# Assignment 2

Sampling-based Motion Planning

Deadline: November 11, 11:59pm
Perfect score: 100 points

**Assignment Instructions:**

**Teams:** Assignments should be completed by teams of students. Up to three members are allowed for CS460 students and up to two members for CS560 students. Since the assignment has different requirements for the undergraduate and graduate version of the course, we will not allow mixed teams in terms of CS460 and CS560. No additional credit will be given for students working in smaller than the allowed teams. You are strongly encouraged to form a team of the maximum size. Please inform the TAs (email: comprobfall2018/AT/gmail.com) as soon as possible about the members of your team so they can update the scoring spreadsheet. Failure to inform the TAs on time will result in your assignment not being graded.

**Submission Rules:** Submit your reports electronically as a PDF document through Sakai (`sakai.rutgers.edu`). For your reports, do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. You also need to submit a compressed file via Sakai, which contains your results and/or code as requested by the assignment. *Each team of students should submit only a single copy of their solutions and indicate all team members on their submission*. Failure to follow these rules will result in a lower grade in the assignment.

**Program Demonstrations:** You will need to demonstrate your program to the TAs and grader on a date after the deadline. The schedule of the demonstrations will be coordinated by the TAs. During the demonstration you have to use the files submitted on Sakai and execute it either on your laptop or an available machine, where the TAs and grader will be located (you probably need to coordinate this ahead of time). You will be asked to describe the architecture of your implementation and algorithmic aspects of the project. You need to make sure that you are able to complete the demonstration and answer the TAs' questions within the allotted 12 minutes of time for each team. If your program is not directly running on the computer you are using and you have to spend time to configure your computer, this counts against your allotted time.

**Late Submissions:** No late submissions are allowed. You will be awarded 0 points for late assignments.

**Extra Credit for LaTeX:** You will receive 6% extra credit points if you submit your answers as a typeset PDF (i.e., using LaTeX). Resources on how to use LaTeX are available on the course's website. There will be a 3% bonus for electronically prepared answers (e.g., on MS Word, etc.) that are not typeset. Have in mind that these bonuses are computed as percentage of your original grade, i.e., if you were to receive 50 points and you have typesetted your report using LaTeX then you get 3 points bonus. If you want to submit a handwritten report, scan it and submit a PDF via Sakai. We will not accept hardcopies. If you choose to submit scanned handwritten answers and we are not able to read them, you will not be awarded any points for the part of the solution that is unreadable.

**Precision:** Try to be precise in your description and thorough in your evaluation. Have in mind that you are trying to convince skeptical evaluators (i.e., computer scientists...) that your answers are correct.

**Collusion, Plagiarism, etc.:** Each team must prepare its solutions independently from other teams, i.e., without using common code, notes or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Unless explicitly allowed by the assignment, do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the course, the department or the university (the standards are available through the course's website). Failure to follow these rules may result in failure in the course.

Course's website:
`https://robotics.cs.rutgers.edu/pracsys/courses/intro-to-computational-robotics/`

# Part A. Sampling-based Motion Planning for a Rigid Body in SE(3)

## 1 Setup

For this part of the assignment you are asked to compute a collision-free path for a rigid body (a piano) among static obstacles (a room) by implementing the Probabilistic Roadmap Method (PRM) as well as its asymptotically optimal variant (PRM$^*$) and evaluating their performance in terms of computational efficiency and path quality. Figure 1 provides a visualization of the environment you will consider.

