

COMP-598 - Assignment 1

Version 1.0 of this assignment:.

Due Oct 9th, 5pm. Late assignments will be penalized.

1 Assignment 1

This assignment involves the development and assessment of sampling based path planners. The starter contains a 2D RRT with uniform sampling, obstacle worlds, map images, and plotting code. You will extend it to the other robot models and planning methods below.

To make it more interesting, we will also makingg the RRT more realistic by examining a simple kinodynamic planner (reviewed below).

Finally, you should think about how an RRT might have problems. To illustrate this, you should concoct a scenario, and illustrate how the scenario is difficult.

For all stages and for your code. you are expected to read it, run it, and then find out where its choices help and where they fail.

There is a zip file called “assignment1_starter_pack.” It provides fixed environments (worlds), map images, one small geometric RRT, and a drawing convention.

The core sample code there is: `uniform_rrt_example.py`

You need to read it yourself and fully understand it.

You are allow to use AI to help you, but you are not allowed to just delegate the assignment to your AI. If you want to do the assignment with AI help, that’ OK, but the assignment is heavy for a pure manual effort because AI assistance is now the de facto standard.

You should do planning and development by hand and you must understand and be able to explain results and the code. The assessment is therefore dependent on the experiments, the comparison of cases, and your explanation of the decisions in the code. A simple code base or a report which only says that a planner succeeded is unsatisfactory.

1.1 Backgrounder: What kinodynamic means

As seen in class (but fast), the word kinodynamic combines motion and dynamics.

Kinematics is the study of motion, or motion constraints, without considering the associated forces (mass, inertia). Examples include geometric boundaries, steering angles, and joint limits for robot arms.

Dynamics is the study of forces and torques that cause motion. It bring in ideas from physics like mass, momentum, and angular velocity. Examples concerns from dynamics include engine power, friction, angular momentum, and gravity.

A geometric planner asks whether a configuration is free and then connects two configurations with a local geometric path. A *kinodynamic* planner also includes quantities such as velocity in the state. It can move only between states that the robot’s controls and dynamics can reach: can’t instantly turn 90 degrees if your vehicle has inertia.

In parts of this assignment the state of a 2D point point robot will be:

$$s = [x, y, v_x, v_y]$$

where x and y are the current position coordinates, and v_x and v_y are the corresponding components of velocity. Thus v_t means the velocity at time t ; the subscripts x and y select its two components.

Thus, corresponding control for the vehicle (that is, the input to change the state) is a bounded acceleration

$$u = [a_x, a_y].$$

Here a_x and a_y are the commanded acceleration components, and u is the control vector chosen by the planner. The bounds on u are the control limits of the simulated robot.

The simple double-integrator model says that velocity changes because of the chosen acceleration and position changes because of velocity. The simulator uses small time steps to approximate this:

$$v_{k+1} = v_k + u_k dt, \quad p_{k+1} = p_k + v_{k+1} dt.$$

In these equations, $p_k = [x_k, y_k]$ is position, $v_k = [v_{x,k}, v_{y,k}]$ is velocity, and $u_k = [a_{x,k}, a_{y,k}]$ is the control used during step k . The integer k counts simulation steps, dt is the duration of one step, and p_{k+1} and v_{k+1} are the position and velocity after that step. The notation is discrete, but it approximates motion over ordinary time t .

The velocity and acceleration limits are applied at every step. A rollout is the short sequence of positions produced by holding one sampled control. The kinodynamic RRT grows a tree of states by trying several such rollouts from the nearest state, discarding rollouts which hit an obstacle, and adding the valid endpoint which is closest to the sampled target under the state metric.

This differs from ordinary RRT in a practical way. A point may be in open space while the robot is moving too quickly to turn or stop before an obstacle. The planner must find both a position and a usable velocity. The bottleneck world in this assignment makes that distinction visible.

Forward Propagation is then: Given a state q , a control u (i.e. a simulated acceleration), and a time step dt , compute the next state by applying the update equations above. State the value of dt used in each run.

