Quiz 1 Review

Mu-Ruei Tseng

September 18, 2024

1 Introduction

- What is a robot? Defining characteristics. Mount Fuji illustration.
  - It is difficult to define a robot. (No clear definition)
  - Equivalent to finding the line between the flat land and gradually rising mount fuji Masahiro Mori
  - Though difficult to define has some defining characteristics:
    - Autonomous
    - Sensor - Driven
    - Physical
    - Goal - Achieving
- Why is robotics hard? The fundamental problem.
  - Robotics need actuators and sensors to interact with the world but........ the real world is highly complicated and unpredictable. Also hardware is often limited and unreliable
- Why can't we focus strictly on computing issues?
  - Because robotics is a combination of computing, sensing and acting and problems in robotics exist in their intersection
- Four fundamental problem types. What does each problem type ask the robot to do?
  - Navigation and Planning
  - Localization and Mapping
  - Manipulation
  - Exploration and Coverage

2 Robot hardware

- Actuators: motors (simple DC and gearhead), stepper, servo, linear actuator.

  Definition: a device that converts energy into motion or force.

  1. DC Motor: converts direct current (DC) electrical energy into mechanical rotation. Usually used when required **high speed without high torque** or precise control over speed.
     - Ex: Cooling fans in computers.
  2. Gearhead Motor: DC motor with an integrated gearbox that provides more torque and control over speed (**decreasing velocity and increasing torque**).

- Ex: Robotic arms, motorized wheels on robots

3. Stepper Motor: Motor designed for precise control of angular or linear position, velocity, and acceleration. Unlike DC motors or gearhead motors, stepper motors **move in discrete steps**.
   - Ex: Controlling the joints of robotic arms or in Automated Positioning Systems.
4. Servo: Uses a **closed-loop feedback system** to ensure accurate control of both position and speed. Typically used in applications requiring high precision, high speed, and high torque, especially in dynamic systems that demand real-time feedback for continuous adjustments. However, **without** the ability to complete a **full rotation**.
   - Ex: Steering control in remote-controlled cars, boats, and airplanes.
5. Linear actuators: generate translational motion in various ways.

Characteristic	**DC Motor**	**Gearhead Motor**	**Stepper Motor**	**Servo Motor**	**Linear Actuator**
Control System	Open-loop, no feedback	Open-loop, no feedback	Open-loop (can be closed-loop with additional components)	Closed-loop, uses feedback (encoder/potentiometer)	Closed-loop or open-loop, depends on design
Precision	Low, continuous rotation	Low to medium, improved by gear reduction	High, moves in discrete steps	Very high, continuous real-time feedback	High, provides accurate linear movement
Speed	High speed, low torque at high speeds	Reduced speed, increased torque due to gears	Medium to low speed, torque decreases with speed	High speed, maintains torque at high speeds	Variable, typically lower speed
Torque	Low torque at high speeds	High torque at low speeds due to gearbox	High torque at low speeds, reduces with increased speed	High torque across all speeds, adjustable with feedback	High force, especially for moving heavy loads
Best Use Case	High-speed applications with low torque requirements	Low-speed, high-torque applications like conveyors or robotics	Precise positioning applications like 3D printers or CNC machines	Precision motion control like in robotics, CNC, and automation	Applications requiring linear motion, such as gates or actuators

Table 1: Comparison of DC Motor, Gearhead Motor, Stepper Motor, Servo Motor, and Linear Actuator

- Modes of locomotion Definition: the movement or the ability to move from one place to another.
  - Terrestrial: wheels, legs, hovercraft, ...
  - Underwater: thrusters, flippers, ...
  - Airborne: propellers, fixed-wing, ...
  - Space: rockets, solar sails, ...
- Sensing: encoder, infrared sensor, camera, RGBD sensor, sonar, lidar, IMU, compass, GPS, inclinometer
  1. Encoders: Measure rotation in joints or wheels.
     - Ex: Used in robotic arms to track joint movement for precise positioning.
  2. Infrared sensors: Measure distance by emitting IR light and detecting the reflected signal's intensity.
     - Ex: Used in automatic door sensors to detect nearby people or objects.
  3. Ultrasonic sensors: Emit sound pulses and measure distance based on time-of-flight.
     - Ex: Common in parking assist systems in cars to detect obstacles when reversing.
  4. Lidar: Measures distance using **phase shifts** in coherent light.

