s sin e to one side of where it should. If the heading error itself keeps growing as the robot moves, the sideways error grows *faster* than linearly.

```
Sideways error of a straight run of length s with a constant heading error e:    s sin(

Worked example: the heading error grows by 1 degree (k = 0.01745 rad) for every metre t
       After 10 m:   about k s^2 / 2 = 0.01745 x 100 / 2 = 0.87 m  (exactly (1 - cos(ks
Doubling the distance quadruples the position error (3.45 / 0.87 = 4.0 to within 1 %).

A fixed heading error of 6 degrees on a straight 5 m run:   5 sin 6 degrees = 0.52 m.
```

Worked example: heading from the wheels against the gyro

A robot drives three laps of a 1 m square (12 turns of 90 degrees, 1080 degrees in total), taking 180 s. The real track width is 150 mm, but the program uses 152 mm. The gyro was calibrated at power-up and is left with a residual bias of 0.05 degrees/s.

```
Wheels:  d(theta) = (dR - dL) / L(assumed).  The real turn is (dR - dL) / L(true), so t
       After the 1080 degrees of real turning, odometry reads 1080 x 0.98684 = 1065.8 d
       The wheel error is proportional to how much the robot has TURNED (and to the dis
Gyro:    the residual bias gives an error of  b t = 0.05 x 180 = 9.0 degrees after 180

So over this short run the gyro is better (9.0 degrees against 14.2), and the crossover
If the same robot spent 30 minutes (1800 s) doing the same 1080 degrees of turning, the
Neither is good enough for a long mission. A robot that fuses them can use the wheels t
is not changing, so any rate the gyro reports is its bias), and the gyro to cover the t
```

7. GPS and its limits

The **Global Positioning System** (and its relatives, together called GNSS) is the obvious absolute position sensor for a robot outdoors: a receiver listens to satellites and works out its latitude and longitude. For robots the limits matter more than the principle.

PROPERTY	TYPICAL VALUE (APPROXIMATE)	CONSEQUENCE FOR A ROBOT
Accuracy, open sky	About 2 to 5 m for a low-cost receiver	Fine for a lawn mower crossing a field; hopeless for a 1 m wide corridor or driving up to a door
Near buildings, trees	Often much worse, with jumps of 10 m or more	Signals bounce off walls (*multipath*) and arrive late, and the position moves although the robot has not
Indoors, underground	No fix at all	The satellite signals are very weak and do not pass through roofs and walls, so GPS cannot be the sole answer for an indoor robot
Update rate	1 Hz for many cheap modules; 5 to 10 Hz for better ones	At 1 m/s a robot moves 1 m between fixes; at 1 Hz and 10 m/s, 10 m. Far too slow for steering by itself
Heading	Only from motion: the direction between successive fixes	Useless when the robot is stopped or slow (the noise of metres swamps a movement of centimetres)
Start-up	From seconds to a minute or more to find a first fix	The robot cannot navigate at once
RTK GPS	1 to 2 cm, and 5 to 20 Hz, with a nearby base station or correction service	Used for farm and survey machines; costs more, needs a clear sky and a radio or network link to the corrections, and fails indoors like ordinary GPS

A degree of latitude is about 111 km on the ground, so 0.00001 degrees (the last digit most modules print) is about 1.11 m. The north-south and east-west steps are not equal: a degree of longitude shrinks with the cosine of the latitude, so it is about 111.32 × cos 55 = 63.9 km at 55 degrees north. GPS is best used for what it is good at: a slow, absolute, drift-free position, fused with the fast but drifting estimate of wheel odometry and the IMU (this is the standard job for a Kalman filter).

8. Absolute positioning: landmarks and beacons

Dead reckoning drifts, and an **absolute fix** resets the drift. Indoors, where there is no GPS, a robot gets its absolute fixes from places and objects whose position it already knows.

- **Landmarks** are features of the environment at known places: a wall (an ultrasonic or laser distance reading tells you the distance from it, which corrects the coordinate perpendicular to the wall), a corner, a door frame, a line crossing a marked junction, a coloured mark, or a printed marker that a camera can read together with its position and orientation. Meeting a

landmark whose location is on the robot's map lets the robot replace its drifted estimate with a fresh one, at least in that one direction.

- **Beacons** are active devices at known places that the robot measures its distance or direction to: infrared or radio transmitters, ultra-wideband (UWB) anchors, or acoustic pingers. Distances to three beacons fix a position (**trilateration**: the position is where three circles meet), and the beacons never drift. The costs are the infrastructure to install and the noise in the range readings.

```
 error
   ^
   |         /|            /|
   |       /  |          /  |
   |     /    |        /    |
   |   /      |      /      |
   | /        |    /        |
   +----------+---+---------+-----> distance travelled
   start     fix 1         fix 2    (start = known pose)

   Between fixes the error grows; each landmark or beacon fix pulls it back down to the
```

Worked example: a robot has driven 10 m along a corridor and its odometry has put it 0.3 m too far along. At the end an ultrasonic sensor measures 0.50 m to the end wall where the map says it should be 0.80 m. The wall reading says the robot is 0.30 m *further* than it thought, so the robot corrects its position along the corridor by 0.30 m. That fixes the along-corridor coordinate only; the sideways coordinate and the heading need a second landmark (a side wall) to correct. Landmarks used this way, together with maps, are the subject of Robotics Fundamentals 8 (localisation).

9. Reference frames and angle wrapping

Every measurement has a **frame of reference**. Wheel odometry works in the **robot frame** (x forward, y to the left, origin at the robot), a lidar reading is in the sensor's frame, and the pose of the robot is in the **world frame** (fixed to the room, or to the starting point). To use a measurement from one frame in another, rotate it by the robot's heading and shift it by its position. A point (x_r, y_r) in the robot frame is, in the world frame:

```
x(world) = x + x_r cos( theta ) - y_r sin( theta )              The robot is at (x, y)
y(world) = y + x_r sin( theta ) + y_r cos( theta )

Worked example: the robot is at (2, 1) facing 90 degrees (along the world y axis).   An
       x(world) = 2 + 1.0 x cos 90 - 0.5 x sin 90 = 2 + 0 - 0.5 = 1.5 m
       y(world) = 1 + 1.0 x sin 90 + 0.5 x cos 90 = 1 + 1 + 0 = 2.0 m              So t

The reverse (world point to robot frame): subtract the robot's position, then rotate by
```

Angle wrapping

An angle of 190 degrees and an angle of −170 degrees point the same way, yet to a computer they are numbers 360 apart. If a robot is facing −170 degrees and wants to face 170 degrees, the subtraction target minus heading gives 340 degrees, and a robot that believed it would turn the long way round, almost a full circle, when 20 degrees the other way would do. The cure is to **wrap** every angle difference into one standard range, usually −π to +π radians (−180 to +180 degrees).

```
wrap( a ) = atan2( sin a, cos a )          or          wrap( a ) = a - 2 pi x floor( ( a

Degrees:     190 -> -170      -190 -> 170      540 -> 180      720 -> 0      -350 -> 10

Heading error:   target 170, current -170.   Naive:  170 - (-170) = 340 degrees.   Wrap

Averaging angles:   the average of 350 and 10 degrees is clearly 0, but ( 350 + 10 ) / 2
       Average the sines and cosines instead:   atan2( sin 350 + sin 10, cos 350 + cos 1
The same trap catches filters:  a complementary filter or exponential filter that blends
       wrapped DIFFERENCE, as in  heading = heading + ( 1 - alpha ) x wrap( compass - he
```

Make a habit of wrapping *differences* of angles (errors) and the final angle after every update; it is the same wrap that appeared in the pose update of Robotics Fundamentals 4. Tilt angles in the range −90 to +90 usually do not need it, but headings always do.

10. Why feedback matters

Everything in this chapter is a way of measuring what the robot actually did, as opposed to what it was told to do. That difference is the heart of robot control. Compare two ways of driving 0.6

m.

```
Open loop (no measurement):    command --> [ motors and wheels ] --> motion        "Ru

Closed loop (feedback):       target ---(+)---> [ controller ] --> [ motors and wheels
                                      ^ - error
                                      +-------------- [ encoder: counts ] <-----------

Worked example: the battery has sagged, and the wheels turn at 0.255 m/s instead of the
       Open loop for 2.0 s:   0.255 x 2.0 = 0.51 m instead of 0.60 m.   An error of 0.0
       Closed loop:   the target of 0.60 m is 0.6 / 0.0001547 = 3878 counts (the encode
       The final error is about one count, 0.15 mm, plus whatever slip the wheels had;
```

A robot with no feedback is relying on every part being exactly as it was at the factory, and the world never is. With feedback the robot is told its own error, and can correct it. Doing that well is harder than it sounds: how hard should it correct, how quickly, and how can it avoid overcorrecting and swinging back and forth? That is the subject of Robotics Fundamentals 7.

FEEDBACK IS ONLY AS GOOD AS THE MEASUREMENT

A control loop that steers by a drifting, delayed or noisy number will faithfully produce drifting, delayed or noisy motion. The filters and calibration of Robotics Fundamentals 5, the interval choice for speed in section 2, and the fusion in section 5 are not optional extras. They decide how good the feedback loop can ever be.

11. Challenges

CHALLENGE 1: ENCODERS AND SPEED

A robot has a geared motor with a magnetic quadrature encoder of CPR = 16 on the motor shaft, a gearbox of 20:1 and wheels of diameter 80 mm, decoded at 4×. (a) Find the counts per motor revolution, the counts per wheel revolution, the angular resolution of the wheel in degrees per count, and the distance per count in millimetres. (b) The robot runs at 0.40 m/s. Find the counts per second, the counts in one 25 ms interval, the speed represented by one count, and the quantisation error as a percentage. Also find the motor speed in rpm. (c) What interval is needed to keep the quantisation error at or below 1 % at 0.40 m/s? What is the error with the 25 ms interval at 0.05 m/s, and how long would it take to collect 100 counts at that speed? (d) The decoder sees the states 00, 01, 11, 10, 00 in turn (A and B written AB), where forward is 00, 10, 11, 01. Which way is the wheel turning and what is

the net count? What does the sequence 00, 11 mean? (e) Find the resolution of a potentiometer with 270 degrees of travel on a 10-bit ADC, and of 12-bit and 14-bit absolute encoders, in degrees. (f) At a no-load motor speed of 9000 rpm, find the number of counts per second, and the share of the processor taken if each interrupt routine takes 10 microseconds.

View solution

CHALLENGE 2: GYRO, ACCELEROMETER, COMPASS AND THE COMPLEMENTARY FILTER

(a) A 16-bit gyro is set to the ±500 degree/s range. Find its sensitivity in LSB per degree/s. Averaged over a stationary period it reads +33; find the bias in degrees/s and the drift in degrees per minute. While the robot turns, the raw reading is 3277: find the rate before and after subtracting the bias, and the angle turned in 3 s at the corrected rate. (b) An accelerometer on the ±2 g range (16384 LSB per g) reads ax = 4096 and az = 15854 (ay = 0). Find ax and az in g, the pitch with atan2, and |a|. The robot then accelerates forward at 0.2 g with the same tilt: find the new ax, the pitch reported, the error, and |a|. Would a gate of 1.00 g ± 0.05 g reject it? (c) A complementary filter has alpha = 0.95 and dt = 0.02 s. Find the time constant. The robot is still at a true tilt of 10 degrees, the accelerometer reads 10 degrees, the filter starts at 0 and the gyro bias reads 0.5 degrees/s. Calculate the filter output for the first five steps and the final value, and estimate the time to cover 95 % of the way. (d) A compass in the horizontal plane reads (mx, my) = (28, 16) microtesla at a true reading of (20, 20). During a full turn the largest and smallest values were mx from −20 to 36 and my from −32 to 24. Find the heading error and the hard-iron offsets, and the corrected heading.

View solution

CHALLENGE 3: DEAD RECKONING, FRAMES, WRAPPING, GPS AND FEEDBACK

(a) A robot is at (3, 2) with heading −30 degrees. A landmark is seen at (2.0, 1.0) in the robot frame. Find its world position. (b) Wrap these angles into the range −180 to +180 degrees: 200, −270, 450. Wrap 4.0 radians into −π to +π. A robot facing −175 degrees must face +170 degrees: find the naive error, the wrapped error, and the way it should turn (counter-clockwise is positive). Find the correct mean of 355, 5 and 10 degrees and the wrong naive mean. (c) A robot turns 90 degrees twice. The wheels read each turn 3 degrees short of the truth because of slip, and the gyro has a residual bias of 0.05 degrees/s. Compare the heading errors from the wheels and from the gyro for a 60 s mission and for a 600 s mission with the same two turns. At what time do the two errors match? Find the sideways error of a final straight run of 4 m in the 6 degree and the 30 degree case. (d) A GPS fix moves from (55.00000, −3.00000) to (55.00003, −2.99995). Taking 111.32 km per degree of latitude and the longitude scale at 55 degrees north, find the northward and eastward displacement in metres, the distance, and the bearing (clockwise from north). A robot at 1.5 m/s with a 5 Hz GPS: how far does it move between fixes? Would a 3 m accuracy suit a 0.6 m wide corridor? (e) Open loop, a robot is

commanded to run at 0.30 m/s for 2 s, but the battery has sagged and it moves at 0.255 m/s. Find the distance and the error. With the encoder of the chapter (0.1547 mm per count), how many counts mark 0.6 m, and how long does the closed-loop run take?

View solution

QUICK REFERENCE

- **Relative sensors** (encoder, gyro, odometry) are smooth but drift; **absolute sensors** (compass, GPS, absolute encoder, landmark) do not drift but are noisier or unavailable. Fusion uses one to correct the other.
- **Quadrature:** A and B 90 degrees apart; four states per cycle (forward 00, 10, 11, 01); 4× decoding counts every edge. Counts per wheel revolution = 4 × CPR × gear; resolution = 360 / (CPR × 4 × gear); distance per count = πD / (CPR × 4 × gear). Use interrupts (Basic Electronics 10).
- **Speed:** $v = \Delta\text{count} \times d_{count} / \Delta t$; quantisation error about 1 / N counts per interval; a longer interval is finer but more delayed; time the gap at low speed.
- **Absolute:** n-bit encoder $360 / 2^n$; potentiometer $T / 2^n$ over travel T; absolute sensors keep their reading through a power cycle.
- **Gyro:** rate = raw / sensitivity; angle += rate × dt; a bias b gives error b t (0.5 degrees/s is 30 degrees a minute); measure the bias at rest and subtract it.
- **Accelerometer:** tilt = atan2 of the axes (pitch = $\text{atan2}(a_x, \sqrt{(a_y^2 + a_z^2)})$); check |a| = 1 g; wrong while accelerating; no heading.
- **Compass:** heading = $\text{atan2}(m_y, m_x)$; hard-iron offset = (max + min) / 2; disturbed by motors and steel; needs tilt compensation.
- **Complementary filter:** angle = α (angle + gyro × dt) + (1 − α) accel_angle; $\tau = \alpha\, dt / (1 - \alpha)$; a gyro bias b leaves an error τ b. Kalman filter: predict with a model, update with a measurement, by a gain set from the uncertainties.
- **Dead reckoning:** distance from the wheels, heading from the gyro; sideways error s sin e, and a growing heading error makes position error grow as s^2. Wheel heading error scales with turning, gyro error with time.
- **GPS:** metres of error, 1 to 10 Hz, no signal indoors, no heading at low speed; RTK gives centimetres with corrections. 0.00001 degrees of latitude is about 1.11 m.
- **Frames:** $x(\text{world}) = x + x_r \cos\theta - y_r \sin\theta$, $y(\text{world}) = y + x_r \sin\theta + y_r \cos\theta$. **Wrap** angle differences to −π..+π with atan2(sin a, cos a); average angles with sines and cosines.
- **Feedback:** open loop trusts the commands; closed loop measures the result and corrects the error.

Next, Robotics Fundamentals 7 uses everything measured here to close the loop. Control: making a robot do what you want. It covers open loop against closed loop, the simple on-off (bang-bang) controller, PID control with a worked example, how to tune it, and why a badly tuned robot oscillates, overshoots or becomes unstable, all driven by the encoder speed and gyro heading from this chapter.

Robotics Fundamentals

Course 1 · Chapter 7 · Control: Making It Do What You Want

Robotics Fundamentals 6 ended on an idea: a robot that only *commands* its motors is trusting that the world is exactly as it was when the commands were written, and a robot that *measures* what its motors actually did can correct itself. This chapter is about the correcting. Given a target (a speed, a heading, a position) and a measurement (the encoder speed or gyro heading of Robotics Fundamentals 6), a **controller** decides what to send to the motor next. We start from the simplest possible controller, on or off, and see why it chatters. We then build up the famous **PID** controller one term at a time (proportional, integral, derivative), simulating a motor speed loop step by step by hand so that every number can be checked. We look at why a position loop *rings* (the reason robots wobble), what a delay in the loop does to the largest gain you can use, how to tune a controller by hand and with the Ziegler-Nichols method (and what that method gets wrong), and finish with cascaded loops, feedforward, and two classic PID jobs: following a line and holding a heading.

The controller gains, motor constants, delays and noise figures in this chapter are **illustrative or typical values** chosen so that the arithmetic is easy to follow, not data for a particular motor. The formulas (the first-order step response, the steady-state error of proportional control, the discrete PID, the damping ratio, natural frequency and overshoot of a standard second-order system, and the Ziegler-Nichols table) are standard. Every simulation table in this chapter was recomputed step by step for the model stated next to it. The code is deliberately short, hardware-neutral Python-style pseudocode; real board code is the job of Programming for Robotics.

1. Open loop, closed loop, and the error

A **plant** is whatever the controller is trying to control: here, a motor with a wheel on it. The controller's output u goes into the plant (for a motor, a voltage, set by a PWM duty cycle) and the plant's response y comes out (a speed or an angle). There are two ways to arrange the loop.

```
OPEN LOOP:    the controller works out u from the target alone and never looks at the r

       target r  -------->  [ controller ]  --- u -------->  [ plant: motor + wheel ]
```

```
CLOSED LOOP:  the controller is told how far the result is from the target, and acts on

       target r ---->( + )-- e --> [ controller ] --- u ---> [ plant: motor + wheel ] -
                     ^ -
                     |
                     +----------------------- [ sensor: encoder ] <-------------------
                                               measurement

       error   e = r - y        (setpoint minus measurement:  e is positive when the ro
```

The sign convention matters and is the same everywhere in this chapter: **error = setpoint − measurement**. A positive error means the output is below the target, so the controller must push it up; a negative error means it is above, so the controller must pull it down. Get this sign

instead of settling.

The plant: a motor as a first-order system

Robotics Fundamentals 3 gave the steady-state motor model, $V = k\omega + IR$ and $\tau = kI$ (Basic Electronics 9). That model tells you where the speed ends up. To control the motor we also need to know how *quickly* it gets there, and the missing ingredient is the inertia J of everything the motor has to accelerate (rotor, gears and wheel, as seen from the motor shaft). Putting $J\,d\omega/dt = kI - T_{load}$ together with the electrical equation (and ignoring the winding inductance, whose time constant is far shorter than the mechanical one for the small motors in this course) gives a standard result: **the speed approaches its final value like a charging capacitor** (compare the RC circuit of Basic Electronics 4).

```
tau(m) d(omega)/dt + omega = K ( u - d )       first-order motor model          (u = ap

       K      = 1 / k              the steady gain: rad/s for each volt
       tau(m) = J R / k^2          the mechanical time constant: how fast the speed res
       d      = R T(load) / k      the load, written as the number of volts it "eats"

Step response from rest at constant u:      omega(t) = K ( u - d ) ( 1 - e^( -t / tau(m

Illustrative motor and wheel (a gearbox is included in k and J):   k = 0.05 V s/rad,
       K      = 1 / 0.05 = 20 rad/s per volt
       tau(m) = 2.5e-4 x 2.0 / 0.05^2 = 5.0e-4 / 0.0025 = 0.20 s
       d      = 2.0 x 0.025 / 0.05 = 1.0 V          (the wheel needs 1 V of drive befor
       With u = 6 V and the load:  final speed = 20 x (6 - 1) = 100 rad/s;   with no lo
```

```
Fraction of the final speed reached after n time constants:

       t / tau      1       2       3       4       5
       fraction   63.2 %  86.5 %  95.0 %  98.2 %  99.3 %          (1 - e^-n)
```

```
% of final speed
  100 |                              ********************
   90 |                    ***********
   80 |               *****
   70 |           ***
   60 |        ***
   50 |      **
   40 |     *
   30 |   **
   20 |  *
   10 | *
    0 |*
      +-------------------------------------------------
       0         1         2         3         4         5     time in units of tau(m)
```

Control engineers call this a **first-order lag**: a command that jumps instantly produces a speed that climbs smoothly, taking about $4\tau_m$ (0.8 s here) to settle to within 2 %. For the simulations below we step the model forward in small, equal time steps dt, using the simplest method (Euler):

```
omega(next) = omega + ( dt / tau(m) ) x ( K ( u - d ) - omega )          with dt = 0.02

Check from rest, u = 6 V, d = 1 V:   omega(1) = 0 + 0.1 x ( 20 x 5 - 0 ) = 10.0 rad/s
(The exact curve reaches 9.5, 18.1, 25.9 rad/s: Euler is a slightly fast approximation.
```

Open loop: why it is not enough

To hold 60 rad/s open loop, calculate the voltage from the model and send it: u = r / K + d = 60 / 20 + 1 = 4.0 V. That is a perfectly good controller on the day the model is right. The model is never right for long.

```
Open loop sends 4.0 V for a target of 60 rad/s.   What actually happens:

       Model exactly right:                          20 x ( 4.0 - 1.0 ) = 60 rad/s
       Battery sags so all voltages fall by 15 %:    20 x ( 0.85 x 4.0 - 1.0 ) = 48 rad/
       The robot meets a carpet, load doubles (d = 2 V):   20 x ( 4.0 - 2.0 ) = 40 rad/s

Closed loop (the controllers of the next sections) measures the speed, sees the error a
```

Open loop is not useless: for a stepper motor driving a light load, or a quick rough move, it is simple, cheap and needs no sensor. A good controller usually *combines* both: an open-loop guess for the bulk of the command (**feedforward**, section 11) and a closed loop to clean up what the guess gets wrong.

2. Bang-bang control and hysteresis

The simplest closed-loop controller is a switch: if the speed is below the target, turn the motor fully on; otherwise turn it off. This is **bang-bang** (or on-off) control, and it is what a household thermostat does.

```
u = U(max)   if e > 0   (measurement below the target)          u = 0   otherwise

Plain bang-bang cannot settle:  the instant the speed crosses the target the output fli
The output switches as fast as the loop can run ("chatter"): hard on the motor driver,
```

The cure is **hysteresis**: two thresholds instead of one. Switch on only when the measurement falls below r − h, and off only when it rises above r + h; between the thresholds, keep doing whatever you were doing. The output now switches at a rate set by the plant, not by the loop speed, at the price of a ripple of ±h in the result.

```
Motor of section 1 (K = 20, tau(m) = 0.2 s, d = 1 V), target 60 rad/s, supply 6 V:
       On  (u = 6 V):   speed heads for 20 x (6 - 1) = 100 rad/s.             Off (u = (
       (that -20 is only the limit of the formula; in the range of interest, near 60, th

Hysteresis h = 3 rad/s:  on below 57, off above 63.   Time to climb from 57 to 63 and to
       t(on)  = 0.2 x ln( (100 - 57) / (100 - 63) )  = 0.2 x ln( 43 / 37 )  = 0.0301 s
       t(off) = 0.2 x ln( (63 + 20) / (57 + 20) )    = 0.2 x ln( 83 / 77 )  = 0.0150 s
```

```
        Period 0.0451 s, a switching rate of 22.2 Hz, and the speed ripples between 57 a

   Hysteresis h = 6 rad/s:  t(on) = 0.2 x ln( 46 / 34 ) = 0.0605 s,   t(off) = 0.2 x ln( 8
   A wider band halves the switching rate and doubles the ripple: that is the whole trade.
```



Bang-bang control is a good fit when the plant is slow and the result need not be precise: heaters, a battery charger, a fan, a pump filling a tank, or an obstacle avoider that simply turns on or off (Robotics Fundamentals 8). It is a poor fit for steering a wheel or arm smoothly, because the ripple and the shock of full power to full reverse are exactly what we do not want. For those the controller should give an output that is *proportional* to how far off the robot is.

3. Proportional control

A **proportional (P) controller** sets the output in proportion to the error: u = K_p e. A big error gets a big push, a small error a gentle one, and at zero error the output is zero. This is what makes it smooth: the push shrinks as the robot approaches the target, so there is no hard switching.

Use the motor model with a proportional gain K_p = 0.2 V per rad/s of error, a target of 60 rad/s, the 1 V load, a supply limit of 6 V, and a 20 ms step (dt = 0.02 s). At each step: e = 60 − ω, u = min(6, 0.2 e), then the Euler update of section 1.

```
 step k   time (s)   omega (rad/s)   error e    Kp e     u after the 6 V limit
    0      0.00          0.00          60.00     12.00        6.00        (12 V wanted,
    1      0.02         10.00          50.00     10.00        6.00
    2      0.04         19.00          41.00      8.20        6.00
```

```
      3      0.06        27.10          32.90      6.58        6.00
      4      0.08        34.39          25.61      5.12        5.12        (no longer sa
      5      0.10        39.20          20.80      4.16        4.16
      6      0.12        41.60          18.40      3.68        3.68
      7      0.14        42.80          17.20      3.44        3.44
      8      0.16        43.40          16.60      3.32        3.32
      9      0.18        43.70          16.30      3.26        3.26        ... and it se

   Row check (k = 4):  omega(5) = 34.39 + 0.1 x ( 20 x ( 5.12 - 1 ) - 34.39 ) = 34.39 + 0.
```

Steady-state error

The speed does not reach 60. It settles where the controller's output is just enough to hold the speed against the load, and with a purely proportional controller that output is only produced *by an error*. At rest in the steady state $\omega = K(u - d)$ and $u = K_p e$, with $e = r - \omega$; solving gives the **steady-state error**:

```
   omega = K ( Kp e - d )  and  e = r - omega    =>    e ( 1 + K Kp ) = r + K d      =>

   Worked example (K = 20, d = 1 V, r = 60 rad/s, Kp = 0.2):      e(ss) = ( 60 + 20 ) / (
   Without the load (d = 0):                                      e(ss) = 60 / 5 = 12 rad/

   Effect of the gain (same motor, load d = 1 V):

          Kp (V per rad/s)        0.1       0.2       0.5
          K Kp (loop gain)        2         4         10
          e(ss) = 80 / (1+K Kp)   26.67     16.00     7.27     rad/s
          final speed             33.33     44.00     52.73    rad/s

   The error shrinks like 1 / (1 + K Kp): a large gain helps, but never removes it.  A big
   which a 6 V supply cannot give, and, as sections 7 and 8 show, a larger Kp makes the lo
   is unstable above about Kp = 0.95, see section 8.)
```

A steady-state error is not a bug in the code; it is a property of proportional control with a load. A motor holding a heavy robot on a slope, a heater holding a room warm in winter, an arm holding up its own weight: each needs a non-zero output to stay where it is, and a P controller can only supply that by leaving an error. That is the job of the next term.

