

Автономні роботи: сприйняття і навігація

Український машинозчитуваний модуль, format-checked

Correll robotics chapters, Claude Ukrainian translation

2026-06-01

Зміст

1	Сенсори	2
1.1	Термінологія	2
1.1.1	Пропріоцепція vs екстероцепція	3
1.2	Сенсори положення суглобів	3
1.3	Сенсори его-руху	4
1.3.1	Акселерометри	4
1.3.2	Гіроскопи	4
1.4	Вимірювання сили	4
1.4.1	Вимірювання тиску або дотику	5
1.5	Сенсори вимірювання відстані	5
1.5.1	Відбиття	6
1.5.2	Фазовий зсув	6
1.5.3	Час прольоту (time-of-flight)	6
1.6	Сенсори глобальної пози	6
2	Зір (Vision)	9
2.1	Зображення як двовимірні сигнали	9
2.2	Від сигналів до інформації	9
2.3	Базові операції з зображеннями	10
2.3.1	Порогові операції	10
2.3.2	Згорткові фільтри	11
2.3.3	Морфологічні операції	12
2.4	Виокремлення структури із зору	12
2.5	Комп'ютерний зір і машинне навчання	13
3	Локалізація	15
3.1	Мотиваційний приклад	15
3.2	Марковська локалізація	16
3.2.1	Оновлення сприйняття (perception update)	16
3.2.2	Оновлення дією (action update)	17
3.2.3	Приклад: марковська локалізація на топологічній карті	17
3.3	Фільтр Байеса (Bayes filter)	18
3.3.1	Приклад: фільтр Байеса на сітці	19
3.4	Фільтр частинок (Particle Filter)	19
3.5	Розширений фільтр Калмана (EKF)	20
3.5.1	Одометрія через фільтр Калмана	21
3.6	Підсумок: ймовірнісна локалізація на карті	22
4	Побудова карти (Mapping)	24
4.1	Представлення карти	25
4.2	Iterative Closest Point (ICP) для розрідженого зіставлення	25
4.3	Octomap: щільна побудова карти вокселями	26
4.4	RGB-D mapping: щільна побудова поверхневих карт	27

5	Одночасна локалізація та побудова карти (SLAM)	29
5.1	Вступ	29
5.1.1	Орієнтири	29
5.1.2	Особливий випадок I: один орієнтир	29
5.1.3	Особливий випадок II: два орієнтири	30
5.2	Коваріаційна матриця	30
5.3	EKF SLAM	30
5.3.1	Алгоритм	31
5.3.2	Багатосенсорний випадок	32
5.4	SLAM на основі графів (Graph-based SLAM)	32
5.4.1	SLAM як задача оцінювання максимальної правдоподібності	32
5.4.2	Числові техніки для Graph-based SLAM	33
6	Планування шляху	35
6.1	Простір конфігурацій	35
6.2	Алгоритми планування на основі графів	36
6.2.1	Алгоритм Дейкстри	36
6.2.2	A*	37
6.3	Планування на основі вибірок (sampling-based)	37
6.3.1	Rapidly Exploring Random Trees (RRT)	38
6.4	Планування на різних масштабах довжини	40
6.5	Планування покриття (coverage path planning)	41
6.6	Підсумок	41

1. Сенсори

Роботи — це системи, що сприймають, діють і обчислюють. Досі ми вивчали базові фізичні принципи руху, тобто локомоції та маніпуляції. Тепер потрібно зрозуміти основні принципи сенсорів робота, що надають необхідні дані для прийняття рішень і керування.

Цілі цього розділу:

- подати огляд сенсорів, поширених у роботизованих системах;
- викласти фізичні принципи їх роботи;
- з'ясувати механізми, відповідальні за невизначеність у сенсорному висновуванні.

Історично розвиток роботизованих сенсорів спричинений не лише робототехнікою, а й суміжними галузями: транспортом (автомобільний, морський, авіація), пристроями безпеки промисловості, сервоприводами в радіокерованих іграшках, а останнім часом — смартфонами, віртуальною реальністю та ігровими консолями. Ці галузі переважно зробили “екзотичні” сенсори доступними за низькою ціною завдяки масовим застосуванням. Наприклад, акселерометри та гіроскопи зараз широко використовуються у смартфонах і коштують менше долара; XBox зробила 3D-сенсоріку глибини (Kinect) доступною; сучасні легкові автомобілі містять велику кількість сенсорів без значного збільшення вартості.

Подумайте про сенсори, з якими ви взаємодієте щодня. Які сенсори у вашому телефоні, на кухні, в автомобілі?

Сенсори важко класифікувати лише за областю застосування. Більшість задач виграє від кожного можливого джерела інформації. Наприклад, локалізації можна досягти, вимірюючи кількість обертів колеса *енкодером* (1.2), але оцінювання стає точнішим з додаванням акселерометрів (1.3) або візуальних сенсорів (2). Усі підходи відрізняються за точністю і типом наданих даних, але жоден з них не розв'язує задачу локалізації самостійно.

1.1. Термінологія

При роботі з сенсорами важливо точно визначати такі терміни, як “швидкість” і “роздільна здатність”, а також додаткову таксономію.

Розрізняють *активні* та *пасивні* сенсори. Активні сенсори випромінюють певну енергію та вимірюють реакцію середовища. Пасивні сенсори вимірюють енергію, що надходить із середовища. Більшість дистанційних сенсорів (окрім стереозору) — активні. Акселерометр, компас, кнопка-натискач — пасивні.

Діапазон — різниця між верхньою і нижньою межею вимірюваної величини. Відрізняється від *динамічного діапазону* — відношення між найвищим і найнижчим значеннями (зазвичай у логарифмічному масштабі — “децибел”). Мінімальна відстань між двома значеннями, які сенсор може розрізнити — *роздільна здатність*.

Точність (ассурасу) — різниця між (середнім) виходом m і справжньою величиною v :

$$\text{accuracy} = 1 - \frac{|m - v|}{v}. \quad (1.1)$$

рисунок пропущено

Рис. 1.1: Хрестик — справжнє значення. Зліва направо: ані прецизійно, ані точно; прецизійно, але не точно; точно, але не прецизійно; точно і прецизійно.

рисунок пропущено

рисунок пропущено

рисунок пропущено

Рис. 1.2: Зліва направо: патерн квадратурного енкодера; вихідний сигнал (рух вперед); патерн абсолютного енкодера (код Грея).

Прецизійність (precision) — відношення діапазону до статистичної дисперсії сигналу. Прецизійність — це *повторюваність*, а точність описує *систематичну похибку* фізики сенсора. Наприклад, GPS зазвичай прецизійний у межах кількох метрів, але точний лише в межах десятків. Це особливо проявляється при зміні конфігурації супутників.

Швидкість надання вимірювань — *пропускна здатність* сенсора. Сенсор з пропускною здатністю 10 Гц дає сигнал десять разів на секунду.

1.1.1. Пропріоцепція vs екстероцепція

Пропріоцепція — сприйняття внутрішнього стану робота (кути суглобів, швидкості, внутрішні моменти й сили).

Екстероцепція — сприйняття всього зовнішнього: стану світу, невизначеності та правильна дія. Останнім часом увагу приділяють *проксимальним* сенсорам — для вимірювання середовища безпосередньо біля тіла робота, а не лише *дистальним* (камери, акустичні).

1.2. Сенсори положення суглобів

Найважливіший пропріоцептивний сенсор — *енкодер*. Енкодери використовуються для виміру положення і швидкості суглоба, а також сили (у поєднанні з пружиною). Поділяються на інкрементальні (відносні; переважно в мобільних роботах) і абсолютні (переважно в маніпуляторах). Загалом покладаються на магнітний або оптичний маяк, що обертається з мотором, і сенсор, що рахує проходи. Найпоширеніший в робототехніці — *квадратурний енкодер*, оптичний, що використовує патерн, який обертається з мотором, і оптичний сенсор для реєстрації переходів чорний/білий (1.2).

Один сенсор достатній для виявлення положення і швидкості, але не дозволяє визначити напрямок руху. Квадратурні енкодери мають два сенсори (А, В), що реєструють чергуючий патерн з відстанню у чверть фази. Якщо А випереджає В, диск обертається за годинниковою стрілкою; якщо В випереджає А, проти. Абсолютні енкодери використовують код Грея — патерн, у якому між сусідніми сегментами змінюється лише один біт.

1.3. Сенсори его-руху

Вимірювання конфігурації суглобів обмежене статичними спостереженнями. Воно не дозволяє роботу виявити, чи він зараз рухається або прискорюється — осо-

бливо важливо для динамічно стійких роботів (гуманоїдів, квадрокоптерів). Рух можна оцінити через принцип *інерції*.

1.3.1. Акселерометри

Акселерометр можна уявити як масу на демпфованій пружині. За закону Гука $F = kx$ вимірюємо силу, яку маса спричиняє на пружину. З $F = ma$ обчислюємо прискорення a маси m . На Землі це приблизно $9,81 \text{ м/с}^2$. На практиці ці системи реалізують мікроелектромеханічними пристроями (MEMS) — наприклад, консольною балкою, відхилення якої вимірюється ємнісним сенсором. Акселерометри вимірюють до трьох осей трансляційних прискорень. Виведення абсолютного положення вимагає подвійного інтегрування, що вносить значний шум — це робить оцінку положення лише з акселерометра непрактичною. Однак гравітація дає сталий вектор прискорення, тому акселерометри добре оцінюють орієнтацію відносно гравітації (*крен і тангаж*).

1.3.2. Гіроскопи

Гіроскоп — електромеханічний пристрій, що вимірює кутову швидкість і (у деяких конфігураціях) орієнтацію. Комплементарний акселерометру. Класично — це диск, що вільно обертається в системі осей; рух системи зберігається інерційним моментом. Діскові гіроскопи досі використовуються, але важко мініатюризуються.

Варіація — *гіроскоп кутової швидкості* (rate gyro), що вимірює кутову/обертальну швидкість. У оптичному гіроскопі лазерний промінь розщеплюють надвоє і пускають по круговому шляху в протилежних напрямках. При обертанні системи проти напрямку одного з променів цей промінь долатиме трохи довшу відстань — це дає вимірюваний фазовий зсув на приймачі, пропорційний *швидкості обертання*. Малорозмірні оптичні rate gyros непрактичні, але MEMS rate gyros широко доступні — використовують іншу технологію: масу, підвішену на пружинах. Маса активно вібрує, тому при обертанні сенсора зазнає сили Коріоліса, пропорційної бічній швидкості й кутовій швидкості.

Rate-гіроскопи вимірюють кутову швидкість навколо трьох осей, інтегрування дає абсолютну орієнтацію. Акселерометр (3 осі трансляції) + гіроскоп (3 осі обертання) → інформація про рух у всіх шести ступенях свободи. Разом з магнітометром (компасом), що дає абсолютну орієнтацію за курсом (yaw), це формує *інерціальний вимірювальний модуль* (IMU). Ця комбінація особливо потужна: акселерометр + гіроскоп дають комплементарну інформацію про крен/тангаж, магнітометр + гіроскоп — про курс. Це лежить в основі систем визначення кутової орієнтації (AHRS) через сенсорне злиття.

1.4. Вимірювання сили

Вимір фізичних сил взаємодії — критично важливий для робототехніки. Дає змогу делікатно піднімати ягоду, безпечно взаємодіяти з людиною через дотик.

При поєднанні мотора, енкодера й пружини отримуємо *серійний пружний актуатор* (Series Elastic Actuator) [?] — обертальні/лінійні енкодери можуть служити простими сенсорами сили чи моменту через закон Гука. Інший метод — вимірювання струму на кожному суглобі: знаючи позу, обчислюємо очікувані сили й струми, а похибки відповідають додатковим силам.

Рис. 1.3: Сенсор сили/моменту (F/T) передає силу і момент між двома ланками робота через три металеві стрижні, що з'єднують внутрішню втулку із зовнішнім кільцем. Кожен стрижень обладнаний тензодатчиком з кожного боку — 12 сенсорів усього.

Найточніший і найпоширеніший — *сенсор сили/моменту (F/T)*. Механічний пристрій, що детектує одну або більше компонент шестивимірного *wrench*-у (3D-сили і 3D-моменту). Більшість комерційних F/T-сенсорів використовують *тензодатчики* — металеву (провідну) фольгу, що змінює форму під дією *wrench* і змінює електричний опір. Типова конфігурація: внутрішня втулка з суцільного металу підвішена у зовнішньому кільці через три симетричних прямокутних металевих стрижні, кожен з 4 тензодатчиками — разом 6 сигналів (попарно), з яких обчислюємо сили й моменти в 3D.

Обмеження F/T: 1) висока вартість через високу точність виготовлення; 2) розмір (типово розмір зап'ястя людини); 3) низьке відношення сигнал/шум; 4) низька пропускна здатність/реактивність; 5) одинична точка даних, розріджена в просторі і часі (особливо для маніпулятора з багатьма точками контакту).

1.4.1. Вимірювання тиску або дотику

Щоб частково усунути ці обмеження, робототехніки працюють над вимірюванням тиску, прикладеного до поверхні робота.

Людський дотик — найстарший, найбільший і один із найважливіших органів чуття.

Сенсор тиску здатний детектувати або контакт/зіткнення як бінарну величину (сенсор дотику), або градієнт тиску. Вертикальний тиск пропорційний 1D-силі по нормалі до сенсора. Сенсори тиску переважно ємнісні (як тачскрін смартфона): два провідні шари, розділені ізолятором; при тиску відстань зменшується, ємність змінюється.

У порівнянні з F/T, сенсори тиску дають менше інформації (1D vs 6D), але мають: 1) високу реактивність; 2) високу щільність сенсингу (до десятків сенсорів на см^2); 3) низьку вартість; 4) легкість мініатюризації.