    - Ex: Used in autonomous vehicles for mapping and object detection.
5. Cameras: Capture color and light intensity, sometimes using mirrors for enhanced imaging.
6. RGBD sensors: Provide color images and depth information by projecting infrared patterns and measuring distortion.
7. Compasses: Measure orientation relative to Earth's **magnetic field**.
    - Ex: Used in smartphones for navigation apps to indicate direction.
8. GPS: Uses satellite signals to determine geographical position.
9. Inclinometers: Measure the **direction of gravity**.
    - Used in construction tools like digital levels to measure surface inclinations.
10. IMUs: Combine accelerometers and gyroscopes to measure linear and angular acceleration.
    - Used in drones for stabilizing flight and controlling motion.

- Proprioception and exteroception
    - Proprioceptive sensors provide information about the robot's internal state.
        * Ex: **Encoders** that measure the rotation of robot joints or wheels.
        * Ex: **Inertial Measurement Units (IMUs)** that track the robot's orientation and acceleration.
    - Exteroceptive sensors provide information about what the robot's environment.
        * Ex: **Lidar** that maps the surroundings by measuring distances to nearby objects.
        * Ex: **Cameras** that capture visual information about the robot's environment.
- Evaluating sensors: Active vs passive. What is the difference? Which sensor types are active? Which are passive?
    - Active sensors: emit energy into the environment and measure the response.
        * Ex: **Lidar** emits laser light and measures the reflected signal to determine distances.
        * Ex: **Ultrasonic sensors** emit sound waves and measure the time of flight of the reflected signal.
    - Passive sensors: detect energy that is naturally present in the environment.
        * Ex: **Cameras** capture visible light that is naturally reflected from objects in the environment.
        * Ex: **Thermal sensors** detect heat emitted by objects, measuring infrared radiation.
- Criteria for evaluating sensors: Speed of operation, Cost, Error rate, Robustness, Computational requirements, Power consumption, Size and weight

2.1 Robot hardware case study

- Nothing in particular from this set, though it does have some good examples of sensors and actuators from the main hardware lecture.

3 Differential drive locomotion

- Differential Drive

  Definition: a robot that has two independent non-steered drive wheels along the common axis.
- Drive vs. steered vs. passive wheels.

  - Drive wheels: only roll forward and backward **- cannot be steered**
  - Steered wheels: can rotate relative to the ground and influence the direction of the robot
  - Passive wheels: neither drive nor steer wheels and only contribute to the robot's stability

- States, actions, transitions. What are they? Notation for each. x, X, u, U, f. State transition equation.
  - State $x \in \mathbf{X}$ is a collection of aspects of the robot and the environment. For differential drive, the state is defined as (x, y, θ).
  - Action $u \in \mathbf{U}$ is a collection of choices the robot can make. For differential drive, the action is defined as $(v_l, v_r, \Delta t)$.
  - f: state transition equation. $x' = f(x, u)$.

- Using the differential drive ICC equations. Special cases for differential drive movement. Updating the state for a differential drive.
  - ICC (Instantaneous Center of Curvature) is the point around which the robot moves with **constant** angular velocity (ω).
  - ICC is located at distance R from the robot
  - If $v_r = v_l$, then $R = \infty \implies$ ICC is at infinity and robot moves in a straight line
  - If $v_r = -v_l$, then $R = 0 \implies$ ICC is at robot center robot rotates in place

- Navigate with a differential drive to a given target.

4 Introduction to ROS

- What is ROS?

  ROS: a set of software libraries and tools for building robot applications.

- Why is ROS necessary?
  1. Distributed computation: ROS simplifies distributed software by enabling efficient message-passing between components.
  2. Code reuse: Provides reusable implementations of robotics algorithms, easily integrating multiple algorithms in a single application.
  3. Seamless simulations: Ensures simulators and robots share consistent APIs for seamless interaction.