4. Integral control

The **integral term** adds up the error over time: $I = K_i \int e\,dt$, in discrete form $I = I + K_i\, e\, dt$ at every step. While any error remains, the sum keeps growing and pushes the output harder, so the error cannot stay non-zero in the steady state: the integral stops changing only when $e = 0$. It has *learned* the load. A PI controller is $u = K_p\, e + I$.

Same motor, same target, now with $K_i = 3$ V per (rad/s · s) in addition to $K_p = 0.2$. The loop also uses the anti-windup rule explained below; the integral is not allowed to grow while the output is saturated.

k	t (s)	omega	e	Kp e	I (held or updated)	u wanted	u sent (6
0	0.00	0.00	60.00	12.00	0.000	15.600	6.00
1	0.02	10.00	50.00	10.00	0.000	13.000	6.00
2	0.04	19.00	41.00	8.20	0.000	10.660	6.00
3	0.06	27.10	32.90	6.58	0.000	8.554	6.00
4	0.08	34.39	25.61	5.12	0.000	6.659	6.00
5	0.10	40.95	19.05	3.81	1.143	4.953	4.95
6	0.12	44.76	15.24	3.05	2.057	5.105	5.10
7	0.14	48.50	11.50	2.30	2.748	5.049	5.05
8	0.16	51.74	8.26	1.65	3.243	4.894	4.89
9	0.18	54.36	5.64	1.13	3.582	4.710	4.71
10	0.20	56.34	3.66	0.73	3.801	4.533	4.53
11	0.22	57.77	2.23	0.45	3.935	4.380	4.38

Row check (k = 5): I = 0 + 3 x 19.05 x 0.02 = 1.143; u = 3.81 + 1.143 = 4.953 V.
The PI result: peak 60.16 rad/s (0.3 % overshoot), within 2 % of the target from about

```
rad/s   o = P only (Kp = 0.2)     x = PI (Kp = 0.2, Ki = 3, with anti-windup)       tar
  70 |
  65 |
  60 |-----------xxxxxxxxxxxxxxxxxxx
  55 |         xx
  50 |       xx
  45 |      xoooooooooooooooooooooo
  40 |     xo
  35 |    x
  30 |
  25 |   x
  20 |  x
```

```
 15 |
 10 | x
  5 |
  0 |x
    +------------------------------
```

Integral windup and anti-windup clamping

The integral term has a weakness. Suppose the output is already at its limit (6 V here), so the robot cannot go any faster, yet the error is large for a long time (a big step, a blocked wheel, a hill too steep). A naive integrator *keeps adding up* the error even though the extra output cannot be delivered. The integral winds up to a huge value, like a spring being wound far beyond what the mechanism can use. When the speed finally reaches the target, the error drops to zero, but the integral is still enormous, and the controller keeps driving at full power until the error has been negative for long enough to unwind it. The result is a big overshoot. Here is the same loop *without* protection:

```
Without anti-windup (the integral always adds):

 k    t (s)   omega     e        Kp e      I          u wanted     u sent
 0    0.00     0.00    60.00    12.00      3.600       15.600       6.00
 1    0.02    10.00    50.00    10.00      6.600       16.600       6.00
 2    0.04    19.00    41.00     8.20      9.060       17.260       6.00
 4    0.08    34.39    25.61     5.12     12.571       17.693       6.00        (the i
 8    0.16    56.95     3.05     0.61     15.155       15.764       6.00        integr
 9    0.18    61.26    -1.26    -0.25     15.079       14.828       6.00        the ta
11    0.22    68.62    -8.62    -1.72     14.254       12.530       6.00        and th

Result:  the speed peaks at 81.47 rad/s, an overshoot of 35.8 % of the target, against
```

The cure is **anti-windup**. The simplest and most common form is **clamping** (also called conditional integration): *do not let the integral grow when the output is saturated and the error would push it further into saturation.* That is exactly what the PI table above did in rows 0 to 4. Two other common forms: clamp the integral itself to a sensible range (for example, never more than the supply voltage), or **back-calculate** (feed the amount by which the output was clipped back into the integral so it unwinds). Any robot with output limits (which is every robot) needs one of these. Windup also appears whenever the plant is stuck: a robot pressing against a wall while the position controller keeps asking for more.

5. Derivative control

The **derivative term** responds to the *rate of change* of the error: K_d de/dt. In discrete form, K_d (e − $e_{previous}$) / dt. It reacts to where the error is *heading*: if the error is shrinking quickly (the robot is rushing toward the target) the derivative is negative and subtracts from the output, braking early, so the robot arrives gently instead of overshooting. It acts like damping: a shock absorber, or a hand that steadies a swinging pendulum. Section 7 quantifies this. It does nothing about a steady error (the derivative of a constant is zero), so it is a partner to P, not a replacement.

Two problems: noise and kick

Differentiating a signal amplifies everything fast in it, including noise. A measurement that jumps by one count of the encoder or one step of the ADC produces a large derivative. And if the setpoint is changed suddenly, the error steps with it, and the derivative of a step is, in principle, infinite.

```
Position loop of section 7 (Kd = 0.0914 V per rad/s), sample time dt = 0.01 s, output l

Noise:  the encoder reading changes by one count, 1 degree = 0.01745 rad, in a single s
       rate seen = 0.01745 / 0.01 = 1.745 rad/s          derivative output = 0.0914 x 1
       Halve dt and the same count step gives twice the rate and twice the spike:  diff

Kick:   the target is changed by 1 rad in one sample (the error steps from 0 to 1 rad):
       derivative on the ERROR = Kd x ( 1 - 0 ) / 0.01 = 0.0914 x 100 = 9.14 V     for
```

Two standard fixes. **Derivative on measurement**: since the setpoint r does not change between steps, de/dt = dr/dt − dy/dt reduces to −dy/dt except at the moment the setpoint moves. Compute the derivative term as $-K_d$ (y − $y_{previous}$) / dt and the kick disappears completely (the measurement cannot jump when the target changes). And **filter** the derivative with a first-order filter (Robotics Fundamentals 5) so noise is smoothed before it is multiplied by K_d; commonly the filter time constant is set at a small fraction of K_d/K_p. Better still, use a sensor that measures the rate directly: a gyro gives the turn rate, and an encoder with a good speed estimate gives velocity, so the “derivative” of a heading or position is *measured*, not differentiated (section 12).

6. The PID controller in discrete time

Adding the three terms gives the **PID controller**. In continuous form, and in the form the robot's program computes at every tick:

```
Continuous:   u(t) = Kp e(t)  +  Ki integral( e dt )  +  Kd de/dt

Discrete (every dt seconds):
       e         = r - y
       P         = Kp e
       I         = I + Ki e dt                        (the running sum, which is the con
       D         = -Kd ( y - y(previous) ) / dt       (derivative on measurement)
       u         = P + I + D                          then limit u to its allowed range,

Alternative form:   u = Kp ( e + (1/Ti) integral(e dt) + Td de/dt )    with   Ki = Kp /
       Ti is roughly "the time over which the integral repeats the proportional correct
       The two forms are the same controller; the tuning rules in section 9 use Ti and

What each term does (an increase of each gain):

       gain      rise time          overshoot      settling time      steady-state erro
       Kp        shorter            larger         little change      smaller (not zero
       Ki        a little shorter   larger         longer             removed
       Kd        little change      smaller        shorter            no effect
       (typical trends for a simple plant, not laws for every system)
```

Choosing dt, and limiting the output

- **The loop must be much faster than the plant.** A common rule is at least 10 updates per time constant of the closed loop you want. For the 0.2 s motor, 20 ms is the largest step worth using, and 10 ms is better; a small robot's speed loop typically runs between 50 Hz and 1 kHz (a rule of thumb, approximate). Euler stepping of the plant, as in this chapter, also needs dt well below τ_m.
- **But not faster than the measurement.** A speed measured from encoder counts has a quantisation error that grows as the interval shrinks (Robotics Fundamentals 6 section 2: 2.6 % at 20 ms, 0.5 % at 100 ms for the 0.3 m/s example). The derivative term makes it worse (previous section). There is no use running the loop faster than new information arrives.
- **Use the real dt.** If the loop time varies (jitter), compute dt from a timer every tick, as with the gyro of Robotics Fundamentals 6; a program that assumes 20 ms but runs at 21 ms makes K_i and K_d wrong by 5 %.

- **Always limit the output** to what the hardware can deliver (the supply voltage, or ±100 % duty cycle), and tie the integral to that limit (anti-windup). For a motor, the command is converted to a PWM duty cycle: duty = u / V_{supply}. Real motors also have a *deadband* (a minimum voltage below which they do not turn because of friction) that is sometimes handled by adding a small constant offset; the integral term also takes care of some of it.

The PID as pseudocode, and a plant to test it on

Hardware-neutral Python-style pseudocode of the controller of the previous section, with derivative on measurement and clamping anti-windup, and a three-line simulation of the first-order motor so that the controller can be tried on a laptop before it ever sees a motor:

```
class PID:
    def __init__(self, kp, ki, kd, out_min, out_max):
        self.kp, self.ki, self.kd = kp, ki, kd
        self.out_min, self.out_max = out_min, out_max
        self.integral = 0.0
        self.prev_meas = None

    def update(self, setpoint, measurement, dt):
        error = setpoint - measurement
        p = self.kp * error
        if self.prev_meas is None:
            d = 0.0                                          # no history on the first cal
        else:
            d = -self.kd * (measurement - self.prev_meas) / dt    # derivative on measu
        self.prev_meas = measurement

        new_integral = self.integral + self.ki * error * dt
        u = p + new_integral + d
        u_limited = min(self.out_max, max(self.out_min, u))

        # clamping: do not integrate if saturated AND the error pushes further into sat
        if u == u_limited or (u - u_limited) * error < 0:
            self.integral = new_integral
        return u_limited


def motor_step(omega, u, load_volts, dt, K=20.0, tau=0.2):
    return omega + (dt / tau) * (K * (u - load_volts) - omega)       # the Euler step o


pid = PID(0.2, 3.0, 0.0, -6.0, 6.0)
```

```
omega = 0.0
for k in range(150):                                    # 3 seconds at dt = 0.02
    u = pid.update(60.0, omega, 0.02)
    omega = motor_step(omega, u, 1.0, 0.02)
```

Running this reproduces the PI table of section 4 row by row, which is the best test of a PID class: before a robot ever uses it, check it against a plant whose answer you can calculate by hand.

7. Position control: second-order behaviour, overshoot and oscillation

So far the quantity controlled was speed. Controlling a *position* (the angle of an arm joint, the distance driven) is harder in an important way: the motor **integrates speed into position**. Even with no integral term in the controller, the plant itself contains one. The motor of section 1 becomes $\theta' = \omega$, $\tau_m\, \omega' + \omega = K\, u$ (no load, for clarity), which is a **second-order** system, and a second-order system can *ring*.

Close the loop with a P controller, $u = K_p\,(r - \theta)$. Substituting gives the standard form of a second-order system, a mass on a spring with some damping:

```
theta'' + (1 / tau(m)) theta' + ( K Kp / tau(m) ) theta = ( K Kp / tau(m) ) r

       Natural frequency:   wn   = sqrt( K Kp / tau(m) )                    (rad/s: I
       Damping ratio:       zeta = 1 / ( 2 sqrt( K Kp tau(m) ) )            (a pure r

In plain terms:
       zeta = 0 : never dies down, rings for ever          zeta = 0.2 to 0.5 : rings se
       zeta = 0.7 : about the best compromise, a small overshoot, then done        zeta
       zeta > 1 : overdamped, slow and creeping, no overshoot at all

For zeta below 1 (a step of size 1 in the target):
       Overshoot (fraction of the step):    OS = exp( - pi zeta / sqrt( 1 - zeta^2 ) )
       Period of the ringing:   T(d) = 2 pi / ( wn sqrt( 1 - zeta^2 ) )

Overshoot against damping:     zeta    0.2     0.3     0.4     0.5     0.6     0.7     (
                               OS %    52.7    37.2    25.4    16.3    9.5     4.6     1
```

Worked example with the motor of section 1 (K = 20 rad/s per volt, τ_m = 0.2 s), a 1 rad step, and no load:

```
 Controller       Kp (V/rad)  Kd (V per rad/s)    wn (rad/s)   zeta     overshoot    fi
 P only             1.0          0               10.00       0.250     44.4 %       0.3
 P only             0.5          0                7.07       0.354     30.5 %       0.4
 P only             0.125        0                3.54       0.707      4.3 %       1.2
 P only             0.0625       0                2.50       1.000      none         -
 PD                 1.0          0.0914          10.00       0.707      4.3 %       0.4

The first row:  wn = sqrt( 20 x 1.0 / 0.2 ) = sqrt( 100 ) = 10.0 rad/s,  zeta = 1 / ( 2
       first peak  pi / ( 10 x 0.968 ) = 0.32 s;  one full swing  2 pi / 9.68 = 0.65 s
The PD row:    2 zeta wn = ( 1 + K Kd ) / tau(m) = ( 1 + 20 x 0.0914 ) / 0.2 = 14.14,
(A step-by-step simulation of these loops with a small time step agrees with the formul
```

```
position (target = 1.0, dashes)     a = P, Kp = 1.0 (zeta 0.25)    b = P, Kp = 0.125 (z
1.5 |        a
1.4 |       a a
1.3 |      a   a
1.2 |           a
1.1 |     a      a         aaaaaa
1.0 |--------ccccccccccccccccccccccccccccccccccccccccc
0.9 |       c      a   bbb
0.8 |    a c        bbba
0.7 |     c       bb
0.6 |    c      bb
0.5 |   a      b
0.4 |   c    bb
0.3 |  a    b
0.2 |  c  bb
0.1 | c bb
0.0 |cbb
    +-------------------------------------------------
```

Why robots oscillate

Read the damping formula again: $\zeta = 1 / (2\sqrt{(K K_p \tau_m)})$. As the gain K_p rises, the damping *falls*. The loop gets a stronger spring and the same weak damping, so it rings. This is the single most important fact about tuning and it explains most wobbling robots:

- **Gain too high.** The controller pushes harder than the plant's own damping can absorb. Overshoot grows, then the swings take longer to die, and finally they do not die at all.
- **Not enough damping.** Friction alone is weak damping, which is why a position loop with only P is easy to make ring. The remedy is the derivative term (the PD row above): it supplies the damping that the plant lacks, so a high gain (fast response) can be kept without the ringing.
- **Delay.** A late measurement means the controller reacts to where the robot *was*, so its correction arrives out of step with the motion (next section).
- **Backlash and friction.** The loose fit in a gearbox (Robotics Fundamentals 3) is a dead zone in which the motor moves but the output does not, so the loop sees no response and pushes harder, then overshoots when the gears engage. An integral term plus static friction (stiction) causes *hunting*: the integral slowly builds up until the joint breaks free, overshoots, and the cycle begins again.
- **Integral too large** (section 4): it overshoots by design and, when large, produces a slow oscillation of its own.

8. Stability, delay and gain, in plain words

A control loop is **stable** if any disturbance dies away, and **unstable** if it grows. Think of steering a boat by watching its wake: if you are slow to react, you correct for a course error that is already gone, and your correction makes the next error larger. Every real loop has some **delay** (time between something happening and the controller acting on it): the sensor filter of Robotics Fundamentals 5, the interval over which speed is measured (Robotics Fundamentals 6), the time to compute, a communications link, the motor's own lag. A delay turns a correction that should oppose the error into one that arrives a half-swing late and *adds* to it. At a frequency where the delay shifts the correction by half a cycle, the loop is stable only if the correction is *smaller* than the error it was trying to fix; above a certain gain, called the **ultimate gain** K_u, each swing is bigger than the last.

Take the speed loop of section 1 and add a delay of three samples (60 ms: sensor filtering and command latency, a plausible value for a small robot, approximate). With P control and a 60 rad/s target the loop behaves as follows (each row is a simulation, 20 ms step, no load, no output limit):

```
 Kp (V per rad/s)   loop gain K Kp   final speed (rad/s)   first swing above final   ne
     0.05               1.0               30.00                  0.50
     0.10               2.0               40.00                 10.05
     0.15               3.0               45.00                 24.22
     0.20               4.0               48.00                 40.68
     0.2427             4.86              49.75                 55.38
     0.30               6.0               51.43                 74.40

(the final value is K Kp r / ( 1 + K Kp ), the same as without the delay;  the minus si
 Ku = 0.243 and the period of the sustained oscillation, Pu = 0.25 s, come from the sim

The same limit exists WITHOUT any extra delay:  the Euler update is itself a one-step d
       i.e. Kp < ( 2 tau(m) / dt - 1 ) / K = ( 20 - 1 ) / 20 = 0.95 for dt = 0.02 s  (K
Rule of thumb: delay sets the speed limit.  The more delay in a loop (a slow sensor, a
```

The practical lessons: use the fastest, least-delayed measurement you can (a gyro rate rather than a heading difference, an encoder rate with a short interval); do not filter harder than you must (Robotics Fundamentals 5), because a filter is a delay; run the loop at a steady, fast rate; and stay well below the ultimate gain. A loop at half of K_u is not “half as stable”; it is comfortably stable, with a moderate overshoot, and has margin for the motor, battery and floor to change.

9. Tuning

Tuning means choosing K_p, K_i and K_d so the response is fast without being unstable. There is a model-based way (calculate them from the plant, as in sections 3 and 7 where ζ and ω_n followed from K and τ_m) and several experimental ways, because the real plant is never as clean as the model.

Manual tuning, in order

1. **Make it safe first.** Wheels off the ground or a slow, limited speed, an output limit set low, and a stop switch to hand (Robotics Fundamentals 10 covers safe testing). A badly tuned loop can swing a heavy arm or run a robot off a table.
2. **Set $K_i = K_d = 0$. Raise K_p** in steps, applying a step in the target each time, until the response is fast and starts to overshoot or ring, then back off to somewhere between a half and two thirds of that value.

3. **Add K_d** (for position or heading loops, where it helps most) until the ringing is controlled and the overshoot is small. Stop when the output starts to look noisy or buzzes. If the loop is a plain speed loop, with a lag but no ringing, K_d is often not needed at all.
4. **Add K_i**, small at first, until the steady-state error is removed in a time that suits the task; too much gives overshoot and slow oscillation. Test with the load changed, and with a big step to see windup.
5. **Re-test everything** at the extremes: low battery, high load, full speed. Choose gains that are good enough everywhere, not best at one operating point.

The Ziegler-Nichols ultimate gain method

The classic experimental recipe (Ziegler and Nichols, 1942) needs no model. Using *only* a P controller, raise K_p until the loop produces a steady, sustained oscillation (neither growing nor dying). The gain at that point is the **ultimate gain** K_u; the period of the oscillation is the **ultimate period** P_u. The standard table converts the two numbers into controller settings (quoted from memory of the original table; check it before relying on it):

```
 Controller     Kp          Ti            Td          (Ki = Kp / Ti,   Kd = Kp Td)
 P              0.5  Ku     -             -
 PI             0.45 Ku     Pu / 1.2      -
 PID            0.6  Ku     Pu / 2        Pu / 8

Worked example on the delayed speed loop of section 8:  Ku = 0.24 (simulation: 0.2427),
       P:    Kp = 0.5 x 0.24 = 0.12
       PI:   Kp = 0.45 x 0.24 = 0.108,   Ti = 0.25 / 1.2 = 0.208 s,  Ki = 0.108 / 0.208
       PID:  Kp = 0.6 x 0.24 = 0.144,    Ti = 0.25 / 2 = 0.125 s,    Td = 0.25 / 8 = 0.(

Step response of the delayed motor loop (target 60 rad/s, no load, 20 ms step, 60 ms de

       Controller                            overshoot     settles within 2 %      note
       Ziegler-Nichols P   (Kp 0.12)          -             -                       rings
       Ziegler-Nichols PI  (0.108, 0.5184)   27.9 %       0.60 s
       Ziegler-Nichols PID (0.144, 1.152, 0.0045)   53.6 %     0.72 s             peak
       Hand-tuned PI (Kp 0.1, Ki 0.5)         23.8 %       0.50 s                  found
       Hand-tuned PI (Kp 0.1, Ki 1.0)         63.8 %       1.08 s                  too m
       "Some overshoot" variant (Kp Ku/3, Ti Pu/2, Td Pu/3):  21.5 %   1.12 s       Kp (
       "No overshoot" variant (Kp 0.2 Ku, Ti Pu/2, Td Pu/3):  14.0 %   0.96 s         K
```

What Ziegler-Nichols gets wrong

- **It is aggressive.** The table aims for a response in which each swing is about a quarter the size of the last. That is fast but rings, as the 54 % overshoot above shows; most robots want something gentler. Treat the result as a *first guess*, then reduce K_p and K_i until the behaviour is acceptable. The variants of the table that claim "some" or "no" overshoot are, again from memory, less aggressive but still overshoot here.
- **It requires driving the system to the edge of instability.** That is awkward with a delicate or heavy mechanism: a robot arm swinging at its ultimate gain can slam into its stops, and in a saturated actuator the "sustained oscillation" is clipped and ill-defined. Measure with low output limits and a stop switch to hand.
- **The measurements are fuzzy.** Finding the exact gain of a steady oscillation is hard on a real robot with noise, backlash and friction, and a 10 % error in K_u or P_u carries straight into the gains.
- **It assumes a plant with a delay and a lag.** A pure position loop (a plant with an integrator, section 7) has no ultimate gain at all without delay, only the delay of the sampling and sensor, and the table was built for process-control plants rather than motors.
- **It optimises the wrong thing for robots.** It was tuned for rejecting disturbances, not for following a changing setpoint smoothly, which is what most robot motion needs.
- **Other methods exist**: relay auto-tuning (the controller switches between two outputs and measures the resulting oscillation, which is gentler than driving the loop unstable), the "reaction curve" method (measure the step response of the plant and fit a delay and a time constant), and model-based design as in sections 3 and 7. Many motor-control libraries include an auto-tune routine of the first kind.

10. Cascaded loops

A position controller does not have to command the motor voltage directly. A common and effective structure is a **cascade**: an **inner loop** controls speed (fast, at 50 to 1000 Hz, using the encoder rate), and an **outer loop** controls position (slower), where the outer controller's output is not a voltage but a *speed setpoint* for the inner loop.

```
position  --->( + )--- e(pos) --> [ P: Kpos ] -- speed target -->( + )-- e(speed) --> [
target       ^ -                                                 ^ -
             |                                                   +------------- encoder speed
```

```
              +------------------------------------------------------ encoder position

   Advantages:   the inner loop makes the motor behave like a simple, well-damped speed so
                 the outer loop is then easy to tune (a plain P controller gives smooth ap
   Rule:         the outer loop must be clearly SLOWER than the inner loop (a factor of th
```

Using the inner PI controller of section 4 (K_p = 0.2, K_i = 3, 6 V limit, 1 V load, which settles within 2 % in about 0.26 s) and an outer proportional gain K_{pos} in rad/s of speed target per radian of error, a 2 rad step of position gives:

```
    Kpos (1/s)     speed target at the start    overshoot     settles within 2 % (0.04 rad
       3                   6 rad/s                  0.0 %            1.30 s
       5                  10 rad/s                  0.0 %            0.78 s
       8                  16 rad/s                  0.0 %            0.50 s
      15                  30 rad/s                  7.8 %            0.48 s

   With an ideal inner loop the outer loop is a first-order lag with a time constant of 1 /
   At 15 the outer loop asks for changes faster than the speed loop can deliver, and the e
```

11. Feedforward

A feedback controller reacts only after an error has appeared. If you already know roughly what output a target needs, **add it directly** and let the feedback correct only the remainder. This is **feedforward**: $u = u_{ff} + u_{PID}$. For the motor, the model gives the first guess $u_{ff} = r / K$ (plus an estimate of the load, if known); for a moving target it also includes terms for speed and acceleration. In an arm, the feedforward often carries the weight of the arm against gravity (Robotics Fundamentals 9).

```
   Motor of section 1, target 60 rad/s, true load d = 1 V (unknown to the controller), Kp =

          P only:                                final speed 44.0 rad/s, error 16
          Feedforward u(ff) = r / K = 60 / 20 = 3.0 V  plus the same P:
                  omega = ( K ( u(ff) - d ) + K Kp r ) / ( 1 + K Kp ) = ( 20 x 2 + 20 x 0.
          Feedforward plus PI:                   the integral only has to supply the missin
                                                 (in simulation: peak 60.26 rad/s, within 2
```

```
Feedforward cannot become unstable, because it contains no feedback; and it removes mos
which is exactly what the feedback is for.
```

12. PID examples: line following and heading hold

Line follower

The line sensor array of Robotics Fundamentals 5 produces a single number x: the position of the line in millimetres, positive to the right. The target is x = 0 (line under the centre), so the

positive (the convention of Robotics Fundamentals 4), and it is turned into wheel speeds with the differential-drive relation $v_R = v + uL/2$, $v_L = v - uL/2$. A line to the right gives x > 0, so e < 0, so u < 0: a clockwise turn, toward the line.

```
Robot: base speed v = 0.20 m/s,  track width L = 0.15 m,  loop time dt = 0.01 s,  line
       Kp = 0.08 rad/s per mm,   Kd = 0.002 rad/s per (mm/s)

       e = -4.69 mm            P = 0.08 x ( -4.69 ) = -0.375 rad/s
       de/dt = ( -4.69 - ( -4.19 ) ) / 0.01 = -50 mm/s          D = 0.002 x ( -50 ) = -(
       Proportional term alone:    u = -0.375 rad/s:      vR = 0.200 + ( -0.375 ) x 0.0
       With the derivative:        u = -0.475 rad/s:      vR = 0.164 m/s,     vL = 0.23(

The line follower is a position-style loop: the controller steers (changes heading), an
       too much Kp makes the robot snake from side to side; the derivative is the standa
       A sensor mounted far ahead of the wheels adds delay between steering and seeing
```

Heading hold

To drive straight (or turn to a bearing) the target is a heading and the measurement is the gyro-based heading of Robotics Fundamentals 6. The error *must be wrapped* (section 9 of that chapter), or a target of −170 degrees and a heading of +175 degrees would give a 345 degree error and a robot that spins the long way round. The controller's derivative can come straight from the gyro: the heading's rate of change is the gyro rate ω_{gyro}, so the derivative term is $-K_d \omega_{gyro}$, with no differentiation of a noisy angle at all.

```
e = wrap( target - heading )              u = Kp e - Kd x gyro_rate

Worked example:  target = -170 degrees,  heading = +175 degrees,  gyro says +10 degrees,
       e = wrap( -170 - 175 ) = wrap( -345 ) = +15 degrees = 0.2618 rad
       Kp = 3.0 per second,  Kd = 0.4:      P = 3.0 x 0.2618 = 0.785 rad/s           D

The gyro rate acts as damping: the faster the robot is already turning, the more the co
which has inertia and the lag of the wheel speed loops (inner loops, section 10), headi
```

13. Challenges

CHALLENGE 1: THE MOTOR MODEL, OPEN LOOP, BANG-BANG AND PROPORTIONAL CONTROL

A motor with its wheel has k = 0.04 V s/rad, R = 1.5 ohm and J = 2.4e-4 kg m^2. (a) Find K and τ_m, and the time to reach 63.2 %, 95 % and 98.2 % of the final speed. (b) The wheel load is equivalent to d = 1.2 V; find the load torque. Open loop, find the voltage that gives 75 rad/s. What speed results if the battery sags so that every voltage is 10 % lower, and what is the error as a percentage of the target? What if the load rises to d = 2.0 V instead? (c) A P controller with K_p = 0.16 V per rad/s and the 1.2 V load, target 75 rad/s: find the steady-state error and the final speed, then the same with no load. What gain K_p would reduce the load error to 3 rad/s, and why is it not practical on a 6 V supply? (d) Bang-bang control with a 6 V supply and a target of 75 rad/s, hysteresis ±4 rad/s: find the speed the motor would head for with the supply on and with it off, the time to climb and to fall across the band, the switching frequency, and the frequency with ±8 rad/s instead.

