# Course 7- KUKA Path Planning Project

## Project Introduction

### Welcome!

Video 1.1

**The KUKA Path Planning Project**

Welcome to the KUKA path planning project! In this project, you will solve a maze given a start and goal state. You will implement a path planning algorithm to search for a path, and then move an object from start to goal state inside the maze with the help of a 6-DOF industrial KUKA arm. Later, you will have a chance to submit your code and watch it on a real KUKA arm located at the KIT Lab in Germany!

**PRACTICE PHASE**

Any RoboND student may try the project and "submit" it to the KIT Robotic Learning Lab during the PRACTICE PHASE of the project (see **Schedule**). There are two sets of poses for the mazes that will be run, and a running leaderboard will be available to see how you're doing! The results, logs, and videos will be available on a continuous basis from these runs, provided by KIT.

**CONTEST PHASE**

After the PRACTICE PHASE is over, the CONTEST PHASE will begin. In order to enter the final CONTEST PHASE, you must submit your code using the "contest-submit" instructions. See the **Project Details** lesson. This final phase will accept entries for only a few days. This phase will use *hidden* poses and you won't be able to see any results until all the entries are run after the close of the contest.

Special note: You must be eligible per the **Contest Terms and Conditions** to participate in the CONTEST PHASE. In particular, the contest is only open to United States residents (please see the **Contest Terms and Conditions** concept for all details).

**Schedule**

| **August 1, 2018** | **Project opens with the first practice pose set Maze 1** | **PRACTICE PHASE** |
| --- | --- | --- |
| August 20, 2018 | Maze 1 closed; second practice pose set opens Maze 2 | PRACTICE PHASE |
| August 24, 2018 | Maze 2 closed; | PRACTICE PHASE |
| August 25, 2018 | contest opens (hidden pose set) Contest Maze | CONTEST PHASE |
| August 28, 2018 | Contest Mazeclosed; final runs begin | CONTEST PHASE |
| August 29, 2018 | Contest Mazeclosed; first practice pose set opens Maze 1 | PRACTICE PHASE |

During the PRACTICE PHASE, results logs and videos will be released as completed. During the CONTEST PHASE, results will be released only after all runs are completed.

### Meet KIT and KUKA

Video 1.2

Trailer on the KIT and KUKA from Intersect 2018

**KIT and KUKA**

The KUKA path planning project is brought to you in partnership with KUKA Robotics and the Karlsruhe Institute of Technology (KIT) Robotic Learning Lab (RLL). KUKA Robotics, a leading manufacturer of industrial robots, has provided the KIT lab with real-world 6-DOF (degrees of freedom) industrial robotic arms for you to interface with.  
The KIT lab has provided the real-world setup as well as the project SDK (software development kit), which is a replica of the physical lab setup in gazebo simulation. They also built the web interface we'll use to submit your code to the hardware, and from which you'll view video captures of your code running on the hardware!

"KUKA is truly excited about the Robot Learning Lab at KIT, and our collaboration with Udacity. Together, we’re pushing robotics research, scalable online education, and worldwide community-building. This unique offering establishes a new model for globally training and educating excellent talents"  
      -- by DR. Rainer Bischoff, Head of Corporate Research, KUKA

### Project Overview

In this project, you’ll implement a path planning algorithm by navigating a maze. In general, path planning problems through mazes are reduced to two dimensions: the x and y position of the object. However, this project will extend the search space by a third dimension, the orientation of the gripper around the vertical z-axis.

A computer screen shot of a game

AI-generated content may be incorrect.

**Execution Cycle**

At each run, the robot will follow this cycle:

1. First, the KUKA gripper will move to the object start position, orient itself with respect to the object, grab the object, and lift it up.
2. At this stage, your path planning code will be executed. Your code will search for a path and navigate the robot through the maze by commanding 2D positions, and an orientation angle, in order to get around corners in the maze
3. You will have a total time of 8 minutes to search for a path and move toward the goal pose. Once reached, your cuboid object will be placed in the goal pose.
4. Finally, the robot will lift the object from the goal pose, return it back to the start pose, drop it, and get ready for another run.

### Contest Terms & Conditions

**Robotics Nanodegree Program Challenge with Kuka and KIT**

We're so happy you've decided to participate in the Robotics Nanodegree Program Challenge with Kuka and KIT (“Contest”) for the Robotics Software Engineer Nanodegree Program!