Людський дотик не обмежується лише тиском — також високочастотною інформацією (вібраціями). Роботи можуть відтворювати це інтегруванням акселерометрів або мікрофонів у м'який перетворювач та класифікацією спектральної інформації.

В ідеалі — *штучна шкіра*, що поєднує різні модальності (тиск, текстура, температура, світло) — поки не отримала широкого застосування.

1.5. Сенсори вимірювання відстані

Існує плавний перехід від пропріоцептивних до екстероцептивних сенсорів — вимір внутрішнього стану тісно пов'язаний із зовнішнім середовищем при контакті. Для дистанційного дослідження середовища критично виміряти відстань до окремих об'єктів.

Малий форм-фактор і низька ціна світлочутливих напівпровідників призвели до поширення оптичних сенсорів на різних фізичних ефектах: відбиття, фазовий зсув, час прольоту. Інші принципи — радіо ("радар") і звук.

Рис. 1.4: Реальна відповідь IR-сенсора відстані як функція відстані. Одиниці навмисно безрозмірні.

1.5.1. Відбиття

Найпростіший принцип: чим ближче об'єкт, тим більше світла/радіо/звуку він відбиває. Для незалежності від кольору об'єкта зазвичай використовують *інфрачервоне* світло; звук залежить від поверхневих властивостей.

Сенсор має дві компоненти: випромінювач і приймач (вимірює силу відбитого сигналу). Типова відповідь IR-сенсора показана на рис. 1.4 — насичена на малих відстанях, квадратично зменшується далі.

1.5.2. Фазовий зсув

На відстанях більших за короткі відбиттєве вимірювання неточне. Лазерні сенсори вимірюють *фазовий зсув* відбитої хвилі. Випромінене світло модулюється довжиною хвилі, що перевищує максимальну вимірювану дистанцію.

Сучасні *лазерні сканери* випромінюють на 5 МГц. Зі швидкістю світла $3 \cdot 10^5$ км/с довжина хвилі — 60 м, що дає сканеру корисний діапазон до 30 м. Знаючи форму випроміненої хвилі, обчислюємо фазовий зсув, звідки — відстань.

Лазерні сканери в поєднанні з обертальними дзеркалами утворюють *лазерні скануючі далекоміри (LiDAR)*. Системи з 64 скануючими лазерами широко використовуються в автономному водінні.

1.5.3. Час прольоту (time-of-flight)

Найточніше вимірювання — за часом прольоту світла. Світло рухається швидко ($3 \cdot 10^8$ м/с), тому потрібна високошвидкісна електроніка, що вимірює періоди менше наносекунди для сантиметрової точності. На практиці поєднують приймач з дуже швидким електронним затвором, що працює на тій же частоті, що й випромінений імпульс. Знаючи час, виводимо відстань за кількістю фотонів, що повернулись.

Ультразвукові сенсори відстані

Вимір часу прольоту значно простіший для звукових хвиль (344 м/с у повітрі). Ультразвуковий сенсор випромінює імпульс і вимірює час до повернення. Через нижчу швидкість звуку — компроміс між дальністю і пропускну здатністю: довша дальність вимагає довшого очікування.

Перевага ультразвукових перед оптичними: замість променя — конус з кутом розкриття 20–40°. Можуть детектувати малі перешкоди без необхідності прямого попадання променя. Це робить їх сенсором вибору в авто-парковочних системах сучасних автомобілів.

1.6. Сенсори глобальної пози

Досі ми обговорювали сенсори, що вимірюють положення суглобів, кутову швидкість, трансляційне прискорення, сили взаємодії та відстань до об'єктів відносно

власної пози. Для надійної навігації роботи потребують поняття *світової системи координат*.

Локалізація об'єкта через тріангуляцію існувала ще в давні часи — моряки орієнтувалися за зорями. Найдосконаліша система — *Глобальна система позиціонування* (GPS). Супутники на орбіті оснащені знанням свого точного положення і синхронізованими годинниками; вони транслюють радіосигнали, закодовані часом випромінювання. Приймачі обчислюють відстань до кожного супутника, порівнюючи час випромінювання й час прийому. Оскільки невідоме як положення (x, y, z) , так і часова різниця між годинниками — потрібно 4 супутники для отримання "фіксу". Початковий фікс — кілька хвилин, потім — кілька разів на секунду. GPS-вимірювання недостатньо прецизійні й точні для робототехнічних застосувань і вимагають злиття з іншими сенсорами (наприклад, IMU).

Існують також *закриті GPS-рішення* — активні або пасивні маяки з відомими локаціями. Пасивні (інфрачервоні відбивні наклейки, 2D-штрихкоди) детектуються камерами і дають позу з відомих розмірів. Активні випромінюють радіо, ультразвук або їх комбінацію — для оцінки відстані. У цій області особливо поширене ультраширокосмугове радіо (UWB).

Висновки на винос

- Більшість сенсорів робота вирішує задачу визначення пози робота або локалізації/розпізнавання об'єктів у його оточенні.
- Кожен сенсор має переваги й недоліки, що квантифікуються в його діапазоні, прецизійності, точності й пропускній здатності. Тому стійкі рішення досягаються лише через поєднання сенсорів з різними принципами роботи.
- Твердотільні сенсори (без механічних частин) можна мініатюризувати й дешево виробляти. Це уможливило появу доступних IMU і 3D-сенсорів глибини — основи локалізації та розпізнавання об'єктів у масових роботизованих системах.

Вправи

1. Лазерний сканер з кутовою роздільною здатністю 0.01 рад і максимальною дальністю 5.6 м. Яка мінімальна відстань d від робота до об'єкта шириною 1 см, щоб гарантовано вловити його хоча б одним променем? Апроксимуйте відстань між променями довжиною дуги.
2. Чому пропускна здатність ультразвукового сенсора різко падає при збільшенні динамічного діапазону, а лазерного — ні?
3. Ви проектуєте автономний електромобіль для перевезення вантажів. Ви турбуєтесь про вартість, обираючи між лазерним сканером і ультразвуковим. Дальність — 15 м. Ультразвук швидкість 300 м/с.
 - (a) Час від ультразвука до перешкоди 15 м.
 - (b) Час від лазера.
 - (c) Якщо рухаєтесь до перешкоди — який сенсор дасть ближче до реального вимірювання й чому?
4. Виберіть навчальну робот-платформу і складіть список її сенсорів.

5. Сконструйте простий range scanner з ультразвука на сервомоторі. Чи бачите прості ознаки — кути, отвори?
6. Знайдіть DIY robotics магазини в інтернеті. Які сенсори вони пропонують? Які інтерфейси?
7. Виберіть фізичний сенсор. Чи можете спроектувати експеримент для характеристики його прецизійності й точності?
8. Спроектуйте сенсор для виявлення вільного простору в посилці для e-commerce.
9. Спроектуйте автономний візок, що автоматично пристиковується до полиць у середовищі.
10. Ви проектуєте контролер для гри "Ratslife". Яку інформацію надає середовище і який сенсор потрібен?

2. Зір (Vision)

Зір — одна з найінформативніших сенсорних систем, доступних і людям, і роботам. Однак ефективна та точна обробка цієї багатой інформації досі залишається ключовим викликом галузі. Цілі цього розділу:

- ввести поняття зображення як двовимірного сигналу;
- дати інтуїцію щодо багатства інформації, прихованої в низькорівневих даних;
- ввести базові алгоритми обробки зображень — згорткові та порогові.

2.1. Зображення як двовимірні сигнали

Зображення захоплюються камерами, які містять матриці приладів зі зарядовим зв'язком (CCD) або подібних напівпровідників (наприклад, КМОП, CMOS), що перетворюють фотони на електричні сигнали. Ці матриці зчитуються попіксельно і перетворюються на цифрові значення, наприклад масив 640×480 трибайтових кортежів RGB. Дивлячись на дані у матриці, чітко видно білу плитку серед чорних у нижньому правому куті шахівниці. Вищі значення відповідають яскравішим кольорам (білий), нижчі — темнішим. Хоча плитки мають однаковий колір, фактичні значення помітно відрізняються. Має сенс думати про ці значення як про 1D-сигнал: “похідна” уздовж горизонтальних рядків укаже на області різких змін, а “частотна” гістограма зображення вкаже, як швидко змінюються значення. Області з плавними градієнтами (чорно-білі плитки) матимуть низькі частоти; області з сильними градієнтами — високочастотну інформацію.

Ця мова відкриває серії концепцій сигналів: *фільтри нижніх частот* (придушують високочастотну інформацію), *фільтри верхніх частот* (придушують низькочастотну), *смугові фільтри* (пропускають лише діапазон), аналіз частотного спектру, “згортання” зображення з іншою двовимірною функцією.

2.2. Від сигналів до інформації

Багато явищ з різним або навіть протилежним змістом виглядають подібно на низькому рівні. Наприклад, різкі зміни кольору не обов'язково означають зміну кольору поверхні. Подібні патерни виникають також від розривів глибини, дзеркальних відблисків, змін освітлення або змін орієнтації поверхні. Це — те, що робить комп'ютерний зір складною задачею.

Цей приклад ілюструє: сигнали й дані самі по собі недостатні для розуміння — потрібен *контекст*. Контекст включає не лише сусідні сигнали, а й високорівневі концептуальні знання: те, що джерела світла створюють тіні й дзеркальні відблиски, що передні об'єкти здаються більшими, тощо. Важливість концептуальних знань ілюструє рис. 2.3: обидва зображення показують ідентичний ландшафт, але

рисунок пропущено

Рис. 2.1: Шахівниця всередині МКС з астронавтом Грегорі Хамітофф. У вставці — приклад фактичних даних сенсора. Чітко видно контури білої плитки.

рисунок пропущено

Рис. 2.2: Всередині МКС (зліва); області з різкими змінами пікселів (справа). Причини: зміна властивостей поверхні (1), розриви глибини (2), дзеркальні відблиски (3), зміни освітлення/тіні (4), зміни орієнтації поверхні (5).

один здається покритим кратерами, інший — пагорбами. Знаючи, що Сонце освітлює сцену з лівого/правого боку, парадокс прояснюється — змінне освітлення робить кратери схожими на горби.

рисунок пропущено

Рис. 2.3: Знімок місця посадки Apollo 15 у різний час доби. Ландшафт однаковий, але здається покритим кратерами (зліва) або пагорбами (справа). Знаючи напрямок Сонця, ефект прояснюється. Image credit: NASA/GSFC/Arizona State University.

Більше того, концептуальних знань часто достатньо, щоб компенсувати брак низькорівневих ознак. Приклад на рис. 2.4 — далматинця можна впізнати, попри відсутність більшості його контурів.

Ці приклади ілюструють переваги й недоліки сигнально-обробного підходу. Алгоритм детектує цікаві сигнали навіть там, де ми їх не бачимо чи не очікуємо (через концептуальне зміщення). Розуміння зображення вимагає не лише низькорівневої обробки, а й розумного поєднання просторових відношень з концептуальними знаннями. Це робиться через згорткові нейромережі (??).

2.3. Базові операції з зображеннями

Базові операції — це фільтри в частотній або просторовій (інтенсивність/колір) області. Зазвичай фільтр діє в інтенсивностній області, але знання його впливу на частоту допомагає розуміти його функцію. Фільтр для виділення країв повинен пригнічувати низькі частоти (області, де значення не змінюються) і підсилювати високі (де змінюються швидко).

2.3.1. Порогові операції

Щоб знайти об'єкти певного кольору або амплітуди краю, *порогування* зображення дає бінарне зображення з "true/false" регіонами. Використовуються оператори $>$, $<$, \leq , \geq . Існують і адаптивні версії, що локально оновлюють пороги (для змін освітлення). Хоча пороговання просте, знаходження правильних значень — складна задача: значення пікселів сильно змінюються з освітленням, тож немає такого як "червоний" або "зелений" під різними умовами.

рисунок пропущено

Рис. 2.4: Зображення далматинця чітко розпізнається попри відсутність низькорівневих ознак (край лише для вух, підборіддя, частини лап).

Рис. 2.5: Зашумлене зображення до (вгорі ліворуч) і після гаусового фільтра (вгорі праворуч). Відповідні зображення країв — нижче.

2.3.2. Згорткові фільтри

Фільтр реалізується оператором *згортки* \star , що згортає функцію $f()$ з функцією $g()$:

$$f(x) \star g(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau) d\tau, \quad (2.1)$$

де $g()$ — *фільтр*. Згортка “ковзає” $g()$ вздовж $f()$, множачи їх. Оскільки зображення дискретні, згортка теж дискретна:

$$f[x] \star g[x] = \sum_{i=-\infty}^{\infty} f[i]g[x - i]. \quad (2.2)$$

Для 2D-зображень — двовимірна:

$$f[x, y] \star g[x, y] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j]g[x - i, y - j]. \quad (2.3)$$

Хоча визначення від $-\infty$ до ∞ , зображення і фільтри скінченні. Згортка комутативна:

$$f[x, y] \star g[x, y] = \sum_{i, j} f[x - i, y - j]g[i, j]. \quad (2.4)$$

Гаусове згладжування

Один із найважливіших фільтрів — *гаусів фільтр*, що має форму гаусівського дзвону:

$$g(x, y) = \frac{1}{10} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad (2.5)$$

Кожен піксель стає середнім сусідів, з власним значенням, зваженим вдвічі більше. Гаусів фільтр діє як *фільтр нижніх частот* — пригнічує високі частоти. Шум у зображенні згладжується.

Детектування країв

Детектування країв — згорткою з ядром Собеля:

$$s_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad s_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}. \quad (2.6)$$

s_x детектує вертикальні краї, s_y — горизонтальні. *Детектор країв Кенні* (Canny) запускає принаймні два таких фільтри.

рисунок пропущено

Рис. 2.6: Приклади морфологічних операторів — ерозія, дилатація та комбінації (з документації OpenCV, BSD).