- Meanings and relationships between various ROS objects: packages, nodes, topics, publishers, subscribers, messages
  1. Packages: the fundamental unit in ROS, **containing nodes, libraries, configuration files, and other resources**. Organize the code and assets (like nodes and messages) used to **create a robot's functionality**.
  2. Nodes: **an executable** in ROS that performs computation. Multiple nodes can run simultaneously and **communicate with each other**.
  3. Topics: Topics are the channel through which **nodes exchange information**. It is **one-way, many-to-many communication**. Multiple publishers can send data to the same topic and multiple subscribers can listen.
  4. Publishers: a **node** that generates and **sends messages** to a specific **topic**.

5. Subscribers: a **node** that **listens** to messages on a specific **topic**.
6. Messages: data structures exchanged between nodes, containing information like sensor readings, control signals, or status updates.

- What is the ROS graph?

  Definition: Represents the **communication** structure **between different nodes** in a ROS system. It consists of nodes, topics, services, and other elements that define how data flows and interactions occur in the system.

- What ROS concepts are used for one-way many-to-many communication? Two-way one-to-one communication?
  - One-way, Many-to-Many Communication: **Topics**. A node (the **publisher**) sends messages to a **topic**, and **multiple nodes (subscribers)** can listen to and **receive** those messages.
  - Two-way one-to-one communication: **Services**. A node (**client**) **sends a request** to another node, and the receiving node (**server**) processes the request and **sends back a response**.

- No specific questions about the details of Project 1, but the main ROS concepts explored by that project may appear on the test.

5 Other wheeled systems

- What is an ICC? How to find ICC? What assumptions is this analysis based on? What happens if no ICC exists?
  - The point around which the robot moves with constant angular momentum is the ICC.
  - Draw lines from center of the wheels, perpendicular to their rolling direction.
    * If all the perpendiculars intersect at a point, that point is the ICC
    * If the lines overlap, the ICC can lie at any point on the overlap
    * If the lines do not intersect, then there is no ICC
  - This is based on 2 assumptions:
    * The wheels only roll (No sliding/slipping)
    * Wheels do not move relative to each other
  - If the ICC does not exist, the robot cannot move **without the wheels slipping/sliding.**

- ICC analysis for bicycle, synchro, tricycle, and Ackerman drive.
  - Synchronous drive: has three steerable drive wheels. All the wheels always **point in the same direction** and at the **same speed**. Turn through Rotate In-Place.
  - Ackermann drive: It is widely used in cars. It allows the wheels to turn at different angles when the vehicle takes a turn. They cannot steer by the same amount because there would be no ICC (due to parallel steering paths), causing the robot to not follow a curved path and resulting in slippage or inefficient movement.

- Alternatives to simple wheels for terrestrial navigation.

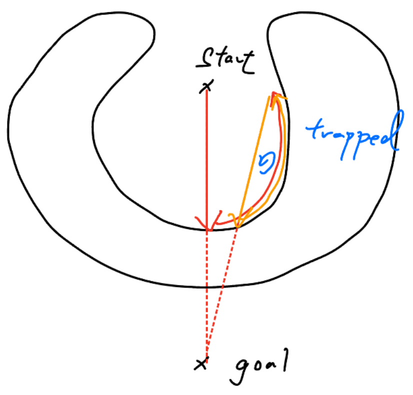

Figure 1: Sample case when Bug0 algorithm fails.

6 Navigation: Bug algorithms

- Basic model. How does the robot move? What can it sense? Be able to tell if the Bug algorithms are applicable in a given situation.

  Bug algorithms **DO NOT** require the robot to **know the obstacles** ahead of time. But requires two important pieces of information:

  1. Sense the **direction** and **distance** to the **goal**.
  2. Able to follow nearby obstacle boundaries and **measure the distance traveled**.

- Why doesn't the naive optimistic algorithm ('Bug0') work?

  Bug0 algorithm:

  1. Move toward the goal until reaching an obstacle.
  2. Follow the obstacle boundary until it's possible to move directly toward the goal.
  3. Repeat.

  The Bug0 algorithm does not work in some cases because it **does not ensure that the robot finds the optimal leave point** when following the obstacle boundary. As a result, the robot may **re-encounter the same obstacle** or become **trapped** in an inefficient path, making it possible for the algorithm to **fail to reach the goal**, especially in environments with complex or irregular obstacles. See Figure 1.