Please be sure to read through the [**Contest General Terms and Conditions(opens in a new tab)**](https://www.udacity.com/legal/contests) (“T&Cs”) as well as these Contest details below before submitting entering the contest as this will govern your participation in the Contest.

**Contest Dates:**

August 25, 2018 (12:00 am GMT +2) - August 28 (11:59 pm GMT +2)

**Eligibility to Enter:**

In addition to the other requirements set forth herein and in the T&Cs, to participate and claim the prize package, entrant must be 1) a US resident; and 2) either currently enrolled in or a graduate of Term 1 of the Program. Only one (1) winner will be selected and there is no option for team submissions.

**Contest Entry:**

**Submission:**

Participants will compete by submitting their code (“Code”) to program a Kuka robotics arm to navigate a physical 2D maze in the fastest time possible. Students who are currently enrolled or who have successfully completed Term 1 of the Robotics Software Nanodegree Program (“Program”) will find the **Kuka Project Challenge** in the extracurricular section the Program classroom. The code must successfully pass the simulator before it can be run on the Kuka robotic arm at the lab in Germany. To “pass” the simulator, the Code must successfully navigate the robotic arm through the maze from the starting point to the goal point without any errors and without hitting or attempting to move through the walls of the maze.

Subject to the Code passing the simulation, it, will be added to the queue to run on the physical robotic arm and run by the KIT team in Karlsruhe, Germany at the Kuka Learning Lab (“Lab”).

**Run-Time:**

Participant submissions will be timed at the Lab using standardized software. The Kuka robotics arm will only begin to move after it has planned the path and poses according to the Code provided by the participant. The run-time for the challenge therefore includes both the robotic arm’s time to plan and then execute its movement along the maze. The clock, controlled by KIT’s program for executing the code, will begin when the Code starts to run and end when the robotic arm has moved the object to the goal location within the maze. Reducing the time required to plan the robotic arm’s path is a critical component to winning this Contest.

To account for small natural variations that occur in the code’s run-time, it will be run on the arm three separate times. The **median or (middle)** run-time will be the participant’s recorded time for the Contest. Participants are therefore encouraged to submit Code that runs consistently well rather than Code that is prone to wide variability. A participant’s best and worst run-time will be discarded and not considered for determining the Winner of the Contest.

**Ties:**

In the event that two or more participants’ Codes have the same run-time (rounded to the nearest 1/100th of a second), the participant with the faster **average** run-time, as determined as the average of the three run-times of their code, will be declared the winner.

* Leaderboard and Leaderboard Participation:

Throughout the challenge, participants will have the opportunity to view their run-time as well as the run-times of fellow participants on a leaderboard in the Kuka Challenge classroom.

Please note that participants are not required to participate in the publicly viewable leaderboard to participate in this Contest. If a participant elects to not to opt-in to the leaderboard, their run-times for each phase will not be visible to other participants. To the extent you opt to be visible in the leaderboard, please keep in mind that this leaderboard will be visible to others in the classroom. You should not share personal information on the leaderboard (e.g., real name, etc.). Please use an anonymous moniker. If you choose to provide your real personal information in the leaderboard, you acknowledge and agree that it will be viewable by others and that such sharing is subject to our Privacy Policy (including the social sharing provisions). If you choose to opt-out after initially opting-in, you may do so, but information you shared prior to opting-out may be residually available/viewable.

The top 5 entrant’s scores will be placed on a leaderboard that can be viewed within the classroom. Student’s real names will not be used on the Leaderboard. Even if you choose not to put your name on the publicly viewable leaderboard, you will be able to see your own run-time so that you can rank it against the leaderboard and your run-time for the final submission will be eligible to win the challenge.

**Winner Notification:**

On September 1, 2018, Udacity will notify the Winner via the email associated with their Udacity account and provide additional information on how to claim the prize. Winner will forfeit eligibility if they fail to respond within 7 days.

**Contest Prize:**

One (1) prize package will be awarded to one (1) winner consisting of:

* One (1) 1 round trip economy plane ticket to Munich, Germany from your home of record inside the U.S.
* One (1) economy class train ticket from Munich, Germany to Karlsruhe, Germany to visit the KIT lab
* 6 nights’ accommodation for one person in Munich & Karlsruhe, Germany capped at $200/night
* Daily per diem for meals for one person capped at USD $75.00/daily
* A 1-day visit with head of KIT lab (to be coordinated directly with KIT)

ARV approximately USD **$3,310.00**

**Disclaimers:**

Certain portions of this Contest are run by and/or under the control of our partners KIT or KUKA. Entrant acknowledges that Udacity has no control over or responsibility for technical or other portions of this Contest that are provided by these partners.