рисунок пропущено

Рис. 2.7: Схема кореляції ознак між зображеннями для виокремлення тривимірної інформації з двовимірних видів.

Різниця гаусіан (DoG)

Difference of Gaussians (DoG): віднімання двох зображень, кожне відфільтроване гаусіаном різної ширини. Обидва пригнічують високочастотну інформацію; їхня різниця — *смуговий фільтр*, з якого вилучено і низькі, і високі частоти. Одне ядро зазвичай у 4–5 разів ширше за інше.

DoG також апроксимує *лапласіан гаусіана* (Laplacian of Gaussian, LoG) — суму других похідних гаусівського ядра. Одне ядро приблизно в 1.6 разів ширше за інше. DoG/LoG виокремлюють високочастотну інформацію (краї), пригнічуючи високочастотний шум.

2.3.3. Морфологічні операції

Інший клас фільтрів — *морфологічні оператори* з ядром, що описує структуру операції, і правилом зміни пікселя за оточенням. Важливі: *ерозія* (мінімум у оточенні) і *дилатація* (максимум у оточенні). Корисні для заповнення дірок у лінії або видалення шуму. Дилатація + ерозія = “Закриття” (Closing); ерозія + дилатація = “Відкриття” (Opening). Віднімання еродованого і дилатованого зображень — детектор країв.

2.4. Виокремлення структури із зору

Зір дає і *семантичну* (якості — що в сцені), і *метричну* (кількості — розміри, відстані) інформацію. Семантика нині сильно покладається на машинне навчання. Метрика — на геометричних відношеннях.

Рис. 2.7 показує відношення між двома кадрами, що спостерігають одну точку. Не розрізняємо тут просторово- і часово-корельовані випадки — це дві різні задачі: *стереозір* (дві жорстко прикріплені камери, просторово корельовані); *структура з руху* (одна камера переміщується, пара зображень корельована через матрицю перетворення). У стереозорі це перетворення — *екстринсики сенсора* (6-DoF, калібровані). У структурі з руху — рух камери, що оцінюється локалізацією.

3D-точка проектується в кадр камери:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = KT \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \quad (2.7)$$

де K — *інтринсична матриця камери* (два параметри оптичного центру і два параметри масштабування — обидва калібровані), T — перетворення між камерою і глобальними координатами. Дві проекції тієї самої точки можна обчислити

Рис. 2.8: Зліва направо: два складних об'єкти, патерн кольорових ліній та його деформація на поверхнях, реконструювана 3D-форма. З [?].

прямою тріангуляцією. З C_L як глобальної системи:

$$\begin{pmatrix} u_R \\ v_R \\ 1 \end{pmatrix} = K T_{LR} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = K T_{LR} K^{-1} \begin{pmatrix} u_L \\ v_L \\ 1 \end{pmatrix}. \quad (2.8)$$

Епіполярна лінія: точка p_{P_L} лежить на промені від центра C_L через точку. Невизначеність — глибина уздовж променя. Проекція цього променя в P_R — епіполярна лінія, вздовж якої треба шукати p_{P_R} . Пошук на лінії значно швидший за пошук у всій площині P_R .

Виокремлення метрики вимагає унікальної ідентифікації однакових точок. Просте рішення — *структуроване світло* (рис. 2.8). Інфрачервоні сенсори глибини використовують *спекл-патерн* (псевдовипадкові точки з різними відстанями). Ідентифікація однакових точок зводиться до пошуку blob-ів подібного розміру близько один до одного.

2.5. Комп'ютерний зір і машинне навчання

Алгоритми, описані тут, досі формують основу конвеєрів розуміння зображень. З появою згорткових нейромереж (??) базова обробка часто інтегрується в єдиний пайплайн, але розуміння того, що роблять згортка, морфологічні операції, пороги — залишається релевантним.

Висновки на винос

1. На відміну від сенсорів з 1, наш мозок безпосередньо обробляє 2D-інформацію. Важко “не думати” про обробку, яку ми виконуємо автоматично, доповнюючи сигнал знаннями.
2. Описані алгоритми зменшують інформацію до нижньо-вимірного простору, видаляючи шум і допоміжну інформацію.
3. Існує компроміс між трактованістю потоку даних і збереженням реальної інформації. З розвитком обчислень і алгоритмів ML сучасні системи зору об'єднують попередню обробку і розуміння в єдиний конвеєр.

Вправи

1. Нижче подано кілька “ядер” для згорткового фільтрування зображень.

1	1	1
1	2	1
1	1	1

0	-1	0
0	-1	0
0	-1	0

1	1	1
1	-4	1
1	1	1

- (а) Знайдіть ядро, що може розмивати зображення.

- (b) Які ознаки можна детектувати двома іншими ядрами?
- 2. Скільки fog-циклів потрібно для реалізації 2D-згортки? Поясніть.
- 3. Використовуйте симулятор робота з камерою у світі з простими об'єктами.
 - (a) Реалізуйте пороговання для виділення об'єкта одного кольору. Чи достатньо простого порога? Чому ні? Чи можна виділити об'єкт нижнім і верхнім порогом?
 - (b) Реалізуйте згладжування: гаусову згортку + морфологічні операції. Експериментуйте з ядрами різної ширини. Переваги/недоліки морфології vs. простий гаусів фільтр?
 - (c) Реалізуйте детектування країв (Sobel). Що ще потрібно для отримання зображення лише з краями?
- 4. Чи можете придумати алгоритм згладжування, що згладжує малий шум, але зберігає краї? Які фільтри для цього поєднаєте?
- 5. Знайдіть в інтернеті бібліотеку комп'ютерного зору. Чи реалізує всі алгоритми? Розв'яжіть завдання вище через її функції.
- 6. Симулюйте дві камери на відомій відстані в одній площині. На простих об'єктах (червоний м'яч) обчисліть їх відстань через стерео-диспаратність.

3. Локалізація

Роботи використовують сенсори й виконавчі механізми, що зазнають невизначеності. Розділ ?? описує, як кількісно оцінювати цю невизначеність функціями густини ймовірності, що пов'язують імовірність з кожним можливим результатом випадкового процесу — наприклад, з показанням сенсора або фактичною фізичною зміною актуатора. Тут *поза* робота — це складена метрика центрального значення для мобільної робототехніки і є фокусом цього розділу.

Існує багато способів локалізувати робота у його середовищі, і одометрія — лише один з них. Інший можливий спосіб — витягати високорівневі ознаки (розділ ??), як-от відстані до стіни з кількох різних сенсорів.

Як ми бачили у ??, невизначеність постійно поширюється без можливості коригувальних вимірювань. Цілі цього розділу — представити математичні інструменти й алгоритми, що дозволять реально *зменшити* невизначеність вимірювань шляхом поєднання з додатковими спостереженнями. Зокрема, у розділі розглядаються:

- використання орієнтирів (landmarks) для підвищення точності дискретної оцінки положення (*марковська локалізація і фільтр Байєса*);
- апроксимація неперервних оцінок положення (*фільтр частинок, particle filter*);
- використання *розширеного фільтра Калмана* (EKF) для оцінювання неперервного положення.

3.1. Мотиваційний приклад

Уявімо коридор з трьома дверима, дві з яких ближче одні до одних, а третя — далі по коридору (3.1). Уявімо, що ваш робот здатний детектувати двері — тобто може визначити, чи знаходиться він перед стіною, чи перед дверима. Такі ознаки можуть слугувати орієнтиром. Маючи карту цього простого середовища і жодної інформації про положення робота, ми можемо використати орієнтири, щоб різко скоротити простір можливих локацій, щойно робот пройшов одні з дверей. Один зі способів представити цю віру — описати положення робота трьома гаусівськими розподілами, кожен з яких центрований перед однією з дверей з дисперсією як функцією невизначеності, з якою робот детектує центр дверей. Це *мультигіпотезна віра* (multi-hypothesis belief): є гіпотеза про знаходження робота перед кожними дверима. Що відбувається при русі робота? Із закону поширення похибки знаємо:

1. Гаусіани, що описують 3 можливі локації робота, рухатимуться разом з ним.
2. Дисперсія кожного гаусіана зростатиме з відстанню, яку проходить робот.

Що, якщо робот доходить до інших дверей? Маючи карту, можемо тепер відобразити три гаусівські розподіли на локації трьох дверей. Оскільки всі три гаусіани зрушилися, але двері розставлені нерівномірно, лише деякі з піків збігатимуться з локаціями дверей. Якщо ми довіряємо детектору дверей набагато більше за одометричну оцінку, можемо тепер видалити всі віри, що не збігаються з дверима.

Рис. 3.1: Робот локалізує себе “детектором дверей” на відомій карті. Зверху: при зустрічі з дверима робот може бути перед будь-якими з трьох дверей. По центру: при русі праворуч гаусівські розподіли, що представляють його локацію, також зсуваються праворуч і розширюються, представляючи зростання невизначеності. Знизу: після детектування других дверей робот може відкинути гіпотези не перед дверима і набуває впевненості у своєму положенні.

Знов-таки, припускаючи, що детектор може детектувати центр дверей з певною точністю, невизначеність нашої оцінки положення обмежена тепер лише невизначеністю детектора дверей.

Все трохи складніше, якщо детектор дверей також зазнає невизначеності: є шанс, що ми перед дверима, але не помітили цього. Тоді було б помилкою видаляти цю віру. Натомість зважуємо всі віри ймовірністю того, що там *могли б* бути двері. Скажімо, наш детектор має 10% хибнопозитивних спрацювань. Тоді є 10% шанс бути в локації, що не перед дверима, навіть якщо детектор каже, що ми перед дверима. Аналогічно, з 20% хибнонегативних спрацювань — детектор каже, що дверей немає, хоча робот перед ними. Тож треба зважити всі позиції перед дверима з 20% ймовірністю та всі позиції не перед дверима з 80% ймовірністю, якщо детектор каже, що дверей немає.

3.2. Марковська локалізація

Обчислення ймовірності бути в певній локації за заданої правдоподібності спостережень — це та сама умовна ймовірність. Формальний спосіб опису — *правило Байєса* (??):

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \quad (3.1)$$

3.2.1. Оновлення сприйняття (perception update)

Як це переноситься на локалізацію? Припустимо, подія A — це знаходження в локації loc , подія B — побачити ознаку $feat$. Тоді

$$P(loc|feat) = \frac{P(loc)P(feat|loc)}{P(feat)} \quad (3.2)$$

Можемо обчислити ймовірність бути в локації loc за умови сприйняття ознаки $feat$. Це *Оновлення сприйняття*. Що нам треба знати:

1. Апріорну ймовірність бути в локації $P(loc)$.
2. Ймовірність побачити ознаку, якби ми були в цій локації: $P(feat|loc)$.
3. Ймовірність зустріти ознаку $P(feat)$.

$P(feat)$ часто виражають як $\sum_{x \in locations} P(feat|x)P(x)$ — ймовірність побачити цю ознаку для кожної можливої локації. Часто цей член беруть рівним 1, записуючи $P(loc|feat)$ як *пропорційне* чисельникові (3.2).

Апріорна ймовірність $P(loc)$ — це *модель віри*. У прикладі з трьома дверима — це значення гаусівського розподілу під відповідними дверима.

Рис. 3.2: Офісне середовище з двох кімнат, з'єднаних коридором. Накладена топологічна карта.

$P(feat|loc)$ — ймовірність побачити ознаку $feat$ за умови знаходження в loc . Якби сенсор був ідеальним, ця ймовірність — просто 1 або 0. Якщо сенсор не ідеальний — це правдоподібність детектування ознаки, якщо вона існує.

Останнє — як представляти можливі локації. У графічному прикладі 3.1 ми припустили гаусівські розподіли. Альтернатива — дискретизувати світ на сітку і обчислити правдоподібність робота бути в кожній клітинці. У 3-дверному світі логічно вибрати клітинки з шириною дверей.

3.2.2. Оновлення дією (action update)

Одне з припущень вище — що ми точно знаємо: робот зрушив праворуч. Розглянемо обробку невизначеності від руху. Одометрія — це ще один сенсор з гаусівським розподілом; якщо одометр показує 1 метр пройденої відстані, він міг пройти трохи більше або менше. Тоді апостеріорна ймовірність переходу з loc' у loc за входу одометра odo :

$$P(loc' \rightarrow loc|odo) = P(loc' \rightarrow loc)P(odo|loc' \rightarrow loc)/P(odo). \quad (3.3)$$

Це знову правило Байеса. $P(loc' \rightarrow loc)$ — апіорна ймовірність робота бути в loc' . $P(odo|loc' \rightarrow loc)$ — ймовірність отримати показання одометра odo після проходження від loc' до loc . Якщо odo розумна для відстані $loc' \rightarrow loc$, ця ймовірність висока.

Оскільки локація робота невизначена, треба обчислити апостеріорну ймовірність для всіх можливих loc' , сумуючи:

$$P(loc|odo) = \sum_{loc'} P(loc' \rightarrow loc)P(odo|loc' \rightarrow loc). \quad (3.4)$$

Це *Оновлення дією*. На практиці треба обчислювати лише для локацій, технічно досяжних з огляду на максимальну швидкість робота. Зверніть увагу, що сума відповідає згортці (??) розподілу ймовірностей локації робота з розподілом помилки одометра.

3.2.3. Приклад: марковська локалізація на топологічній карті

Ми вивчили два методи оновлення віри. По-перше, робот може використовувати зовнішні орієнтири — *оновлення сприйняття*, спирається на екстероцепцію. По-друге, робот може спостерігати власні внутрішні сенсори — *оновлення дією*, спирається на пропріоцепцію. Поеднання називається *марковською локалізацією*. Думайте про оновлення дією як про збільшення невизначеності, а оновлення сприйняттям — як про її зменшення.