References: (1) Randomized Kinodynamic Planning, Steven M. LaValle and James Kuffner, The International Journal of Robotics Research, 2001. https://www.researchgate.net/publication/247330158_Randomized_Kinodynamic_Planning

(2) Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation, Kleinbort et al, 2022. <https://arxiv.org/pdf/1809.07051>

What to do

1. Extend the starter and produce vanilla RRT results for a 2D point robot, a 3D point robot, and the rod in (x, y, θ) , where θ is the rod's orientation angle. Check the collision tests, the nearest-node metric, the steering step, and the parent-chain path. Make at least one small change to a parameter and record what changed.
2. For the RRT algorithm, you select points to grow the tree using sampling. Compare the effects of (A) uniform sampling, (B) exact goal bias (picking the goal itself), and (C) Gaussian sampling bias centered at the goal. Use the cluttered world and *at least* one world of your own invention. Choose the parameters rather than copying one successful seed. Explain what happens near obstacles and give a case in which the Gaussian choice is worse, and play with the standard deviation of the Gaussian a bit. As σ varies from very small to very large, how does this relate to the other sampling distributions (uniform and at-the-goal)? You will need to explain this with figure(s) and in your own human-generated explanation.
3. Compare ordinary RRT, Momentum-RRT, and one variation of your own in the serpentine and straight-channel worlds. Here Momentum-RRT means directional persistence in geometric extension: each node remembers the unit direction h of its incoming edge. If d points from that node toward the new sample, extend along the unit direction of $(1 - \alpha)d + \alpha h$, where $0 \leq \alpha < 1$. At the root, or if this sum is zero, use d . Choose and report α ; $\alpha = 0$ gives ordinary RRT. This is a search heuristic; its nodes do not contain velocity. The double-integrator planner in Item 5 does. For your own variation, change one sampling or extension rule, state it precisely, and test all three planners in both worlds. Explain where each change helps and where it hurts. Examples for your rule include changing the sampling distribution after repeated collisions, choosing a parent from several nearby nodes, or changing the extension length according to local clearance.
4. Use the image-to-rectangle code on a binary or grey-scale map (or make a map any other way you wish). Make a map with a thin obstacle and a narrow passage. Test whether the map conversion loses or enlarges obstacles. Include one test which would be missed by coarse point sampling along a segment. Explain the effect of rectangle boundaries and of the threshold used to binarize the image. The supplied obstacle-only maps in `student_starter/maps/` are test inputs. Add at least one map of your own.
5. Study the double-integrator planner. Use the open-room / narrow-middle channel example and one variation which changes the width or the length of the middle passage. Here $a_{\max}$ is the maximum allowed acceleration magnitude, $v_{\max}$ is the maximum allowed speed, and dt is the simulation time step. Vary at least two of these quantities, the number of simulation steps, the velocity weight, and the number of trial controls. The velocity weight is the coefficient used by the state-distance metric to decide how much a velocity mismatch matters; the number of trial

controls is how many candidate accelerations are tested from each selected tree node. Find a setting which works reliably and a setting which fails or takes substantially longer. Include the speed profile and explain whether the failed case is caused by dynamics, sampling, collision checking, or the stopping rule.

6. Run vanilla RRT in the same open-room / narrow-middle world. Compare its tree and path with the kinodynamic result. Discuss a case where the geometric planner gets through quickly and a case where carrying velocity makes the dynamic planner more constrained.

1.2 Implementation details

Use the supplied Python code as a starting point. You may reorganize it when there is a reason. Keep the planner separate from the robot and environment code. A change to the collision test should not silently change the meaning of the 2D, 3D, rod, and kinodynamic cases.

Every reported run must have a seed (random initialization value) that you should record and report. For comparisons, use the same seed list for each method and the same iteration or time budget. Twelve seeds is a reasonable minimum for a small comparison; use more when a success rate is close to 50 percent. Report the time budget, the map dimensions, the extension step or simulation parameters, and the software version used to make the figures.