- Online vs. offline algorithms.
  - Online algorithm: Planning and execution interleaved (planning **on-the-fly**). Bug algorithms are online algorithms.
  - Offline algorithm: Forming a **complete plan before moving**.

- Hit points and Leave points
  - Hit point: the location where the robot first encounters an obstacle. At this point, the robot can **no longer move** directly **toward the goal** because the obstacle is in its path.
  - Leave point: the location where the robot decides to **stop following the boundary** of the obstacle and **resume moving** directly **toward the goal**.

- Bug1. Be able to execute the algorithm. Be able to state and explain an upper bound on path length. What if the goal cannot be reached?

  Bug 1 algorithm:

  1. Move toward the goal until reaching an obstacle.
  2. Turn left and follow the obstacle boundary. Keep track of which point on the boundary is closest to the goal.
  3. Return to this point, following the obstacle boundary in whichever direction is shorter.
  4. Repeat to 1. until the robot reaches the goal.

  Worse Case: $D + \frac{3}{2}\sum_{i=1}^{M} p_i$, where D is the distance between start and goal, M is the number of obstacles, and p_i is the perimeter of obstacle i.

  More Conservative: ensuring a **thorough search** for the optimal path around obstacles, but at the expense of efficiency.

  How to detect if it cannot reach the goal: The closest point on an obstacle to the goal leads to an immediate collision.

- Bug2. Be able to execute the algorithm. Be able to state and explain an upper bound on path length. What if the goal cannot be reached?

  Bug 2 algorithm (**More aggressive**):

  1. Move toward the goal until reaching an obstacle.
  2. Turn left and follow this obstacle until it returns to the line **connecting the start and goal positions**
  3. If the robot can leave the obstacle toward the goal from this point **and** it is closer to the goal than the hit point, then leave toward the goal. Otherwise, continue around the obstacle.
  4. Repeat to 1. until the robot reaches the goal.

  Worse Case: $D + \frac{1}{2}\sum_{i=1}^{M} n_i p_i$, where n_i is the number of intersection points between obstacle i and the start-to-goal line. D, M, p_i are the same as that in Bug1.

  More Aggressive: does not fully explore the obstacle's boundary but leaves the obstacle once finds the first suitable leave point.

  How to detect if it cannot reach the goal: When a complete cycle is made without finding a new leave point.

- Comparison between Bug1 and Bug2. Be able to explain when Bug1 is better. Be able to explain when Bug2 is better.
  - Bug1 is better in environments with complex or irregularly shaped obstacles.
  - Bug2 is more efficient in environments with simple, convex-shaped obstacles or where obstacles are spread out.

7 Navigation: Visibility graphs

- Applicability: Under what conditions can the visibility graph method be used?
  - Visibility graph algorithm is a navigation algorithm to find the **shortest path** to the goal in an environment with known planar **polygonal** obstacles
- Definition: What are the nodes? What are the edges? Be able to draw the visibility graph for a given set of obstacles.

  - A visibility graph a weighted graph that includes all paths as edges and between obstacle vertices, the start or the goal
  - Nodes of the visibility graph include the obstacles vertices, the start node x_s and the goal node x_g
  - Edges of the visibility graph are valid paths. A valid path is an obstacle-free line segment between a pair of nodes. Edges are weighted with weights equal to the distance between the nodes it connects

- How do we know that the shortest path must be along the visibility graph?
  - The visibility graph includes all straight-line segments between visible vertices, start, and goal.
  - Any path that does not follow the edges of the visibility graph would either intersect an obstacle or take a longer detour.
  - The shortest path is composed of straight-line segments between visible points, which are exactly the edges of the visibility graph. Thus, by definition, the shortest path must be part of the visibility graph.

- Difference between visibility graph and reduced visibility graph. Be able to identify edges that are in the visibility graph but not in the reduced visibility graph.

  An edge between obstacle vertices in the visibility graph is excluded from the reduced visibility graph if, when extended slightly on either side, it intersects with the obstacle.

8 Navigation: Potential fields

- When are potential fields applicable? Intuitive explanation of how potential fields work.

  Potential Field Navigation: Model the robot as a **particle** responding to an **attractive force** from its **goal** and repulsive forces from obstacles.