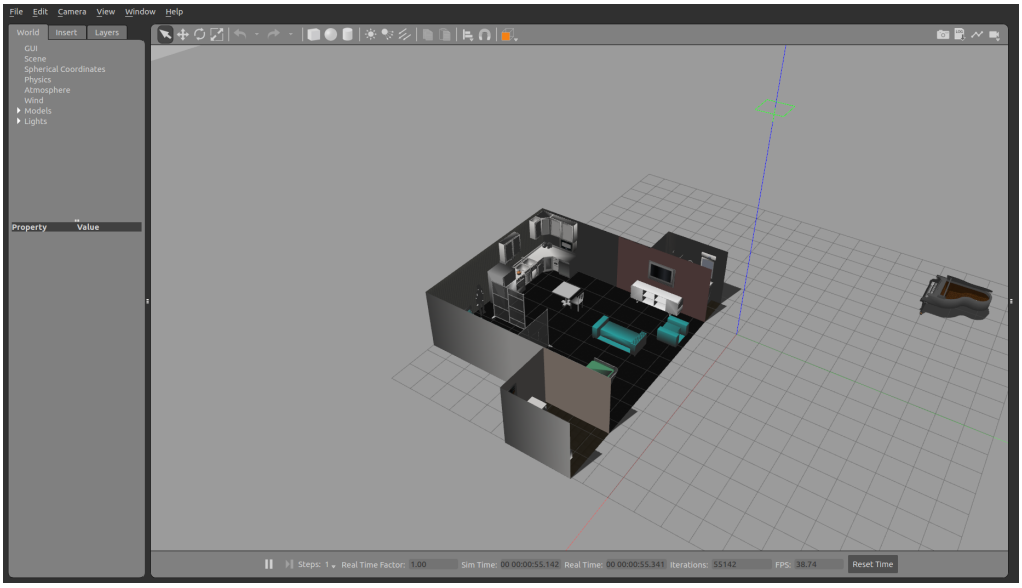


Figure 1: The environment in Gazebo where you need to compute collision-free paths for a piano.

### 1.1 Software Infrastructure and Available Input

You will be provided access to the following pieces of software:

- The collision checking library PQP.
- A library for computing nearest neighbors, developed by the PRACSYS group at Rutgers: `https://bitbucket.org/pracsys/graph_nearest_neighbors/`

You will also have access to 3D models representing the shape of the obstacles in the environments, as well as the 3D model of your "robot", i.e., the piano, which needs to move in a collision-free manner.

The software for both Part A and Part B of this assignment has been publicly hosted at the following link: `https://github.com/rajanya/comprobfall2018-hw2`. Please follow the instructions that we forwarded along with Assignment 1 to create a private fork of this repository, or download it as a .zip file and create your own private repository for collaborating with your group members. The README file inside the repository contains further instructions on how to use the code.

For this component of the assignment, your communication with Gazebo will remain rather simple and similar to the first assignment. In other words, you will just specify a sequence of configurations for the piano so that it moves in the simulation environment from an initial to a goal configuration without collisions.

### 1.2 Review of PRM and PRM$^*$

Figure 2 provides an outline of a version of the Probabilistic Roadmap Method, which incrementally samples collision-free configurations and connects them to neighboring nodes on the roadmap given a distance function and a local planner. Each configuration in this assignment corresponds to a point in SE(3).

```
Input: number of iterations n, configuration space Q with distance function
distance(◦, ◦)
Algorithm: V=0, E=0
for n iterations
    repeat
        q ← sample(Q)
    until( collision_free_configuration(q) == true )
    V ←V∪ {q}
    N ← neighborhood(q,V, E, distance(◦, ◦))
    for all q'∈ N
        path ← local_planner(q,q')
        if ( collision_free_path(path) == true ) then
                E ← E∪{(q,q')}
        endif
    endfor
endfor
```

Figure 2: The outline of the Probabilistic Roadmap Method (PRM) as presented during the lectures.

At a high-level, you will need to construct a graph $G(V, E)$, where nodes $V$ correspond to collision-free configurations of the robot and edges $E$ correspond to collision-free paths that connect pairs of nodes. This construction will happen via a sampling-based process. At each iteration of the algorithm you first sample a point $q$ in SE(3) until you confirm that it is collision-free. Once you have discored such a point, you added it to the set of roadmap nodes $V$. Then, you define the neighborhood $N$ in the set $V$ of the newly added node $Q$ given a distance function $distance(\cdot, \cdot)$. For all nodes $q'$ in the neighborhood, you define the local path connecting $q$ and $q'$. Then, you evaluate whether the local path is collision free or not. If it is, then you can add the edge $(q, q')$ in the set $E$ and store the corresponding local path.

Once you are given an initial $q_{init}$ and a goal $q_{goal}$ configuration, you can query the roadmap (this process is not shown in the above figure) in order to discover a path. This requires first connecting $q_{init}$ with collision-free paths to nodes in its neighborood $neighbood(q_{init}, V, E, distance(\cdot, cdot))$ and similarly doing so for $q_{goal}$. Once these two configurations have been added on the roadmap and they belong in the same connected component, then you can use a search algorithm (e.g., Dijkstra's or $A*$ if you define a heuristic) on the corresponding graph in order to find a path between $q_{init}$ and $q_{goal}$ in the continuous space. This is the path that you should communicate to Gazebo as a solution to your problem.