AI coding agents may be used. Keep a short log of the prompts or commands which led to substantial code changes. You remain responsible for checking the code. In particular, inspect a few paths point by point, test a collision case that you designed, and be prepared to explain the nearest-node metric, the local planner, and the stopping condition without asking the agent.

Do not replace a measurement with a screenshot of a successful run. Save the raw run data or print enough information to reproduce the table. Keep failed runs in the data. A timeout, an invalid start state, and a planner which found no eligible extension should be identified separately.

The figures may be made with any graphics package. The supplied drawing code produces PNG files. You may use Matplotlib, ffmpeg, or other tools to make GIF or MP4 animations. Use figures which make the obstacles, the tree, and the final path distinguishable. Thickening a line is useful when a figure is viewed in a report, but it does not improve a collision test.

1.3 Analysis

For each comparison, give the success rate, median and range of iterations, number of nodes, CPU time, and path length for successful runs. Include the number of failures and the reason for failure when it is known. For the kinodynamic case also report the maximum speed, the final speed, and the number of controls tried per extension.

Discuss memory use qualitatively as well as CPU time. Find ways to measure and report this if you can. Give a best case and a bad case for at least one planner. An order estimate is sufficient when a precise measurement would be misleading, but state what is being counted.

The important comparison is not simply which method has the smallest median. Look for cases in which a heuristic improves a narrow passage and cases in which the same heuristic destroys exploration. Check whether a shorter path was obtained by spending more time. Check whether path simplification changes the reported length without changing the search tree. For the dynamic planner, comment on whether the state metric rewards a useful state or merely a nearby position.

Use plots or small tables for the results. Include at least one plot against the changed parameter and one plot or figure showing a good case beside a bad case. A single representative animation is useful, but it is not a substitute for the seed summary.

1.4 Evaluation

Submit the source code, a report of approximately 6–10 pages, and the data needed to regenerate the main tables and figures. The report should contain:

1. a short description of the planner interface and the robot models;
2. the experiment questions and the parameter ranges;
3. results for the point, rod, image-map, channel, and kinodynamic cases;

4. for your heuristic, one good case and one bad case, each tied to a specific mechanism (not just the two numbers being compared);
5. the limitations of your collision checking and stopping conditions, each backed by a concrete case or discretisation you identify in the code, not a generic caveat; and
6. a short, checkable account of AI-agent assistance: at least one change you tried, measured, and rejected, with the data that made you reject it.

Label every figure with the map, planner, seed or seed count, and important parameters. Give path length units and time units. If a run did not finish, say so and show the budget which was used.

You may be asked to explain your approach, implementation, and results verbally (i.e. in person, live). The explanation may use a small test case instead of the largest experiment.

1.5 Grading emphasis

Some requirements above are easy to satisfy mechanically – a seed, a budget, a labelled figure, a table of medians – and are checked mainly for presence, not depth. These carry little weight on their own: a seed number, reported budget/dimensions/step size/software version, figure labelling, and the summary table of success rate/nodes/CPU/path length.

The important thing is for you to show your understanding of the material.

Most weight is on the parts that cannot be produced without exploring the domain with the code:

- the Gaussian-worse-than-uniform case (Item 2) – must be a case you found, with parameters you chose, not the supplied default;
- the good/bad case for your heuristic, explained by mechanism, not just by the two numbers;
- the rod’s angular-weight effect on nearest-node choice, and whether the kinodynamic metric rewards a useful state or merely a nearby position – both require pointing at specific path data;
- the diagnosis of *why* the failing kinodynamic setting fails (dynamics, sampling, collision checking, or stopping rule) – a specific, falsifiable claim, not a restatement of “it failed”;
- a collision-checking limitation you found by reading the actual discretisation, not a generic “it might miss thin obstacles”;
- the rejected-change account – must include what you measured before rejecting it, since an unfalsifiable log is worth little; and
- the live explanation, which is the one item nothing else on this list substitutes for.