Як приклад — одна з перших успішних реальних робот-систем, що використовувала марковську локалізацію в офісному середовищі [?]. Офіс складається з двох кімнат і коридору, що моделюється *топологічною картою* (3.2). У топологічній карті області, де може перебувати робот, моделюються вершинами, а проходи між ними — ребрами. Локація робота — розподіл ймовірностей за вершинами цього графа.

Робот має такі сенсорні здатності:

	Стіна	Закр. двері	Відкр. двері	Відкр. коридор	Фое
Нічого не виявлено	70%	40%	5%	0,1%	30%
Виявлено закр. двері	30%	60%	0%	0%	5%
Виявлено відкр. двері	0%	0%	90%	10%	15%
Виявлено відкр. коридор	0%	0%	0,1%	90%	50%

Табл. 3.1: Умовні ймовірності детектування ознак роботом Dervish у лабораторії Стенфорду.

- Детектувати закриті двері ліворуч або праворуч.
- Детектувати відкриті двері ліворуч або праворуч.
- Детектувати, чи це відкритий коридор.

Сенсори ненадійні. Дослідники експериментально знайшли ймовірності отримати певну сенсорну відповідь для конкретних фізичних положень (3.1).

Наприклад, успіх детектування закритих дверей лише 60%, тоді як фое виглядає як відкриті двері в 15% випадків. Це відповідає умовній ймовірності детектування ознаки в локації.

Початковий розподіл віри: $p('1-2') = 0,8$, $p('2-3') = 0,2$. Робот їде, доки не повідомить: “відкритий коридор ліворуч і відкриті двері праворуч”. Це насправді локація 2, але робот міг бути будь-де. Наприклад, є 10% шанс, що “відкриті двері” — насправді відкритий коридор (тоді робот у локації 4). Як обчислити новий розподіл? Описуються можливі траєкторії з відповідними ймовірностями (див. оригінал для детальної арифметики). Підсумовуючи апостеріорну ймовірність за всіма траєкторіями, отримуємо нову віру.

3.3. Фільтр Байєса (Bayes filter)

Ми побачили, як формально включати сенсорні вимірювання в оцінку позиції через правило Байєса. Тепер дамо алгоритм локалізації через мультигіпотезний ітеративний процес, що не залежить від конкретного класу моделі руху або сенсорів (наприклад, гаусівських моделей шуму, що їх використовують фільтри Калмана).

Формалізуємо модель руху як $P(x'|x, u)$ — ймовірність бути в стані x' за умови, що ми були в стані x і виконали дію u . Модель сенсора: $P(z|x)$ — ймовірність побачити спостереження z , якщо ми у стані x . Ймовірність бути у стані x : $P(x)$.

Мета фільтра Байєса — оцінити стан робота з часом (x_t , де t — крок часу) за історією дій і спостережень. Обчислюємо апостеріорну ймовірність — *віру*:

$$Bel(x_t) = P(x_t | u_1, z_1, u_2, z_2, \dots, u_t, z_t).$$

За марковським припущенням (поточний стан залежить лише від попереднього стану x_{t-1} і дії u_t):

$$P(x_t | x_{0:t-1}, z_{1:t-1}, u_{1:t}) = P(x_t | x_{t-1}, u_t),$$

і

$$P(z_t | x_{0:t}, z_{1:t-1}, u_{1:t}) = P(z_t | x_t).$$

Рис. 3.3: Марковська локалізація на сітці. Лівий стовпець: правдоподібність бути у конкретній клітинці як рівень сірого (темне = висока). Правий стовпець: фактична локація робота. Стрілки показують попередній рух. Спочатку положення робота невідоме, але зареєстрований рух угору робить локації у верхній частині карти імовірнішими. Після зустрічі зі стіною положення подалі від стін стають малоімовірними. Після рухів праворуч і вниз можливі положення стискаються до невеликої області.

Виводимо рекурсивне визначення віри:

$$Bel(x_t) = P(x_t | u_1, z_1, \dots, u_t, z_t), \quad (3.5)$$

$$Bel(x_t) = c \cdot P(z_t | x_t, u_1, z_1, \dots, u_t, z_t) \cdot P(x_t | u_1, z_1, \dots, u_t), \quad (3.6)$$

$$Bel(x_t) = c \cdot P(z_t | x_t) \cdot P(x_t | u_1, z_1, \dots, u_t), \quad (3.7)$$

$$Bel(x_t) = c \cdot P(z_t | x_t) \cdot \sum_{x_{t-1} \in X} P(x_t | u_t, x_{t-1}) \cdot Bel(x_{t-1}). \quad (3.8)$$

Це підставове рівняння дозволяє виконувати оновлення віри за станом, інкорпоруючи сенсорне вимірювання та/або прогноз руху. Алгоритм фільтра Байєса:

```

BayesFilter(Bel, d, X):
  while d не порожня:
    c = 0
    if d[0] – сенсорне вимірювання:
      z = d.pop(0)
      for x ∈ X:
        Bel'(x) = P(z|x) Bel(x)
        c += Bel'(x)
      for x ∈ X:
        Bel'(x) = c-1 Bel'(x)
    elif d[0] – дія:
      u = d.pop(0)
      for x ∈ X:
        Bel'(x) = ∑xt-1 P(x|u, xt-1) · Bel(xt-1)
    Bel = Bel'
  return Bel

```

Ця потужна ідея ітеративного інкорпорування сенсорних вимірювань і прогнозів руху лежить в основі цілої сім'ї методів оцінювання стану.

3.3.1. Приклад: фільтр Байєса на сітці

Замість грубої топологічної карти можна моделювати середовище тонкою сіткою. Кожна клітинка позначена ймовірністю знаходження робота в ній (3.3). Робот детектує стіни з певною впевненістю — наприклад, короткодістанційним ультразвуковим сенсором.

3.4. Фільтр частинок (Particle Filter)

Хоча марковська локалізація на сітці може давати переконливі результати, вона обчислювально дуже дорога, особливо коли середовище велике, а роздільна здатність сітки — тонка. Це частково тому, що ми мусимо переносити вперед імовірність бути у певній локації для кожної клітинки сітки, незалежно від того, наскільки мала ця імовірність. Елегантне рішення — *фільтр частинок*. Він працює так:

1. Представити положення робота N частинками, випадково розподіленими навколо оціненого початкового положення. Або гаусівські розподіли навколо початкових оцінок, або рівномірний розподіл (3.4).
2. Щоразу, коли робот рухається, рухаємо кожну частинку так само, але додаючи шум так, як спостерігали б на реальному роботі. Без оновлення сприйняття частинки розходяться все далі.
3. При події сприйняття оцінюємо кожну частинку моделлю сенсора: яка була б імовірність такої події сприйняття, як ми спостерігали, у цій локації? За правилом Байєса оновлюємо положення кожної частинки.
4. Час від часу або під час подій сприйняття, що роблять певні частинки неможливими, частинки з надто низькою імовірністю видаляються, а ті з найвищою — реплікуються (resampling).

рисунок пропущено

Рис. 3.4: Приклад фільтра частинок. Можливі положення і орієнтації спочатку рівномірно розподілені. Частинки рухаються за моделлю руху робота. Частинки, які вимагали б руху крізь стіну за відсутності події сприйняття стіни, видаляються (зірочки). Після події сприйняття частинки, надто далекі від стіни, стають надто малоімовірними і ресемплюються в околі стіни. З часом фільтр частинок сходиться.

Головна перевага фільтрів частинок: вони — *непараметричні* оцінювачі довільних розподілів імовірностей, здатні обробляти нелінійні функції, що недоступні фільтру Калмана.

3.5. Розширений фільтр Калмана (EKF)

На відміну від лінійних моделей KF, у EKF моделі переходу станів і спостережень не мусять бути лінійними — лише диференційовними. Крок прогнозу дії:

$$\hat{x}_{k'|k-1} = f(\hat{x}_{k-1}, u_{k-1}) \quad (3.9)$$

де $f()$ — функція попереднього стану x_{k-1} і керуючого входу u_{k-1} . Гарний приклад — одометричне оновлення; $f()$ описує пряму кінематику робота, x_k — його положення, u_k — задану швидкість коліс.

Коваріаційна матриця положення:

$$P_{k'|k-1} = \nabla_{x,y,\theta} f P_{k-1|k-1} \nabla_{x,y,\theta} f^T + \nabla_{\Delta r,l} f Q_{k-1} \nabla_{\Delta r,l} f^T, \quad (3.10)$$

де Q_k — коваріація процесного шуму (проковзування коліс), якобіани прямої кінематики беруться відносно положення робота (x, y, θ) і відносно проковзування лівого/правого коліс.

Крок оновлення сприйняттям:

$$\hat{\mathbf{x}}_{k|k'} = \hat{\mathbf{x}}_{k'|k-1} + \mathbf{K}_{k'} \tilde{\mathbf{y}}_{k'}, \quad (3.11)$$

$$\mathbf{P}_{k|k'} = (\mathbf{I} - \mathbf{K}_{k'} \mathbf{H}_{k'}) \mathbf{P}_{k'|k-1}. \quad (3.12)$$

Розраховуємо все двічі: спершу оновлюємо з $k-1$ до проміжного k' за рухом, потім — до k за сприйняттям. Додаткові змінні:

1. Інновація $\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})$.

2. Коваріація інновації $\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k$.

3. (Близьке до оптимального) калманівське підсилення $\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1}$.

$h()$ — модель спостереження, \mathbf{H} — її якобіан, \mathbf{R}_k — коваріація шуму спостереження.

3.5.1. Одометрія через фільтр Калмана

Покажемо, як мобільний робот з лазерним сканером і картою середовища може коригувати оцінку положення оптимально, спираючись на ненадійну одометрію і ненадійне сприйняття.

1. Прогноз. Припускаємо знайомство з обчисленням $\hat{\mathbf{x}}_{k'|k-1} = f(x, y, \theta)^T$ і його дисперсії $\mathbf{P}_{k'|k-1}$. Коваріаційна матриця проковзування коліс:

$$\mathbf{Q}_{k-1} = \begin{bmatrix} k_r |\Delta s_r| & 0 \\ 0 & k_l |\Delta s_l| \end{bmatrix}, \quad (3.13)$$

де $\Delta s_l, \Delta s_r$ — рух лівого/правого коліс, k_l, k_r — сталі.

2. Спостереження. Припустимо, що детектуємо лінійні ознаки $\mathbf{z}_{k,i} = (\alpha_i, r_i)^T$, де α і r — кут і відстань лінії в системі координат робота. Ці ознаки мають дисперсії $\sigma_{\alpha,i}, \sigma_{r,i}$, що формують діагональ \mathbf{R}_k . Спостереження — матриця 2×1 .

3. Оновлення вимірюваннями. Припускаємо, що однозначно ідентифікуємо лінії, які бачимо, і отримуємо їхнє реальне положення з карти. Ознаки в карті зберігаються в глобальних координатах, тому перетворюємо їх у те, як їх бачив би робот. З $\hat{\mathbf{x}}_k = (x_k, y_k, \theta_k)^T$ і відповідним записом $\mathbf{m}_i = (\alpha_i, r_i)$:

$$h(\hat{\mathbf{x}}_{k|k-1}) = \begin{bmatrix} \alpha_{k,i} \\ r_{k,i} \end{bmatrix} = h(\mathbf{x}, \mathbf{m}_i) = \begin{bmatrix} \alpha_i - \theta \\ r_i - (x \cos \alpha_i + y \sin \alpha_i) \end{bmatrix} \quad (3.14)$$

і обчислюємо якобіан \mathbf{H}_k як часткові похідні.

4. Зіставлення. Маємо вимірювання \mathbf{z}_k і прогноз $h(\hat{\mathbf{x}}_{k|k-1})$. Обчислюємо інновацію

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}). \quad (3.15)$$

5. Оцінювання. Маємо всі складники для кроку оновлення сприйняттям:

$$\hat{x}_{k|k'} = \hat{x}_{k'|k-1} + K_{k'} \tilde{y}_{k'}, \quad (3.16)$$

$$P_{k|k'} = (I - K_{k'} H_{k'}) P_{k'|k-1}. \quad (3.17)$$

Це оновлення положення зливає одометричний вхід та інформацію з ознак середовища з урахуванням їх дисперсій. Якщо дисперсія попереднього положення велика (бо ми не знаємо, де ми), а дисперсія вимірювання мала (наприклад, GPS або чітко розпізнаваний символ на стіні), фільтр Калмана надає більшої ваги сенсору. Якщо сенсори погані (наприклад, не можемо розрізнити лінії/стіни), більше ваги — одометрії.

Коваріація інновації та підсилення Калмана:

$$S_k = H_k P_{k|k-1} H_k^\top + R_k, \quad (3.18)$$

$$K_k = P_{k|k-1} H_k^\top S_k^{-1}. \quad (3.19)$$

3.6. Підсумок: ймовірнісна локалізація на карті

Щоб локалізувати робота на карті, виконуємо такі кроки:

1. Обчислюємо оцінку нового положення прямою кінематикою та знанням швидкостей коліс, доки робот не натрапить на унікально ідентифіковану ознаку.
2. Обчислюємо відносне положення ознаки (стіна, орієнтир, маяк) до робота.
3. Використовуємо знання глобального положення ознаки для прогнозу того, що робот має бачити.
4. Обчислюємо різницю між тим, що робот бачить, і тим, що, на його думку, він має бачити (наприклад, через фільтр Калмана).
5. За результатом (4) оновлюємо віру, зважуючи кожне спостереження на його дисперсію.

Висновки на винос

- Без додаткових сенсорів і з шумною одометрією поширення похибки веде до постійно зростаючої невизначеності положення робота — незалежно від використання марковської локалізації або фільтра Калмана.
- Коли робот може сприймати ознаки з відомими локаціями, правило Байєса дозволяє оновити апостеріорну ймовірність можливого положення. Ключова ідея: умовну ймовірність бути в певній позиції за заданого спостереження можна вивести з правдоподібності зробити це спостереження за заданою позиції.
- Повне рішення для дискретних локацій — *марковська локалізація*.
- *Розширений фільтр Калмана* — оптимальний спосіб злити спостереження різних випадкових величин з гаусівським розподілом.