View solution

CHALLENGE 2: PI CONTROL, WINDUP, POSITION LOOP, DAMPING AND DERIVATIVE PROBLEMS

(a) Simulate the speed loop K = 20, τ_m = 0.2 s, dt = 0.02 s, load d = 0.5 V, output limit ±5 V, target 40 rad/s, K_p = 0.1, K_i = 2.0, starting from rest. Tabulate steps 0 to 5 (ω, e, K_p e, I, the wanted u, and the u sent) twice: once with the integral always adding, once with clamping anti-windup (do not integrate if saturated and the error pushes further). Compare the integral values at step 5. What will the integral equal when the loop has settled? (b) A position loop with the same K and τ_m, K_p = 0.8 V/rad. Find ζ, ω_n and the overshoot with K_d = 0. Then add K_d = 0.06 V per rad/s: find ζ, ω_n, the overshoot, the time of the first peak, the period of the ringing and the approximate 2 % settling time. (c) At dt = 0.01 s with K_d = 0.06, the target changes by 0.5 rad in one step and the encoder has a resolution of 0.0105 rad. Find the derivative output when taken on the error at the instant of the change, and the noise it takes from a

one-count change in the measurement; what is the first value when the derivative is taken on the measurement instead?

View solution

CHALLENGE 3: TUNING, STABILITY, FEEDFORWARD AND PID IN A ROBOT

(a) A speed loop with P control oscillates steadily at $K_u = 0.35$ with a period $P_u = 0.18$ s. Give the Ziegler-Nichols settings (K_p, T_i, T_d, and K_i and K_d) for the P, PI and PID controllers. (b) Find the overshoot for $\zeta = 0.5$, and the damping ratio that gives exactly 10 % overshoot. (c) A loop with $K = 20$ and $\tau_m = 0.2$ s is updated every $dt = 0.05$ s by the Euler rule with no extra delay. Find the largest K_p for which it is stable, and the value if dt is cut to 0.02 s. (d) A motor with $K = 25$ has a true load of $d = 1.2$ V and a P gain of 0.16; the target is 75 rad/s. Find the final speed and error for P alone, and with feedforward $u_{ff} = r / K$ (no load assumed). (e) A line follower: $v = 0.25$ m/s, $L = 0.16$ m, $dt = 0.01$ s, $x = -6.5$ mm now and -5.9 mm one step ago, $K_p = 0.06$ rad/s per mm, $K_d = 0.0015$ rad/s per (mm/s). Find e, de/dt, u and both wheel speeds, and say which way the robot turns. (f) Heading hold: target -170 degrees, heading $+175$ degrees, gyro $+10$ degrees/s, $K_p = 3$, $K_d = 0.4$. Find the wrapped error and the turn rate command u, and say which way the robot turns.

View solution

QUICK REFERENCE

- **Error** = setpoint − measurement. **Open loop** trusts the model; **closed loop** measures and corrects. Open loop at 4.0 V gives 60 rad/s only while the battery and load are as modelled.
- **Motor model:** $\tau_m\, \omega' + \omega = K\,(u - d)$; $K = 1/k$; $\tau_m = J\,R / k^2$; $d = R\,T / k$; step response $K\,(u - d)(1 - e^{-t/\tau})$: 63.2 %, 86.5 %, 95.0 %, 98.2 % after 1, 2, 3, 4 τ. Euler: $\omega \mathrel{+}= (dt/\tau)(K\,(u - d) - \omega)$.
- **Bang-bang:** full on or off; chatters. Hysteresis ±h gives a ripple of ±h at a rate set by the plant: $t = \tau \ln((\text{final} - \text{start})/(\text{final} - \text{end}))$.
- **P:** $u = K_p\, e$; leaves $e_{ss} = (r + K\,d)/(1 + K\,K_p)$ with a load; larger K_p shrinks it but never removes it, and makes the loop ring.
- **I:** $I \mathrel{+}= K_i\, e\, dt$; removes the steady-state error (it learns the load: $u_{ss} = r/K + d$). **Windup:** saturation plus integrating gives a big overshoot (35.8 % against 0.3 % in the example); fix by clamping the integral when saturated.
- **D:** $-K_d\,(y - y_{prev})/dt$ (on *measurement*, to avoid kick); adds damping, amplifies noise, so filter it or use a gyro or speed measurement directly.

- **Discrete PID:** $u = K_p\, e + I + D$, limited to the output range; $K_i = K_p/T_i$, $K_d = K_p\, T_d$; choose dt at least 10 per closed-loop time constant, but no faster than the measurement; use the real dt.
- **Second order (position loop):** $\omega_n = \sqrt{(K\, K_p / \tau_m)}$; $\zeta = 1 / (2\sqrt{(K\, K_p\, \tau_m)})$; overshoot $= \exp(-\pi\zeta/\sqrt{(1-\zeta^2)})$; $t_{peak} = \pi / (\omega_n \sqrt{(1-\zeta^2)})$; settling about $4 / (\zeta\, \omega_n)$; $\zeta = 0.7$ gives 4.6 %. With PD: $2\,\zeta\,\omega_n = (1 + K\, K_d)/\tau_m$. Higher K_p means lower ζ: robots oscillate when gain is high and damping low.
- **Stability:** delay (sensor, filter, computation, loop rate) sets an ultimate gain K_u above which each swing grows; stay well below it. Without any extra delay, Euler at dt needs $K_p < (2\tau_m/dt - 1)/K$.
- **Tuning:** K_p up to just under ringing, then K_d for damping, then a little K_i, then test at the extremes. **Ziegler-Nichols:** P 0.5 K_u; PI 0.45 K_u, $P_u/1.2$; PID 0.6 K_u, $P_u/2$, $P_u/8$; fast but aggressive (about 54 % overshoot here), needs a near-unstable test, a first guess only.
- **Cascade:** fast inner speed loop, slower outer position loop whose output is a speed target; the outer loop must be clearly slower. **Feedforward:** $u = u_{ff}$ + PID, with $u_{ff} = r / K$; it does most of the work and the feedback trims it.
- **Line follower:** $e = -x$, output a turn rate u, $v_R = v + uL/2$, $v_L = v - uL/2$. **Heading hold:** e = wrap(target − heading), $u = K_p\, e - K_d \times$ gyro rate.

Next, Robotics Fundamentals 8 steps up from *how* a robot moves to *what it decides to do*. Behaviour and decision-making: reactive behaviours, state machines, line following, obstacle avoidance and wall following built on the controllers of this chapter, and a first look at simple planning, mapping and localisation (using the landmarks and the frames of Robotics Fundamentals 6) so that a robot can choose where to go as well as get there.

Robotics Fundamentals

Course 1 · Chapter 8 · Behaviour and Decision-Making

Robotics Fundamentals 7 ended with controllers that make a robot do exactly what it is told: hold a speed, hold a heading, follow a line. This chapter is about *what it is told*. Something has to decide that now is the time to follow the line, that a wall in front overrides the line, that the battery is low and the dock matters more than the floor, and that the robot has been stuck for ten seconds and should try something else. That something is the robot's **behaviour layer**, the "think" of the sense-think-act loop of Robotics Fundamentals 1. We start with the cheapest kind of decision, a **reactive behaviour** that maps sensors straight to motors, and with **arbitration**, the rule for choosing between behaviours that disagree. Then we build the main tool for organising behaviours over time, the **finite state machine**, including how to stop a noisy sensor from making it flicker. We put numbers on the classic behaviours (line following, obstacle avoidance, wall following, homing to a dock), move on to **simple planning** on a grid map (occupancy grids, breadth-first search, and A* worked out by hand on a real grid), take a first look at **localisation** (belief, particle filters, and a worked trilateration from three beacons), and finish with behaviour trees and the unglamorous but vital subject of testing and failure handling.

The thresholds, gains, speeds, probabilities and sensor figures in this chapter are **illustrative or typical values** chosen so that the arithmetic is easy to follow, not data for a particular robot or sensor. The formulas (differential-drive wheel speeds, the repulsive potential, the log-odds update, breadth-first search, A* with the Manhattan heuristic, the particle weight and the trilateration equations) are standard. Every number in the worked examples was recomputed, and the search examples were actually run on the grids shown. The code is deliberately short, hardware-neutral Python-style pseudocode, used only for an arbitration rule, a state machine and A*; real board code is the job of Programming for Robotics.

1. Reactive behaviours

A **reactive behaviour** computes its output from the sensor readings of this instant, with no map of the world and no memory of the past: *sense, then act*. The line follower of Robotics Fundamentals 7 section 12 is one (line position in, turn rate out); so is a bumper that makes a robot back up. There is no model to get out of date, so reactive behaviours are fast (one sensor read and a little arithmetic), cheap, and robust against a changing world. Their limits are equally

clear: with no memory they cannot count, remember where they have been, or pursue a goal that is out of sensor range, and two simple rules can chase each other in circles.

Braitenberg vehicles

The neuroscientist Valentino Braitenberg described a series of imaginary vehicles in which a few wires from sensors to wheels produce behaviour that looks purposeful. They are the cleanest picture of a reactive robot: the "program" is only the wiring. Take two light sensors (reading 0 to 1, larger meaning brighter) and two wheel motors, each wheel speed set by v = b + g × sensor, where b is a base speed and g a gain.

```
UNCROSSED  (left sensor drives the left wheel, right sensor the right wheel):    the br
CROSSED    (left sensor drives the right wheel, right sensor the left wheel):    the br

Light to the left:  sL = 0.8,  sR = 0.3.     b = 0.05 m/s,   g = 0.4 m/s per unit of re

       UNCROSSED:   vL = 0.05 + 0.4 x 0.8 = 0.37 m/s     vR = 0.05 + 0.4 x 0.3 = 0.17 m
       CROSSED:     vL = 0.17 m/s                        vR = 0.37 m/s

A NEGATIVE gain (brighter means slower) makes the vehicle slow down as it nears the sou
charges at it. Four wires, two sensors, no processor, no memory, and the vehicle alread
```

Real robots use the same idea with ordinary distance sensors: two forward-angled ultrasonic or infrared sensors steer a robot *away* from whichever side has the nearer obstacle: the nearer the obstacle on the left, the faster the left wheel is driven, so the robot swings right (the "uncrossed" wiring, with a reading of nearness in place of brightness). The weakness is the same too: a reactive vehicle in a dead-end corridor sees the same left and right readings, turns neither way, and drives into the end. Something with a little memory or a little goal has to take over, which is what the rest of the chapter provides.

Arbitration: when behaviours disagree

A real robot has several reactive behaviours at once (follow the line, avoid obstacles, stop if the bumper is hit), and at any moment they may ask for different things. Rodney Brooks's **subsumption architecture** (1980s) organises them in layers of priority, where a higher layer can *suppress* (override) the output of those below it. The simplest practical form is **priority arbitration**: list the behaviours from the most to the least important, ask each in turn, and obey

the first one that has something to say. Safety behaviours sit at the top; the pleasant, goal-seeking behaviours sit below; a default behaviour at the bottom always has an answer.

```
priority   behaviour           active when                command it issues  (v in m/s
--------   ------------------  -------------------------  ----------------------------
   1       STOP     (bumper)   the bumper is pressed      (0, 0)
   2       AVOID    (front)    front distance < 0.30 m     v = 0 and turn toward the c
   3       LINE     (array)    a line is seen             v = 0.18,  w = Kp x e = 0.08
   4       WANDER   (nothing)  always                     (0.15, 0)

Asked in order from 1 to 4: the FIRST behaviour that has a command wins, and every beha
```

```
def stop_b(s):
    return (0.0, 0.0) if s["bump"] else None

def avoid_b(s):
    if s["front"] < 0.30:
        return (0.0, 1.0 if s["left"] > s["right"] else -1.0)
    return None

def line_b(s):
    if s["line_x"] is None:
        return None                                  # no line seen: nothing to say
    return (0.18, 0.08 * (-s["line_x"]))             # e = 0 - x, in millimetres

def wander_b(s):
    return (0.15, 0.0)

BEHAVIOURS = [("STOP", stop_b), ("AVOID", avoid_b), ("LINE", line_b), ("WANDER", wander_

def arbitrate(s):
    for name, behaviour in BEHAVIOURS:
        command = behaviour(s)
        if command is not None:
            return name, command
    return "STOP", (0.0, 0.0)                        # if nothing answers, stand still
```

```
Four sensor snapshots, run through the rule above:

bump    front   left   right   line x (mm)    winner    command (v, w)       why
```

```
True    0.10    0.9    0.4      +3          STOP      (0.00, 0.00)        the bumper
False   0.22    0.9    0.4      +5          AVOID     (0.00, +1.00)       0.22 < 0.3
False   0.80    0.9    0.4      -8          LINE      (0.18, +0.64)       e = +8 mm,
False   0.80    0.9    0.4     none         WANDER    (0.15, 0.00)        nothing el
```

The alternative to winner-takes-all is to **blend** the commands (a weighted average, or the vector sum used by potential fields in section 3). Blending gives smooth, graded behaviour, but averages can be worse than either input: if AVOID says turn left at +1.0 rad/s and a goal behaviour says turn right at −1.0 rad/s, the average is 0.0 and the robot drives straight into the obstacle between them. Use priorities for anything safety-related, and blending only among behaviours that are all fine with each other.

2. Finite state machines

Reactive rules have no memory. Many tasks do: *I am cleaning, I have been told to go home, I am charging*. A **finite state machine** (FSM) adds exactly the memory needed: the robot is always in exactly one of a small, named set of **states**, and each state says what the robot is doing (which behaviours are active, which targets are set). **Events** (a bump, a low battery, an arrival, a timer running out) cause **transitions** from one state to another. Everything the machine has to remember is the name of its current state, which makes it easy to test: for every state and every event, you can write down what happens, and a missing entry is a visible gap rather than a surprise.

A worked example: a vacuum robot

```
    DOCKED --start--> CLEAN --battery_low--> RETURN --docked--> CHARGE --full--> DOCKED
                       |  ^
                 bump  |  | avoid_done
                       v  |
                     AVOID --battery_low--> (RETURN)
    A timeout in AVOID, RETURN or CHARGE moves the machine to FAULT;   FAULT --reset--> I
```

```
State       What the robot does in this state                    Events it responds t
---------   ---------------------------------------------------  --------------------
DOCKED      wheels stopped, waiting on the charger               start
```

```
CLEAN       cover the floor (zigzag), brushes on                  bump
                                                                  battery_low
AVOID       back up 0.1 m and turn 90 degrees                     avoid_done
                                                                  battery_low
                                                                  timeout (5 s)
RETURN      drive to the dock (homing, section 3); a bump is      docked
            handled INSIDE this state and does not leave it       timeout (120 s)
CHARGE      wheels stopped, charging                              full
                                                                  timeout (8 hours)
FAULT       wheels stopped, lights flashing, error logged         reset

Any (state, event) pair that is not in the table is ignored (for example, bump while in
```

```
TABLE = {
    ("DOCKED", "start"):       "CLEAN",
    ("CLEAN",  "bump"):        "AVOID",
    ("CLEAN",  "battery_low"): "RETURN",
    ("AVOID",  "avoid_done"):  "CLEAN",
    ("AVOID",  "battery_low"): "RETURN",
    ("AVOID",  "timeout"):     "FAULT",
    ("RETURN", "docked"):      "CHARGE",
    ("RETURN", "timeout"):     "FAULT",
    ("CHARGE", "full"):        "DOCKED",
    ("CHARGE", "timeout"):     "FAULT",
    ("FAULT",  "reset"):       "DOCKED",
}
TIMEOUT = {"AVOID": 5.0, "RETURN": 120.0, "CHARGE": 28800.0}      # seconds allowed in

class Machine:
    def __init__(self):
        self.state, self.entered = "DOCKED", 0.0

    def handle(self, event, now):
        nxt = TABLE.get((self.state, event))                      # no entry: the event
        if nxt is not None:
            print(f"{now:6.1f} s  {self.state} --{event}--> {nxt}")      # log every tr
            self.state, self.entered = nxt, now

    def tick(self, now):                                          # call once per loop
        limit = TIMEOUT.get(self.state)
        if limit is not None and now - self.entered > limit:
            self.handle("timeout", now)
```

```
# A short run:  m = Machine();  m.handle("start", 0);  m.handle("bump", 10);  m.tick(16
#      0.0 s  DOCKED --start--> CLEAN         10.0 s  CLEAN --bump--> AVOID
#     16.0 s  AVOID --timeout--> FAULT        20.0 s  FAULT --reset--> DOCKED       (6 
```

Debouncing a decision: timers and hysteresis

Events come from noisy sensors, and a machine that reacts to every flicker flips between states. Robotics Fundamentals 7 section 2 met this with bang-bang control; the cure is the same, with one more tool. Three standard defences, used together:

- **Require the condition to hold for a time** (a *debounce* timer or a count of consecutive samples) before generating the event.
- **Use two thresholds** (*hysteresis*): enter a condition at one level, leave it at a different, safer level, so a reading sitting right on the line cannot toggle the machine.
- **Add a minimum time in a state** (a *dwell* time) so that a transition cannot immediately be reversed.

```
Battery charge (%), read 10 times a second, noisy by about +-2 around a true value of 1
Rule A (naive):      battery_low when the reading is below 20.
Rule B (debounced):  battery_low only when 5 readings IN A ROW are below 20 (that is 0.
Rule C:              Rule B to enter, and the condition is cleared only when the readin

sample          0    1    2    3    4    5    6    7    8    9   10   11   12   13
time (s)      0.0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.0  1.1  1.2  1.3
reading (%)    21   19   21   20   19   21   19   18   19   18   19   18   18   17
count below 20  0    1    0    0    1    0    1    2    3    4    5    6    7    8

Rule A: the condition turns on at sample 1, off at 2, on at 4, off at 5, on at 6:  5 ch
Rule B: first reaches a count of 5 at sample 10, t = 1.0 s:  ONE event.        Rule C: 

The price of debouncing is delay.  Sizing it:   battery:  0.5 s at a speed of 0.2 m/s i
       a cliff (floor-edge) sensor:  the same 0.5 s would carry the robot 0.1 m, which 
So the debounce time is chosen per sensor, from what the consequences of a false trigge
```

Avoiding state explosion

A state machine grows badly when it must track several *independent* things in one set of states. Suppose the robot has 4 drive modes, 3 battery conditions (ok, low, critical) and 2 payload conditions (loaded, empty). One machine that has to name every combination needs 4 × 3 × 2 =

24 states, and each new feature *multiplies* the count. Three small machines running side by side, one for each independent aspect, need only 4 + 3 + 2 = **9 states**, and each is simple to read. A second tool is the **hierarchical state machine**: a state such as CLEAN contains its own sub-states (zigzag, edge cleaning, spiral), and a rule written once on the outer state ("battery_low leaves CLEAN") covers all of them, instead of being repeated on each sub-state. The advice: one machine per independent concern, group states under a parent when they share transitions, and if the table starts to look like a plate of spaghetti, the design (not the drawing) needs another look.

3. Classic behaviours, with numbers

Line following

The controller is the PID of Robotics Fundamentals 7 section 12, fed with the weighted line position of Robotics Fundamentals 5 section 4; this section adds the *decisions around it*. With e = −x (x in mm, positive to the right) and u = K_p e a turn rate, wheel speeds follow from v_R = v + uL/2 and v_L = v − uL/2. Slowing on bends keeps the sensor array on the line: v = v_{max} / (1 + c |e|).

```
Robot: vmax = 0.30 m/s, c = 0.05 per mm, Kp = 0.08 rad/s per mm, track width L = 0.15 m

       e = -x = +12 mm                  u = 0.08 x 12 = +0.96 rad/s   (counter-clockwis
       v = 0.30 / ( 1 + 0.05 x 12 ) = 0.30 / 1.6 = 0.1875 m/s
       vR = 0.1875 + 0.96 x 0.075 = 0.2595 m/s        vL = 0.1875 - 0.96 x 0.075 = 0.11

Decisions around the controller:
       FOLLOW  -- line lost for 0.3 s (3 readings of no line, debounced) -->  SEARCH:
       SEARCH  -- line found --> FOLLOW.     SEARCH for more than 3 s  ( 3 x 1.0 rad/s =
       Last-seen side matters: a sharp corner makes the line leave the array on the sid
       A junction (all sensors dark for a stretch) is a different EVENT, not a lost lin
```

Obstacle avoidance

The simplest avoider is a three-state machine: DRIVE, TURN, and a recovery state. Use a **threshold with hysteresis**: start turning when the front distance falls below 0.30 m, and only go back to driving when it has risen above 0.45 m (a 0.15 m band), so the robot does not twitch at the edge of the threshold. Choose the turn direction toward the side with more room (the left

and right readings, as in section 1), and keep turning in that direction until the front is clear, rather than re-deciding every loop.

The threshold itself comes from a stopping-distance budget (section 7): at 0.5 m/s the robot needs about 0.10 m to stop, so 0.30 m leaves a margin of about three. The sensor must also see that far without a blind spot (Robotics Fundamentals 5: an ultrasonic sensor has a minimum range below which it reads nonsense).

Stuck and escape recovery. A reactive avoider can trap itself: a corner or the legs of a chair make it avoid left, then right, then left. Detect the loop and break it: if there are **3 avoid events in 10 seconds** (a count with a timer, like the debounce above), enter ESCAPE: reverse 0.30 m (at 0.10 m/s, 3 s), turn through a larger, randomised angle (90 to 180 degrees), and then resume. If ESCAPE has happened 3 times without progress (position has not changed by more than 0.5 m, which needs odometry from Robotics Fundamentals 6), the robot has no more tricks: it stops and reports being stuck.

```
DRIVE --front < 0.30 m--> TURN --front > 0.45 m--> DRIVE           DRIVE or TURN --3 a
```

Potential fields: a blended way to avoid

A **potential field** replaces if-then rules with forces. The goal pulls the robot with an attractive force; every obstacle within a range d_0 pushes it away with a repulsive force that grows rapidly as the obstacle gets close. The robot steers along the sum. The usual repulsive force comes from a potential $U = \tfrac{1}{2}\, k\, (1/d - 1/d_0)^2$ for $d < d_0$ (zero beyond), whose slope gives a force, pointing away from the obstacle, of magnitude:

```
F(rep) = k ( 1/d - 1/d0 ) / d^2     for d < d0, else 0          F(att) = k(a) x ( dista

Worked example (typical, illustrative):  k = 0.01,  d0 = 0.5 m,  k(a) = 0.2.   Robot at
       goal 2.0 m straight ahead:                   F(att) = 0.2 x 2.0 = 0.40 along +x
       obstacle A, 0.25 m ahead (at x = 0.25):      F = 0.01 x ( 4 - 2 ) / 0.0625 = 0.0
       obstacle B, 0.40 m to the left (at y = +0.40):  F = 0.01 x ( 2.5 - 2 ) / 0.16 =

       Sum:  Fx = 0.40 - 0.32 = 0.08      Fy = -0.03125       magnitude 0.086,  directi
       With a heading controller of gain 1.5 per second:   w = 1.5 x ( -21.3 degrees =

Weakness: LOCAL MINIMA.  A goal directly behind a wide wall: the pull forward and the pu
```

Wall following

Following a wall (to explore a room, or to find a door) is a position-style loop like the line follower: the error is the distance to the wall minus the wanted distance, and the output is a turn rate. With the wall on the *right* and a sensor looking at it, if the robot is too far ($e = d - d_{ref} > 0$) it must turn right (clockwise, negative), so $u = -K_p\, e$. A single sensor has a flaw: it cannot tell a robot that is close and parallel from one that is far and angled toward the wall (both give the same reading, but the second is about to hit it). With **two sensors** a distance s apart along the robot's side, the difference gives the angle: $\theta = \text{atan}((d_F - d_R)/s)$, and the perpendicular distance is $d_{perp} = ((d_F + d_R)/2)\cos\theta$. A positive θ means the front is farther from the wall than the rear, so the robot is heading away from it.

```
Wall on the right, wanted distance 0.30 m, speed v = 0.20 m/s, track width L = 0.15 m.

One sensor:   d = 0.38 m,   e = +0.08 m,   Kp = 2.0 rad/s per m:      u = -2.0 x 0.08 =

Two sensors 0.10 m apart:   dF = 0.40 m (front),  dR = 0.34 m (rear):
       theta  = atan( ( 0.40 - 0.34 ) / 0.10 ) = atan( 0.6 ) = 30.96 degrees = 0.5404 r
       d(perp) = ( ( 0.40 + 0.34 ) / 2 ) x cos( 30.96 degrees ) = 0.37 x 0.857 = 0.3173
       u = -( Kp x error + Ktheta x theta ) = -( 2.0 x 0.0173 + 1.5 x 0.5404 ) = -( 0.0
       vR = 0.20 - 0.845 x 0.075 = 0.137 m/s,   vL = 0.20 + 0.845 x 0.075 = 0.263 m/s

Corners:   an INSIDE corner (wall ahead): the front sensor sees it, so give the obstacl
           an OUTSIDE corner (the wall ends): the reading jumps up suddenly; cap the co
```

Homing and docking

To return to a charger, a robot needs something to find: an infrared beacon on the dock that gives the **bearing** β (angle to the dock, counter-clockwise positive) and a rough **range** r. The rule is the line follower once more: turn in proportion to the bearing, $w = K_b\, \beta$, and slow down as the range shrinks, $v = \min(v_{max}, k_v\, r)$, so the last centimetres are covered gently enough for the contacts to meet.

```
Kb = 1.5 per second,  vmax = 0.25 m/s,  kv = 0.5 per second.   The beacon is 20 degrees

       w = 1.5 x ( 20 degrees = 0.349 rad ) = 0.524 rad/s  (counter-clockwise: turn lef
       At range 0.30 m:  v = min( 0.25, 0.15 ) = 0.15 m/s.     At range 0.10 m:  v = 0.(
```

```
The last metre is made forgiving mechanically: a funnel-shaped dock guides a robot that
back off 0.5 m and try again, at most 3 times, then FAULT (the table above): a rule of
```

4. Simple planning: grid maps and finding a path

Reactive behaviours and state machines choose *what to do now*. To choose *how to get there* needs a **map** and a **planner** that searches the map for a route.