## Project Details

### Project Specification

Typically, path planning algorithms are applied on *known* maps. However, in this project, you’ll have to search almost blindly, which makes for a challenging problem. Here’s what we know about the maze:

A white maze with black lines

AI-generated content may be incorrect.

**Only its width, and length!**  
In addition to the maze dimensions (1.2m x 1.6m), the *start and goal states* are provided as inputs to your algorithm. The *path toward the goal state* is your output. Inside a launch file, you’ll edit the start pose and the goal pose of the object.

Since the map is unknown and the object is not equipped with any sensor, you might be wondering…how do we detect obstacles and move the cuboid object with the robotic arm?

We’ve included two services that you can call in your python script; Move and CheckPath:

* Move: Command the robot by sending a 2D pose. The robot will move the object on a linear path to this pose. Your moving point should be at least **5mm** away from the starting one. And if you are to rotate the arm, then you need to make sure you are rotating by at least **0.35** radians.
* CheckPath: Verify if a linear path between two 2D poses is valid.

**Note**: Even if you gain familiarity with the maze through repetition or by examining videos and logs, your algorithm must **not** be permitted to make use of this knowledge by hard-coding the map or learning from previous episodes. Each time the robot begins its maze problem, it should be completely ignorant of where the walls are or what the map of the maze looks like!

To summarize, you will have two main files to edit in this project: a launch file where you can specify the start/goal poses, and a python script to code your path planning algorithm.

There are 4 launch files in this project which you’ll launch later to start the Gazebo and RVIZ simulation, and all the other nodes:

1. **moveit\_planning\_execution.launch**(rll\_planning\_project package): Starts the setup simulation in Gazebo and Rviz.
2. **planning\_iface.launch**(rll\_planning\_project package): Starts the planning interface.
3. **path\_planner.launch**(rll\_planning\_project package): Starts your path planning algorithm code.
4. **run\_project.launch**(rll\_project\_runner package): Starts a single planning and path execution.

You'll have a chance to launch each of these files separately in different terminals or use a provided shell script file that will launch all the nodes in separate instances of xterm terminals. The first method is preferable for debugging since you can easily identify the errors.

### Getting Started

To get started, navigate to the project workspace concept. Under home/workspace/catkin-ws/src, you will see the project SDK(**rll\_sdk**) and planning(**rll\_planning\_project**) packages. You will be mainly working with the planning package where you’ll edit a launch file to change the object start/goal pose, and you'll edit a python script to code the path planning algorithm.

**Python script: path\_planner.py**

Let’s first take a look at the Python path planning script. Navigate to /home/workspace/catkin\_ws/src/rll\_planning\_project/scripts and open the path\_planner.py script. Here, you will find a sample solution where we first retrieve the input values, print them, call the CheckPath service and check if it’s possible to move directly from the start pose toward the goal pose. If it’s valid, the Move service is called to move the object linearly toward the goal pose.

**Launch file: planning\_iface.launch**

Moving on, let’s take a look at the launch file you'll need to modify. Navigate to /home/workspace/catkin\_ws/src/rll\_planning\_project/launch and open planning\_iface.launch. There’s a long list of parameters needed for the project. You’ll only need to edit the start 2D pose and goal 2D pose parameters.

A screenshot of a computer program

AI-generated content may be incorrect.

**Stopping a Run**

Hit Enter key in the main terminal to instantly kill the project execution and all the ROS nodes. You need to wait at least 30 seconds before starting a new run because the nodes will take some time to fully terminate. If you encountered an error, don't worry about it - just relaunch the nodes either manually or with the shell script file provided.

Video 2.2

**Video Review**

Look closely at the video and try to identify the 4 cycles mentioned earlier:

1. First, the KUKA gripper will move to the object start position, orient itself with respect to the object, grab the object, and lift it up.
2. At this stage, your path planning code will be executed. Your code will search for a path and navigate the robot through the maze by commanding 2D positions, and an orientation angle, in order to get around corners in the maze
3. You will have a total time of 8 minutes to search for a path and move toward the goal pose. Once reached, your cuboid object will be placed in the goal pose.
4. Finally, the robot will lift the object from the goal pose, return it back to the start pose, drop it, and get ready for another run.