- Можливі випадкові величини: оцінка положення робота з одометрії та спостереження статичних маяків з відомою локацією (але невизначеним сенсингом) у середовищі.
- Для використання підходу потрібні диференційовні функції, що пов'язують вимірювання зі змінними стану, а також оцінка коваріаційної матриці сенсорів.
- Апроксимація, що поєднує переваги марковської локалізації (множинні гіпотези) і фільтра Калмана (неперервне представлення) — *фільтр частинок*.

Вправи

1. Припустіть, що стеля обладнана інфрачервоними маркерами, які робот ідентифікує з певною впевненістю. Розробіть ймовірнісну схему локалізації для обчислення $p(\text{маркер}|\text{показання})$.
 - (a) Виведіть вираз для $p(\text{маркер}|\text{показання})$, припускаючи, що маєте оцінки $p(\text{показання}|\text{маркер})$ та $p(\text{маркер})$.
 - (b) Припустіть: ймовірність правильної ідентифікації маркера — 90%, помилкова — 10%, ви не бачите маркер під собою — 50%. Розгляньте вузький коридор з 4 маркерами. Точно відомо, що ви стартували з боку маркера 1 і рухаєтесь у прямій лінії праворуч. Перше показання — “маркер 3”. Обчисліть ймовірність бути дійсно під маркером 3.
 - (c) Чи може робот бути під маркером 4?

4. Побудова карти (Mapping)

Побудова карти — це процес створення представлень середовища для подальшого використання алгоритмами автономії або для інформування людей. Карти інформують прийняття рішень алгоритмами планування і керування, надаючи інформацію про поверхні й перешкоди, об’єкти, з якими робот може взаємодіяти, або топологічну інформацію — як кімнати з’єднані одна з одною. Якщо карти середовища вже надано, роботи можуть будувати плани на них без необхідності будувати карту самостійно; вони можуть навіть локалізуватися на цих картах, просто збираючи інформацію *in situ* і звіряючи з апіорними картами. Побудова карти також дає людям уявлення про те, що робот *бачить*, орієнтуючи проєктувальників або операторів робота. Тому інформація карти — критичний ключ до роботи в реальних середовищах.

Існує два класи величин: *метричні* (наприклад, фізичні розміри середовища) та *семантичні* (наприклад, тип кімнати або об’єкти інтересу). Метрична інформація зазвичай отримується геометричними техніками (Розділ 1), семантична — через машинне навчання. У цьому розділі зосередимось переважно на метричній побудові карти.

Дистанційні сенсори (range sensors) виявилися найефективнішими для автономії. Дальніметричні дані збираються у “скани” — списки точок, які називають *хмарою точок* (point cloud). Хмари точок дозволяють пряму побудову 3D-моделі середовища. Сенсори, що видають хмари точок, точні й дедалі поширені: Velodyne 3D-LiDAR (64 скануючих лазери в одному корпусі) був ключем у DARPA Grand Challenge; жодна команда DARPA Subterranean Challenge не обходилася без LiDAR. Однак більшість LiDAR-ів використовують обертові лазерні масиви — рухомий сенсор збирає інформацію під різними кутами обертання, що *аліасує* вимірювання з рухом сенсора. Це усувається, якщо рух сенсора повільний. Існують і сенсори без цього обмеження — наприклад, RGB-D-камери (колір + глибина).

Задача побудови карти варіюється від тривіальної (за ідеальної локалізації) до доволно складної — еквіваленту картографії, де задачі локалізації та побудови тісно переплітаються. Задача *одночасної локалізації та побудови карти* (SLAM) розглядається в розділі 5; в 4.2 описано один з ключових алгоритмів — зіставлення сканів.

Хмара точок дозволяє підгонку ліній і площин методом RANSAC, що може слугувати ознаками в EKF-локалізації, а також для покращення одометрії, детектування замикань циклу та побудови карт. Хмари точок можна ймовірно злити у воксельні сітки зайнятості та щільні поверхневі представлення. Цілі розділу:

- ввести *алгоритм Iterative Closest Point* (ICP) для зіставлення хмар точок;
- показати, як ICP можна покращити, надавши початкові гіпотези через RANSAC;
- використати хмари точок для генерації щільних карт через сітки зайнятості;
- продемонструвати RGB-D-побудову карти.

4.1. Представлення карти

Щоб планувати шлях, треба представити середовище цифрово. Розрізняють два підходи: *дискретні* і *неперервні* апроксимації. У дискретній — карта поділена на рівні (сітка, шестикутна) або різні за розміром (кімнати) секції. Останні — *топологічні карти* або *графові карти*. Кожна область відповідає вершині, ребра з'єднують області, між якими робот може переміщатись. Наприклад, road-map — це топологічна карта з перехрестями-вершинами і дорогами-ребрами (6.2). У пам'яті граф зберігається списком/матрицею суміжності або інцидентності.

Неперервна апроксимація вимагає визначення внутрішніх (перешкоди) і зовнішніх меж (зазвичай полігони); шляхи кодуються послідовностями точок з дійсними числами. Попри переваги пам'яті, дискретні карти домінують у робототехніці.

Часто комбінують дискретні та неперервні: GPS-roadmaps зберігаються як топологічні з GPS-координатами вершин, але можуть містити накладення аерофото і вуличної фотографії.

4.2. Iterative Closest Point (ICP) для розрідженого зіставлення

У найпростішій формі карту можна створити зі скан-зрізів 2D-дальніметричних даних (наприклад, з лазерного сканера). За відсутності точної оцінки руху між двома вимірюваннями (з одометрії чи IMU) виклик — асоціювати послідовні скани.

Стандартне рішення — *Iterative Closest Point* (ICP). Запропоновано на початку 1990-х для реєстрації 3D-дальніметричних даних до CAD-моделей. У робототехніці ICP застосовується для зіставлення сканів 2D-лазерних сканерів: перетворення, що мінімізує похибку між двома послідовними знімками середовища, пропорційне руху робота. Складність: незрозуміло, які точки утворюють пари, які точки — викиди (через шумні сенсори), і які точки треба відкинути як неперекривні. Зшивання серії знімків теоретично дає 2D-карту середовища. Однак похибка між знімками — як у одометрії — накопичується.

ICP працює і в 3D — дозволяє виводити зміну 6D-пози камери і створювати 3D-карти. ICP корисний для ідентифікації об'єктів з 3D-бази та для зшивання послідовних дальніметричних зображень у 3D-карту [?].

Варіанти ICP розкладаються на шість кроків:

1. Відбір точок в одній або обох хмарах/сітках.
2. Зіставлення/пара з зразками в іншій хмарі/сітці.
3. Зважування відповідних пар.
4. Відкидання певних пар.
5. Присвоєння метрики похибки на основі пар.
6. Мінімізація метрики похибки.

Зіставлення точок. Ключовий крок — зіставити точку з відповідною в іншому вимірюванні. Наприклад, 67-й промінь лазерного сканера потрапляє у стіну;

після зрушення на 10 см найближче попадання може здійснити 3-й промінь. Прямого потрапляння в ту саму точку зазвичай не буває — ненульова похибка навіть для оптимального паросполучення. Поширені методи: найближча точка в іншій хмарі; перетин нормалі точки-джерела з поверхнею-призначенням (для зіставлення з сіткою); SIFT-ознаки за візуальним виглядом. Пошук прискорюється представленням хмари точок у k -вимірному дереві.

Зважування пар. Деякі пари кращі за інші. Підходи: більше ваги парам з меншою відстанню; врахування кольору (RGB-D) або відстаней SIFT-ознак; врахування очікуваного шуму (лазерний сканер точніший ортогонально до площини, ніж під гострим кутом).

Відкидання пар. Викиди — від сенсорного шуму чи неповного перекриття. Стандарт — відкидати пари, де одна з точок лежить на межі хмари (бо вони, ймовірно, відповідають точкам у неперекривних областях). Часто також — відкидання пар із надто великою відстанню (пороговий еквівалент відстань-зваженого підходу).

Метрика похибки і мінімізація. Після відбору, зіставлення, зважування й відкидання — потрібна метрика похибки. Простий вибір — сума квадратів відстаней між парами. Часто розв’язується аналітично. Нехай

$$A = \{a_1, \dots, a_n\}, \quad (4.1)$$

$$B = \{b_1, \dots, b_n\} \quad (4.2)$$

— хмари точок у \mathbb{R}^n . Мета — знайти вектор $t \in \mathbb{R}^n$, що мінімізує функцію похибки $\phi(A + t, B)$:

$$\phi(A + t, B) = \frac{1}{n} \sum_{a \in A} \|a + t - N_B(a + t)\|^2, \quad (4.3)$$

де $N_B(a + t)$ — найближчий сусід $a + t$ у B . Значення t впливає на паросполучення, тому застосовується ітеративний підхід: спочатку $t = 0$, встановлюємо пари, обчислюємо δt , зрушуємо всі точки у A на δt , починаємо знову. Триває до локального мінімуму.

Поширеною стала метрика “точка-до-площини” (point-to-plane) — сума квадратів відстаней від кожної точки-джерела до площини, що містить точку-призначення і перпендикулярна до її нормалі. Особливо доречно при зіставленні хмари точок з сіткою/CAD-моделлю. У такому випадку — оптимізатор типу Левенберга-Марквардта.

4.3. Octomap: щільна побудова карти вокселями

Найпоширеніша карта для перешкод — *сітка зайнятості* (occupancy grid map). Середовище дискретизується на *воксели* довільної роздільної здатності (наприклад, 1 см × 1 см), і на них позначаються перешкоди. У ймовірнісній сітці зайнятості клітинки маркуються *імовірністю* містити перешкоду — особливо важливо при невизначеності положення робота. Недоліки сітки: великі вимоги до пам’яті та обчислювального часу для проходження великих структур.

Розв’язання — зберігати сітку як *k -вимірне дерево* (k -d tree). k -d дерево рекурсивно ділить середовище на k частин (наприклад, при $k = 4$ область ділиться на

рисунок пропущено

Рис. 4.1: Сітка зайнятості та відповідне квадрандеро (квадтрі — k -d дерево).

рисунок пропущено

Рис. 4.2: Схема генерації TSDF з 2D-дальніметричних даних. Воксели заповнюються лише відстанями в межах “відстані усічення”.

4 шматки), за критерієм підрозбиття (як-от ділити лише якщо область заповнена на 5–95%). Кожен шматок може ділитися далі — до максимально допустимої роздільної здатності або поки критерій не виконано. Не всі вершини треба ділити до найменшої роздільної здатності — лише області з перешкодами. Для 3D-даних — *Octree* (8-вимірне дерево).

Значення кожного запису в k -d дереві — ймовірність зайнятості вокселя. Може обчислюватися моделями сенсора: абсолютний поріг або ймовірнісна модель з false-positive і false-negative rate.

4.4. RGB-D mapping: щільна побудова поверхневих карт

Octomap ефективний для планування, але має недоліки: фіксована роздільна здатність вокселів не дозволяє розрізняти малі перешкоди (вони виглядатимуть більшими); високочастотна інформація всередині вокселя (наприклад, кривизна поверхні) — нерозрізнима.

Розв’язання — заповнювати воксели не ймовірністю зайнятості, а *найімовірнішою відстанню до найближчої поверхні*. Якщо поверхня лежить за вокселем (далі від сенсора), відстань додатна; якщо *перед* — від’ємна. Це *signed distance field* (SDF) — *поле зі знаком відстані*. SDF генерується слідуванням за відстанню вздовж променя до поверхні і записом значення у воксель, з ймовірнісним оновленням при кожному кадрі з каналу глибини. SDF неявно представляє поверхню. *Усічене SDF* (Truncated SDF, TSDF): воксели, значення яких перевищили б *porіg усічення* (truncation distance), залишаються незаповненими — економія пам’яті та прискорення алгоритмів реконструкції.

TSDF природно представляє багатомасштабні перешкоди і має додаткову перевагу для алгоритмів планування — *дає відстань до найближчої перешкоди без посередньо*. Це корисно як ризик-метрика. Недоліки: 1) вимагає високоточної інформації про позу (часто ICP на кожен скан); 2) неявне представлення поверхонь не дає прямих 3D-візуалізацій — потрібен рендерер (наприклад, методом [?], що дає карти, як на 4.3). Результат бувають надзвичайно високої роздільної здатності навіть на грубій вокселізації. З RGB можна створювати повні 3D-walkthrough середовища.

Проблема постійного використання ICP — похибки кожного перетворення поширюються у процес генерації карти у вигляді дрейфу карти. SLAM (5) виправляє попередні похибки при детектуванні замикання циклу, але оновлення TSDF при замиканні циклу вимагає неперервного збереження і глобального перерахунку

рисунок пропущено

Рис. 4.3: Злита хмара точок з проходу офісним середовищем за допомогою “Kintinuous”. Зображення: John Leonard.

всіх даних — велика ціна.

Оскільки ICP працює лише, коли обидві хмари вже близько вирівняні (що не завжди справедливо для швидко рухомого робота з шумними сенсорами; Xbox Kinect має похибку 3 см на кілька метрів дальності, тоді як лазерні сканери — міліметри), RGB-D Mapping використовує RANSAC для початкового перетворення. RANSAC: вгадує можливі перетворення для 3 пар SIFT-ознак, рахує число “інлаєрів” при зіставленні двох хмар.

Висновки на винос

1. Виклик побудови карти середовища походить від невизначеності і в локалізації, і в сенсингу.
2. Техніки подолання невизначеності в локалізації та сенсингу можна, своєю чергою, використати для зростання впевненості в перших. Наприклад, при наявності надійних ознак (кутів, стін) результати ICP можуть покращити одометричні оцінки.
3. За відсутності надійної локалізації задача побудови карти перетворюється на SLAM (5).