Given this basic framework, there are multiple variations of the PRM algorithm that you can consider:

- You can aim for computational efficiency and apply the "connected components" heuristic, i.e., do not try to connect nodes $q$ and $q'$ if they are already connected with a path on the roadmap. This version of the algorithm generates a tree and significantly reduces the amount of collicions checking that the method requires.

- You can aim for a more dense roadmap, which can improve path quality and define the neigborhood $N$ of each new node $q$ to be a constant number, e.g., $k = 3, 5$ or $10$. You then connect $q$ to the nodes in $N$ for which you can discover a collision-free path.

- You can aim for the asymptotically optimal variant PRM$^*$ [1], where the number of neighbors depends on the number of nodes on the roadmap, i.e., $k(n) \geq e(1 + \frac{1}{d})log(n)$, where $d$ is the dimensionality of your configuration space and $e$ is the base of the natural logarithm (if you are interested in more details, see page 9 of the PRM$^*$ paper [1]).

For your implementation, you have to make a choice on how to represent $SE(3)$ configurations, how to sample them, how to compute distances between them and how to interpolate between two neighboring configurations. If you decide the go the route of quaternions, a useful reference that can guide your choices is the paper "Effective Sampling and Distance Metrics for 3D Rigid Body Path Planning" by James Kuffner [2], one of the co-authors of the original RRT paper (today chief executive officer of the Toyota Research Institute).

Note that the collision checking package PQP uses matrices in order to set the configuration of objects internally. If you use quaternions, you will have to switch to matrices for collision checking purposes. You can also decide to have your entire software using transformation matrices to represent configurations.

### 1.3 Questions and Deliverable

1. Describe your choice for representing configurations in SE(3) and how you are sampling collision-free configurations. Argue that your sampling process is indeed generating uniform at random configurations in the free part of the configuration space. [5 points]

2. Describe your choice of a distance function in SE(3) and why do you think that it is appropriate for finding good neighborhoods. [5 points]

3. Describe your implementation for the local planner in SE(3), as well as how you collision check the corresponding path, and why these are the right choices for the problem. [5 points]

4. Implement the following versions of the PRM* algorithm and show that you can construct correct roadmaps in each case:

   - The connected components heuristic version [5 points extra credit for CS460, 5 points for CS560]
   - The constant $k$ version [7 points for CS460, 5 points for CS560]
   - The PRM$^*$ version [8 points for CS460, 5 points for CS560]

5. Sample 50 query points $q_{init}$ and $q_{goal}$ so that the piano is resting stably on the ground in each one of these configurations, it is inside the room and does not intersect with any other geometry. [5 points]

6. Solve the corresponding path planning problems for these 50 queries with the various versions of the PRM algorithm. Generate graphs where the x axis corresponds to computation time and the y axis corresponds to path quality. Report the resulting path quality (and the first time that a solution is found) for the various versions of the PRM algorithm and as the size of the roadmap increases. [10 points]

7. Visualize your solutions on Gazebo and demonstrate that they are indeed collision-free [10 points for CS460, 5 points for CS560]

# Part B. Sampling-based Motion Planning for a Physically Simulated System

## 2 Setup

For this part of the assignment you are asked to compute feasible controls for a car-like vehicle that result in a collision-free trajectory in a maze-like environment by implementing the Rapidly-Exploring Random Tree (RRT) algorithm as well as variants and evaluating their performance in terms of computational efficiency and path quality. The dynamics of the car-like vehicle can be accessed only through the physics engine. Figure 3 provides a visualization of the environment you will consider.