Grid maps and occupancy grids

The simplest map is a **grid**: the floor is cut into squares of a chosen size, and each cell is free, blocked, or (before the robot has looked) unknown. World coordinates become cell indices by division: column = floor(x / cell), row = floor(y / cell). With cells of 0.25 m, the point (x = 1.30 m, y = 0.60 m) lies in column floor(5.2) = 5, row floor(2.4) = 2. The cell size is a trade-off between detail and memory: a 20 m by 20 m floor is 80 × 80 = 6,400 cells at 0.25 m but 400 × 400 = 160,000 cells at 0.05 m (at one byte per cell, 160 kB; it also takes 25 times as many cells for a planner to search). To stop the planner from steering the robot's centre next to a wall, **inflate** each obstacle by the robot's radius (0.15 m is 0.6 of a 0.25 m cell, so one extra cell of margin round every blocked cell), which lets the planner treat the robot as a point.

Sensors are noisy and a cell is rarely certain, so a map built from measurements stores a **probability of occupancy** in each cell (an **occupancy grid**). Every cell starts at 0.5, “don't know”. Each time a range sensor beam *ends* in a cell (a hit), that cell becomes more likely to be occupied; each cell the beam *passes through* on the way becomes more likely to be free. Updating the probabilities directly means a small multiplication-and-division for every reading. The neat trick is to store **log-odds** instead, where each new reading just adds a number.

```
odds = p / ( 1 - p )         log-odds   l = ln( odds )         back to a probability:

Sensor model (typical, illustrative):  a hit says "occupied" with 70 % confidence, a pa
       l(hit)  = ln( 0.7 / 0.3 ) = +0.847          l(free) = ln( 0.3 / 0.7 ) = -0.847

One cell, four readings in turn:  hit, hit, free, hit
       reading        hit     hit     free    hit
       l              0.847   1.695   0.847   1.695
       p              0.700   0.845   0.700   0.845          (net: three hits and one f
       Check with Bayes directly on the first two hits:  0.7 x 0.7 / ( 0.7 x 0.7 + 0.3 :
```

```
A cell a beam passes through three times without a hit:   l = -3 x 0.847 = -2.542,   p =
Clamp l to +-4 (p between 0.018 and 0.982):  otherwise a cell seen as a wall for 20 rea
```

Searching the grid: breadth-first search

A **graph search** treats each free cell as a node, joined to its 4 neighbours (up, right, down, left) by edges of cost 1. **Breadth-first search** (BFS) explores outward from the start in rings: first the cells 1 move away, then those 2 moves away, and so on, using a first-in, first-out queue. The first time it reaches the goal it has used the fewest moves, so for equal-cost moves it always finds a shortest path. Its weakness is that it explores in *all* directions, including the wrong ones. The example grid below is used for BFS and then for A*; S is the start, G the goal, `#` a blocked cell, and rows and columns are numbered from 0 at the top left. A cell is written (row, column).

```
        c0    c1    c2    c3    c4    c5              Start S = (2,0).   Goal G = (
   r0    .     .     .    ###    .     .              Each cell is 0.25 m.   Moves:
   r1    .     .     .    ###    .     G
   r2    S     .     .    ###    .     .
   r3    .     .     .     .     .     .
   r4    .     .     .    ###    .     .

BFS: the number of moves from S to each cell, found by the rings of the search (the goa

        c0    c1    c2    c3    c4    c5
   r0    2     3     4    ###    8     9
   r1    1     2     3    ###    7    [8]  G
   r2   [0]    1     2    ###    6     7    S
   r3    1     2     3     4     5     6
   r4    2     3     4    ###    6     7

BFS takes cells off its queue in order of distance and stops when G comes off the queue
The shortest path has 8 moves, one of several: (2,0) (2,1) (2,2) (3,2) (3,3) (3,4) (2,4
```

A*: searching with a guess

A* (“A star”) keeps the shortest-path guarantee but explores mainly toward the goal. For each cell it tracks g, the cost of the best path found so far from the start, and a **heuristic** h, an *estimate* of the remaining cost to the goal. It always expands the open cell with the smallest f = g + h. On a 4-connected grid the standard heuristic is the **Manhattan distance** h = |row −

row_{goal}| + |col − col_{goal}| (the number of moves if there were no walls). It never overestimates the true remaining cost (*admissible*, and consistent), which is what guarantees that the first path found is a shortest one. With h = 0 A* becomes Dijkstra's algorithm (BFS with unequal costs); the larger the heuristic stays below the truth, the more the search is guided. If diagonal moves are allowed (cost 1.414), use the octile distance instead, or Manhattan will overestimate.

The rules used below, so that the hand-work is repeatable: pick the open cell with the lowest f; if tied, the lowest h (closer to the goal); if still tied, the one added to the open list first. For each neighbour (in the order up, right, down, left) that is not a wall: new g = g(current) + 1, and if that is better than the neighbour's recorded g (or it has none), record it and add it to the open list with its f. Goal G = (1,5), so h(r,c) = |r − 1| + |c − 5|, and h(S) = 1 + 5 = 6.

```
  #   expand   g   h   f    neighbours added or improved (cell: g, h, f)
  1   (2,0)    0   6   6    (1,0): 1,5,6    (2,1): 1,5,6    (3,0): 1,7,8
  2   (1,0)    1   5   6    (0,0): 2,6,8    (1,1): 2,4,6                                  (
  3   (1,1)    2   4   6    (0,1): 3,5,8    (1,2): 3,3,6
  4   (1,2)    3   3   6    (0,2): 4,4,8    (2,2): 4,4,8                                  (
  5   (2,1)    1   5   6    (2,2): 2,4,6  (improved from g = 4 to 2)    (3,1): 2,6,8
  6   (2,2)    2   4   6    (3,2): 3,5,8                                                  (
  7   (0,2)    4   4   8    none
  8   (0,1)    3   5   8    none
  9   (3,2)    3   5   8    (3,3): 4,4,8    (4,2): 4,6,10
 10   (3,3)    4   4   8    (3,4): 5,3,8                                                  (
 11   (3,4)    5   3   8    (2,4): 6,2,8    (3,5): 6,2,8    (4,4): 6,4,10
 12   (2,4)    6   2   8    (1,4): 7,1,8    (2,5): 7,1,8
 13   (1,4)    7   1   8    (0,4): 8,2,10   (1,5): 8,0,8
 14   (1,5)    8   0   8    GOAL reached, path cost g = 8

  Order of expansion on the grid (. = never expanded;  14 cells expanded, against 25 for

         c0    c1    c2    c3    c4    c5
    r0    .     8     7    ###    .     .
    r1    2     3     4    ###   13    14   (G)
    r2    1     5     6    ###   12     .   (1 is S)
    r3    .     .     9    10    11     .
    r4    .     .     .    ###    .     .

  Path back from G by following the recorded parents:  (1,5) <- (1,4) <- (2,4) <- (3,4) <
  Notice that f stayed at 6 (the straight-line guess) while the search ran toward the wal
```

```
from heapq import heappush, heappop

def astar(grid, start, goal):                       # grid: list of strings, "#" = bloc
    def h(cell):
        return abs(cell[0] - goal[0]) + abs(cell[1] - goal[1])       # Manhattan distan

    def neighbours(cell):
        for dr, dc in ((-1, 0), (0, 1), (1, 0), (0, -1)):            # up, right, down
            r, c = cell[0] + dr, cell[1] + dc
            if 0 <= r < len(grid) and 0 <= c < len(grid[0]) and grid[r][c] != "#":
                yield (r, c)

    g = {start: 0}
    parent = {start: None}
    closed = set()
    added = 0                                       # counts insertions, to break tie
    open_list = [(h(start), h(start), 0, start)]    # (f, h, order, cell): the heap p
    while open_list:
        f, hh, order, cell = heappop(open_list)
        if cell in closed:
            continue                                # an out-of-date entry for a cell
        closed.add(cell)
        if cell == goal:
            path = []
            while cell is not None:
                path.append(cell)
                cell = parent[cell]
            return path[::-1], g[goal]
        for nxt in neighbours(cell):
            new_g = g[cell] + 1
            if nxt not in g or new_g < g[nxt]:
                g[nxt], parent[nxt] = new_g, cell
                added += 1
                heappush(open_list, (new_g + h(nxt), h(nxt), added, nxt))
    return None, None                               # open list empty: the goal canno
```

Practical notes. **Path cost** need not be distance only: a planner can charge more for cells near walls (inflation zones), for turns (many planners add a small penalty per change of direction, which would favour the straight run along row 3 over the staircase above), or for rough floor; A* simply minimises the total. **Replanning:** when the robot finds a new obstacle on the path (a chair that was not on the map), mark the cell blocked and search again from the robot's current

cell, which on a small grid takes a fraction of a second; incremental methods (D* and its relatives) reuse earlier work. **Path following:** the planner gives a list of cells; turning it into motion is the job of the controllers of Robotics Fundamentals 7 (steer toward the next waypoint with a heading controller, speed limited near obstacles).

5. Localisation

A map is only useful if the robot knows *where it is on it*. Robotics Fundamentals 6 showed that dead reckoning drifts and that an absolute fix pulls the error back. Localisation is the art of combining the two: the robot keeps a **belief**, a probability distribution over where it might be ("probably near (2.1, 1.0), within about 0.2 m"), and repeats two steps. **Predict:** as the wheels turn, move the belief along with the odometry, and make it *wider*, because every movement adds uncertainty. **Update:** when a sensor reading arrives (a distance to a wall, a beacon range, a landmark), keep the possible positions that agree with it and discard those that do not, making the belief *narrower*. The Kalman filter of Robotics Fundamentals 6 section 5 is this loop for a single bell-curve belief. When the belief is not a single bump (the robot could be in either of two identical corridors), something more flexible is needed.

The particle filter, conceptually

A **particle filter** represents the belief as a cloud of guesses (*particles*), each a complete possible pose, often several hundred to thousands (a typical range, depending on the map). In each loop: **1. Move** every particle by the odometry reading plus some random noise, so the cloud spreads like the uncertainty. **2. Weight** each particle by how well the real sensor reading matches what that particle *would* read from its own pose on the map; a particle that agrees gets a high weight, one that disagrees a weight near zero. **3. Resample**: draw a new cloud, picking particles in proportion to their weights, so that good guesses are copied and bad ones die out. The cloud collapses onto the poses that fit the evidence, and the robot's best estimate is the cloud's average (if it is one group). A cloud that starts spread evenly over the whole map can find the robot's place without being told, which dead reckoning cannot do ("the kidnapped robot" problem).

```
A robot on a corridor, 1-D for clarity.  A wall is at x = 5.0 m; a range sensor reads tl
Five particles, equally weighted. Weight rule: w = exp( -( z - predicted range )^2 / ( 2

        particle x   predicted range 5.0 - x    z - predicted    weight w     normalised
          1.0             4.0                     -1.1           0.0012        0.001
```

```
          1.5             3.5                  -0.6          0.1353        0.090
          2.0             3.0                  -0.1          0.9460        0.629
          2.5             2.5                  +0.4          0.4111        0.273
          3.0             2.0                  +0.9          0.0111        0.007
        sum of weights = 1.5047;   normalised = each w / 1.5047;   expected copies = 5 x

        Weighted mean position = 1.0 x 0.001 + 1.5 x 0.090 + 2.0 x 0.629 + 2.5 x 0.273 +
        A typical resampled cloud: 2.0, 2.0, 2.0, 2.5, 1.5 (random: the numbers of copie

   Cost: the work grows with the number of particles; too few and the right answer may not
```

Trilateration from beacons

With range beacons at known places (Robotics Fundamentals 6 section 8) the robot can compute a position *directly*. Each measured range d_i puts the robot on a circle around beacon i.

pairwise cancels the squared terms and leaves two straight-line equations. For beacon A at the origin, B on the x axis at (b, 0) and C on the y axis at (0, c), the solution has a closed form:

```
   Circles:   x^2 + y^2 = dA^2          ( x - b )^2 + y^2 = dB^2          x^2 + ( y - c )^
   Subtract the first from each of the others:     x = ( dA^2 - dB^2 + b^2 ) / ( 2 b )

   Worked example:  A = (0, 0),  B = (4, 0),  C = (0, 3) metres.   The robot is really at
          x = ( 2.50^2 - 3.20^2 + 4^2 ) / ( 2 x 4 ) = ( 6.25 - 10.24 + 16 ) / 8 = 12.01 /
          y = ( 2.50^2 - 1.80^2 + 3^2 ) / ( 2 x 3 ) = ( 6.25 - 3.24 + 9 ) / 6  = 12.01 / 6

   Sensitivity:  add 0.10 m of error to dB alone (3.30):   x = ( 6.25 - 10.89 + 16 ) / 8 =
                 add 0.10 m of error to dA alone (2.60):   x = 1.565 m, y = 2.087 m:   an
                 typical of the geometry; it gets WORSE when the robot is outside the beac
   With more than three beacons, find the position that minimises the sum of squared range
   With only two beacons, or three in a straight line, there are two answers, a mirror pai
```

SLAM, in concept only

SLAM (simultaneous localisation and mapping) is the problem of building a map and locating the robot on it *at the same time*, with neither given in advance. It is a chicken-and-egg problem: to add what the sensor sees to the map, you need to know where the robot is; to know where the robot is, you need the map. The solution, in outline: the robot moves and measures, estimates its pose from odometry, matches each new scan against the map so far (or against

landmarks it has seen before), and corrects both the map and its pose estimate together. Its hardest moment is **loop closure**: coming back to a place already seen, recognising it, and pulling the whole drifted trail back into line. Common forms include filters (Kalman, particle) and *graph* SLAM, which keeps all the poses and measurements as a network of constraints and finds the arrangement that fits them best. SLAM with a lidar is what many floor robots use; it is implemented in ready-made software packages, and building one from scratch is far beyond this course. What matters here is why it exists and what it needs: good range data, a sensible odometry, and a world that holds still.

6. Behaviour trees, briefly

A state machine says *where the robot is* and what causes it to move on. A **behaviour tree** says *what it should try*. The tree is run ("ticked") from the top at every loop; each node returns **success**, **failure** or **running**. A **Selector** (fallback) node tries its children left to right and stops at the first that does not fail; a **Sequence** node runs its children in order and stops at the first that does not succeed; the leaves are **conditions** (a sensor test) and **actions**.

```
ROOT: Selector      (try children left to right; stop at the first that succeeds or is
  |-- Sequence "recharge"
  |      |-- Condition: battery < 20 %
  |      `-- Action:    go to the dock
  |-- Sequence "avoid"
  |      |-- Condition: front distance < 0.30 m
  |      `-- Action:    turn away
  `-- Action:    clean the floor        (the default: it only gets a turn if the two ab
```

```
Finite state machine                                          Behaviour tree
------------------------------------------------------------  ----------------------------
Natural for modes with a clear sequence (docked, cleaning,    Natural for prioritised alt
charging) and for exactly-specified transitions               this, else that") that are
Transitions are explicit arrows: easy to check, but many      Adding a behaviour means ad
arrows as the machine grows (state explosion)                 need a new arrow to every o
Remembers its state between ticks                             Re-checks conditions from t
                                                              a node returns "running"
Best for small robots, hardware-level safety logic            Best for larger sets of beh
```

Neither is better in general, and they are often combined: a state machine for the robot's operating modes, a behaviour tree (or the priority list of section 1) inside the mode that needs many alternatives. Behaviour trees are named here so that you recognise them; building one is a topic for Programming for Robotics or a software course.

7. Testing and failure handling

A robot's behaviour layer is the part most likely to meet situations nobody planned for, and the part where a mistake moves a heavy object near people. Four habits turn "it worked once" into "it fails safely".

- **Timeouts.** Every state that waits for something (an arrival, a dock, a door) gets a maximum time, after which the machine takes a defined exit (the table of section 2: AVOID 5 s, RETURN 120 s, CHARGE 8 hours). Without them a robot can wait forever in a state that no event will ever end.
- **A watchdog.** A **watchdog timer** is a countdown that the main program must keep *resetting* ("kicking") while it runs healthily; if the program freezes or crashes, the countdown reaches zero and hardware (not the frozen software) cuts the motor power. A typical choice is 5 times the loop period (100 ms for a 20 ms loop).
- **Safe stop.** Define what "safe" means for each failure: usually motors off and brakes on, but an arm that holds a load or a robot on a slope may need to be held, not released (Robotics Fundamentals 10). The safe state must also be the state the hardware falls into when power or signal is lost, not only a state that software can choose.
- **Logging.** Write down every state transition with its time and its cause, and the key sensor values at a fixed rate. When the robot does something strange, the log is the only witness. At 20 records per second of 40 bytes, that is 800 B/s or 2.9 MB per hour, easily stored on a card. A good log lets you replay a failure in simulation.

```
Stopping distance budget (typical, illustrative values):  speed v = 0.5 m/s,  braking d
       Reaction:   sensor period 50 ms + processing 20 ms + motor command 10 ms = 80 ms
       Braking:    v^2 / ( 2 a ) = 0.25 / 4 = 0.0625 m
       Total:      0.040 + 0.0625 = 0.1025 m.   With a safety margin of 2, the stop thr
       At 1.0 m/s: 0.08 + 1.0 / 4 = 0.33 m, MORE than the 0.30 m threshold: the robot c
       The largest speed that stops in 0.30 m is about 0.95 m/s, with NO margin:  v x 0

Watchdog: the program freezes while the robot is moving at 0.5 m/s.  With a 100 ms watc
       Allow for it in the clearance that you leave around people and walls.
```

```
Habits: test each behaviour alone, with wheels off the ground; then in a simulator; then
```

8. Challenges

CHALLENGE 1: REACTIVE BEHAVIOURS, ARBITRATION AND STATE MACHINES

(a) A Braitenberg vehicle has sensors reading sL = 0.2 and sR = 0.7 (the light is on the right), b = 0.1 m/s, g = 0.5 m/s per unit and track width L = 0.12 m. Find both wheel speeds, the forward speed and

behaviours of section 1 (STOP, AVOID, LINE, WANDER), give the winner and the command for these snapshots: (i) no bump, front 0.25, left 0.3, right 0.8, line at +4 mm; (ii) no bump, front 0.9, line at −10 mm; (iii) no bump, front 0.9, no line. (c) A delivery robot has the states WAIT, TO_PICKUP, LOADING, TO_DROP, UNLOADING, HOLD_PICKUP and HOLD_DROP, and the events order, arrived, loaded, unloaded, blocked and clear. Write its transition table (WAIT goes to TO_PICKUP on order; arriving moves on in turn; blocked on either trip goes to the matching HOLD state, and clear goes back to that trip), then trace the events order, arrived, loaded, blocked, clear, arrived, unloaded and give every state visited. Explain why two HOLD states were needed. (d) A battery reading is sampled every 0.2 s: 24, 19, 22, 19, 18, 19, 19, 18, 19, 19. battery_low needs 4 readings in a row below 20. At which sample, and at what time, does it fire, against the naive rule (the first reading below 20)? (e) A robot has 5 drive modes, 3 battery conditions, 2 payload conditions and 2 communications conditions. How many states does a single machine need, and how many do four separate machines need?

View solution

CHALLENGE 2: LINE FOLLOWING, OBSTACLE AVOIDANCE, WALL FOLLOWING AND HOMING

(a) A line follower has vmax = 0.40 m/s, c = 0.04 per mm, Kp = 0.06 rad/s per mm, L = 0.18 m, and the line is at x = +15 mm. Find e, the turn rate, the speed, both wheel speeds, and the direction of turn. If the line is then lost and the robot spins toward the last-seen side at 1.0 rad/s for the 3 s limit, how many degrees does it turn, and which way? (b) A potential field with k = 0.02, d0 = 0.60 m and ka = 0.30: the goal is 1.5 m straight ahead, obstacle A is 0.30 m straight ahead, obstacle B is 0.45 m to the right. Find the three forces, their sum, its size and direction, and say which way the robot veers. (c) Wall following, wall on the right, wanted distance 0.35 m, two sensors 0.12 m apart reading dF = 0.28 m and dR = 0.36 m, with Kp = 2.5 and Ktheta = 1.2, v = 0.20 m/s, L = 0.14 m. Find theta (and which way the robot is heading relative to the wall), the perpendicular distance, the error, the turn rate command, and both wheel speeds. (d) Homing with Kb = 1.2 per second, vmax = 0.20 m/s, kv = 0.4 per second: the beacon is 35 degrees to the right at 0.8 m. Find w and v, and the speed at a range of 0.2 m.

View solution

CHALLENGE 3: OCCUPANCY GRIDS, PLANNING, LOCALISATION AND STOPPING SAFELY

(a) An occupancy grid uses a sensor model of 80 % confidence (a hit means occupied with p = 0.8, a pass-through free with p = 0.8). Find the log-odds step, and the log-odds and probability of one cell after the readings hit, hit, free, hit, hit. Then after 4 pass-throughs from the prior of 0.5. What does clamping at ±4 give as the probability limits? (b) On the grid with rows `S . # . .` / `. . # . G` / `. . . # .` / `# . . . .` (S at (0,0), G at (1,4), # blocked, cells 0.2 m), use moves in the order up, right, down, left. Find the Manhattan distance from S to G, then by BFS the number of moves to G and the distance written in every reached cell, and then run A* with the tie rules of section 4 (lowest f, then lowest h, then first added): list the expansions with g, h, f, the path found and its length in metres, and compare the number of expansions with BFS. (c) Beacons A = (0, 0), B = (5, 0), C = (0, 4) measure dA = 3.20, dB = 3.90 and dC = 2.50 m. Find the position and check it against the true position (2.0, 2.5). (d) A wall is at x = 6.0 m and the range reading is z = 3.6 m with sigma = 0.5 m. Particles sit at x = 1, 2, 3, 4 and 5 m. Find each predicted range, weight and normalised weight, the expected copies in a cloud of 5, and the weighted mean position. (e) A robot at 0.8 m/s with deceleration 2.5 m/s^2 and a total reaction time of 60 ms: find the stopping distance, the stop threshold with a margin of 2, and the largest speed at which a 0.30 m threshold still leaves a margin of 2.

View solution

QUICK REFERENCE

- **Reactive behaviour:** sensors straight to motors, no map and no memory; fast and robust, but it cannot plan or remember. Braitenberg: wheel speed = b + g × sensor; crossed wiring turns toward the stimulus, uncrossed turns away.
- **Arbitration:** priority list (first behaviour with a command wins; safety first, default last) for anything that matters; blending only among behaviours that agree, because an average of two avoids can steer straight at the obstacle.
- **State machine:** states, events, a transition table (the table is the program; unlisted pairs are ignored); log every transition; give waiting states a timeout.
- **Debounce:** require the condition for a time or a count (5 samples at 0.1 s), use hysteresis (enter at 20 %, leave at 25 %) and a dwell time; size the delay to the cost of a late reaction (battery 0.5 s, cliff 15 ms).
- **State explosion:** independent aspects multiply (4 × 3 × 2 = 24) when put in one machine, but add (4 + 3 + 2 = 9) as separate machines; use hierarchy for shared transitions.

- **Line following:** $e = -x$, $u = K_p\, e$, $v_{R,L} = v \pm uL/2$, $v = v_{max}/(1 + c|e|)$; add lost-line search toward the last-seen side with a time limit.
- **Obstacle avoidance:** stop at 0.30 m, resume above 0.45 m (hysteresis), turn toward the clearer side, 3 avoids in 10 s means ESCAPE. **Potential field:** $F_{rep} = k(1/d - 1/d_0)/d^2$ for $d < d_0$; sum with the goal pull; beware local minima.
- **Wall following:** $u = -(K_p(d_{perp} - d_{ref}) + K_\theta\theta)$ for a wall on the right; $\theta = \text{atan}((d_F - d_R)/s)$, $d_{perp} = ((d_F + d_R)/2)\cos\theta$. **Homing:** $w = K_b\beta$, $v = \min(v_{max}, k_v\, r)$; retry a limited number of times.
- **Occupancy grid:** store log-odds; $l = l + l(\text{reading})$; $p = 1/(1 + e^{-l})$; $l(\text{hit}) = \ln(0.7/0.3) = +0.847$ for a 70 % model; clamp at ±4. Column = floor(x/cell), row = floor(y/cell); inflate obstacles by the robot radius.
- **BFS** finds a shortest path (equal costs) but explores everywhere (25 cells in the example). **A*** expands the lowest $f = g + h$ with the Manhattan $h = |\Delta\text{row}| + |\Delta\text{col}|$ (admissible), and found the same 8-move, 2.0 m path with 14 expansions.
- **Localisation:** belief = predict (wider) + update (narrower). Particle filter: move, weight by $w = \exp(-(z - z_{pred})^2/2\sigma^2)$, resample. Trilateration: $x = (d_A^2 - d_B^2 + b^2)/2b$, $y = (d_A^2 - d_C^2 + c^2)/2c$. SLAM builds the map and the pose together; loop closure fixes the drift.
- **Behaviour tree:** selector (first child that does not fail), sequence (all must succeed), re-evaluated every tick; state machines suit modes, trees suit prioritised alternatives.
- **Failure handling:** timeouts, a hardware watchdog (about 5 times the loop period), a defined safe stop, and a log of every transition. Stopping distance = $v \times \text{latency} + v^2/2a$; keep a margin of 2 or more.

Next, Robotics Fundamentals 9 moves from a robot that drives to a robot that *reaches*: arms and manipulators. Joints and links, forward kinematics (where is the hand, given the joint angles?) and inverse kinematics (what joint angles put the hand there?) worked out by hand for a two-link arm, grippers, and the torque a joint must supply to hold a payload at arm's length, using the motor sizing of Robotics Fundamentals 3 and the position control and feedforward of Robotics Fundamentals 7.

Robotics Fundamentals

Course 1 · Chapter 9 · Arms and Manipulators

Every robot so far has moved *itself* through the world: wheels on a floor, with a position and a heading to keep track of (Robotics Fundamentals 4) and behaviours to choose between (Robotics Fundamentals 8). This chapter is about a robot that stays put and moves *things*. A **manipulator** is a chain of rigid links joined by motorised joints, with a hand, or **end effector**, on the end. The central questions are the two that decide every arm: **where is the hand, given the joint angles?** (forward kinematics) and **what joint angles put the hand there?** (inverse kinematics). We work both out by hand for a two-link arm, extend them to a third link that sets the hand's orientation, and name the convention (Denavit-Hartenberg) that keeps the bookkeeping straight for bigger arms. Then we look at where the arm can and cannot reach, and at the **singular** poses where it loses a direction of motion; at the Jacobian, which links joint speeds to hand speed and hand force to joint torque; at the torque a joint needs to hold a payload (the part that decides which servo or motor you can buy); at moving smoothly from one pose to another; at why a real arm is never quite as accurate as its geometry says; at grippers and the force needed to hold what they grip; and at keeping people safe around all of it.

