Rheinische Friedrich-Wilhelms-Universität Bonn
Prof. Dr. Maren Bennewitz
Institut für Informatik
Abteilung VI
Humanoid Robots Lab
Adresse:
Friedrich-Hirzebruch-Allee 8
53115 Bonn

Humanoid Robotics

Assignment 5 (ungraded)

Discussion in Tutorial on 09.06.2026.

Motion Planning:

1. Collision Representation Trade-off **(Total: 3 points)**

A 3-link robot arm must plan motion in a 3D environment with a single static point obstacle (e.g., a peg or wall corner). Each link is a straight cylinder of length 1 meters and radius 0.2 meters. For collision checking, the following geometric approximations may be used:

- Full triangle mesh
- Axis-aligned box
- One sphere per link (radius = 1m)
- Multiple spheres per link (e.g., 5 overlapping spheres per link)

a. Rank these four options in terms of: **(2 points)**
(i) Ease of collision checking (1 = easiest)
(ii) Accuracy of representation (1 = most accurate)

b. Which representation offers the best balance for real-time manipulation in semi-cluttered scenes (e.g., picking from shelves)? Justify your answer in one sentence. **(1 point)**

2. RRT Motion Planning with Collision Checking **(12 points)**

In this task, you will implement and evaluate several variants of the Rapidly-Exploring Random Tree (RRT) algorithm for a mobile robot navigating a 2D environment represented by PNG file where black represents obstacles and white represents free space. The start and goal positions are specified for each world.

You will design your RRT planner in a modular and extensible way, guided by well-defined abstract interfaces. The assignment emphasizes software design, algorithmic understanding, and empirical evaluation. Your ultimate goal is to assess how different planning components affect feasibility, efficiency, and path quality. Some combinations may lead to failures, and it is important to report them.

The codebase includes abstract base classes for **Sampler**, **Connector**, and **CollisionChecker** interfaces. Also, we provide concrete implementations for **UniformSampler** and **PointCollisionChecker**. A partially implemented **RRTPlanner** is provided with visualization, tree tracking, and path saving utilities. A timer utility is also included for consistent runtime measurement.

Important: All your code must conform to the APIs of the provided abstract classes

a. Direct Connector Implementation: **(1 point)**
Implement direct connection (greedy connect to sample). Use appropriate method of point collision checking for the same.

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik Abteilung VI Humanoid Robots Lab

b. RRT Pipeline Implementation: **(2 points)**

Now, that we have a concrete implementation for Sampler, Connector and CollisionChecker, use an instance of each to develop a modular RRT planner. You should not hardcode sampling or collision logic. Implement the necessary logic in the run method of the RRT planner. Ensure your code is modular and extensible to support different types of samplers, connectors and collision checkers.

c. Goal Biased Sampling Strategy: **(1 point)**

Implement a **GoalBiasedSampler** where the planner should sample the goal with a fixed probability.

d. **Incremental Connection (fixed step size):** **(1 point)**

Implement an **IncrementalConnector** that connects the nearest node to the sampled point in small steps. This connector checks if the path from the nearest node to the sampled point is collision-free by moving in small increments (step size).

e. Collision Checking: **(2 points)**

You are already provided with a PointCollisionChecker. In addition, implement:
- **CircleCollisionChecker**: Treat the robot as a circular disc with radius 5 and 10 pixels
- **RectangleCollisionChecker**: Treat the robot as an axis-aligned rectangle (e.g., 10×5 px)

Ensure your implementation works for both point and segment validation.

f. Tabulation and Discussion: **(5 points)**

World	Sampler	Connector	Checker	Radius/Size	Notes
world1	uniform	direct	point	—	sanity check, baseline
world1	goalbiased	direct	point	—	effect of goal-bias in open space
world1	uniform	direct	circle	5 px	validate circle fit in open
world1	uniform	direct	rectangle	10×5 px	validate rectangle fit in open
world2	uniform	incremental(5)	point	—	narrow: fine step
world2	uniform	incremental(10)	point	—	narrow: coarse step
world2	uniform	direct	point	—	direct in S
world2	goalbiased	incremental(5)	point	—	effect of bias
world3	uniform	direct	point	—	baseline
world3	uniform	direct	circle	10 px	larger circle tests
world3	uniform	incremental(10)	point	—	step size effect
world3	goalbiased	direct	point	—	bias effect in clutter
world4	uniform	direct	point	—	baseline in two-room map
world4	uniform	direct	circle	5 px	tests collision in room
world4	uniform	direct	circle	10 px	tests collision in room
world4	uniform	incremental(10)	point	—	step vs branch choice
world4	goalbiased	direct	point	—	Effect of bias
world4	goalbiased	incremental(5)	point	—	full narrow, bias, fine-step
world4	goalbiased	incremental(10)	circle	5 px	all factors combined
world4	goalbiased	incremental(10)	circle	10 px	all factors combined

Use the provided experimental design table and run a total of 20 meaningful planner configurations across 4 different world maps. Your experiments must include combinations of different samplers, connectors, collision checkers, and robot sizes.

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik Abteilung VI Humanoid Robots Lab

A **Timer** utility function and a function for **calculating the path length** has been provided. Similarly, we have provided a function for saving results. Also, we have provided options to test configurations manually or a single yaml file or all the 20 yaml file configurations. We use a fixed seed 42 for random generation. So, results will be identical across runs.

You must tabulate the results (success/failure, runtime overall, runtime for collision, total iterations, path length) and write a short discussion addressing the following questions:
- What effect does goal biasing have on speed and success?
- How do collision checkers affect feasibility?
- What patterns do you observe as robot size increases?
- How does step size in the connector influence path smoothness or failure rate?
- Which configurations perform well in which types of environments?

3. Extending RRT to Bi-RRT Motion Planning **(Bonus: 5 points)**

The RRT algorithm builds a tree from the start state toward the goal. Bi-RRT (Bidirectional RRT) extends this idea by growing **two** trees — one from the **start** and one from the **goal** — and trying to connect them in the middle. In this bonus task, you must implement a modular version of Bi-RRT using the same Sampler, Connector, and CollisionChecker interfaces.

a. Bi-RRT Planner Implementation: **(3 points)**

Implement a class BiRRTPlanner that:
- Grows two trees: one from the start and one from the goal
- Alternates tree expansion between the two
- Attempts to connect the two trees when nodes are within a predefined threshold

Your implementation must:
- Reuse your Sampler, Connector, and CollisionChecker classes
- Return a complete path if a connection is found (start → meeting point → goal)
- Visualize the process similarly to RRT

b. Evaluation and Comparison with RRT: **(2 points)**

Run your Bi-RRT planner on the following 5 configurations:

World	Sampler	Connector	Checker	Notes
world1	uniform	direct	point	Trivial open-space: measures Bi-RRT overhead vs. RRT baseline
world2	uniform	incremental (5)	point	Narrow S-corridor: RRT vs Bi-RRT
world4	uniform	incremental (5)	point	Branching two-room
world2	uniform	incremental (5)	circle (r=10)	Large robot in narrow corridor
world4	uniform	incremental (5)	circle (r=10)	Large robot in branching map

For each configuration, compare Bi-RRT to your original RRT planner in terms of:
Success or failure, Runtime Overall, Path length, Number of iterations until connection. Present your results in a small table and answer the following:

- Did Bi-RRT reach the goal faster or more reliably?
- In which worlds did Bi-RRT show a clear advantage over standard RRT?