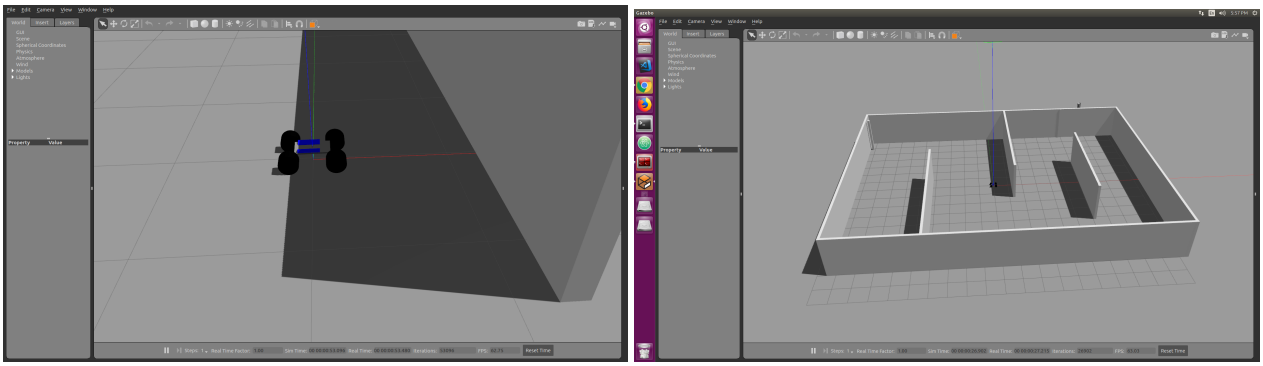

Figure 3: The Ackermann vehicle in the Gazebo environment and the corresponding workspace for computing trajectories.

## 2.1 Software Infrastructure and Available Input

For this part of the assignment, you will integrate tighter with Gazebo and the underlying physics engine that it uses.

In partiular, the topic `/gazebo/model_states` outputs messages of type `gazebo_msgs/ModelStates`, which consists of the following information:

```
string[] name # model names
geometry_msgs/Pose[] pose # desired pose in world frame
geometry_msgs/Twist[] twist # desired twist in world frame
```

`geometry_msgs/Pose` consists of the pose, which is simply the position and orientation of the rigid body in the world frame. `geometry_msgs/Twist` consists of the twist, which corresponds to the linear and angular velocities of the rigid body in the world frame.

The model that we are interested in is "`ackermann_vehicle`". Before and after propagating a control, you must save the "`model_state` of the `ackermann_vehicle` alone. These will form the nodes of the RRT.

If you wish to move between nodes after an edge has been added to your RRT, or if you simply wish to reset the Ackermann vehicle to a previously visited state, you must publish to the topic `/gazebo/set_model_state`, that accepts messages of type `gazebo_msgs/ModelState`, which is defined below.

```
# Set Gazebo Model pose and twist
string model_name # model to set state (pose and twist)
geometry_msgs/Pose pose # desired pose in reference frame
geometry_msgs/Twist twist # desired twist in reference frame
string reference_frame # set pose/twist relative to the frame of Body/Model
# leave empty or "world" or "map" defaults to world-frame
```

In the above message, the `model_name` will be "`ackermann_vehicle`", whereas the pose and twist would be previously obtained pose and twist information from the `/gazebo/model_states` topic. This will reset the `ackermann_vehicle` to the desired pose and twist.

## 2.2 Questions and Deliverable

1. Implement the basic kinodynamic RRT algorithm, where you perform random sampling of a state, you discover the closest node on the tree and then you sample a random control and duration so as to propagate an edge from the closest node. [5 points]

2. Implement the integration of the propagation process with Gazebo. Define the start state at the bottom left of the maze environment and a "goal region" at the top right. User your RRT implementation integrated with Gazebo in order to discover solution trajectories. Demonstrate these trajectories in Gazebo as described above. [20 points for CS460, 15 points for CS560]

3. Explain your choice for a distance function in order to achieve voronoi-bias in the state space [5 points]

4. Implement a variation of the kinodynamic RRT that is greedier, i.e., it uses information about distance to the goal region in order to bias the propagation of the tree. Demonstrate that you can improve solution time and path quality in this environment. [15 points for CS460, 10 points for CS560].

5. Run multiple experiments (50) of the RRT and its greedier version that you proposed. On a graph of path quality versus computation time provide both the mean and the standard deviation of the solution quality achieved as well as the first solution time. [15 points]

# References


[1] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[2] J. J. Kuffner, "Effective sampling and distance metrics for 3d rigid body path planning," in *Proc. of the IEEE Intern. Conference on Robotics and Automation (ICRA)*, 2004.