The link lengths, masses, torques, speeds, accuracies and friction coefficients in this chapter are **illustrative or typical, approximate values** chosen so that the arithmetic is easy to follow, not data for a particular arm, servo or gripper. The formulas (planar forward and inverse kinematics, the Jacobian, $\tau = J^{T} F$, the torque of a weight at a distance, the trapezoidal profile, the friction grip condition, the cantilever deflection) are standard. Every number in the worked examples was recomputed, and the pseudocode was actually run, including a round trip (inverse kinematics, then forward kinematics) that returns to the starting point. The code is deliberately short, hardware-neutral Python-style pseudocode, used only for forward kinematics, inverse kinematics and the trapezoidal profile; real board code is the job of Programming for Robotics.

1. Joints, links and arm types

A **link** is a rigid piece; a **joint** connects two links and allows one kind of motion between them. Robotics Fundamentals 2 section 2 listed the joint types. Two do almost all the work in arms. A **revolute** joint (hinge) rotates about one axis and is described by an angle θ, driven by a servo, a geared motor or a stepper. A **prismatic** joint (slider) moves along one axis and is described by a length d, driven by a lead screw, a belt or a linear actuator. Each such joint has one degree of

freedom, so, as Robotics Fundamentals 1 section 4 showed, the number of joints of a serial chain is its number of degrees of freedom (DOF): one motor per DOF.

The **link lengths** are the fixed numbers of the design: L_1 from shoulder to elbow, L_2 from elbow to wrist, and so on, measured between joint axes. The joint variables (θ_1, θ_2, ...) are what the motors change. Everything in kinematics is a relation between the two.

How many DOF does an arm need? The hand is a rigid body, and Robotics Fundamentals 1 section 4 counted that placing a body anywhere in space and pointing it any way takes 6 numbers (3 for position, 3 for orientation). A six-joint arm can therefore reach a point *and* choose how the hand is turned there; an arm with fewer joints gets only some of that freedom. On a table-top, a 3-DOF arm can place a hand at a point, a fourth DOF (a wrist turn) adds an orientation, and a fifth adds a tilt. A **redundant** arm has more joints than the task needs (a 7-DOF arm reaching round an obstacle), at the price of an inverse problem with many solutions. Gruebler's formula counts the DOF of a mechanism, but, as Robotics Fundamentals 1 pointed out, it counts only; it does not check that the geometry really moves.

ARM TYPE	JOINTS	TYPICAL REACH AND PAYLOAD (APPROXIMATE)	STRENGTHS	WATCH FOR
Cartesian / gantry	Three prismatic axes at right angles (X, Y, Z), often plus a wrist	Very scalable: from 0.2 m desktop machines (3D printers, plotters) to 10 m or more gantries; payload from grams to hundreds of kg	Simplest maths (x, y, z are the joint values); rigid; easy to make accurate; large workspace is a plain box	Bulky for the volume reached; the whole frame must be larger than the workspace; rails need protection and maintenance
SCARA	Two revolute joints with parallel vertical axes (a planar 2-link arm) plus a vertical prismatic axis, plus a wrist spin; 4 DOF	Reach about 0.2 to 1 m; payload about 1 to 20 kg; fast, with repeatability around 0.01 to 0.05 mm on good models	Stiff downward (vertical axes carry the load) and quick sideways; ideal for pick-and-place and assembly from above	Cannot tilt the hand; vertical stroke is short; the planar kinematics of this chapter applies directly
Articulated 6-axis	Six revolute joints: base turn, shoulder, elbow, and a three-axis wrist	Reach about 0.5 to 3 m; payload from about 3 kg to several hundred kg; repeatability about 0.02 to 0.1 mm	Reaches around and under objects at any hand orientation; the standard factory	Complex inverse kinematics, many singular poses; the large shoulder

ARM TYPE	JOINTS	TYPICAL REACH AND PAYLOAD (APPROXIMATE)	STRENGTHS	WATCH FOR
			arm (welding, painting, handling)	torque limits payload at full reach
Delta (parallel)	Three arms hanging from a fixed top plate, each driven at the top, joined by light rods to one small platform; 3 DOF, sometimes plus a wrist spin	Reach 0.2 to 1.3 m; light payloads (grams to about 1 to 8 kg); very fast, over 100 picks a minute	The heavy motors are fixed to the frame, so the moving mass is tiny: ideal for fast light pick-and-place (food, packaging)	Small workspace and a dome shape; the kinematics is not a simple chain, and the payload is low
Hobby servo arm	Typically 4 to 6 revolute joints, each a hobby servo, with a small gripper	Reach 0.2 to 0.4 m; payload tens to a few hundred grams	Cheap, light, simple to wire and drive; the best way to learn all of this chapter on a desk	Backlash, flexible plastic parts and servo torque limits; repeatability is millimetres, not hundredths of a millimetre

The figures in the table are typical orders of magnitude from catalogues and are approximate; the point is the pattern. Cartesian machines are simple and stiff; the SCARA is the fast, planar specialist; the articulated 6-axis is the general-purpose reaching arm; the delta trades size and payload for speed; and the servo arm is the learner's version of an articulated arm. The rest of the chapter uses a **planar two-link arm** (the shoulder and elbow of a SCARA, or of an articulated arm seen from the side) as the worked model, because its algebra fits on a page and every idea carries over to the bigger arms.

2. Forward kinematics

Forward kinematics (FK) answers: given the link lengths and the joint angles, where is the hand? For differential drive, Robotics Fundamentals 4 section 3 turned wheel speeds into a position; here the joint angles do the same job. For a planar arm, each link adds a vector. The first link points at angle θ_1 from the x axis; the second is turned a further θ_2 relative to the first, so it points at $\theta_1 + \theta_2$ from the x axis. The hand is the sum of the two vectors. Angles are in radians in programs and counter-clockwise positive; degrees are used below for readability.

```
Planar 2-link arm (not to scale):  shoulder O at the origin; elbow E; end effector P.

  y
  ^                          P  (x, y)      the hand
  |                        /
  |                      /    L2            link 2 points at ( t1 + t2 ) from the +x ax
  |                    /
  |                  E  (elbow)
  |               . '  .
  |           . '   /
  |       . '     /   L1                     link 1 points at t1 from the +x axis
  |   . ' t1    /
  O ----------------------------------> x   the shoulder is fixed to the base

  Elbow position:   Ex = L1 cos t1                    Ey = L1 sin t1
  Hand position:    x  = L1 cos t1 + L2 cos( t1 + t2 )
                    y  = L1 sin t1 + L2 sin( t1 + t2 )          (t2 > 0 bends the elbow

Worked example:  L1 = 0.30 m,  L2 = 0.20 m,  t1 = 40 deg,  t2 = 60 deg   (so link 2 poi
       cos 40 = 0.7660     sin 40 = 0.6428     cos 100 = -0.1736     sin 100 = 0.9848
       x = 0.30 x 0.7660 + 0.20 x ( -0.1736 ) = 0.22981 - 0.03473 = 0.19508 m
       y = 0.30 x 0.6428 + 0.20 x   0.9848    = 0.19284 + 0.19696 = 0.38980 m
       Elbow at ( 0.2298, 0.1928 ) m.   Check: the distance from the elbow to the hand
```

```
from math import cos, sin

def fk(L1, L2, t1, t2):                       # angles in radians; returns the hand pos
    x = L1 * cos(t1) + L2 * cos(t1 + t2)
    y = L1 * sin(t1) + L2 * sin(t1 + t2)
    return x, y
```

A third link: setting the hand's orientation

Two joints fix the position of the hand and nothing else: the direction it points is whatever the angles happen to give. A third revolute joint (a **wrist**) of link length L_3 gives control of that direction. The third link points at $\varphi = \theta_1 + \theta_2 + \theta_3$, which is also the **orientation** of the end effector (the angle at which a gripper points), and the position gains one more vector:

```
x   = L1 cos t1 + L2 cos( t1 + t2 ) + L3 cos( t1 + t2 + t3 )
y   = L1 sin t1 + L2 sin( t1 + t2 ) + L3 sin( t1 + t2 + t3 )
phi = t1 + t2 + t3                                                    (the hand's

Why this makes the INVERSE problem easy:  the point where the wrist joint sits (call it
       Wx = x - L3 cos( phi )          Wy = y - L3 sin( phi )
and W depends only on t1 and t2, so W is found with the 2-link formulas of section 3;

Worked example:  L1 = 0.30,  L2 = 0.20,  L3 = 0.10 m;  t1 = 0,  t2 = 90 deg,  t3 = -90
       Elbow at ( 0.30, 0 );   link 2 points at 90 deg, so the wrist is at ( 0.30, 0.20
       Same hand position and orientation, other elbow:  t1 = 67.38,  t2 = -90,  t3 = 2
```

Denavit-Hartenberg: a bookkeeping convention

For a planar arm the sums above are easy. For a six-joint arm in 3D, with joint axes pointing in many directions, a rule is needed to say how each joint is described, so that anyone reading the arm's data sheet can write the same equations. The **Denavit-Hartenberg (DH) convention** (Denavit and Hartenberg, 1955) is that rule. It fixes a coordinate frame on every link in a standard way and then describes each joint by just **four numbers**: a link length a and a link twist α (both fixed by the shape of the arm), a link offset d and a joint angle θ (one of which is the variable of the joint: θ for a revolute joint, d for a prismatic one). The hand's pose is then the product of one standard 4 × 4 matrix per joint. For the planar 2-link arm of this chapter the table has two rows, with $a_1 = L_1$, $a_2 = L_2$, every twist and offset zero, and θ_1, θ_2 the variables: the product of the two matrices contains exactly the two equations above. DH is named here so you recognise the table in a manual (from memory, there are two common variants, “classic” and “modified”, which place the frames slightly differently, so check which one a data sheet uses); deriving it is the subject of a robotics-kinematics course.

3. Inverse kinematics

Inverse kinematics (IK) is the harder direction: given the wanted hand position (x, y), find the joint angles. It is what an arm controller has to solve every time it is told “go to this point”. For the 2-link arm there is a clean answer from geometry. The shoulder, the elbow and the hand form a triangle with sides L_1, L_2 and $r = \sqrt{(x^2 + y^2)}$, the straight-line distance from shoulder to hand. The **law of cosines** on that triangle gives the elbow angle, and the shoulder angle is the direction to the hand minus the angle the first link sits away from that direction.

```
Target ( x, y ),   r^2 = x^2 + y^2

Reachability first:     | L1 - L2 |  <=  r  <=  L1 + L2            (outside this ring r

Elbow:     the law of cosines on the triangle gives the interior angle at the elbow; th
           cos t2 = ( r^2 - L1^2 - L2^2 ) / ( 2 L1 L2 )           t2 = +acos( ... ) or
Shoulder:  t1 = atan2( y, x )  -  atan2( L2 sin t2,  L1 + L2 cos t2 )       (the direct

Use atan2, not atan( y / x ):  atan2 keeps the quadrant and works when x = 0.

Elbow-down (t2 > 0): the elbow bends counter-clockwise, so for a target on the right th
Elbow-up   (t2 < 0): the elbow bends clockwise, and the elbow sits ABOVE that line.

Worked example:  L1 = 0.30,  L2 = 0.20,  target ( 0.35, 0.25 ) m.
       r^2 = 0.1225 + 0.0625 = 0.1850,   r = 0.4301 m.     Reachable?  0.10 <= 0.4301 <=
       cos t2 = ( 0.1850 - 0.0900 - 0.0400 ) / ( 2 x 0.30 x 0.20 ) = 0.0550 / 0.1200 = 0

       ELBOW-DOWN:  t2 = +62.72 deg.   atan2( 0.25, 0.35 ) = 35.54 deg.    atan2( 0.20 s
                    t1 = 35.54 - 24.41 = 11.13 deg.
                    Check by FK:  x = 0.30 cos 11.13 + 0.20 cos 73.85 = 0.2944 + 0.0556

       ELBOW-UP:    t2 = -62.72 deg.   atan2( 0.20 sin( -62.72 ),  0.3917 ) = atan2( -0
                    t1 = 35.54 + 24.41 = 59.95 deg.
                    Check by FK:  x = 0.30 cos 59.95 + 0.20 cos( -2.77 ) = 0.1503 + 0.19

Two different poses of the arm (11.13, 62.72) and (59.95, -62.72) put the hand on the sa
```

```
from math import acos, atan2, cos, sin

def ik(L1, L2, x, y, elbow_down=True):
    r2 = x * x + y * y
    c2 = (r2 - L1 * L1 - L2 * L2) / (2 * L1 * L2)
    if abs(c2) > 1.0:                                  # outside |L1 - L2| <= r <= L1 + L2
        return None                                     # unreachable: say so, never move
    t2 = acos(c2) if elbow_down else -acos(c2)
    t1 = atan2(y, x) - atan2(L2 * sin(t2), L1 + L2 * cos(t2))
    return t1, t2

# round trip:  t1, t2 = ik(0.30, 0.20, 0.35, 0.25);  fk(0.30, 0.20, t1, t2)  returns (0
```

With the third link of section 2 the recipe is: subtract L_3 along the wanted orientation to find the wrist point W, solve the 2-link problem for W, and then set $\theta_3 = \varphi - \theta_1 - \theta_2$. In the earlier example, the target hand position (0.40, 0.20) with $\varphi = 0$ gives W = (0.30, 0.20), so $r^2 = 0.13$ and $\cos\theta_2 = (0.13 - 0.09 - 0.04)/0.12 = 0$, which is $\theta_2 = \pm 90°$. Elbow-down gives $\theta_1 = \text{atan2}(0.20, 0.30) - \text{atan2}(0.20, 0.30) = 0$ and $\theta_3 = 0 - 0 - 90 = -90°$; elbow-up gives $\theta_1 = 33.69 + 33.69 = 67.38°$ and $\theta_3 = 0 - 67.38 + 90 = 22.62°$, the two poses quoted above.

Two things to remember about IK in general. First, it can have **no solution** (out of reach), **one** (at the edge of the reach), **two** (as here) or, for a redundant arm, **infinitely many**; a program must handle all of these and prefer the solution that is reachable within the joint limits and nearest to the current pose, so the arm does not swing round the long way. Second, for a six-joint arm with a wrist whose three axes meet at one point (the usual design) there is still a closed-form answer, in the same spirit as the wrist point trick above; for other arms the problem is solved numerically, by repeatedly nudging the angles with the Jacobian of section 5.

4. Workspace and singularities

The **workspace** is the set of points the hand can reach. For the planar 2-link arm with unlimited joint travel it is a ring (an *annulus*): a full circle of radius $L_1 + L_2$ (the arm straight out) with a hole of radius $|L_1 - L_2|$ (the arm folded back on itself, the shorter link along the longer). The 3D workspace of a real arm is the same idea turned into a solid. The **dexterous** part of the workspace is where the hand can also take any orientation; it is smaller, because the wrist needs room to turn.

```
   Workspace of the 2-link arm, L1 = 0.30 m, L2 = 0.20 m, no joint limits:  # = reachable,

                      #
                ################
             ##########################
          ##############################
        ##################################
      ####################################
     ####################################
    ######################################
   ######################################
  ######################################
  ######################################
   ################       ################
  ################    0    ################
```

```
    ################       ################
    ######################################
    ######################################
     ####################################
      ##################################
       ################################
         ############################
           ########################
               ################
                       #

          Outer radius  L1 + L2      = 0.50 m   (arm fully STRETCHED:  t2 = 0)
          Inner radius  | L1 - L2 |  = 0.10 m   (arm fully FOLDED:     t2 = +/-180 deg)
          Real joints do not turn 360 degrees.  With the elbow limited to +/-150 deg the i
          sqrt( L1^2 + L2^2 + 2 L1 L2 cos 150 ) = sqrt( 0.09 + 0.04 - 0.1039 ) = 0.161 m,
```

Targets near the edge of the ring are expensive. Near the outer edge the arm is almost straight, and a small step of the hand outward needs a large change of elbow angle (section 5 puts numbers on this); and every extra bit of reach is paid for in torque (section 6). The cause is geometric: the forward kinematics maps two angles to two coordinates, and the map folds over on itself at the edges of the ring, where its inverse breaks down.

Singularities

A **singular pose** is one in which the arm loses the ability to move the hand in some direction, however fast the joints turn. For the 2-link arm the singular poses are exactly the two edges of the workspace: **fully stretched** ($\theta_2 = 0$) and **fully folded** ($\theta_2 = \pm 180°$). In either pose the two links lie along one line. A shoulder rotation then swings the hand in a circle at right angles to that line, an elbow rotation swings it in a circle at right angles to that line as well, and neither joint can move the hand *along* the line: it is a direction in which the hand is stuck. In the stretched pose, the hand also cannot go farther out, simply because the arm has no more length. Close to a singularity the arm is not stuck but **ill-conditioned**: it can move in the lost direction, but only by turning the joints extremely fast. Section 5 puts numbers on this.

Singularities matter in practice. A controller that asks a nearly singular arm to follow a straight line through the singular pose commands joint speeds that grow without limit, and the arm either hits its speed limit and leaves the path or jerks violently. The usual remedies: plan paths that stay away from singular poses (the workspace edges are the first ones to avoid; a six-axis

arm has others inside the workspace, such as the wrist axes lining up), limit the commanded joint speed near them, or slow the hand and accept a path error there.

5. The Jacobian: velocity and force

The forward kinematics tells where the hand is. The **Jacobian** tells how the hand *moves* when the joints move, and so it is the tool for speed and for force. It is the matrix of derivatives of the hand coordinates with respect to the joint angles. For the 2-link arm, differentiating x and y of section 2, and writing $s_1 = \sin\theta_1$, $s_{12} = \sin(\theta_1 + \theta_2)$, c_1, c_{12} likewise:

```
Velocity relation:        [ vx ]       [ -L1 s1 - L2 s12      -L2 s12 ]   [ t1_dot ]
                          [    ]   =    [                             ] x [        ]
                          [ vy ]       [  L1 c1 + L2 c12       L2 c12 ]   [ t2_dot ]

Determinant:    det J = ( -L1 s1 - L2 s12 )( L2 c12 ) - ( -L2 s12 )( L1 c1 + L2 c12 )

Singular when det J = 0:   sin t2 = 0,  that is  t2 = 0 (stretched) or t2 = 180 deg (fo

Worked example:  L1 = 0.30,  L2 = 0.20,  t1 = 40 deg,  t2 = 60 deg   (the pose of secti
       J11 = -( 0.30 x 0.6428 + 0.20 x 0.9848 ) = -( 0.19284 + 0.19696 ) = -0.38980
       J21 =    0.30 x 0.7660 + 0.20 x ( -0.1736 ) =  0.22981 - 0.03473   =  0.19508
       det J = ( -0.38980 )( -0.03473 ) - ( -0.19696 )( 0.19508 ) = 0.013538 + 0.038424

       Joint speeds  t1_dot = +0.5 rad/s,  t2_dot = -0.2 rad/s:
       vx = -0.38980 x 0.5 + ( -0.19696 )( -0.2 ) = -0.19490 + 0.03939 = -0.1555 m/s
       vy =  0.19508 x 0.5 + ( -0.03473 )( -0.2 ) =  0.09754 + 0.00695 =  0.1045 m/s
```

Near a singularity: what the determinant means

Going the other way, joint speeds for a wanted hand velocity, needs the inverse, $\dot{\theta} = J^{-1} v$, and the inverse divides by the determinant. Wherever $\sin\theta_2$ is small the joint speeds grow, as the table shows for the arm at $\theta_1 = 30°$ asked to move the hand *outward along the line from the shoulder* at 0.05 m/s:

```
L1 = 0.30,  L2 = 0.20,  t1 = 30 deg,  hand moving radially outward at 0.05 m/s

  t2        reach r    det J = L1 L2 sin t2    t1_dot (rad/s)    t2_dot (rad/s)
  90 deg    0.361 m         0.0600                +0.092            -0.300
```

```
   45 deg    0.464 m         0.0424              +0.210           -0.546
   15 deg    0.496 m         0.0155              +0.636           -1.597
    5 deg    0.4995 m        0.0052              +1.910           -4.776
    1 deg    0.500 m         0.0010              +9.549          -23.874        (abou

  Every row asks for the same slow 5 cm/s.  The elbow speed grows as 1 / det J (the produ
  Moving TANGENTIALLY (across the line from the shoulder) is not a problem at any of thes
```

Force and torque: the static relation

The same matrix also relates forces. By the principle of virtual work (the power the joints put in equals the power the hand puts out, $\tau^T \dot{\theta} = F^T v = F^T J \dot{\theta}$ for every $\dot{\theta}$), the joint torques needed to make the hand push on the world with force F (in the static case, with nothing moving) are:

```
  tau = J^T x F        ( J transposed: swap its rows and columns )        F = the force

  Same pose as above (t1 = 40 deg, t2 = 60 deg, hand at ( 0.195, 0.390 ) m).  The hand pr
        tau1 = J11 Fx + J21 Fy = ( -0.38980 )( 0 ) + ( 0.19508 )( -5 ) = -0.9754 N m
        tau2 = J12 Fx + J22 Fy = ( -0.19696 )( 0 ) + ( -0.03473 )( -5 ) = +0.1736 N m

  Check by moments (the force times its perpendicular distance, with the lever of Robotic
        about the shoulder (0, 0):          tau1 = x Fy - y Fx = 0.19508 x ( -5 ) - 0 =
        about the elbow (0.2298, 0.1928):   tau2 = ( x - Ex ) Fy - ( y - Ey ) Fx = ( 0.1

  Pressing sideways, F = ( -4, 0 ) N:    tau1 = ( -0.38980 )( -4 ) = +1.5592 N m        t
  To HOLD a weight W up, the hand must push UP on it: use F = ( 0, +W ).   The joint torq

  Notice what a SINGULAR pose does to this relation: at t2 = 0 the matrix is singular, so
  A stretched arm holding a load pulled straight along it is not working its motors: a fa
```

6. Payload and torque at the joint

The torque a joint needs is the oldest formula in the book, from Robotics Fundamentals 2 section 2: **force times the perpendicular distance**. A weight m g hanging at distance r along an arm that makes angle α above the horizontal has a perpendicular distance of r cos α, so:

```
  tau = m g r cos( alpha )          alpha = angle of the arm above (or below) the HORIZON
```

```
Worst case:  alpha = 0 (the arm horizontal):  cos = 1,  so  tau(max) = m g r.     Straig
So SIZE EVERY JOINT FOR THE HORIZONTAL POSE (or the worst pose its limits allow), even

Add up the contributions for EVERY mass that sits beyond the joint:
       a link's own weight:  m g x ( distance to its CENTRE, for a uniform link = half
       a gripper and the payload:  m g x ( distance to the tip ).
```

Each joint carries everything *beyond* it, so the joint nearest the base carries the most, and the

puts the heavy motors **at the base** (or drives distant joints through belts or rods) and keeps the outer links light: it cuts the mass at long distances. The next example is a hobby arm on a desk, worked joint by joint.

```
Arm:  L1 = 0.20 m,  L2 = 0.15 m, both horizontal (worst case).  g = 9.81 m/s^2.   Masses
       link 2 0.05 kg (centre at 0.20 + 0.075 = 0.275 m);  gripper 0.05 kg and payload (

SHOULDER torque (everything beyond the shoulder):
       link 1:           0.08 x 9.81 x 0.100 = 0.0785 N m
       elbow servo:      0.06 x 9.81 x 0.200 = 0.1177 N m
       link 2:           0.05 x 9.81 x 0.275 = 0.1349 N m
       gripper + load:   0.15 x 9.81 x 0.350 = 0.5150 N m
       total             tau(shoulder) = 0.8461 N m

ELBOW torque (only link 2, the gripper and the payload):
       link 2:           0.05 x 9.81 x 0.075 = 0.0368 N m
       gripper + load:   0.15 x 9.81 x 0.150 = 0.2207 N m
       total             tau(elbow) = 0.2575 N m

If the arm is raised 30 degrees above horizontal (both links in a line):  multiply by c
```

Now choose actuators. The data-sheet **stall torque** is the most the servo can produce at all, with the output stopped, and, as Robotics Fundamentals 3 section 2 warned, it is not a working figure: at stall the current and heat are at their maximum, the gears are at their highest stress, and the servo cannot be held there for long. A common design rule is to **derate** to about half of the stall torque for a load that must be held continuously (typical practice, not a law; more where the duty is heavy or the servo is cheap, less for brief loads), and to leave an extra margin where the arm also has to accelerate.

```
Shoulder servo:  stall torque 2.5 N m (about 25 kg cm, a typical large hobby servo).   I
       Required 0.846 N m < 1.25 N m: it passes, with a margin of 1.25 / 0.846 = 1.48 (a
       Most payload the shoulder can hold at the derated limit:  the arm alone and the g
       so  m(max) = ( 1.25 - 0.503 ) / ( 9.81 x 0.35 ) = 0.747 / 3.434 = 0.218 kg    (ag
Elbow servo:  stall torque 1.0 N m, derated to 0.50 N m.   Required 0.2575 N m:  margin
Required stall torque for the shoulder at a 50 % derating:  0.846 / 0.5 = 1.69 N m, so a

Acceleration adds torque:  tau = I x alpha.  The gripper and payload (0.15 kg) at 0.35 m
       Starting the arm at 8 rad/s^2 needs another 0.0184 x 8 = 0.147 N m at the should
```

If the joint is driven by a motor and gearbox rather than a servo, the same sizing applies to the output of the gearbox. As Robotics Fundamentals 3 section 3 showed, the output torque is $\tau_{out} = \tau_{motor} N \eta$, with gear ratio N and efficiency η; and its section 6 gave the full procedure for sizing a motor from the load, the speed and the continuous rating. A gearbox that cannot be back-driven (a worm drive, or a very high ratio) also holds the arm up when power is off, which is useful and which makes it harder to move the arm by hand. A joint that is only *just* strong enough also sags and overheats: the margin is not a luxury.