  Best in environments with well-defined, relatively simple obstacle configurations. It allows the robot to move toward the goal while **avoiding obstacles**. It's an **online** algorithm that calculates the navigation path according to the negative gradient of the potential field in real time.

- Domain and range of the potential function. The robot follows the **negative gradient** of the potential function.

- Definition of typical potential function as the sum of attractive and repulsive forces. Details for attractive and repulsive components.

- Notation: x, $U(x)$, ζ, η, Q_i^*, d_i

  1. x: the state
  2. $U(x)$: the potential at state x
  3. ζ: a scaling factor for the attractive potential from the goal
  4. η: a scaling factor for the repulsive potential from the obstacle
  5. Q_i^*: the maximum "range of influence" for obstacle i.
  6. d_i: distance from state x to the closest point on obstacle i

- Local minima. What are they? Why do they cause problems for potential fields? Possible solutions.

  Local minima: points other than the goal (the "global minimum") at which the gradient is zero.

  The local minima cause the robot to get "stuck". There is **no guarantee** that the robot will reach its goal.

  - Solution: Find a better potential function that **doesn't have local minima** (A navigation function) or give **momentum** when reaches a local minimum to escape.

9 Others

- There are three navigation algorithms: visibility graphs, bug algorithms, and potential fields. When is it possible to use each one? When is it a good idea?

See Table 2.

Algorithm	**When to Use**	**When is it a Good Idea?**	**Limitations**
Visibility Graphs	- Static, fully known environments - Polygonal obstacles	- Finding the **shortest path** in environments where **all obstacles** and the goal are **known**	- Not suitable for dynamic environments - Computationally expensive in large environments
Bug Algorithms	- Unknown or partially known environments - 2D planar navigation	- Simple, real-time navigation - Environments where efficiency is not critical - Low computational resources	- Inefficient paths - Struggles in cluttered environments
Potential Fields	- Dynamic or unknown environments - Small or simple environments	- Fast, reactive navigation - Local **obstacle avoidance**	- Risk of local minima - Poor performance with complex or non-convex obstacles

Table 2: Comparison of Navigation Algorithms: Visibility Graphs, Bug Algorithms, and Potential Fields

10 Important Equations

10.1 Differential Drive

$$R = (\frac{l}{2})\frac{v_r + v_l}{v_r - v_l}, \quad \omega = \frac{v_r - v_l}{l} \tag{1}$$

, where R is the distance from the ICC to the robot center, l is the distance between the two wheels, and v_l, v_r are the velocities for the left and right wheels respectively.

Location of the ICC:

$$c = [x - R\sin(\theta), y + R\cos(\theta)] \tag{2}$$

New state:
$$\begin{bmatrix} x' \\ y' \\ \theta \end{bmatrix} = \begin{bmatrix} \cos(\omega\Delta t) & -\sin(\omega\Delta t) & 0 \\ \sin(\omega\Delta t) & \cos(\omega\Delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - c_x \\ y - c_y \\ \theta \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \\ \omega\Delta t \end{bmatrix} \tag{3}$$

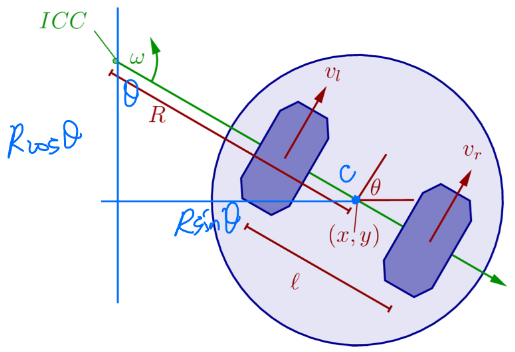

Figure 2: Differential Drive

$$U(x) = U_{goal}(x) + \sum_{i}^{n} U_{obst}^{(i)}(x) \tag{4}$$

$$U_{goal}(x) = \frac{1}{2}\zeta[d(x, x_{goal})]^2, \quad U_{obst}^{(i)}(x) = \begin{cases} \frac{1}{2}\eta(\frac{1}{d_i(x)} - \frac{1}{Q_i^*})^2, & \text{if } d_i(x) \leq Q_i^* \\ 0 & \text{otherwise} \end{cases} \tag{5}$$