Вправи

1. Симулюйте LiDAR-сенсор у симуляторі. Розробіть структуру сіткової карти, що дозволить намалювати положення робота. Використовуйте сталий кутовий зсув LiDAR і позу робота для обчислення координат карти кожного показання.
2. Запустіть симульованого робота на смузі перешкод і запишіть карту симульованим LiDAR. Реалізуйте ICP для оцінки трансляції між послідовними сканами і порівняйте з одометричною оцінкою.
3. Використовуйте ICP для покращення оцінки стану робота при різних настройках проковзування коліс.

5. Одночасна локалізація та побудова карти (SLAM)

Роботи можуть відстежувати своє положення й орієнтацію (*позу*), використовуючи модель шуму силового агрегату та пряму кінематику для поширення цієї похибки у просторову функцію густини ймовірності (??). Якщо робот бачить унікально ідентифіковані орієнтири з відомими положеннями, дисперсія розподілу зменшується. Це досягається для дискретних локацій правилом Байєса (3.2) і для неперервних — розширеним фільтром Калмана (3.5). Ключова ідея: кожне спостереження зменшує дисперсію оцінки положення. Фільтр Калмана оптимально зливає два спостереження, зважуючи їх обернено пропорційно дисперсії. Досі ми припускали, що локації орієнтирів відомі. У цьому розділі введемо:

- поняття *коваріації* (тобто того, що описують недіагональні елементи коваріаційної матриці);
- як одночасно оцінювати положення робота і положення орієнтирів на карті (*одночасна локалізація та побудова карти, SLAM*).

5.1. Вступ

SLAM — наріжна задача автономної мобільної робототехніки. Робот, доставлений у невідому локацію, повинен мати змогу досліджувати область і будувати метрично точні карти й оцінки пози, спираючись лише на бортову сенсоріку. Це корисно для будь-якого польового робота — наземного, позаземного, підводного або в недослідженому штучному середовищі.

5.1.1. Орієнтири

Сенсорні вимірювання зазвичай зводять до *ознак* (??). Ознаки, які можна надійно зіставити між вимірюваннями, представляють когерентні структури реального світу (стіна, кут) — це *орієнтири*: геометричні об'єкти, що інформують рух у світі.

5.1.2. Особливий випадок I: один орієнтир

Розглянемо середовище з одним орієнтиром невідомого положення. Робот вимірює відносну дальність і кут до нього з певною дисперсією. Положення цього вимірювання $m_i = [\alpha_i, r_i]$ у глобальних координатах невідоме, але обчислюється за оцінкою положення робота \hat{x}_k . Дисперсія компонентів m_i — дисперсія положення робота плюс дисперсія спостереження.

Рухаючись до орієнтира, робот отримує додаткові спостереження. Хоча невизначеність положення зростає з рухом, орієнтир m_i дозволяє зменшити дисперсію апіорного положення (поки орієнтир нерухомий). Повторні спостереження з різних кутів і відстаней покращують оцінку положення орієнтира, а отже й положення робота.

Ймовірнісне оновлення виконується каркасом EKF (3.5): дисперсія оцінки положення орієнтира додається до дисперсії процесу сенсоріки.

5.1.3. Особливий випадок II: два орієнтири

Розглянемо карту з двома орієнтирами. Робот зберігає обидва в карті, але другий — з вищою дисперсією через зростання позиційної дисперсії з часом. Хоча спостереження двох орієнтирів незалежні, відношення їхніх дисперсій залежить від траєкторії: якщо робот рухається по прямій між ними — дисперсії близькі; якщо виконує серію поворотів — різниця значна.

Уявімо: робот довго їде і накопичує велику дисперсію. Потім за короткий час спостерігає обидва орієнтири послідовно. Тоді функція густини ймовірності *відстані між орієнтирами* буде вузькорозподіленою. Це — *коваріація* двох випадкових величин. У теорії ймовірностей коваріація вимірює, наскільки дві змінні змінюються разом. Коваріація між локаціями двох орієнтирів, відвіданих одразу один за одним, набагато більша, ніж між тими, що далеко розставлені. Це не означає більшої невизначеності — лише кореляцію.

Тож можна використати коваріацію між орієнтирами для ретроспективного коригування оцінок: якщо робот повертається до першого орієнтира, він зменшує дисперсію власної оцінки положення. Знаючи, що він не далеко проїхав з моменту спостереження останнього орієнтира, він може скоригувати і його оцінку.

5.2. Коваріаційна матриця

При оцінюванні величин з кількома змінними (наприклад, положення робота: x, y, θ) матрична нотація зручна. У поширенні похибки записуємо дисперсії вхідних величин на діагональ коваріаційної матриці. Для диференціального колісного робота дисперсії лівого/правого коліс — у 2×2 матрицю, отримана 3×3 матриця має $\sigma_x, \sigma_y, \sigma_\theta$ на діагоналі. Усі інші елементи — нулі, бо дисперсії коліс — *незалежні випадкові процеси*: коваріація між ними — нуль. Але для положення робота це не так: невизначеність одного колеса впливає на всі вихідні випадкові величини одночасно — це виражається ненульовими коваріаціями (поза діагоналлю).

У SLAM ведемо пози всіх відомих роботові орієнтирів у вектор-стовпчику. Їхні дисперсії — на діагоналі великої коваріаційної матриці. При послідовному відвідуванні дисперсії корелюють — недіагональні елементи стають ненульовими.

5.3. EKF SLAM

Ключова ідея EKF SLAM: розширити вектор стану з положення робота до вектора, що містить положення всіх орієнтирів. Замість

$$\hat{\mathbf{x}}_{k'|k-1} = (x, y, \theta)^T \quad (5.1)$$

маємо

$$\hat{\mathbf{x}}_k = (x, y, \theta, \alpha_1, r_1, \dots, \alpha_N, r_N)^T, \quad (5.2)$$

для N орієнтирів — це $(3 + 2N) \times 1$ вектор. Оновлення дією (прогноз) аналогічне випадку відомих орієнтирів. Коваріаційна матриця — $(3 + 2N) \times (3 + 2N)$, на діагоналі — дисперсії положення робота і кожного орієнтира.

Оновлення сприйняттям: спостерігається один орієнтир за раз. Якщо робот спостерігає кілька — виконуються кілька послідовних оновлень сприйняттям. Тільки ті елементи вектора спостереження $(3 + 2N) \times 1$, що відповідають спостережуваному орієнтиру, ненульові.

5.3.1. Алгоритм

Ініціалізація

Початковий вектор стану — нулі:

$$\mathbf{x}_0 = (0, 0, 0)^T, \quad (5.3)$$

коваріація — маленьке число ϵ :

$$\mathbf{P}_0 = \begin{bmatrix} \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{bmatrix}. \quad (5.4)$$

(Не можна точно знати величину; також нульова матриця необертна.)

Оновлення

Двокроковий процес: спершу прогноз, потім сприйняття.

Прогнозне оновлення. Якщо f — нелінійна модель переходу і \mathbf{u} — керуючі входи:

$$\hat{\mathbf{x}}_{k'|k-1} = f(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}). \quad (5.5)$$

Коваріаційне прогнозне оновлення:

$$\hat{\mathbf{P}}_{k'|k-1} = \mathbf{F}_{\hat{\mathbf{x}}_{k-1}} \mathbf{P}_{k-1} \mathbf{F}_{\hat{\mathbf{x}}_{k-1}}^T + \mathbf{N}, \quad (5.6)$$

де $\mathbf{F}_{\hat{\mathbf{x}}_{k-1}} = \partial f(\mathbf{x}, \mathbf{u}) / \partial \mathbf{x}|_{k-1}$ — яacobіан моделі переходу за станом, \mathbf{N} — коваріація шуму актуаторів. Лише стан *робота* (а не положення орієнтирів) залежить від k — тож більшість $\hat{\mathbf{P}}_{k'|k-1}$ не оновлюється на цьому кроці.

Оновлення сприйняттям. Функція спостереження $h(\mathbf{x})$ нелінійна, з адитивним шумом коваріації \mathbf{R} . Вимірювання — \mathbf{y}_k . Яcobіан спостереження — $\mathbf{H}_{\hat{\mathbf{x}}_{k'}} = \partial h(\mathbf{x}) / \partial \mathbf{x}|_{k'}$. Оновлення стану:

$$\hat{\mathbf{x}}_{k|k-1} = \hat{\mathbf{x}}_{k'|k-1} + \mathbf{K}_{k'}(\mathbf{y}_k - h(\hat{\mathbf{x}}_{k'|k-1})), \quad (5.7)$$

$$\hat{\mathbf{P}}_{k|k-1} = \hat{\mathbf{P}}_{k'|k-1} - \mathbf{K}_{k'} \mathbf{Z}_{k'} \mathbf{K}_{k'}^T, \quad (5.8)$$

де

$$\mathbf{Z}_{k'} = \mathbf{H}_{\hat{\mathbf{x}}_{k'}} \hat{\mathbf{P}}_{k'|k-1} \mathbf{H}_{\hat{\mathbf{x}}_{k'}}^T + \mathbf{R}, \quad (5.9)$$

$$\mathbf{K}_{k'} = \hat{\mathbf{P}}_{k'|k-1} \mathbf{H}_{\hat{\mathbf{x}}_{k'}}^T \mathbf{Z}_{k'}^{-1}. \quad (5.10)$$

Цей крок вимагає правильної *асоціації даних* — зіставлення ознак з орієнтирами. Виконується через описові вектори ознак з порогом подібності або через алгоритм Угорський (Hungarian) для оптимального присвоєння. Можна уникнути, якщо орієнтири унікально марковані.

Загальна складність оновлень $\mathcal{O}(kn^2)$, де k — число орієнтирів, n — число станів. Можливі прискорення через *маргіналізацію* [?].

Рис. 5.1: Пози робота (трикутники) і унікальні орієнтири (зірочки) формують позовий граф на 2D-карті. Ребра між позами — одометричні вимірювання. Ребра між позами та орієнтирами — дальність і кут. При замиканні циклу (loop closure), тут — повторне відкриття орієнтира 3, усі пози між подіями можуть бути скориговані.

5.3.2. Багатосенсорний випадок

Сенсори в робототехніці мають компроміси: візуальний сенсор дає інформацію про структуру і позу-до-позу, але страждає при слабкому освітленні або без текстур. Сенсор глибини страждає при надлишковій геометрії (наприклад, коридор). IMU дає короткострокову одометрію, але дрейфує. Розмір, вага, потужність і вартість обмежують вибір сенсорів.

Сенсори інтегруються в EKF SLAM простим розширенням оновлення (5.3.1): додаються кроки оновлення з різних сенсорів на різних частотах.

5.4. SLAM на основі графів (Graph-based SLAM)

Зазвичай робот отримує початкову оцінку положення бортовими сенсорами (одометрія, оптичний потік), використовує її для локалізації орієнтирів (стін, кутів, графічних патернів), і нарешті уточнює оцінку пози зіставленням сенсорної інформації послідовних полів огляду (наприклад, алгоритмом ICP, 4.2). Як тільки робот вдруге відвідує той же орієнтир, він може оновити оцінку. Послідовні спостереження не незалежні, а корельовані; уточнена оцінка поширюється вздовж шляху робота. У EKF SLAM нові надійні вимірювання коригують похибки попередніх.

Інтуїтивніший погляд — як на “граф” з масами у вузлах і пружинами на ребрах. Кожна можлива поза (маса) обмежена сусідньою позою пружиною. Чим вища невизначеність відносного перетворення між двома позами (наприклад, з одометрії), тим слабша пружина. Щоразу, як робот набуває впевненості, пружина стискається. Зрештою всі пози “натягуються” так, щоб мінімізувати загальну напругу в графі. Це досягається числовою мінімізацією загальної похибки градієнтним спуском. Це — *SLAM на основі графів* [?].

5.4.1. SLAM як задача оцінювання максимальної правдоподібності

Класична постановка SLAM описує задачу як максимізацію апостеріорної ймовірності всіх точок траєкторії за одометричним входом і спостереженнями:

$$p(x_{1:T}, m | z_{1:T}, u_{1:T}), \quad (5.11)$$

де $x_{1:T}$ — всі дискретні положення з $t \in (1, T)$, z — спостереження, u — одометричні вимірювання. Розв’язання вимагає:

1. Моделі оновлення руху: $p(x_t | x_{t-1}, u_t)$.
2. Моделі сенсора: $p(z_t | x_t, m_t)$.

У SLAM на основі графів траєкторія робота формує вузли графа, ребра — перетворення (трансляція + обертання) зі своєю дисперсією. Graph-SLAM — це задача

максимізації правдоподібності: знайти ті значення змінних стану, що найбільш імовірні за наявних спостережень.

Нагадаймо нормальний розподіл:

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \quad (5.12)$$

Можна асоціювати такий розподіл з кожним переходом вузол-вузол. Позначимо вимірювання переходу між вузлами i і j як z_{ij} , його очікуване значення — \hat{z}_{ij} .

Логарифмування дає зручність: $\log \prod z_{ij} = \sum \log z_{ij}$. Логарифм нормального розподілу — лінійна функція в x . Використовуючи інформаційну матрицю $\Omega_{ij} = \Sigma_{ij}^{-1}$:

$$l_{ij} \propto (z_{ij} - \hat{z}_{ij}(x_i, x_j))^T \Omega_{ij} (z_{ij} - \hat{z}_{ij}(x_i, x_j)). \quad (5.13)$$

Задача оптимізації:

$$x^* = \arg \min_x \sum_{\langle i,j \rangle \in \mathcal{C}} e_{ij}^T \Omega_{ij} e_{ij}, \quad (5.14)$$

де $e_{ij}(x_i, x_j) = z_{ij} - \hat{z}_{ij}(x_i, x_j)$ — похибка між вимірюванням та очікуваним значенням.