7. Joint space and Cartesian space

There are two ways to tell an arm to go from pose A to pose B. In **joint space** each joint is moved from its start angle to its end angle, usually with all joints starting and finishing together; this is simple and needs no IK along the way, so singularities do not trouble it, but the hand follows a curved, hard-to-predict path. In **Cartesian space** the hand is told to follow a chosen path in space (typically a **straight line**), and at each small step the inverse kinematics finds the joint angles; the path is what you wanted, but it costs an IK solution per step, and it fails if any point on the line is out of reach or near a singularity.

```
L1 = 0.30,  L2 = 0.20.   Move the hand from A = ( 0.30, 0.20 ) to B = ( -0.30, 0.20 ), d

       IK for A:  t1 = 0.00,   t2 = 90.00 deg          IK for B:  t1 = 112.62,  t2 = 90

JOINT SPACE:   move each joint linearly.  Halfway:  t1 = ( 0 + 112.62 ) / 2 = 56.31 deg,
       FK of the halfway pose:  x = 0.30 cos 56.31 + 0.20 cos 146.31 = 0.1664 - 0.1664 =
       The hand passes through ( 0, 0.361 ), a bulge of 0.161 m above the straight line

CARTESIAN (straight line y = 0.20):   IK at five points along the line, elbow-down:
```

```
        x (m)     t1 (deg)    t2 (deg)
        +0.30       0.00        90.00
        +0.15      11.72       124.23
         0.00      48.59       138.59       (the elbow folds to its tightest here: r = 0
        -0.15      85.46       124.23
        -0.30     112.62        90.00

        The straight line needs the elbow to swing 90 -> 138.6 -> 90 deg, while joint-sp
        the Cartesian line is needed whenever the path itself matters (welding a seam, d

   A line through the shoulder, such as ( 0.20, 0 ) to ( -0.20, 0 ), passes through r = 0,
```

The trapezoidal velocity profile

An arm cannot jump to full speed: that would need infinite acceleration, shake the arm and the payload, and exceed the torque of section 6. The standard smooth move is the **trapezoidal velocity profile** (named for the shape of its speed-against-time graph): **accelerate** at a constant a_{max} until the maximum speed v_{max} is reached, **cruise** at v_{max}, then **decelerate** at the same rate to stop exactly at the target. It applies to a joint angle (rad, rad/s, rad/s^2) or to a distance along a Cartesian line, the same formulas hold.

```
   speed
     ^
   vmax |        _____________________
        |       /                     \
        |      /                       \          Area under the curve = the distance mov
        |     /                         \
        |    /                           \
      0 +---+----------------------------+---+-------> time
        0   ta                         T - ta   T

   Ramp time:              ta = vmax / amax                      Distance covered in one ra
   Long move  ( D >= vmax^2 / amax, enough room for both ramps ):     cruise time = ( D -
   Short move ( D <  vmax^2 / amax ):  there is no cruise: the speed rises and falls as a

   Worked example 1 (long move):  a joint turns D = 90 deg = 1.5708 rad,  vmax = 2.0 rad/s
          Ramp time  ta = 2.0 / 8.0 = 0.25 s.   Ramp distance  d = 2.0^2 / ( 2 x 8.0 ) = 0
          Cruise time = 1.5708 / 2.0 - 0.25 = 0.7854 - 0.25 = 0.5354 s.      TOTAL  T = 1.

   Worked example 2 (short move):  D = 20 deg = 0.3491 rad, same limits.   vmax^2 / amax =
          ta = sqrt( 0.3491 / 8 ) = 0.2089 s,   T = 0.4178 s,   v(peak) = 8 x 0.2089 = 1.6
```

```
Several joints:  give every joint the SAME total time T = the longest time any joint ne
       which keeps a joint-space path a smooth arc rather than a zigzag.
```

```
from math import sqrt

def plan(D, vmax, amax):                        # distance D (rad or m) -> ramp time, to
    if D >= vmax * vmax / amax:                  # long move: there is a cruise
        ta = vmax / amax
        return ta, D / vmax + ta, vmax
    ta = sqrt(D / amax)                         # short move: a triangle
    return ta, 2.0 * ta, amax * ta

def position(t, D, amax, ta, T, vpk):           # where the joint (or hand) is at time t
    if t <= ta:
        return 0.5 * amax * t * t               # accelerating
    if t <= T - ta:
        return 0.5 * amax * ta * ta + vpk * (t - ta)       # cruising
    if t <= T:
        s = T - t
        return D - 0.5 * amax * s * s           # decelerating
    return D                                    # arrived

# plan(1.5708, 2.0, 8.0) gives (0.25, 1.0354, 2.0);   position( 0.25, ... ) is 0.25 rad
```

A **smoother** alternative, used where jerk (the rate of change of acceleration) must be kept small, is the S-curve profile, which rounds the corners of the trapezoid; a cubic polynomial between waypoints is another common choice. The trapezoid is the one to know first. Feedforward and the position loop that follows these profiles are the subject of Robotics Fundamentals 7 sections 7 and 11: the profile is the *reference*, the controller makes the joint follow it.

8. Precision, repeatability, backlash and deflection

An arm that computes its pose perfectly still misses the target by a few millimetres. Two quantities describe that. **Repeatability** is how closely the arm returns to the *same* taught point every time (the scatter); **accuracy** is how closely it reaches a point it is *told* to reach in space (the offset from the true coordinates). An arm can be very repeatable and not very accurate: it goes

back to the same wrong place every time, which is fine if it is taught the points by hand and a problem if the points come from a camera or a drawing. Repeatability is the cheaper to buy; typical figures are about 0.02 to 0.1 mm for industrial arms and millimetres for a hobby arm (approximate).

The sources of error are the same few, all of them magnified by the length of the arm. A joint angle error δθ moves the tip by about r δθ, where r is the distance from that joint to the tip:

```
Tip error from an angle error:   e = r x delta_theta  ( radians )      Resolution matte

BACKLASH (Robotics Fundamentals 3 section 3):  free movement of the gears when the dire
       Backlash grows with every gear stage, and it is magnified by the arm.  Worst for
       Remedies: a gearbox with low backlash (planetary or harmonic), preloaded or spli
       so the control loop sees the true joint angle (Robotics Fundamentals 7).

DEFLECTION (Robotics Fundamentals 2 section 1, the cantilever):    delta = F L^3 / ( 3
       Link 2 of the arm: an aluminium flat bar, 20 mm wide, 3 mm thick, L = 0.20 m, ca
       Mounted FLAT (the 3 mm is vertical):   I = 0.020 x 0.003^3 / 12 = 4.5e-11 m^4
       Mounted ON EDGE (the 20 mm is vertical):  I = 0.003 x 0.020^3 / 12 = 2.0e-9 m^4
       Deflection grows with the CUBE of the length: a link twice as long droops 8 time

Thermal growth, wear, a loose screw and an out-of-true link also add error.   Calibrati
```

The practical lessons: keep links short and stiff (ribs, tubes and on-edge bars); keep the moving mass low; choose joints with little play; put encoders where the angle matters; and calibrate. A hobby arm will never be an industrial arm, but understanding where the millimetres come from tells you which one is worth fixing.

9. Grippers and end effectors

The **end effector** is whatever the arm carries to do the job: a gripper, a vacuum cup, a welding torch, a screwdriver, a camera. A gripper is the most common, and its design is a force problem: it must hold the object against its weight and against the acceleration of the arm, without crushing it.

END EFFECTOR	HOW IT HOLDS	GOOD FOR	WATCH FOR
Parallel-jaw gripper	Two flat fingers move together and squeeze; the grip is friction between finger and object	Boxes, blocks and parts of known size; strong, simple, one motor	Needs a flat or well-fitted contact; an object of unknown shape can slip or be crushed; jaw force must be sized (below)
Vacuum (suction cup)	Air pumped out of a soft cup; atmospheric pressure presses the object on the cup: force = pressure difference × cup area	Flat, smooth, non-porous things: sheets, trays, boxes, glass; touches from one side only	Rough, wet or porous surfaces leak; needs a pump and an air supply; a power cut drops the load
Soft / compliant gripper	Flexible fingers (silicone, or a fin-ray shape) that wrap round the object; or a membrane that conforms and then stiffens	Fragile or irregular objects (fruit, eggs, odd shapes); forgiving of positioning error	Lower grip force and a less precise position; the material wears; hard to model, so the force is found by testing

Grip force

With friction grippers the object is held up by friction at the finger pads. Each finger pushes with a normal force N; the friction it can supply is at most μ N (the friction coefficient μ between pad and object, as in the traction formula of Robotics Fundamentals 3 section 5). With n fingers in contact, the total friction must be at least the weight, and, because the arm accelerates, the weight is *increased* by an upward acceleration a. A safety factor covers uncertain friction, shocks and wear:

```
Grip condition:   mu x N x n_fingers   >=   m ( g + a ) x SF          so          N  >=

a = the largest acceleration of the gripper IN THE DIRECTION OPPOSING the grip (upward
SF = safety factor, typically 2 for a known object and surface, higher for unknown ones

Worked example:  a 0.5 kg object, held by 2 rubber-faced fingers (mu = 0.5, a typical v
       Load to hold:  m ( g + a ) = 0.5 x ( 9.81 + 2.0 ) = 5.905 N.   With the safety f
       N = 11.81 / ( 0.5 x 2 ) = 11.81 N per finger.      So each pad must press with a
       Is the actuator enough?  A servo of 0.30 N m stall, derated to 0.15 N m, driving
       The fix is cheap if found early:  larger pads (higher mu), a stronger servo, a l
Too much force is also a fault:  the object must not be crushed.  Compare N with what t

Vacuum:  F = delta_P x A.   A cup of diameter 30 mm ( A = pi x 0.015^2 = 7.07e-4 m^2 )
       With the same acceleration and SF = 2:  the largest mass is  28.3 / ( 2 x ( 9.81
```

10. Arm safety

An arm is a machine that can hurt people, and a small one can still pinch a finger. The energy is the thing to respect: the kinetic energy of the hand and payload is ½ m v^2, which is 0.5 J for 1 kg at 1 m/s, and a joint torque from section 6, divided by the distance from the joint to the point of contact, is the force that point can exert on a trapped finger. Keep these habits.

- **Respect the payload limit.** Work out the torque (section 6) for the worst pose with the heaviest object, including its acceleration; stay inside the derated limit; and write the rated payload on the arm. An overloaded arm sags, overheats, strips its gears, and drops what it holds.
- **Find the pinch points and the crush points.** Wherever two links close on each other (the inside of an elbow as it folds, the gap between a link and the base, a gripper's jaws), a finger can be caught. Guard them, use covers, limit the joint range with hard stops that keep a gap, and keep hands out of the workspace while power is on.
- **Limit the speed and the force.** The lowest speed that does the job is the right one, and a force or current limit on every joint caps what a collision can do. Use the stopping-distance arithmetic of Robotics Fundamentals 8 section 7 (reaction time plus braking) for the arm's own coasting distance.
- **Hold the load when the power goes.** A joint under gravity falls when its motor is off unless something holds it: a brake, a non-back-drivable gearbox (Robotics Fundamentals 3 section 3), or a counterbalance. Decide what the safe state is for each joint before the first test (Robotics Fundamentals 8 section 7), and fit a hardware emergency stop that cuts power whatever the software is doing.
- **Test slowly, with clearance.** First move each joint alone with the payload removed and the speed capped, then the full path, then the load; keep a hand on the stop switch and everyone outside the reach (the outer radius $L_1 + L_2$ plus the length of the gripper and its load).

Collaborative arms are designed to work next to people without a fence. From memory, the standards (ISO 10218 for industrial robots and ISO/TS 15066 for collaborative operation) name four ways to do this, and they are named here only so you recognise them: a **safety-rated monitored stop** (the arm halts when a person enters and resumes when they leave), **hand guiding** (the person moves the arm by hand, with enabling controls), **speed and separation monitoring** (the arm slows as a person comes nearer and stops at a minimum distance) and **power and force limiting** (the arm is built so that contact forces and pressures stay below

published limits for each body region, using low moving masses, rounded covers, torque sensing and limited speed). Power and force limiting is the reason collaborative arms are slow and light. Applying any of this to a real workplace needs the current standards and a proper risk assessment, which this course cannot replace. Robotics Fundamentals 10 (planned) covers power and safety for the robot as a whole: fusing and wiring, emergency stops, safe testing and the ethics of building machines that act in the world.

11. Challenges

CHALLENGE 1: FORWARD AND INVERSE KINEMATICS

A planar arm has $L_1 = 0.25$ m and $L_2 = 0.15$ m. (a) Find the hand position for $\theta_1 = 50°$ and $\theta_2 = -70°$, and say which kind of elbow pose that is. (b) State the reachable range of distance from the shoulder, and say whether each target is reachable: (0.30, 0.20), (0.45, 0.05) and (0.06, 0.04). (c) For the target (0.30, 0.20), find both inverse kinematics solutions (θ_1, θ_2) and verify each by forward kinematics. (d) A third link $L_3 = 0.08$ m is added at the wrist. The hand must reach (0.32, 0.10) pointing at $\varphi = -30°$. Find the wrist point, then θ_1, θ_2 and θ_3 for the elbow-down solution, and verify by forward kinematics.

View solution

CHALLENGE 2: JACOBIAN, SINGULARITIES AND STATIC TORQUE

The arm has $L_1 = 0.25$ m and $L_2 = 0.15$ m, at $\theta_1 = 25°$ and $\theta_2 = 70°$. (a) Find the hand position and the elbow position. (b) Write the Jacobian numerically and find its determinant two ways (from the matrix, and from $L_1 L_2 \sin\theta_2$). (c) For joint speeds $\dot{\theta}_1 = 0.4$ rad/s and $\dot{\theta}_2 = 0.6$ rad/s, find the hand velocity and speed. (d) The hand pushes on a surface with $F = (-2, -6)$ N. Find the joint torques from $\tau = J^T F$ and check both by moments about the shoulder and the elbow. (e) The hand must move straight outward from the shoulder at 0.04 m/s. Find the joint speeds when $\theta_2 = 70°$ and when $\theta_2 = 8°$, and explain the difference. (f) Name the two singular elbow angles and say what the arm looks like in each.

View solution

CHALLENGE 3: PAYLOAD, MOTION, DEFLECTION AND GRIP

(a) A single-joint servo arm: servo stall torque 1.8 N m, derated to 50 %, arm length 0.22 m, arm mass 0.06 kg (weight at its middle), gripper 0.04 kg at the tip. Find the torque of the arm and gripper alone, and the largest payload at the tip that the derated servo can hold with the arm horizontal. (b) With a

0.30 kg payload, find the torque with the arm horizontal and with the arm 40° above horizontal, and say which pose sizes the servo. (c) A gearmotor has a continuous torque of 0.12 N m, a 20:1 gearbox and an efficiency of 0.85: find its output torque and compare with part (a)'s need for 0.30 kg. (d) A joint turns 120° with v_{max} = 1.5 rad/s and a_{max} = 6 rad/s^2: find the ramp time, cruise time and total time. Then do the same for a 15° move. (e) A 0.8 kg object is held by 2 fingers with μ = 0.4, upward acceleration 3 m/s^2 and SF = 2: find the normal force per finger. A 25 mm vacuum cup at 50 kPa lifts the same object under the same acceleration: find the force and the largest mass at SF = 2. (f) A flat aluminium bar 25 mm wide and 3 mm thick, 0.25 m long, carries 0.30 kg at its end (E = 69 GPa): find the deflection when the 3 mm is vertical, and say what turning it on edge does. Then find the tip error from 0.8° of backlash at a reach of 0.40 m.

View solution

QUICK REFERENCE

- **Joints and links:** revolute (angle θ) and prismatic (length d); one motor per DOF; a six-joint arm reaches any position and orientation within its reach. Types: Cartesian (simple, stiff), SCARA (fast, planar), articulated 6-axis (general), delta (fast, light), hobby servo arm (cheap, backlash).
- **Forward kinematics (2 links):** $x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2)$, y likewise with sin. Third link: add L_3 at angle $\varphi = \theta_1 + \theta_2 + \theta_3$. Denavit-Hartenberg: four numbers per joint (a, α, d, θ), a bookkeeping convention (two variants).
- **Inverse kinematics:** reachable only if $|L_1 - L_2| \le r \le L_1 + L_2$; $\cos\theta_2 = (r^2 - L_1^2 - L_2^2)/(2L_1L_2)$, $\theta_2 = \pm\text{acos}$ (elbow-down +, elbow-up −); $\theta_1 = \text{atan2}(y, x) - \text{atan2}(L_2 \sin\theta_2, L_1 + L_2 \cos\theta_2)$. With a wrist: find W = hand − $L_3(\cos\varphi, \sin\varphi)$ first. Always verify by forward kinematics.
- **Workspace:** a ring between $|L_1 - L_2|$ and $L_1 + L_2$. **Singularities** at the edges (stretched $\theta_2 = 0$, folded $\theta_2 = 180°$): a direction of motion is lost, and joint speeds near them grow like $1/\sin\theta_2$.
- **Jacobian:** $v = J\dot{\theta}$; $J = [[-L_1s_1 - L_2s_{12}, -L_2s_{12}], [L_1c_1 + L_2c_{12}, L_2c_{12}]]$; $\det J = L_1L_2 \sin\theta_2$; $\dot{\theta} = J^{-1} v$; static joint torques $\tau = J^T F$ for the force the hand exerts.
- **Joint torque:** $\tau = m g r \cos\alpha$, worst at horizontal ($\alpha = 0$); sum every mass beyond the joint, each link at its centre; the base joint carries the most. Derate a servo to about 50 % of stall; $\tau_{out} = \tau_{motor} N \eta$ for geared motors; add $I\alpha$ for acceleration.
- **Motion:** joint space is quick and curved; Cartesian lines are straight but need IK at every step and a reachability check along the line. Trapezoid: $t_a = v_{max}/a_{max}$; long move $T = D/v_{max} + t_a$ (if $D \ge v_{max}^2/a_{max}$); short move $T = 2\sqrt{(D/a_{max})}$; give all joints the same T.
- **Precision:** repeatability (scatter) vs accuracy (offset); tip error = $r\,\delta\theta$; backlash and resolution are magnified by reach; deflection = $FL^3/3EI$ (cube of length; on edge is $(h/b)^2$ stiffer); calibrate; encoders

on the link side.

- **Grippers:** parallel-jaw (friction), vacuum (F = ΔP A), soft (compliant). Grip condition μ N n ≥ m(g + a) SF, so N = m(g + a) SF/(μ n); check the actuator can supply N without crushing the object.
- **Safety:** size for the payload limit; find pinch points; limit speed and force; hold the load when power fails (brake, non-back-drivable gearbox); a hardware stop; test slowly with clearance. Collaborative modes (from memory): monitored stop, hand guiding, speed and separation monitoring, power and force limiting.

Next, Robotics Fundamentals 10 closes the course with the part that makes a robot real and safe to run: power, safety and building your first robot. Battery sizing and runtime, wiring and fusing, emergency stops and safe testing, a short look at ethics, and a complete build plan with a parts list, which pulls together the mechanics, motors, sensors, control and behaviour of the nine chapters before it, and points on to Programming for Robotics.

Robotics Fundamentals

Course 1 · Chapter 10 · Power, Safety and Building Your First Robot

Nine chapters have designed the parts of a robot one at a time: the frame (Robotics Fundamentals 2), the motors and wheels (Robotics Fundamentals 3), the drive and its kinematics (Robotics Fundamentals 4), the sensors (Robotics Fundamentals 5), position and feedback (Robotics Fundamentals 6), control (Robotics Fundamentals 7), behaviour (Robotics Fundamentals 8) and arms (Robotics Fundamentals 9). None of it runs without a **power system**, and none of it should be switched on near people without **safety** built in. This final chapter does both, then brings everything together. It covers the batteries (chemistry, capacity, voltage sag, runtime), the power budget and the choice of regulator, the low-voltage cutoff and how to charge and store a LiPo pack without starting a fire, the wiring (wire gauge, voltage drop, connectors, fuses, a main switch, a common ground, a star layout), the emergency stop and the habits of safe testing, and a short look at the ethics of building machines that act in the world. Then comes **the build**: a complete plan for a small two-wheel line-following, obstacle-avoiding robot, with a specification, a parts list, design calculations that reuse every earlier chapter, a block diagram, a wiring table, an assembly order, a test plan with expected readings, and a troubleshooting table. The chapter ends with a recap of the whole course and the path on to Programming for Robotics.

> **NOTHING HERE WAS BUILT OR TESTED BY THE AUTHOR**
>
> This is a **plan**, worked out on paper. The author has not built this robot, measured its current, or run its code on a board. Every number below is either a calculation you can repeat (and that was recomputed with a script) or a typical or approximate value for a part (marked as such) taken from memory or from Basic Electronics 9 and Basic Electronics 11. The pin numbers are an *example* for one board, to be checked against your own board's pinout. Where your measurements disagree with a figure here, **trust your meter and the data sheet of the part in your hand**, and treat the disagreement as the interesting result. Prices are approximate pounds sterling and change often.

The same caution applies to safety. This chapter teaches habits and arithmetic for hobby-scale robots (batteries of a few watt-hours, motors of a few watts). It is not a safety certification and does not replace the manufacturer's instructions for a charger, a battery or a motor, or the law for anything you sell or use at work. Where lithium batteries are concerned, the manufacturer's instructions win over this chapter.

1. Batteries: the chemistries

A robot's battery decides how long it works, how fast it can start, how much it weighs and how dangerous it is. Basic Electronics 9 section 8 introduced the four numbers that describe any battery (nominal voltage, capacity, maximum discharge current expressed as a C-rate, and internal resistance) and the main cell types. Here is the comparison again for robot use, with lead-acid added because larger robots use it.

CHEMISTRY	CELL VOLTAGE (NOMINAL; FULL TO EMPTY)	ENERGY PER KG (APPROXIMATE)	INTERNAL RESISTANCE (TYPICAL)	GOOD FOR	WATCH FOR
NiMH (AA, rechargeable)	1.2 V; 1.4 to 1.0 V	60 to 120 Wh/kg	20 to 50 mΩ per cell	A first robot: safe, forgiving, replaceable AA cells, simple chargers	Voltage falls slowly with charge, so the gauge is hard to read; ordinary cells lose charge in weeks, low-self-discharge types hold it far longer; heavy for the energy
Li-ion (18650 and similar)	3.6 or 3.7 V; 4.2 to 3.0 V	150 to 250 Wh/kg	20 to 60 mΩ per cell	Long runtime in a small space; good cells deliver 10 A or more	Needs a protection circuit and a proper lithium charger; a short or a puncture can start a fire; buy genuine cells from a known seller
LiPo (pouch pack)	3.7 V; 4.2 to 3.0 V; 2S 7.4 V, 3S 11.1 V	150 to 200 Wh/kg	5 to 30 mΩ per cell	Light packs with very high C-rates: fast robots, drones, anything needing strong bursts	The most dangerous in everyday use: swelling, fire after a puncture, a short or a deep discharge; needs a balance charger

CHEMISTRY	CELL VOLTAGE (NOMINAL; FULL TO EMPTY)	ENERGY PER KG (APPROXIMATE)	INTERNAL RESISTANCE (TYPICAL)	GOOD FOR	WATCH FOR
					and care in storage (section 5)
LiFePO4	3.2 V; 3.65 to 2.5 V; 4 cells 12.8 V	90 to 160 Wh/kg	Low	A safer lithium chemistry with a very long cycle life; good for robots that are charged daily	Heavier than LiPo; the very flat discharge curve means voltage says little about the charge left; needs its own charger profile
Lead-acid (sealed, SLA)	2.0 V; about 2.1 to 1.75 V; 6 cells 12 V	30 to 40 Wh/kg	Low in a large battery, rising with age	Big, heavy robots (ride-on, lawn, workshop carts): cheap, rugged, huge surge current, forgiving charger	Very heavy for the energy; harms itself if left discharged; must stay upright; vents hydrogen if overcharged

The energy-per-kg figures are typical orders of magnitude from memory and vary a lot between products; the pattern is what matters. **Weight is the robot's enemy**: every gram of battery has to be accelerated, carried up slopes and braked (Robotics Fundamentals 3 section 6), so a small mobile robot usually wants lithium, while a large slow one can afford lead-acid, and a beginner who wants forgiveness takes NiMH. The build in section 9 uses a 2S LiPo pack to show the careful handling it needs, and notes a NiMH alternative.

2. Sizing a battery: capacity, C-rate, sag and runtime

Four calculations size a battery, and they are the ones from Basic Electronics 9 and Robotics Fundamentals 1 section 5 written in one place. **Energy** is voltage times capacity. **Runtime** is usable energy divided by average power. The **C-rate** checks that the pack can supply the peak current. The **voltage sag** checks what the robot sees at the moment of a start or a stall, which is

what resets the board. A simple derating then covers the real-world losses without needing the Peukert exponent (a refinement for lead-acid that this chapter leaves out).

```
Energy:            E = V x C                          E in Wh, V in volts, C in Ah
Usable energy:     E(usable) = 0.8 x E                (the 0.8 of Robotics Fundamentals
Runtime:           t = E(usable) / P = 0.8 V C / P    P = the AVERAGE power drawn from
C-rate:            I(max) = (C-rate) x C              a "20C" 1.0 Ah pack can give 20 x
Voltage sag:       V(terminal) = V(open) - I x R(internal)          (the pack voltage
Simple derating:   t = 0.8 x k(cold) x k(age) x V C / P     one plain multiplier per ef

Worked example 1, sag:  a 2S LiPo pack of about 60 mohm (Basic Electronics 9 section 8)
       V(drop) = 5.03 x 0.060 = 0.30 V:   a pack that rests at 8.40 V (full) gives abou
       Six AA NiMH cells of about 40 mohm each (0.24 ohm in all) at the same current:
       (the brown-out of Basic Electronics 9 section 10), so check the sag at the START

Worked example 2, runtime:  the 1000 mAh 2S pack of the build at the budget of section
       E(usable) = 0.8 x 7.4 x 1.0 = 5.92 Wh          t = 5.92 / 2.58 = 2.29 h   = 137.
       Cold and aged:  k(cold) = 0.8, k(age) = 0.9, so 0.8 x 0.9 = 0.72:   E = 5.92 x 0
       Pessimistic: the robot at its peak working power the whole time, 4.18 W:   t = 5

Worked example 3, C-rate:  the 1000 mAh pack must supply a worst-case (stall) current o
```

Two more points about runtime. First, a pack that is quoted at a high C-rate is not the same as a pack that is *good at* that rate: look at the internal resistance and test the sag. Second, the capacity on the label is measured at a gentle current at room temperature; a heavy load, a cold day and an old pack each take a share, which is why the simple multipliers above exist. Measure the real runtime once the robot works, and keep that figure rather than the calculated one.