**Gazebo and Rviz**

A screenshot of a computer

AI-generated content may be incorrect.

**Gazebo**: Visual maze(red color) which is a replica of the real maze located at the KIT Lab in Germany.

**Rviz**: Collision maze(yellow color) which is the same as the visual maze but with thicker walls and some extra ones. Inside the collision maze, you can still see the visual maze that has thinner walls. The red block indicates the goal pose of the object.

As a safety measure, we added a tolerance value to the visual maze to prevent any collision between the object or robot and the maze walls in the real world.

### Scoring Criteria

**Scoring**

After launching all the nodes in this project, the robot will go through the different cycles and then generate a score. Your score is simply the total time it took the robot to plan and execute a path from start to goal. The clock starts right after the object is lifted from the start pose and stops whenever your robot reaches the goal pose. On the hardware system, your code will be run three times, with the median time selected as your reported score.

Things to always keep in mind while planning and executing a path:

* Do not exceed 8 minutes for planning plus execution - this will result in a failed run.
* Do not send any invalid pose - this will result in a failure and your code will be immediately aborted!
* Always call the Move service before moving to a new pose.

### Path Planning

Earlier, you launched the sample solution which executed a path from start to goal. Now, let’s make this problem more challenging by changing the start and end pose values of the object.

Navigate to the /home/workspace/catkin\_ws/src/rll\_planning\_project/launch directory, open the planning\_iface.launch file, and change the start and 2D pose of the object. Try your code for different configurations of start and goal poses.

The Contest Maze is over and now we are back to the Practice Phase Maze 1.

Practice Phase Maze 1 Start/Goal configuration:

*A screenshot of a computer code

AI-generated content may be incorrect.*

Now, code a path planning algorithm to search for a path and move the object from its new start pose to its new goal pose. At each launch, note your score down and try to improve your planner to be as fast as possible.

**Pre-Submitting**

Once you finish coding the path planning algorithm and succesfully testing it in simulation, you'll want to jump in and submit your code to the hardware. Before you do that, you need to test your code more than once! Actually, **three times** since your code will run **three times** on the hardware, with the median time selected as your reported time. To test your code three times in simulation, change the run\_three\_times argument value in the planning\_iface.launch file to true:

*A white background with black text

AI-generated content may be incorrect.*Then, run your code and check if your able to succesfully move the object from start to goal three times continuously. The reason why we're asking you to do that is to make sure that you accounted for multiple runs by resetting your global variables. After that, feel free to move on and submit your code.

### Project Workspace

### Submission Instructions

**Submitting your project during the PRACTICE PHASE**

Once you’ve gotten your project working in the simulator, you can submit it to run on the real KUKA arm! Once submitted, your “job” will be put in a queue at the KIT Robotics Learning Lab (RLL). Your job will first be run in a simulator, to make sure it can run successfully on the hardware.

There are only a few steps for your submission:

1. Manually fetch your user code and install it in your workspace
2. Enter submit on the command line in a terminal window
3. To check on your jobs, enter check\_jobs on the command line in a terminal window.

**Install your user code - the jwt file**

The jwt file is your user code and must be available at /home/workspace/JWT/jwt in order to submit a project. You can download your user code by [**clicking here(opens in a new tab)**](https://project-assistant.udacity.com/auth_tokens/new).  
You may need to **navigate to the link a second time** after logging in for the file to automatically download. Once you've downloaded your jwt, copy the contents into: /home/workspace/JWT/jwt.

**Submit your job**

Your python script must reside at /home/workspace/catkin\_ws/rll\_planning\_project/scripts/path\_planner.py. This is the only file that will be uploaded. If the submission succeeded, you’ll see the “Submission Complete” feedback. Note that this does not indicate any status of the code itself, only that the job was successfully submitted to the queue. You may only have one job at a time in the queue.

During the submission process, you must positively “opt in” if you wish to see your time and rank on the leaderboard.

**Note:** In case the submission process fails, refresh the workspace and try again.

**Check on your job(s)**

Once your job is submitted, it will be given a job number and placed in a queue at the KIT RLL. To check on your job, enter check\_jobs on the command line. You’ll receive a status update on all the jobs you’ve entered. Here is an example of two failed jobs and another sitting in the queue partially done:

A screenshot of a computer code

AI-generated content may be incorrect.