5.4.2. Числові техніки для Graph-based SLAM

Класичний підхід: лінеаризувати задачу при поточній конфігурації і звести до $Ax = b$. Альтернатива — стохастичний градієнтний спуск: на кожному кроці бере підмножину обмежень. Для Graph-SLAM SGD ітеративно працює над одним обмеженням за раз.

Графи стають великими (мільйони вузлів). Це адресують побудовою *мінімального остовного дерева* (MST) графа обмежень. MST будується через пошук у глибину (DFS) одометричними обмеженнями. При замиканні циклу (loop-closure) DFS повертається назад. Оновлення поз, зачеплених новим обмеженням, вимагає модифікації лише вузлів на шляху між двома задіяними орієнтирами.

Висновки на винос

- SLAM — ключова здатність мобільних роботів для автономного функціонування у світі.
- Існують стійкі реалізації для середовищ із сильними орієнтирами — тими, що надійно локалізуються й ідентифікуються.
- SLAM значно виграє від додаткових сенсорів (особливо маякових, як-от GPS).
- Як працювати в середовищах з *динамічними* об'єктами (тобто з картами, що змінюються) — досі відкрита проблема.

Вправи

1. Розробіть базову SLAM-систему EKF з відомими орієнтирами:

- Реалізуйте одно-орієнтирну SLAM. Базова одометрія в симуляторі + детектор кута/відстані. Ініціалізуйте перше вимірювання середнім і дисперсією з одометрії і покажіть, як додаткові вимірювання обмежують похибку одометрії через фільтр Калмана.

- (b) Введіть другий орієнтир і поверніть робота до першого після відвідування другого. Що можете сказати про дисперсію другого орієнтира після корекції?
 - (c) Реалізуйте симуляційне середовище з кількома орієнтирами (Ratslife — добрий приклад). Експериментально визначте середню дисперсію при локалізації.
 - (d) Реалізуйте EKF SLAM.
2. EKF SLAM вимагає унікально ідентифікованих орієнтирів. Розгляньте реалізацію лише детекторами кутів і стін. Як зробити, щоб такі орієнтири здавались унікальними, і які обмеження підходу?
3. Розробіть базовий Graph-SLAM:
- (a) Реалізуйте графову структуру даних, що зберігає пози робота та орієнтирів.
 - (b) Реалізуйте DFS для обчислення найкоротшого шляху між двома вузлами.
 - (c) Реалізуйте базовий Graph-SLAM. При замиканні циклу оновіть оцінку пози через положення орієнтира усередненням. Використайте нову оцінку для оновлення попередніх поз вздовж найкоротшого шляху назад.

6. Планування шляху

Планування шляху дозволяє автономним мобільним роботам і маніпуляторам знайти шлях для переміщення між двома точками. *Шлях* — це набір поз від початкової конфігурації до кінцевої конфігурації, що задовольняє певний набір специфікацій (наприклад, уникнення перешкод для мобільної бази або дотримання конкретного силового профілю в кінцевому ефекторі маніпулятора). Він відрізняється від поняття *траєкторії* тим, що траєкторія — це виконання шляху в часі. Залежно від вибору алгоритму планування шлях може задовольняти різні ступені оптимальності за певними критеріями — наприклад, мінімізація довжини шляху, мінімізація кількості поворотів або мінімізація гальмування. Алгоритми пошуку найкоротшого шляху важливі не лише для робототехніки, а й для мережевого маршрутизування, відеоігор та розуміння згортання білків.

Планування шляху вимагає відповідного представлення середовища (як-от карта з розділу 4) і перцептивного розуміння положення робота відносно цього представлення. Поки що припускаємо, що робот здатний локалізувати себе, оснащений картою і здатний уникати тимчасових перешкод на своєму шляху. Цілі розділу:

- ввести поняття “простору конфігурацій” (configuration space) для планування;
- зрозуміти різницю між алгоритмами планування на основі графів та на основі вибірок (sampling-based);
- пояснити базові алгоритми шляху — Дейкстра, A*, RRT;
- зрозуміти варіації задачі планування шляху, як-от планування покриття (coverage path planning).

6.1. Простір конфігурацій

У переважній більшості алгоритмів планування шляху робот трактується як точкова маса без об’єму. Щоб шлях можна було виконати на роботі, важливо врахувати фізичне втілення робота та його ненульове об’ємне зайняття, що ускладнює планування. Робот можна звести до точкової маси, “розширивши” (grow) всі перешкоди на його радіус. Це працює для круглого робота. Це узагальнюється на роботів будь-якої форми, розширюючи кожну перешкоду на довжину найбільшого виступу робота від його центра. Таке представлення відоме як *простір конфігурацій*, оскільки воно зводить представлення робота до його керованих ступенів свободи (наприклад, координат x і y у площині для робота, здатного до пласкої трансляції). Приклад наведено на 6.1. Простір конфігурацій можна потім використати як основу для сітчастої карти або неперервного представлення.

рисунок пропущено

Рис. 6.1: Карта з перешкодами та її представлення у просторі конфігурацій, яке можна отримати розширенням кожної перешкоди на величину виступу робота.

Рис. 6.2: Загальна задача планування шляху від вершини I до вершини VI. Найкоротший шлях — I-II-III-V-VI довжиною 13.

6.2. Алгоритми планування на основі графів

Задача пошуку “найкоротшого” шляху від однієї вершини до іншої через зв’язний граф цікава для багатьох областей — найпомітніше, для мережевого маршрутизування, де вона використовується для знаходження оптимального маршруту інтернет-пакета даних. Термін “найкоротший” тут визначається як мінімальна сумарна вартість ребер, що може бути фізичною відстанню (у роботизованому застосуванні), затримкою (у мережевому застосуванні) або будь-якою іншою метрикою, релевантною для завдання. Приклад графа з довільними довжинами ребер показано на 6.2.

6.2.1. Алгоритм Дейкстри

Один із найраніших і найпростіших алгоритмів планування шляху — алгоритм Дейкстри [?]. За заданим графом Дейкстра — це ітеративний процес: починаючи з “початкової” вершини, алгоритм позначає всіх її прямих сусідів вартістю досягнення. Потім інспектує сусідню вершину з найнижчою вартістю та всі її суміжні вершини й позначає їх вартістю досягнення через розглянуту вершину. Якщо ця вартість виявляється нижчою, вона оновлюється. Коли всіх сусідів вершини перевірено, алгоритм переходить до вершини з наступною найнижчою вартістю. Коли алгоритм досягає цільової вершини і немає вершини з нижчою вартістю до цілі, він завершується, і робот може слідувати ребрами, що вказують на найнижчу вартість ребра.

У прикладі на 6.2 Дейкстра спершу позначить вузли II, III та IV вартостями 3, 5 і 7 відповідно. Потім досліджуватиме всі ребра вузла II, який наразі має найнижчу вартість. Це призведе до відкриття того, що вузол III фактично можна досягти за $3 + 1 < 5$ кроків, і вузол III буде перепозначено вартістю 4. Щоб повністю оцінити вузол II, Дейкстра має дослідити решту ребер перед переходом і позначить вузол VI вартістю $3 + 12 = 15$.

Вузол з найнижчою вартістю тепер — вузол III (вартість 4). Можемо перепозначити вузол VI вартістю 14, що менше за 15, і позначити вузол V вартістю $4 + 5 = 9$, тоді як вузол IV лишається на $4 + 3 = 7$. Хоча ми вже знайшли два шляхи до цілі, один з яких кращий, не можна зупинитися, бо ще є вузли з недослідженими ребрами і загальною вартістю меншою за 14. Дійсно, продовження дослідження з вузла V веде до найкоротшого шляху I-II-III-V-VI вартістю 13 без залишку вершин для дослідження.

Оскільки Дейкстра не зупиниться, доки немає вузла з меншою вартістю, ніж поточна вартість до цілі, ми можемо бути впевнені, що найкоротший шлях буде знайдено, якщо він існує. Тому Дейкстра є *повним* (complete) і оптимальним.

Оскільки Дейкстра завжди досліджує вузли з найменшою загальною вартістю першими, дослідження середовища нагадує хвильовий фронт, що йде від початкової вершини до цілі. Це, звичайно, дуже неефективно, особливо коли Дейкстра досліджує вузли далеко від цілі. Наприклад, якщо додати кілька вузлів ліворуч від вузла I у 6.2, Дейкстра дослідить усі ці вузли, поки їхня вартість не перевищить найменшу знайдену вартість до цілі. Це також видно при спостереженні роботи

Дейкстри на сітці (6.3).

рисунок пропущено

Рис. 6.3: Алгоритм Дейкстри знаходить найкоротший шлях від 'S' до 'G', припускаючи, що робот може рухатися лише латерально (не діагонально) з вартістю один за клітинку сітки. Зверніть увагу на малу кількість недосліджених клітинок після знаходження найкоротшого шляху (сірий), оскільки Дейкстра завжди розглядає клітинку з найнижчою вартістю першою.

Зауважимо, що сітку можна звести до графа, в якому кожна вершина (окрім тих на межах) має чотирьох або вісьмох сусідів.

6.2.2. A*

Замість дослідження у всіх напрямках, знання приблизного напрямку до цілі може допомогти уникнути дослідження вузлів, не потрібних для успіху завдання. Як люди, ми легко інтерпретуємо завдання у 6.3 і розуміємо, що більшість станів у верхньому лівому та нижньому правому кутах не варто досліджувати, якщо ми хочемо знайти розв'язок швидко. Такі знання можна закодувати в алгоритмі пошуку через *евристичну функцію* — обґрунтований здогад або оцінку. Наприклад, можна надавати пріоритет вузлам з нижчою оціненою відстанню до цілі. Для цього кожен вузол позначаємо не лише фактичною відстанню, що знадобилася для досягнення (як у Дейкстри), а й оціненою вартістю до цілі — наприклад, обчислюючи евклідову відстань або *манхеттенську відстань* між поточною вершиною і ціллю. Цей алгоритм відомий як A* [?] і проілюстрований у 6.4 з використанням манхеттенської метрики. Залежно від середовища A* може виконати пошук набагато швидше за Дейкстру і в найгіршому випадку працює так само.

Розширення A*, що адресує проблему дорогого перепланування при появі перешкод на шляху робота, відоме як D* [?]. На відміну від A*, D* починає з цільової вершини і має здатність змінювати вартості частин шляху, що містять перешкоду. Це дозволяє D* перепланувувати навколо перешкоди, зберігаючи більшу частину вже обчисленого шляху.

A* і D* стають обчислювально дорогими, коли або простір пошуку великий (наприклад, через тонку роздільну здатність, потрібну для завдання), або коли розмірності задачі високі (наприклад, при плануванні для маніпулятора з багатьма ступенями свободи). Розв'язки для цих проблем можуть давати алгоритми планування на основі вибірок.

6.3. Планування на основі вибірок (sampling-based)

Розділ 6.2 ввів серію *повних* алгоритмів, тобто алгоритмів, гарантовано (зрештою) знаходять розв'язок, якщо він існує. Однак повні алгоритми часто непрактичні —

рисунок пропущено

Рис. 6.4: Знаходження найкоротшого шляху від 'S' до 'G' алгоритмом A* з тими ж припущеннями. Подібно до Дейкстри, A* оцінює лише клітинку з найнижчою вартістю, але враховує оцінку решти відстані.

Рис. 6.5: Проти годинникової стрілки від верхнього лівого: випадкове дослідження 2D-простору пошуку шляхом випадкового семплювання точок і їх з'єднання з графом, доки не буде знайдено допустимий шлях між початком і ціллю.

через великий простір станів, малу доступну пам'ять або обмежений час виконання. Це часто стосується роботів з багатьма ступенями свободи, як-от маніпуляторів. Важливо, що більшість алгоритмів є лише *повними за роздільною здатністю*, тобто повними лише при достатньо тонкій роздільній здатності середовища: оскільки простір станів потрібно дискретизувати, деякі розв'язки можуть бути пропущені через таку дискретизацію.

Планувальники на основі вибірок — альтернатива графовим, що оцінюють усі можливі розв'язки, та неповним обернено-кінематичним розв'язкам на основі якобіана. У плануванні руху на основі вибірок можливі шляхи генеруються випадковим семплюванням і зберігаються у деревовидній структурі, доки не знайдеться розв'язок або не закінчиться відведений час. Оскільки ймовірність знайти шлях прямує до одиниці зі збільшенням числа зразків до нескінченності, такі планувальники є *ймовірно повними*. Помітні приклади: *Rapidly-exploring Random Trees* (RRT, “швидко зростаючі випадкові дерева”) [?] і *Probabilistic Roadmaps* (PRM, “ймовірнісні дорожні карти”) [?].

Приклад роботи RRT показано на 6.5; за суттю, RRT вирощує одне дерево з початкової точки робота, доки одна з його гілок не вдариться у ціль. Цей приклад ілюструє, як планувальник на основі вибірок може швидко дослідити велику частину простору і уточнити розв'язок з часом. Натомість ймовірнісні дорожні карти створюють дерево, випадково семплюючи точки в просторі станів, перевіряючи їх на відсутність колізій, з'єднуючи їх із сусідніми точками за досяжними шляхами (відповідно до кінематики робота), а потім використовуючи класичні графові алгоритми найкоротшого шляху для пошуку шляхів на отриманій структурі. Перевага PRM: карта створюється лише раз (за припущення стабільного середовища) і може використовуватися для багатьох запитів. PRM — це алгоритм *мультизапитового* планування шляху, тоді як RRT — алгоритм *однозапитового* планування шляху. З роками межі між цими алгоритмами розмилися, і існують одно- та мультизапитові варіанти і RRT, і PRM. Загалом “срібної кулі” — алгоритму або евристики — не існує, і навіть вибір параметрів сильно залежить від конкретної задачі.