3. The power budget: average against peak

A power budget lists every load, with an **average** current (which sets the runtime) and a **peak** current (which sets the size of the regulator, the wire, the fuse and the pack's C-rate). Basic Electronics 11 section 2 built one for a phone-driven robot; here is the full budget for the robot of section 9. The motors use the sizing recipe of Robotics Fundamentals 3 section 6 (a 0.8 kg robot, 0.3 m/s cruise, the typical 6 V gearmotor of that chapter: 2.0 Ω, k = 0.006, I_0 = 0.15 A, 50:1 gearbox of efficiency 0.729). The currents of the sensors and the board are typical figures from memory; replace them with the data sheets of your parts.

```
Power budget of the build:  2S LiPo (7.4 V nominal); 5 V logic rail from a 90 % buck co

   A. The 5 V logic rail                                  average   peak working
   Raspberry Pi Pico (on VSYS)                             30 mA          50 mA
   5-channel IR line module (5 x 20 mA, LEDs always on)   100 mA         100 mA
   HC-SR04 ultrasonic sensor                               15 mA          20 mA
   2 x encoders (10 mA each)                               20 mA          20 mA
   Driver logic and status LEDs                             5 mA          25 mA
   Total on the 5 V rail                                  170 mA         215 mA
   Drawn from the pack (5 V x I / 0.90 / 7.4 V)          0.128 A        0.161 A

   B. The two motors, from the pack                                 each motor     du
   Flat cruise at 0.3 m/s (no margin)                                  0.167 A     42
   Working average (assumed 1.5 x cruise)
   Worst working: 10 deg slope, a = 0.8 m/s^2, margin 1.5              0.399 A     48
   Stall: rotor held, duty capped at 81 % (6 V of 7.4 V)               3.00 A      81
   Driver, wire and connector losses (5 % of motor current, assumed)

   C. Totals at the pack                          average    peak working   fault (
   Logic (A)                                     0.128 A         0.161 A         0
   Motors (B)                                    0.211 A         0.385 A         4
   Losses (B)                                    0.011 A         0.019 A
   Total current (sum of the unrounded values)   0.349 A         0.565 A         5
   Power at 7.4 V                                 2.58 W          4.18 W

How the motor rows were found (the recipe of Robotics Fundamentals 3 section 6, m = 0.8
       Cruise:  F = m Crr g = 0.8 x 0.03 x 9.81 = 0.235 N;   tau(wheel) = 0.235 x 0.032
                wheel speed 0.3 / 0.0325 = 9.23 rad/s, motor 50 x 9.23 = 461.5 rad/s;
       Worst:   F = 0.8 x (0.8 + 9.81 sin 10 + 0.03 x 9.81 cos 10) = 0.8 x 2.793 = 2.23
                V(need) = 2.769 + 0.399 x 2.0 = 3.57 V;   duty 48 %;   pack current per
       Stall:   the driver applies at most 6 V (duty capped at 6 / 7.4 = 81 %), so each
       Runtime from the average: 2.58 W gives 137.6 min on the 1000 mAh pack (section 2
```

THE MOTORS ARE 5 AMPS IN A FAULT AND A QUARTER OF AN AMP AT WORK

Look at the three columns of table C. The *average* is 0.35 A, so a small pack lasts hours. The *peak working* figure, 0.57 A, sets the fuse. The *stall* figure, 5.03 A, is more than ten times the working peak: it is what the wire, the switch, the connectors and the driver must survive (briefly) when a wheel is blocked, and what the fuse and the software must cut. Budget every figure, not only the average.

4. Regulators: buck against linear

The motors run straight from the pack through the driver, but the board and the sensors need a steady 5 V (or 3.3 V). Basic Electronics 9 section 9 compared the two ways of making it. A **linear regulator** drops the surplus voltage as heat: it passes the load current unchanged, so the pack supplies the same current as the load and the wasted power is $(V_{in} - V_{out})$ I. A **buck (switching) converter** converts power, not current: it draws $P_{in} = P_{out} / \eta$ with a typical efficiency η of 85 to 95 %, so from a higher voltage it draws proportionally less current. The price of the buck is switching noise and a slightly bigger part.

```
Linear:   I(in) = I(out);   P(lost) = ( V(in) - V(out) ) x I(out);   efficiency = V(out
Buck:     P(in) = P(out) / eta (typical 0.85 to 0.95);   I(in) = P(in) / V(in).

The build: the 5 V rail at 170 mA (average) from the 7.4 V pack.
       Linear:  P(pack) = 7.4 x 0.170 = 1.258 W;   heat in the regulator = ( 7.4 - 5.0
       Buck 90 %:  P(pack) = 5.0 x 0.170 / 0.90 = 0.944 W;   I(pack) = 0.944 / 7.4 = 0.
       Saving: 1.258 - 0.944 = 0.314 W, which is 12 % of the robot's 2.58 W average.
       Runtime with the linear regulator: the pack current average is 0.170 + 0.211 + 0

It matters more with bigger rail currents and bigger packs:  a 1.0 A rail from an 11.1 V
```

The rule of thumb that follows: use a **buck converter** for any rail that carries more than about 100 to 200 mA from a pack well above the rail voltage; use a **linear (LDO) regulator** where the current is tiny, where the input is only a little above the output, or where a very quiet supply matters (a sensitive analogue sensor). A common good design is a buck to a rail a little above the target, then a small LDO for the quiet part. Keep the buck's switching noise away from the line and distance sensors (shorter wires, a ground plane or a clean star, and a filter capacitor), and choose a converter rated for the *peak* rail current with margin: 215 mA is 11 % of a 2 A module.

5. Low-voltage cutoff, charging and storing a LiPo pack

The low-voltage cutoff

A lithium cell that is run flat is damaged (Basic Electronics 9 section 8: never below 3.0 V per cell, and 3.3 V per cell *under load* is a sensible cutoff). The robot must watch its own battery and stop

before that. The usual circuit is the voltage divider of Basic Electronics 10 section 4 feeding an analogue input, and the usual logic is a state of the behaviour machine (Robotics Fundamentals 8 section 2): a **warning** threshold, a **cutoff** threshold, and a debounce so that the voltage dip of one start does not trigger it. Read the pack at rest or smooth the reading (Robotics Fundamentals 5 section 9), because the reading under load sags by I R.

```
Divider for the 2S pack:  R1 = 56 kohm (top), R2 = 33 kohm (bottom) into a 3.3 V, 12-bi
       The divider current at 8.4 V is 8.4 / 89000 = 94 uA, negligible next to the pack

   Pack voltage     Per cell     At the pin     ADC count     Action
   8.40 V           4.20 V       3.115 V        3865         full, at rest
   7.40 V           3.70 V       2.744 V        3405         nominal
   7.00 V           3.50 V       2.596 V        3221         WARN: flash the LED; fini
   6.60 V           3.30 V       2.447 V        3037         CUTOFF: motors off, latch
   6.00 V           3.00 V       2.225 V        2761         the absolute floor: damag

The reading is a filtered average (say the mean of 16 samples), the cutoff needs 2 s of
a robot that restarts as the load drops and the voltage recovers will run the pack flat
```

Charging and storing a LiPo pack safely

Charging is the most dangerous thing most hobbyists do with a battery. A lithium cell charged beyond 4.20 V, charged after a deep discharge, charged while damaged, or charged without balancing can overheat, swell and burn, and a LiPo fire is hot and hard to put out. These rules are not optional.

- **Never leave a charging pack unattended.** Stay in the room, within sight, and stop at the first sign of swelling, a hot smell, hissing or heat. Not overnight, not while you are out, not "just for a minute" in another room.
- **Use a proper lithium charger with the right cell count** (a balance charger for a LiPo pack with a balance connector), set to the right chemistry and cell count, at a current of no more than **1C** (1.0 A for the 1000 mAh pack; gentler is safer). Do not charge with a bench supply set to "about right" and do not use a charger meant for another chemistry.
- **Charge on a non-flammable surface** (a ceramic tile, a metal tray or a fire-resistant LiPo bag), away from paper, curtains, sofas and beds, with a clear space around it, and keep a suitable extinguisher or a bucket of dry sand within reach and a clear way out (ask your local fire service which extinguisher suits lithium battery fires).

- **Never charge a damaged, swollen, punctured, crushed or very old pack,** or one that was deeply discharged (below about 3.0 V per cell) and has sat that way. Retire it: discharge it slowly with a bulb, and take it to a battery recycling point, not the household bin.
- **Let a warm pack cool** before charging, and let a freshly charged pack cool before use; never charge a pack that is hot to the touch.
- **Store at about 3.8 V per cell** (7.6 V for a 2S pack) if it will not be used for more than a week, in a fire-resistant bag or box, in a cool place, not in the robot. Many balance chargers have a "storage" mode that does this. A pack stored full or empty ages quickly.
- **Transport and handling.** Protect the leads from shorting (cap the connector or tape it), do not carry loose packs with keys or coins, and do not leave one in a hot car or in direct sun.

If a LiPo pack starts to swell, hiss or smoke: stop charging if you can do so safely, move away, and do not try to carry it through the house; follow the guidance of the battery's maker and the fire service. The robot of section 9 runs from a pack that is small (7.4 Wh), which limits the energy but not the need to treat it with respect; the NiMH alternative in the parts list is the gentler choice for a first build.

6. Wiring and fusing

Wire gauge: heating and voltage drop

Wire is chosen for two things: the heat it can carry (its current rating, set by the insulation and the length of the run) and the voltage lost over its length (Basic Electronics 9 section 12). The **round-trip length** counts, because the current flows out and back. Voltage drop is $V = I\,R_{wire}$ and the power lost as heat is $P = I^2\,R_{wire}$.

```
Wire resistance (typical, copper, from Basic Electronics 9 section 12):  22 AWG 53.0 mo

Worked example: the motor supply of the build, 0.3 m from the star point to the driver

   Wire (AWG)                          R for 0.6 m (out and back)    V drop at 4.865 A
   22                                                  0.0318 ohm              0.155 V
   20                                                  0.0200 ohm              0.097 V
   18                                                  0.0125 ohm              0.061 V
   26 jumpers + 2 contacts of 75 mohm                   0.230 ohm               1.12 V

At the 0.565 A working peak the drop in 22 AWG is only 0.565 x 0.0318 = 0.018 V: the st
```

```
Choice for the build: 20 AWG silicone-insulated wire for the pack, the switch, the e-st
The jumper row is the lesson: motor current through a breadboard or a thin jumper loses
```

Connectors, the switch and the ground

- **Connectors.** Choose a connector rated above the current it carries, and keyed so it cannot go in backwards. Typical maker's ratings (approximate, from memory; always read the data sheet): Dupont and breadboard contacts about 1 A (signals only); JST-XH about 3 A (balance leads, small loads); XT30 about 30 A; XT60 about 60 A. The build uses an XT30 for the pack and screw terminals or solder for the motor supply. Crimp and solder properly: a poor crimp has a high resistance, heats up, and is a classic cause of "it only fails when the motors work hard".

the pack voltage *in DC* (a DC rating is lower than the AC rating printed on many rocker switches: a switch marked 6 A at 250 V AC may be rated only a fraction of that at DC). A 5 A DC-rated switch is the minimum for the 5.03 A stall of the build.

- **A common ground.** Every part that exchanges a signal must share a ground, or the signal has no reference. But the *way* the grounds are joined matters (below). The motor current's return path must not share a thin wire with a sensor's reference.

Fuses

A fuse protects the **wire and the battery** from a short or an overload; it does not protect the load. Two limits fix its size: it must be **above the normal current** (so that it never blows during ordinary work) and **below the rating of the thinnest wire it protects** (so that it blows before the wire overheats). The usual rule is to choose the nearest standard value at **125 to 150 % of the normal (largest sustained) current**, then check it against the wire rating and against the start-up and stall transients. Place it in the positive lead *as close to the battery as possible*, so that only a few centimetres of wire are unprotected; use a fuse with a voltage rating at or above the pack voltage; and, for a lithium pack, a fast-acting or a slow-blow type chosen to tolerate the short start-up peaks (Basic Electronics 9 section 12 advised a fast-blow type close to the positive terminal, and warned that a resettable polyfuse is not enough on its own against a strong lithium pack).

```
Fuse sizing for the build:   normal (largest sustained, peak working) current from tabl
       125 % to 150 %:   1.25 x 0.5654 = 0.707 A     to     1.50 x 0.5654 = 0.848 A
       Standard values near that:  0.75 A (sold, but less common) or 1 A.   Choose 1 A:
```

```
         Check against the wire:  20 AWG is rated 11 A, 22 AWG 7 A:   1 A is far below bo
         Check against transients:  at stall the pack supplies 5.03 A, five times the fus
                  by design (it protects the driver, whose rating is about 1.2 A continuo
                  blows during ordinary driving means the robot, not the fuse, needs atte

   A second example (a different robot):  motors and logic draw 2.4 A at their largest sus
```

Star distribution, strain relief and separate supplies

The motors switch large, noisy currents, and a board shares the same pack. If the motor current and the board's current flow through the same stretch of wire, the motor's voltage drops and spikes show up on the board's supply and on its ground, and the board resets or reads nonsense (the brown-out and grounding of Basic Electronics 9 sections 10 and 11). The cure is a **star**: every branch has its own wires from one central point, so the branches do not share resistance.

```
   Star power distribution (positive side above, ground side below; each branch has its ow

     battery +  --[ fuse 1 A ]--[ main switch ]--- STAR (+) ----+-----------------[ E-STOP
                                                                |
                                                                +-------------------------
                                                                |
                                                                +--[ 56 kohm ]--+--[ 33 ko
                                                                                |
     battery -  ---------------------------------------- STAR (-) -----------+----------

     Rules:   the motor driver's ground wire is the thickest, and goes straight to the sta
              The bulk capacitor sits at the driver's VM pin and ground pin (Basic Electro
              The buck's input and output each get a capacitor close to the module, and it
              The e-stop opens only the motor branch: the board stays alive to log and to
```

Strain relief means that a pull on a wire is taken by something other than the joint. A solder joint or a screw terminal fails when it is flexed or pulled, and the failure is intermittent and hard to find. Clamp each wire bundle to the frame with a tie or a clip a few centimetres from each connector so that the wire, not the joint, takes the strain; leave a small loop of slack; use stranded wire (not solid) anywhere that moves; and keep wires away from wheels, gears and moving joints. A tidy loom is also a safety feature: loose wires catch on wheels and short against each other.

For the logic supply, **separate the motor and logic supplies** whenever the motors are large enough to disturb the pack: a dedicated buck to the logic, the motors fed straight from the star, bulk capacitors at the driver (Basic Electronics 9 section 5 on servos made the same point with a separate servo supply, and Basic Electronics 10 section 8 on driving a motor from a board safely). In a small robot like the build, one pack feeds both through a star, which works; on a larger robot, or one with servos or a gripper (Robotics Fundamentals 9), give the motors their own pack or converter and join only the grounds.

7. E-stops, safe testing and fail-safe design

A robot can fail in ways the author did not imagine: a program freezes, a sensor reads nonsense, a wire comes loose, the floor is slippery, a person steps into the path. **Safety** is the set of habits and hardware that make the failure harmless. The central question is not “will it fail?” but “when it does, what does it do?”

Hardware kill switch against software stop

STOP	WHAT IT DOES	WORKS WHEN THE PROGRAM HAS CRASHED?	USE IT FOR
Hardware e-stop (a normally-closed button in the motor supply)	Opens the motor power circuit directly; no software in the path	Yes: a frozen, crashed or misbehaving program cannot override it	The emergency stop that anyone can hit; the last line of defence
Watchdog timer	A hardware countdown that resets the board (and so drops the motor outputs) if the program stops kicking it	Yes, within its timeout (Robotics Fundamentals 8 section 7: 5 times the loop period, 100 ms for a 20 ms loop)	Catching a frozen program without a person
Software stop (the STOP and LOW_BATT states)	The behaviour machine commands zero speed	No: needs a running program and working sensors	Normal, controlled stops: a blocked path, a lost line, a low battery
Driver or fuse limit	The driver's own thermal and current limits; the fuse opens on overcurrent	Yes	Protecting parts and wire from a stall or a short

The lesson of the table is that the **emergency stop must not depend on the thing that might be broken**. A software button that tells the program to stop is not an emergency stop; a mushroom button that opens the motor supply is. Use all four layers: the software handles ordinary stops, the watchdog catches a frozen program, the driver and fuse limit a stall, and the hardware e-stop handles everything else.

Fail-safe design principles

- **Fail to the safe state.** Decide what "safe" means for each failure (Robotics Fundamentals 8 section 7): for a small wheeled robot it is motors off. For an arm holding a load, or a robot on a slope, it may be "held", not "released" (Robotics Fundamentals 9 section 10: a brake or a non-back-drivable gearbox).
- **The safe state is what the hardware does when power or signal is lost, not only what software chooses.** A normally-closed e-stop opens when pressed *or* when its wire breaks. Pull-down resistors (10 kΩ) on the driver's standby and PWM pins keep the motors off while the board is starting up or has reset, when its pins float.
- **Stop on silence.** Treat the absence of a good signal as a fault: a distance sensor that reports nothing, an encoder that stops counting while the motors are driven (a stall or a broken wire), a command link that goes quiet. A stall check, "duty above 30 % and the wheel speed under 10 % of the wanted speed for 0.3 s: cut the motors", catches a jammed wheel long before the fuse does.
- **Limit everything.** Cap the duty, the speed, the acceleration and the current so that no single bug can command the maximum. A lower limit costs a little performance and removes a whole class of accidents.
- **Make the safe action the easy one.** The e-stop is big, red, reachable without leaning over the robot, and needs a deliberate action (the start button) to run again: releasing the stop must not restart the motors by itself.
- **Test the stops.** A stop that has never been tested may not work. Press the e-stop, freeze the program, pull a sensor lead and flatten the battery on the bench, on purpose, before you trust them (the tests of section 9).

Safe testing, in order

1. **Power off: look first.** Check the polarity, the connectors, the fuse and for any loose strand or bare metal (continuity and short checks, section 9).
2. **Wheels off the ground.** Put the robot on a block so that the wheels spin freely. Run every motor, the encoders, the sensors and the stops with nothing to run into. A bench supply with

a current limit of about 0.5 A in place of the pack makes the first power-on gentle: a short or a stall folds the supply back instead of the pack burning.

3. **Speed-limit the first floor runs.** Cap the duty at about 25 %: from the motor model, 25 % of 7.4 V is 1.85 V, and (1.85 − 0.17 × 2.0) / 0.006 = 252 rad/s at the motor, which is about 0.16 m/s at the wheels. Raise the cap in steps only after each step behaves.
4. **Clear the area.** A flat, clear floor, no people or pets in the way, a hand within reach of the e-stop, and a stopping distance (below) known and respected.
5. **Add one thing at a time.** Straight line, then turns, then the line, then the obstacle stop; each test passes before the next begins (Robotics Fundamentals 8 section 7: test each behaviour alone).
6. **Log and review.** Record the state changes and the battery voltage; a log of what the robot did is the only witness to a failure.

Stopping distance, with the numbers of the build

The stopping arithmetic of Robotics Fundamentals 1 section 2 (reaction distance plus braking distance) and Robotics Fundamentals 8 section 7 fixes how early the robot must act. For the build:

```
Latency of the build:  ultrasonic update 60 ms + control loop 20 ms + compute 5 ms + mo

       v = 0.3 m/s:   reaction 0.3 x 0.115 = 0.0345 m;   braking 0.3^2 / ( 2 x 2.0 ) = 
       v = 0.4 m/s:   reaction 0.0460 m;   braking 0.0400 m;   stop = 0.0860 m.

Software freeze at 0.3 m/s with the 100 ms watchdog (5 x the 20 ms loop):   0.3 x 0.100
A person hits the e-stop at 0.4 m/s, with a typical human reaction of about 0.25 s (appr
       The stop is only as quick as the person, so on first runs keep the speed down an
```

Arms and other pinch points

Anything with joints can trap a finger. If you extend the build with an arm (section 9, extensions), apply Robotics Fundamentals 9 section 10: size each joint for the payload at its worst pose, find the pinch points (the inside of an elbow, a gripper's jaws), cap speed and force, decide how each joint holds its load when power is cut, and test slowly with everyone outside the reach. A wheel is a pinch point too: keep fingers and cables away from the gap between a wheel and the chassis when the robot is powered.

8. Ethics and responsibility

A robot is a machine that acts in the world, and what it does is the builder's responsibility. Four points, briefly.

- **Privacy.** A camera, a microphone or a robot that moves about a home records people, often without their thinking about it. Decide what is recorded, where it is stored, who can see it, and tell the people it concerns; do not point a robot at places where people expect privacy, and treat recorded images and sound as sensitive data (in the UK, data protection law can apply to cameras that capture people outside a purely private setting; take proper advice for anything beyond your own home).
- **Safety of people and pets.** A small robot can trip, pinch or frighten. Keep it slow, lightweight and soft-edged where it shares a room; keep fingers, hair, pet tails and small children away from wheels and arms; fit the stops of section 7; and do not run an untested robot unattended, least of all with a lithium battery on board.
- **Dual use.** Most robotics ideas can be used to help or to harm: the same navigation and vision serves a delivery robot and a weapon. Think about how what you build could be misused, share it responsibly, and be wary of work whose evident purpose is to hurt people or to watch them without consent.
- **Honest claims.** Say what the robot actually does, and what it was tested for. “Runs about 1.5 hours” is true if you measured it; “avoids all obstacles” is false if a black chair leg beats the sensor. Overclaiming about autonomy, safety or accuracy misleads the people who rely on it, and a robot sold or used at work has legal obligations (safety marking, product liability) that a hobby project does not.

9. The build: a small line-following, obstacle-avoiding robot

The design below is a **plan** for the simplest robot that uses every subsystem of the course: two driven wheels and a caster (differential drive, Robotics Fundamentals 4), geared motors with encoders (Robotics Fundamentals 3 and Robotics Fundamentals 6), a line sensor array and an ultrasonic sensor (Robotics Fundamentals 5), a feedback loop for the wheels and the line (Robotics Fundamentals 7), a behaviour state machine (Robotics Fundamentals 8), and the power and safety of this chapter. It is hardware-neutral: any board with the right pins will do (Basic Electronics 11 section 1 compares them), and the code is described in pseudocode only.

Real board code, in MicroPython or C, is the work of the planned course Programming for Robotics.

9.1 Requirements and specification

#	REQUIREMENT	HOW IT IS MEASURED (PASS IF)
R1	Follows a black 19 mm line (electrical tape) on a light floor at an average of about 0.2 m/s, with bends of radius at least 0.3 m	3 laps of the test oval (lap about 3.1 m) without losing the line for more than 3 s and without entering STOP
R2	Stops for an obstacle on the line	Stops at least 0.10 m from a card placed in its path; waits up to 10 s for it to clear, then stops for good
R3	Drives straight when told to	Lateral drift of at most 30 mm over 2 m (after calibration)
R4	Turns accurately on the spot	A commanded 360 degree turn ends within 5 degrees of the start heading
R5	Runs for at least 90 minutes on one charge	Computed 137.6 min (section 3); the measured runtime is at least 90 min
R6	Fails safe	E-stop, watchdog and low-battery cutoff each stop the motors in the tests of section 9.8
R7	Cost	Parts under about 180 pounds, tools and a multimeter excluded (approximate)

The maximum speed is 0.4 m/s and the line-following base speed is 0.30 m/s, slowed on bends by the rule $v = v_{max} / (1 + c|x|)$ (Robotics Fundamentals 8 section 3). The build is intentionally modest: a robot that meets this table is a real, working machine, and every number in it can be checked.

9.2 Parts list (approximate prices in GBP)

PART	NOTES	APPROXIMATE PRICE (£)
Chassis kit: plate, ball caster, 2 wheels of 65 mm	A cheap two-wheel acrylic or metal kit; the wheels fix the 65 mm of the calculations; confirm the track width	8 to 15

PART	NOTES	APPROXIMATE PRICE (£)
2 geared 6 V motors with quadrature encoders (about 50:1)	The motors of the calculations; encoders giving about 12 counts per motor revolution, 600 per wheel revolution	20 to 35
Raspberry Pi Pico (or an Arduino Uno or Nano)	3.3 V logic (Pico) or 5 V (Arduino): Basic Electronics 10 and Basic Electronics 11 compare the boards	4 to 6
Dual motor driver module (TB6612FNG in the example)	Rated for the motors: the TB6612FNG is about 1.2 A continuous and 3.2 A peak per channel (from memory), below the 3 A stall of these motors, so it relies on the fuse and the stall rule; a 3 A class driver is the safer buy	4 to 7
5-channel IR reflectance (line) module, digital outputs	12 mm pitch, powered from 3.3 V so its outputs are safe for the Pico	3 to 6
HC-SR04 ultrasonic sensor	5 V part; its echo needs a divider for a 3.3 V board	2 to 4
5 V buck converter module, 2 A	Logic rail (section 4)	2 to 5
2S LiPo pack, 1000 mAh, 20C, with an XT30 connector	Alternative: a holder of 6 AA NiMH cells (7.2 V nominal), gentler for a first build	8 to 14
2S balance charger	Mandatory with a LiPo pack; skip if you use NiMH with its own charger	15 to 30
Fire-resistant LiPo bag	For charging and storage (section 5)	4 to 8
Fuse holder with 1 A fuses (slow-blow and fast-blow spares)	In the positive lead near the pack	2 to 4
Main rocker switch, at least 5 A DC	Check the DC rating	1 to 3
E-stop push button, normally closed, at least 5 A DC	In series with the motor supply; red, large, within reach	3 to 7

PART	NOTES	APPROXIMATE PRICE (£)
Wire (20 and 22 AWG silicone), connectors, heat-shrink	Section 6	6 to 12
Perfboard or breadboard, headers and Dupont leads	A breadboard for signals only, never for motor current	4 to 8
Resistors and capacitors	56 k and 33 k (battery divider), 1 k and 2 k (echo divider), 10 k pull-downs, a 100 to 470 uF bulk capacitor, 100 nF decoupling	2 to 4
Standoffs, screws, cable ties, 19 mm black tape	For mounting, strain relief and the test track	4 to 8
Total	Multimeter, soldering iron and small tools not included	92 to 176

Prices vary by supplier, and these are rough ranges from memory. A cheaper kit (yellow TT-type motors without encoders) will not match the calculations below: those motors are bigger and slower and have no feedback, which changes the sizing, the odometry and the test results. The calculations are for the parts listed.