Note that the last job has a “duration” time associated with it. However, the job has not run on the “real” yet, so this is just the simulation run time. If the job runs successfully on the “real” arm, you should see an entry something like this:

A screenshot of a computer code

AI-generated content may be incorrect.

At this point, the score (time) will be entered in the leaderboard database. If you have opted in, your time and rank will be visible on the leaderboard under whatever name you’ve chosen to post with.

**Submitting your project during the CONTEST PHASE**

The same command, submit, should be used to submit your project. The submission system will operate a little differently during this phase:

* During submission, you will be presented with the Terms and Conditions for the contest and must positively agree to them to enter the contest.
* You will not receive any log links or video feedback until the contest is over, though you will receive job status with the check\_jobs command.
* All jobs that pass the simulator will be added to a queue to run on the real KUKA arm. The real runs for each entry will only take place after the contest closes.

### Project Walkthrough

Video 2.7

### Hints

**KUKA Project Hints!!!**

1. There are two main differences between classic path planning algorithms that you’ve learned earlier and the path planning algorithm that you’ll have to write in the KUKA project
2. Classic path planning algorithms assume a known map where the position of obstacles and free spaces are known. However, in the KUKA project the maze is unknown! The only information you have is the map dimensions, start pose, and goal pose.
3. Classic path planning algorithms applied earlier assume a change in position only. The orientation angle of the robot is kept fixed. With the KUKA project, however, you need to change both the position of the gripper and its orientation to move anywhere inside the maze.
4. Applying what you learned earlier about classic path planning algorithms to the KUKA project
5. Since the maze is unknown, you have to implement "a search as you go" technique. Using the map dimensions you can represent the maze in a matrix form. Begin searching from the start pose, identifying if a cell is free or occupied, and as you're searching update the matrix until you reach the goal pose.
6. Define a set of angles that enables you to move anywhere inside the maze. To update the status (occupied or free) of a cell in the matrix, you can change the orientation angle of your gripper and check if you're able to move with each of the predefined angles towards the cell. The cell will be updated as free if any of the predefined angles returns a valid response. On the contrary, the cell will be updated as occupied if all the predefined angles return an invalid response.

**Visual Example**

A grid with different colored squares

AI-generated content may be incorrect.

This is an example of how you can represent Maze 1 practice phase in a matrix form. As a reminder, this is Maze 1 start and goal configuration:

A screenshot of a computer code

AI-generated content may be incorrect.

* Transforming the maze **(1.2mx1.6m)** into a matrix size **(12x16)**. I am using a very small resolution here for demonstration. You need to test which resolution works best for you.
* Transforming Maze 1 start and goal poses to matrix coordinates:
  + Start pose **(0.38, 0, 0)** to matrix coordinate **(9, 8, 0)**
  + Goal pose **(-0.37, 0.5, 0)** to matrix coordinate **(2, 13, 0)**

Now that you’ve transformed world coordinates into matrix ones, implement a **search as you go** technique where you start with the **S** cell and then update the status of adjacent cells.

Updating the status of adjacent **Cell 1**:

Defining a set of angles in radians to navigate the maze: **[0, 0.78, 1.57]**

* **(9, 8,0)** to **(9,9,0)** Valid or Invalid?
* **(9, 8,0)** to **(9,9,0.78)** Valid or Invalid
* **(9, 8,0)** to **(9,9,1.57)** Valid or Invalid?

**Cell 1** will be considered as occupied if all these questions return an invalid answer!

Now, move on to the second adjacent cell, check for movement with different angles, and update your matrix. Then, move to another undiscovered cell and so on until you build a map and reach the goal **G**.

* How to start searching and planning?

As a good start, solve an easy problem where both the start pose and goal pose are on the same line. Search your way toward the goal by keeping the angle of your gripper constant. Once you solve this problem, assign a set of angles to each direction of movement as you’ve seen in the earlier visual example.

A grid with squares and numbers

AI-generated content may be incorrect.

Discovering if cells **1 to 4** are occupied or free by keeping the gripper angle constant:

* Cell 1 - **(9, 8,0)** to **(9,9,0)** Valid or Invalid?
* Cell 2 - **(9, 8,0)** to **(8,8,0)** Valid or Invalid?
* Cell 3 - **(9, 8,0)** to \*\*(9, 7,0)\*\*Valid or Invalid?
* Cell 4 - **(9, 8,0)** to **(10,8,0)** Valid or Invalid?

### Maze #1 Leaderboard

### Maze #2 Leaderboard

### Contest Maze Leaderboard