6.3.1. Rapidly Exploring Random Trees (RRT)

Нехай \mathcal{X} — d -вимірний простір станів. Це може бути стан робота в термінах трансляцій і обертань (6 вимірів або підмножина) або суглобовий простір з одним виміром на суглоб. Вибір представлення впливає на те, як обчислюється досяжність точки, але не впливає на сам алгоритм.

Нехай $\mathcal{G} \subset \mathcal{X}$ — d -вимірна сфера в просторі станів, що вважається ціллю, \max_dist — найбільша допустима довжина ребра, t — дозволений час, k — максимальна кількість вершин у дереві, $goal_bias$ — частка часу, протягом якої алгоритм має намагатися з'єднатися з цільовим станом. Планувальник RRT слідує наступному псевдокоду:


```

Tree = Init(X, G, start, max_dist, t, k, goal_bias);
iteration = 0
WHILE (ElapsedTime() < t AND iteration < k
AND NoGoalFound(Tree, G)) DO:
    iteration = iteration + 1
    IF RandomPercentage() < goal_bias THEN
        q_rand = SampleRandomGoal(G);
    ELSE
        q_rand = SampleRandomState(X);
    ENDIF
    q_nearest = NearestVertex(q_rand)
    q_new = Extend(q_nearest, q_rand, max_dist)
    edge = CreatePath(q_nearest, q_new);
    IF IsAllowablePath(edge) THEN
        Tree.addVertex(q_new);
        Tree.addEdge(edge);
    ENDIF
ENDWHILE
return Tree

```

Цей процес можна ітерувати, поки дозволяє час; параметри k і $goal_bias$ опціональні. RRT відомий як *anytime-алгоритм*: будь-яке переривання користувачем після знаходження початкового розв'язку все одно дасть якесь рішення. За відповідної метрики відстані вартість шляху можна зберігати в кожному вузлі дерева, що дозволяє відстежувати найкоротший шлях до цілі, якщо в зоні цілі кілька вершин. У цьому алгоритмі є чотири ключові моменти:

1. визначення наступної точки q_rand для додавання до дерева (SampleRandomGoal, SampleRandomState, Extend);
2. визначення, куди і як з'єднати цю точку з деревом з урахуванням кінематики робота (NearestVertex, CreatePath);
3. перевірка, чи цей шлях прийнятний (IsAllowablePath) — тобто без колізій;
4. згладжування шляху (не показано в алгоритмі).

Вибір наступної кращої точки. Простий підхід — випадково обрати точку в просторі станів і з'єднати її з найближчою існуючою точкою у дереві. Інші рішення можуть надавати перевагу вузлам з малим ступенем виходу (тобто без багатьох з'єднань) і обирати точки поблизу, щоб полегшити експансію в малодосліджених регіонах.

З'єднання точок з деревом. Інтуїтивно, нову точку q_rand слід з'єднати з найближчою точкою вже в дереві або з ціллю. Це вимагає ітерування по всіх вузлах і обчислення відстані до точки-кандидата, що є обчислювально дорогим процесом; отриманий $q_nearest$ — з найкоротшою відстанню. Вибір правильної структури даних для збереження графа в пам'яті може звести обчислювальну вартість до сублінійної в середньому за кількістю вершин.

Цей метод не гарантує найкоротший шлях. Як альтернатива, RRT* вирощує дерево так, щоб завжди мінімізувати загальну довжину шляху від кореня до кожної вершини. Це робиться у два кроки. По-перше, розглядаються лише точки в дереві

в межах d -вимірної сфери фіксованого радіуса від q_rand , і знаходиться точка, що мінімізує загальну довжину шляху від початку (а не просто найкоротшу відстань від q_rand). Це гарантує, що нова вершина q_rand з'єднана з найкоротшим досяжним шляхом від кореня. По-друге, відбувається крок *перепідв'язання* (rewiring), де вершини поблизу q_rand оцінюються на предмет того, чи ребро між ними та q_rand було б коротшим за поточне. Якщо так і ребро допустиме, граф перепідв'язується так, щоб новознайдена вершина стала новим батьком q_rand .

Після знаходження найближчої вершини функція `Extend` використовує `max_dist` для обмеження максимальної довжини ребра, замінюючи q_rand точкою q_new на лінії, що з'єднує $q_nearest$ і q_rand , на відстані `max_dist` від $q_nearest$. На цьому етапі також зручно врахувати специфічну кінематику робота і його рухові можливості. Наприклад, для автомобіля локальний планувальник може згенерувати придатну траєкторію, що враховує орієнтацію транспортного засобу.

Перевірка колізій. Ефективні алгоритми тестування колізій заслуговують на окремий розділ. Хоча задача інтуїтивна для 2D-планування в просторі конфігурацій і розв'язується простим тестом “точка-в-полігоні” (бо робот зводиться до точки), для маніпуляторів — кількох жорстких тіл, з'єднаних разом, які можуть мати самозіткнення — це складніше. Зазвичай перевірку колізій таких об'єктів виконують перетворенням на трикутні сітки, що потім тестуються на перетин. Останнім часом дедалі частіше використовуються фізико-ігрові двигуни (game engines) із вбудованою перевіркою колізій.

Зазвичай перевірка колізій займає до 90% часу виконання задачі планування шляху; тому методи зниження обчислювальних витрат бажані. Наприклад, алгоритм “лінивої перевірки колізій” (lazy collision evaluation) відрізняється тим, що не оцінює кожную точку на можливе зіткнення. Натомість він спочатку знаходить придатний шлях і лише потім оцінює кожне ребро. Сегменти зі зіткненнями видаляються, алгоритм продовжується, але зберігаються лише вільні від колізій сегменти.

Після знаходження можливого шляху простір семплювання можна звести до еліпсоїда, що обмежує максимальну довжину шляху. Цей еліпсоїд можна побудувати, “натягнувши дріт” максимальної довжини шляху між початком і ціллю і відтиснувши його назовні. Інтуїтивно, лише точки в межах цієї еліптичної області можуть давати коротший шлях за поточний відомий, тож не варто витрачати час на вирощування дерева поза цим еліпсоїдом. Цей підхід особливо ефективний при паралельному запуску кількох копій планувальника і обміні найкоротшими шляхами після їх знаходження [?].

Згладжування шляху. Оскільки планування випадково семплює з дискретних і довільно грубих карт, отримані шляхи зазвичай зубчасті і нерегулярні — далекі від оптимальних на практиці. Це можна значно покращити згладжуванням шляху. Один зі способів — з'єднання точок шляху сплайнами, поліноміальними кривими або фрагментами траєкторій, що відомо є реалізованими для конкретної платформи. Альтернативно, можна використати модель платформи і регулятор зі зворотним зв'язком (як в ?? для мобільних роботів та ?? для маніпуляторів) для генерації траєкторії, яку робот фактично здатний виконати. У поєднанні з динамікою цей підхід відомий як *прогнозне керування моделлю* (model-predictive control, MPC).

Рис. 6.6: Планування шляху на різних масштабах довжини, що вимагає різноманітних представлень карт і парадигм планування. Стрілки показують інформацію, що передається між рівнями.

6.4. Планування на різних масштабах довжини

Реальність виконання складної автономної поведінки в реалістичних сценаріях полягає в тому, що на практиці одного представлення карти та алгоритму планування може бути недостатньо. Планування маршруту автомобіля, наприклад, — це багатоступінчастий процес, де автономія робота переплітається з людським інтелектом: як показано на 6.6, потрібна ієрархія дедалі деталізованіших карт і алгоритмів планування. Спочатку виконується грубий пошук вуличною мережею (наприклад, навігаційним застосунком), потім — точніший планувальник, що визначає, які смуги обирати і як проходити кільцеві перехрестя; на обох цих рівнях абстракції ідеальні графові алгоритми. Далі планувальник на основі вибірок може визначити, як фактично рухати автомобіль між смугами і яку траєкторію використати для уникнення перешкод. Нарешті, такі траєкторії потрібно перетворити на швидкості коліс і кути рульового керування — можливо, використовуючи якусь форму керування зі зворотним зв'язком.

6.5. Планування покриття (coverage path planning)

Досі ми розглядали лише задачу знаходження (найкоротшого) шляху. Варіацією є *планування покриття*. Це актуально для застосувань, як-от прибирання, скошування трави або фарбування, і зазвичай мінімізує час до завершення та надлишковість під час покриття. Ця задача тісно пов'язана з задачею найкоротшого шляху. Наприклад, покриття підлоги можна досягти, виконуючи пошук у глибину (DFS) або в ширину (BFS) на графі, де кожна вершина має розмір інструмента покриття робота.

DFS чи BFS можуть генерувати ефективні шляхи покриття, але вони далекі від оптимальних, бо багато вершин відвідуються двічі. Шлях, що з'єднує всі вершини графа, але проходить кожную вершину рівно один раз, називається *Гамільтоновим шляхом*. Гамільтонів шлях, що повертається у початкову вершину — *Гамільтонів цикл*. Ця задача також відома як *задача комівояжера* (Traveling Salesman Problem, TSP), у якій потрібно обчислити маршрут, що відвідує кожне місто з туру лише раз, і яка відома як NP-повна.

6.6. Підсумок

Планування шляху — це постійна дослідницька проблема. Знаходження безколізійних шляхів для механізмів з високим числом ступенів свободи (як-от кілька рук, що працюють у спільному просторі, мультиробототехнічні системи або системи, що залучають динаміку) досі обчислювально інтенсивна задача. Хоча планувальники на основі вибірок можуть значно прискорити час знаходження певного розв'язку, вони не оптимальні і мають труднощі з вузькими проходами. Не існує “срібної кулі” для всіх задач планування шляху, і евристики, що дають масивне

прискорення в одному сценарії, можуть бути шкідливими в інших. Також параметри алгоритмів переважно ad-hoc, і правильне їх налаштування під конкретне середовище може значно підвищити продуктивність.

Висновки на винос (take-home lessons)

- Перший крок планування шляху — вибір представлення карти, придатного для застосування (4).
- Другий крок — звести робота до точкової маси, що дозволяє планувати у просторі конфігурацій (або C-простір).
- Це дозволяє застосовувати загальні алгоритми найкоротшого шляху на основі графів, які мають застосування у багатьох областях, не обмежених робото-технікою.
- Алгоритм планування на основі вибірок знаходить шляхи семплюванням випадкових точок у середовищі. Евристики використовуються для максимізації дослідження простору та зміщення напрямку пошуку. Це робить алгоритми швидкими, але ні оптимальними, ні повними.
- Оскільки результуючі шляхи випадкові, кілька запусків можуть дати зовсім різні результати.
- Не існує універсального алгоритму планування шляху, і слід обережно обирати правильну парадигму (наприклад, однозапитову vs мультизапитову), евристики та параметри.

Вправи

1. Як змінюється обчислювальна складність алгоритму Дейкстри при переході від 2D до 3D простору пошуку?
2. A* використовує "евристику" для зміщення пошуку в напрямку цілі. Чому можна використати лише евристику, а не фактичну довжину?
3. За припущення, що точки семплюються рівномірно випадково в рандомізованому планувальнику, обчисліть граничну поведінку відношення (площа точок у дереві)/(площа семплованих точок) при числі семплів, що прямує до нескінченності, без дублікатів. Припустіть, що загальна площа A_{total} і площа вільного простору A_{free} відомі.
4. За припущення, що kd-дерево використовується як структура для найближчого сусіда і точки семплюються рівномірно випадково, обчисліть час виконання вставки точки у дерево розміру N . Використайте big-O нотацію, наприклад $\mathcal{O}(N)$.
5. Які інші практичні аспекти часу виконання, крім обчислювальної складності, треба враховувати при плануванні на основі вибірок? Які способи їх подолати?
6. Напишіть програму, що читає просту карту з текстового файлу, де '1' позначає перешкоди, а '0' — вільний простір.
 - (a) Реалізуйте алгоритм Дейкстри для знаходження найкоротшого шляху між двома точками у вільному просторі.
 - (b) Реалізуйте A* для знаходження найкоротшого шляху.
 - (c) Як ці дві реалізації порівнюються за обчислювальною складністю?

7. Напишіть програму, що читає зображення, де білі ділянки — вільний простір, а чорні — перешкоди. Реалізуйте базовий RRT для знаходження найкоротшого шляху між двома точками.
8. Дослідіть інтернет на предмет бібліотек планування шляху мовою на ваш вибір. Які інструменти ви знайдете? Як вони визначають карту? Чи виконують уникнення перешкод? Чи має значення кінематика робота?
9. Розширте свою реалізацію планування шляху для використання з диференціальним колісним роботом. Опишіть кроки для Дейкстри/A* і для RRT.
10. Розширте алгоритм планування для використання з дволанковим маніпулятором. Чи плануватимете у суглобовому просторі або у просторі конфігурацій, і які переваги/недоліки кожного підходу?
11. Як змінюється обчислювальна складність при переході від одного 6-DoF маніпулятора до торса з двома 6-DoF маніпуляторами? Чи можна зберегти оригінальну складність? Які компроміси?
12. Розгляньте робочу задачу складання, у якій робот отримує об'єкти з відомої локації і збирає їх на столі. Коли можна покладатися на просту обернену кінематику, а коли потрібне планування шляху?
13. Як змінюється задача планування, якщо враховувати не лише положення, а й сили й моменти? Чи можна використати варіацію RRT для розв'язання?
14. Завантажте інструмент планування шляху, що дозволяє пробувати різні алгоритми.
 - (a) Порівняйте якість і швидкість розв'язків.
 - (b) Що треба врахувати при використанні рандомізованих планувальників? Чи достатньо одиничного експерименту?
15. Реалізуйте планувальник покриття для одного робота на сітчастій карті за допомогою DFS. Оцініть надлишковість для різних початкових локацій.