9.3 Design calculations (reusing the course)

```
MOTORS (Robotics Fundamentals 3 section 6):  m = 0.8 kg, wheel diameter 65 mm (r = 0.03
       Top speed 0.4 m/s:   wheel 0.4 / 0.0325 = 12.31 rad/s = 117.5 rpm;   gear ratio
       Worst working load (10 deg slope, a = 0.8 m/s^2, margin 1.5):  F = 2.235 N,  tau
       Traction limit:  0.78 x 0.8 x 0.8 x 9.81 x cos 10 = 4.82 N, more than the 2.23 N

KINEMATICS AND ODOMETRY (Robotics Fundamentals 4 sections 3 and 5):  track width L = 0.
       Distance per tick = 0.2042 / 600 = 0.340 mm;   ticks per metre = 2938;   2 m = 5
       Turning:  v(left) = v - w L / 2,  v(right) = v + w L / 2.     A turn on the spot
       Speed from ticks at the 50 Hz loop: one tick per 20 ms is 0.340 mm / 0.020 s = 0

DRIFT (Robotics Fundamentals 4 section 6):   a tyre 1 % bigger than the other gives a h
       The spec (30 mm over 2 m) needs  eps = 2 x 0.030 / 2^2 = 0.015 rad/m, so the dia

SENSORS (Robotics Fundamentals 5):   line array of 5 sensors at a 12 mm pitch, position
       The 19 mm tape covers one or two sensors, so the weighted position takes values
       Ultrasonic: echo time for 0.25 m = 2 x 0.25 / 343 = 1.458 ms (c = 343 m/s at 20
```

```
CONTROL (Robotics Fundamentals 7):   outer loop, the line follower of section 12 of tha
       Speed:  v = 0.30 / ( 1 + 0.05 |x| ).   At the extreme x = 24 mm:  v = 0.30 / 2.2
       Inner loop: a PI on each wheel's speed (cascade, Robotics Fundamentals 7 section

BEHAVIOUR (Robotics Fundamentals 8):   states WAIT, FOLLOW, SEARCH, BLOCKED, STOP, LOW_

POWER AND SAFETY:   budget, runtime, fuse, wire, cutoff: sections 2 to 7 (137.6 min, 1 A
```

```
Behaviour state machine (events on the arrows, timeouts in brackets):

  WAIT      --start button-->                       FOLLOW
  FOLLOW    --no line for 0.3 s-->                  SEARCH          (spin toward the
  FOLLOW    --front distance < 0.25 m-->             BLOCKED         (stop, wheels br
  SEARCH    --line found-->                         FOLLOW
  SEARCH    --3 s with no line-->                   STOP            (report "line los
  BLOCKED   --front distance > 0.35 m for 0.5 s-->  FOLLOW
  BLOCKED   --10 s with the way still blocked-->    STOP
  any       --pack below 6.6 V for 2 s-->           LOW_BATT        (motors off, latc
  STOP / LOW_BATT   --only a power cycle (and a charge, for LOW_BATT)--> WAIT

  A second mode (a jumper to ground selects it) for a robot with no line on the floor:
```




```
POWER                                                  SENSE - THINK - ACT

battery 2S LiPo 7.4 V                                 +-----------------------
       |                                              |  Raspberry Pi Pico
  [ fuse 1 A ]                                        |  50 Hz loop:
       |                                              |  odometry, line,
  [ main switch ]                                     |  state machine,
       |                                              |  wheel PI loops,
       +----------------+--------------------+        |  watchdog 100 ms
       |                |                    |        +-----------+-----------
[ E-STOP, NC ]     [ 5 V buck ]         [ divider ]                 |
       |                |                                 4 direction pins, 2 PWM pi
 [ motor driver ]       +--> Pico VSYS, HC-SR04                     |
    |          |        +--> (through the Pico's 3.3 V) line module,  [ motor driver ]
[ left ]    [ right ]         encoders, driver logic                 |
[ motor ]   [ motor ]                                               +--> status LED
+ encoder   + encoder
```

```
All the grounds meet at ONE star point on the battery minus (section 6).   The e-stop o
```

9.5 Wiring table, pin by pin

The table gives each connection first in hardware-neutral terms (what it needs) and then as an **example for one named board, the Raspberry Pi Pico**. The GP numbers are the Pico's GPIO numbers, not physical pin numbers, and are **an example from memory: check every pin against your own board's pinout diagram** (some pins have fixed roles, and on the Pico W the on-board LED is wired differently).

FUNCTION	HARDWARE-NEUTRAL REQUIREMENT	PICO EXAMPLE (CHECK IT)	NOTES
Left encoder A, B	Two digital inputs, edge-capable (interrupt or a fast poll); encoder powered at the board's logic level	GP2, GP3	Quadrature (Robotics Fundamentals 6 section 2); swap A and B if the count runs backwards
Right encoder A, B	As above	GP4, GP5	Swap A and B on the right so that forward counts up on both
Driver standby	A digital output that enables the driver; pulled low so the motors are off at reset	GP9 (STBY) with a 10 kΩ pull-down to ground	Fail-safe: a floating pin leaves the driver off
Left motor direction	Two digital outputs (forward, reverse)	GP10 (AIN1), GP11 (AIN2)	Both low or both high brakes or coasts, see the driver data sheet (Basic Electronics 9 section 3)
Left motor speed	One PWM output, 20 kHz or so (Basic Electronics 9 section 2)	GP12 (PWMA)	10 kΩ pull-down; the same PWM slice as GP13 (one frequency for both)
Right motor speed	One PWM output	GP13 (PWMB)	10 kΩ pull-down
Right motor direction	Two digital outputs	GP14 (BIN1), GP15 (BIN2)	As the left

FUNCTION	HARDWARE-NEUTRAL REQUIREMENT	PICO EXAMPLE (CHECK IT)	NOTES
Line sensors 1 to 5	Five digital inputs (on or off); sensor module powered from the board's logic voltage so its outputs are at that level	GP16, GP17, GP18, GP19, GP20; module VCC to 3V3	Left to right; position = weighted average (Robotics Fundamentals 5 section 4). The board's 3.3 V regulator supplies about 100 mA here, which is within its rating (check the data sheet)
Ultrasonic trigger	A digital output (a 10 microsecond pulse)	GP6 to TRIG	HC-SR04 powered from the 5 V rail
Ultrasonic echo	A digital input reading the echo pulse width; the 5 V echo must be divided for a 3.3 V input	GP7 from ECHO through a 1 kΩ (series) and 2 kΩ (to ground) divider	5 V x 2 / 3 = 3.33 V at the pin (the level shifting of Basic Electronics 10 section 7 and circuit 4 of its section 10)
Battery monitor	An analogue input, with a divider that keeps it below the ADC reference	GP26 (ADC0) from 56 kΩ (top) and 33 kΩ (bottom)	3.115 V at 8.4 V; table in section 5
Start button	A digital input with a pull-up; closes to ground	GP21, internal pull-up, button to ground	The run command; releasing the e-stop does not restart the robot
Status LED	A digital output through a resistor	GP25 (on-board LED on the plain Pico)	Steady while running, flashing when the battery is low
Driver motor supply (VM)	The pack voltage, through the fuse, the main switch and the e-stop; a bulk capacitor at the pins	From the star (+) via the e-stop, 100 to 470 uF to ground at the driver	20 AWG wire; the thickest ground from the driver straight to the star (-)
Driver logic supply (VCC)	The board's logic voltage	3V3 (OUT)	Check the module: some tie logic and motor supplies together
Board and sensor supply	A regulated rail from the buck, not from the motor branch	5 V buck output to VSYS through a Schottky diode (so USB and the battery cannot back-feed each other); HC-SR04 straight from the 5 V rail	Check the Pico data sheet for the allowed supply pins and the diode arrangement before connecting USB and the battery together

With an **Arduino Uno** (5 V logic) the same plan works, with changes: the Uno has only two external-interrupt pins (D2 and D3), so each encoder's channel A goes to one of them and channel B to an ordinary pin (or use pin-change interrupts); PWM comes from pins 3, 5, 6, 9, 10 or 11; the echo divider is not needed; the battery divider can use A0 (a 5 V reference gives 3.1 V at 8.4 V, still safe); and the line module can run from 5 V. Check the pin capabilities against Basic Electronics 10 section 1.

9.6 Assembly order

1. **Build the frame.** Fit the motors, wheels and caster to the chassis; check that the robot sits level and that the centre of mass is over the wheels and not far behind them (Robotics Fundamentals 2 section 3); measure the real track width and wheel diameter and write them down.
2. **Mount the battery and the star board.** Fix the pack low and central with a strap, put the fuse holder, the main switch and the e-stop where they can be reached, and lay out the star board. **Do not connect the battery yet.**
3. **Wire the power side with the battery disconnected:** the fuse, the switch, the e-stop, the buck module and the divider; strain-relief each bundle. Keep the fuse out until the continuity checks of section 9.8 pass.
4. **Mount the board and the driver;** wire the driver's motor outputs, its logic pins and its pull-downs; solder the motor wires (never through a breadboard).
5. **Add the encoders,** then the line module and the ultrasonic sensor with its echo divider. Mount the line module 3 to 8 mm above the floor and about 40 mm ahead of the axle; mount the ultrasonic sensor at the front, pointing forward and level.
6. **Run the power-off tests** (continuity, shorts, resistance of the motors), then the voltage tests with the fuse in and the wheels on a block (section 9.8, tests 1 and 2).
7. **Test the pieces in order:** encoders, then the driver, the stops, the line module and the ultrasonic sensor, each alone, with the wheels in the air, before any floor test.
8. **Then the floor:** speed-limited and with a clear floor, then straight line and spin calibration, then the line, then the obstacle test, raising the speed cap in steps.

9.7 The top-level control loop (hardware-neutral pseudocode)

Only the top-level loop is shown, in short Python-style pseudocode. The function names stand for what your board's library calls them; the odometry update, the line position, the PID and the state machine are the code of Robotics Fundamentals 4, Robotics Fundamentals 5, Robotics

Fundamentals 7 and Robotics Fundamentals 8. The pseudocode was run with simple stand-in functions to check that the structure holds together.

```
LOOP_DT = 0.02                                  # 50 Hz;  the hardware watchdog bit
TRACK, V_MAX, C_SLOW = 0.15, 0.30, 0.05          # m, m/s, per mm of line error
KP, KD = 0.05, 0.002                             # line follower gains (starting va
D_STOP, D_GO, V_CUT = 0.25, 0.35, 6.6            # obstacle thresholds (m) and the
DUTY_MAX = 0.25                                  # first-run cap; raise it in steps

def behaviour(state, x_line, last_side):         # returns (forward speed, turn rat
    if state == "FOLLOW":
        e = -x_line                              # mm: positive when the line is to
        w = KP * e + KD * line_error_rate()      # counter-clockwise positive
        return V_MAX / (1 + C_SLOW * abs(x_line)), w
    if state == "SEARCH":
        return 0.0, 1.0 * last_side              # spin toward the side where the l
    return 0.0, 0.0                              # WAIT, BLOCKED, STOP, LOW_BATT: s

state = "WAIT"
while True:
    t0 = now()
    kick_watchdog()                              # the hardware resets the board if
    d_l, d_r = read_wheel_distances()            # metres since the last call, from
    x_line, seen, last_side = read_line()        # weighted position in mm; seen =
    front = read_front_distance()                # metres, median-filtered
    volts = read_battery_volts()                 # the average of 16 samples

    state = next_state(state, seen, front, volts, start_pressed())      # the transitio
    v, w = behaviour(state, x_line, last_side)
    v_l, v_r = v - w * TRACK / 2, v + w * TRACK / 2                      # inverse kine
    drive_wheel(LEFT,  v_l, d_l / LOOP_DT, DUTY_MAX)                    # the wheel-spe
    drive_wheel(RIGHT, v_r, d_r / LOOP_DT, DUTY_MAX)
    wait_until(t0 + LOOP_DT)                     # keep the loop period fixed
```

9.8 Test plan with expected readings

Each test has an expected reading (a number you can compare with your meter or your log) and a pass condition. Run them in order, and do not move to the next until the last passes. The expected values come from the calculations above and are approximate; real parts differ, and a reading that is merely different from the figure is not necessarily a fault, while a reading that is wildly different (or a short) always is.

#	TEST	HOW	EXPECTED (APPROXIMATE)	PASS IF
1	Continuity and shorts (power off, battery unplugged)	Multimeter on resistance: pack + to pack - with the switch on, fuse in and e-stop released; fuse, switch and e-stop each alone; the star grounds; each motor's winding	Pack + to - high, no short (the 89 kΩ divider is in parallel, so tens of kΩ); fuse under 1 Ω; switch and e-stop under 0.5 Ω closed and open (OL) when pressed; ground points under 0.5 Ω to the star; each motor about 2 Ω (the data sheet's value)	No short (never below about 10 kΩ across the pack); stops open the circuit; windings are close to the data sheet
2	Voltages (fuse in, wheels on a block)	Multimeter on DC volts at the pack, the driver's VM, the buck output and the board's 3.3 V	Pack 8.2 to 8.4 V when full; VM within 0.05 V of the pack; buck output 5.0 V (within 0.25 V); 3.3 V within 0.1 V; the divider pin 3.115 V at 8.4 V, an ADC count near 3865	All within the ranges; the ADC reading agrees with the meter within about 2 %
3	Currents (wheels in the air, meter in series with the pack)	First with the motors off, then each motor at the 25 % duty cap	Logic only about 0.13 A (0.08 to 0.18 A); with both motors at 25 % duty in the air about 0.20 A (0.15 to 0.30 A)	Under about 0.4 A: more than 0.5 A in the air means a binding gearbox, a short or a driver fault
4	Encoder count per revolution	Mark a wheel; turn it by hand exactly one turn, then ten turns, and read the count; check both signs	600 per wheel revolution (12 counts x 50:1): 6000 for ten turns; both counts rise when the wheel turns forward	600 within +/- 3 per turn and 6000 within +/- 10 for ten; signs right on both wheels
5	Straight-line drift (R3)	At the 25 % cap, drive 2 m (5876 ticks) along a tape measure; measure the sideways offset at the end	Under 30 mm after calibration; before calibration 1 % of tyre mismatch gives 133 mm	Lateral drift at most 30 mm; if more, calibrate (below) and retest
6	Spin test (R4)	Command a 360 degree turn on the spot; mark the start and finish headings	End within 5 degrees of the start; a track width 3 % out would leave the robot 11.6 degrees short	Heading error within 5 degrees (a track width within 1.4 %)
7	Line follow (R1)	A test oval of 19 mm tape (two straights of 0.6 m and two semicircular ends of	About 15 s per lap at an average of 0.2 m/s, so about 46 s for three laps; sideways wobble	3 laps; no state of STOP; no SEARCH longer than 3 s

#	TEST	HOW	EXPECTED (APPROXIMATE)	PASS IF
		radius 0.3 m, lap 3.08 m); 3 laps	within about 12 mm on the straights	
8	Obstacle (R2)	Place a card across the line 1 m ahead; start at 0.3 m/s; log the distance when it stops	The robot stops about 0.19 m from the card (the 0.25 m threshold minus the 0.057 m stopping distance); the echo time at 0.25 m is 1.46 ms	Stops, never touches the card, at least 0.10 m away; resumes within 1 s when the card is removed, or stops for good after 10 s
9	Stops and the cutoff (R6)	(a) Wheels in the air, press the e-stop. (b) Insert a deliberate infinite loop in the program. (c) Replace the pack by a bench supply and lower it slowly to 6.5 V. (d) Do not test the fuse by shorting the pack; check it by continuity only	(a) The wheels stop within about 0.5 s and do not restart on release. (b) The watchdog resets the board and the motors stop within about 0.1 s. (c) LOW_BATT after about 2 s below 6.6 V, latched	All three stops work and the motors do not restart without a deliberate action

```
Calibrating the drift (test 5), a worked example:   the robot drives 2.0 m "straight" a
       Lateral offset y = eps x D^2 / 2, so  eps = 2 y / D^2 = 2 x 0.040 / 2.0^2 = 0.02
       The diameter mismatch is  eps x L = 0.020 x 0.15 = 0.003 = 0.3 %.   That is abov
       Fix: scale the distance per tick of one wheel by 1.003 (the wheel that travelled
       The check on the arithmetic:  a 0.3 % mismatch over 2 m gives  0.003 / 0.15 x 2.

Calibrating the spin (test 6):  if the robot ends 11.6 degrees short of 360, the track
```

9.9 Troubleshooting

SYMPTOM	LIKELY CAUSE	CHECK AND FIX
The board resets when the motors start	Brown-out from the pack's sag or a shared wire (Basic Electronics 9 section 10)	Bulk capacitor at the driver; separate star branches for motor and logic; check the pack's charge and its sag (section 2); thicker motor wire

SYMPTOM	LIKELY CAUSE	CHECK AND FIX
A motor does not run	Standby low, e-stop open, fuse blown, a loose connector	Check STBY, the e-stop, the fuse by continuity, then the motor wires at the driver with a meter on VM
The fuse blows during driving	A stall (a jammed wheel, a snagged wire) or a short	Look at the log for the stall rule; free the wheel; check for a short with the meter; never fit a bigger fuse to stop it blowing
It drives in an arc on a straight command	Wheel diameter mismatch, motor difference, encoder sign or scale	Run test 4, then calibrate by test 5; check the wheels turn the same number of ticks per revolution
It overshoots turns or is 12 degrees short	Wrong track width in the code	Measure the track width between the tyre contact patches; calibrate by test 6
An encoder counts backwards or erratically	A and B swapped, a loose wire, noise from the motor	Swap A and B; twist the encoder pair; keep it away from the motor leads; add 100 nF at the pin
The line follower snakes from side to side	Kp too high for the speed, the sensor too far ahead, or too much delay (Robotics Fundamentals 7 section 7)	Halve Kp, add Kd, lower the speed, move the array closer to the axle, then retune in the order of Robotics Fundamentals 7 section 9
It loses the line on bends	Too fast, the array too high, tape too narrow or glossy	Lower the base speed or raise C_SLOW; set the array 3 to 8 mm above the floor; check the sensor thresholds on the real floor; test the SEARCH rule
It stops for nothing, or does not stop for a chair leg	False echoes, a soft or angled surface, a narrow object outside the cone	Median filter; lower the speed; add a second sensor or a bump switch (Robotics Fundamentals 5 section 5); test with a flat card first
The battery reads wrong	Wrong divider values, a loose ground, noise, or reading under load	Check the pin voltage with a meter (3.115 V at 8.4 V); average 16 samples; compare at rest and under load
A pack is warm, puffy or smells hot	Over-discharge, a short, abuse or age	Stop at once; do not charge or use it; follow section 5

9.10 Extension ideas

- **Add a gyro** (Robotics Fundamentals 6 section 4) and fuse it with the wheel heading by a complementary filter, for a drift-free straight line and an accurate spin.
- **Replace the ultrasonic sensor** by a time-of-flight sensor (Robotics Fundamentals 5 section 3) for a narrower, faster reading, or add side sensors for wall following (Robotics Fundamentals 8 section 3).
- **Sense the motor current** (Robotics Fundamentals 5 section 7) to detect a stall directly and to measure the real power budget.
- **Log and map:** record the pose and the sensor readings, then build an occupancy grid and plan a path with A* (Robotics Fundamentals 8 section 4) as a second program.
- **Redesign the frame** as a 3D-printed or laser-cut part (Robotics Fundamentals 2 section 4), with a proper battery tray, a protected e-stop and a bumper.
- **Add a small servo arm** (Robotics Fundamentals 9) with a separate servo supply and the arm safety rules of that chapter, and re-budget the power and the centre of mass.
- **Telemetry:** a Pico W or ESP32 can stream the log over Wi-Fi so that you can watch the battery, the state and the PID terms live.

10. Course complete: a recap of Chapters 1 to 10

CHAPTER	TITLE	THE ONE THING TO KEEP	WHERE IT APPEARS IN THE BUILD
1	What Is a Robot?	A robot senses, decides and acts in a loop closed through the world; latency and stopping distance set how early it must react.	The 50 Hz loop, the stopping distance (section 7), runtime (section 2)
2	Mechanics and Structure	Frames, levers, centre of gravity and the support polygon decide whether a robot is stiff, light and stays on its wheels.	The chassis, the battery position, assembly step 1
3	Motion: Motors, Gears and Wheels	Torque, speed and power; size a motor from the load, the gear ratio and the stall torque, and check the current.	Motor sizing and the power budget (section 3, section 9.3)
4	Drive Systems and Kinematics	Differential drive turns wheel speeds into motion, odometry integrates it, and small errors in wheel size and track width drift fast.	Kinematics, odometry, drift and spin calibration

CHAPTER	TITLE	THE ONE THING TO KEEP	WHERE IT APPEARS IN THE BUILD
5	Sensors: Seeing the World	A sensor has range, noise, resolution and a sample rate; filter, calibrate and choose by the job.	The line array, the ultrasonic sensor, the median filter
6	Position, Orientation and Feedback	Encoders measure the wheels, an IMU measures the turn, and fusion and feedback keep the estimate honest.	Encoder counts, the speed loop, the gyro extension
7	Control: Making It Do What You Want	Closed-loop PID with tuning, windup and the causes of oscillation; cascades and feedforward.	The line-follower PID and the wheel-speed loops
8	Behaviour and Decision-Making	Reactive behaviours, state machines, timeouts and watchdogs; fail safe, then plan and localise.	The state machine, the stops and the watchdog
9	Arms and Manipulators	Forward and inverse kinematics, the Jacobian, payload torque, grippers and arm safety.	The safety rules and the arm extension
10	Power, Safety and Building Your First Robot	Size the battery, budget the power, wire and fuse it, stop it safely and test in order; then put it all together.	This chapter, and the whole build

You can now reason about a small robot from the battery to the behaviour: why a motor was chosen, what the wheels do, what a sensor can and cannot tell you, why a loop oscillates, and what makes a machine safe to run. The calculations are all things you can repeat with a pencil, and the build above is a way to find out how much of it is true on your own bench.

The path on: Programming for Robotics, and possible later courses

Programming for Robotics (planned; not yet written) is the next course on this site, and it is the other half of this one. Robotics Fundamentals has kept the code hardware-neutral and short on purpose. Programming for Robotics will turn the same ideas into real code on real boards: reading sensors and driving motors, timing and interrupts for encoders, the PID loop and the state machine as programs, odometry and logging, and the top-level loop of section 9.7 written in MicroPython or C. If you build the robot of this chapter, that course is where its control loop becomes running code.

Possible *later* robotics courses, mentioned only as ideas and not planned: project-style courses whose topic is building a robot, perhaps around a commercial kit, in the manner of this chapter's build plan but carried through to a finished and tested machine. If those appear, they will build on the habits of this chapter: size it, budget it, fuse it, test it in order, and never trust a stop that you have not tested.

11. Challenges

CHALLENGE 1: POWER BUDGET AND FUSE CHOICE

A small delivery cart runs from a 3S LiPo pack (11.1 V nominal, 2200 mAh, 20C, internal resistance about 90 mΩ). Its two motors each draw 0.9 A on average, 1.6 A at their worst working load and 7.5 A at stall (all measured at the pack). The 5 V logic rail, from a 90 % efficient buck converter, carries a controller (0.25 A average, 0.35 A peak), sensors (0.12 A, 0.12 A) and a gripper servo (0.10 A average, 0.70 A peak). (a) Find the rail current average and peak, and the current they draw from the pack. (b) Find the total pack current for the average, the worst working peak and the stall, and the average power. (c) Find the usable energy and the runtime, using the 0.8 factor. (d) Choose a fuse by the 125 to 150 % rule, say which wire gauges are acceptable (20 AWG rated 11 A, 22 AWG 7 A, 24 AWG 3.5 A), and find the voltage drop of the chosen run if it is 0.4 m each way (20 AWG 33.3 mΩ/m, 22 AWG 53.0 mΩ/m). (e) Find the C-rate at stall and the voltage sag. (f) Compare a linear regulator with the buck for the rail: pack power and heat, average and peak.

View solution

CHALLENGE 2: SAFETY REVIEW OF A GIVEN DESIGN

A hobbyist describes a robot. It runs from a 3S LiPo pack (3000 mAh, 20C) wired directly to two motor drivers with 26 AWG jumper leads on a breadboard, 0.25 m each way, with no fuse. The power switch cuts only the 5 V supply of the controller. The “stop” is a button read by the program. The controller gets its 5 V from a linear regulator that also feeds a servo. The motor driver's enable pin is not pulled down. The robot drives at 0.8 m/s, stops when an ultrasonic sensor reads under 0.15 m, has a total latency of 0.15 s and brakes at 2.0 m/s^2. There is no watchdog and no battery monitor. The pack is charged overnight on a sofa by a bench supply set to 12.6 V, and then left in the robot, fully charged, for months. (a) List at least ten faults. (b) Rank the five most dangerous. (c) Find the stopping distance, whether the 0.15 m threshold is enough, and the fastest speed that stops within 0.15 m, with and without a margin of 2. (d) Find the heat in the 26 AWG leads at a stall current of 10 A, and the pack's maximum short-circuit current. (e) Give a corrected design, one fix per fault.

View solution

CHALLENGE 3: ADAPT THE BUILD TO A HEAVIER PAYLOAD

The build of section 9 is to carry a 1.4 kg payload, so the total mass becomes 2.2 kg. The speed stays at 0.3 m/s, the acceleration at 0.8 m/s^2, the slope at 10 degrees and the same 6 V motors, 50:1 gearbox and 65 mm wheels are kept (2.0 ohm, k = 0.006, I0 = 0.15 A, stall torque 0.018 N m, efficiency 0.729, Crr = 0.03, mu = 0.8, traction safety factor 0.78). The 7.4 V pack and the 5 V logic budget are unchanged (170 mA average, 215 mA peak on the rail, 90 % buck), and a 5 % loss allowance is kept on the motor current. (a) Find the force, the wheel torque, the motor torque with a margin of 1.5, the current, the percentage of stall torque, the voltage needed and the duty at the worst working load; check traction. (b) Find the cruise current and duty on the flat, the pack current for the average (1.5 times cruise), and the total average pack current and power. (c) The robot must run for 3 hours. Find the capacity needed with the 0.8 factor, and the capacity needed if the cold and aged factors (0.8 and 0.9) also apply; choose a standard pack. (d) Find the peak working pack current and the fuse by the 125 to 150 % rule. (e) State, in one sentence each, which parts of the original plan stay and which must change.

View solution

QUICK REFERENCE

- **Battery:** E = V C; E(usable) = 0.8 E; t = 0.8 V C / P; C-rate = I(peak) / C; V(terminal) = V(open) − I R; simple derating t = 0.8 k(cold) k(age) V C / P. NiMH forgiving; Li-ion and LiPo energetic and dangerous; LiFePO4 safer and heavier; lead-acid cheap and very heavy.
- **Power budget:** list every load at average and peak; the average sets the runtime, the peak sets the regulator, wire, fuse and C-rate; budget the stall case too. The build: 0.349 A, 2.58 W average; 0.565 A peak working; 5.03 A at stall; 137.6 min on 1000 mAh.
- **Regulators:** linear loses $(V_{in} - V_{out})$ I as heat; buck draws P_{out} / η (typically 85 to 95 %); use a buck above about 100 to 200 mA from a high input, a linear where the current is tiny or quiet matters.
- **Low-voltage cutoff:** divider into the ADC; warn at 3.5 V per cell, cut off at 3.3 V per cell under load for 2 s in a row, latched; never below 3.0 V per cell.
- **LiPo:** never charge unattended; proper balance charger at 1C or less; non-flammable surface; storage at about 3.8 V per cell; never charge damaged or swollen packs.
- **Wire and fuse:** $V = I R_{wire}$ over the round trip; fuse at 125 to 150 % of the normal current, below the wire rating, close to the battery; check the stall transient; DC rating on switches and e-stops; motor current never through a breadboard.

- **Layout:** star distribution from one point, separate motor and logic branches, the thickest ground to the star, bulk capacitor at the driver, strain relief on every bundle.
- **Safety:** a hardware e-stop that does not depend on the program, plus a watchdog (5 times the loop period), driver and fuse limits and software stops; fail to the safe state; pull-downs on driver pins; stop on silence; limit duty, speed and current; test wheels-off-ground first, one thing at a time. Stopping distance = $v \times \text{latency} + v^2/2a$.
- **The build:** two motors with encoders (600 ticks per wheel revolution, 0.340 mm per tick), a 5-sensor line module, an ultrasonic sensor, a Pico or an Arduino, a motor driver, a buck and a fused pack; states WAIT, FOLLOW, SEARCH, BLOCKED, STOP and LOW_BATT; calibrate the drift ($\text{eps} = 2y/D^2$, mismatch = eps L) and the spin; test in order.
- **Ethics:** privacy, the safety of people and pets, dual use, and honest claims.

That completes **Robotics Fundamentals**. The next step on this site is Programming for Robotics (planned), where the loops, the filters, the PID and the state machine of these chapters become code that runs on a board.