IASD 2021/22 Project Assignment #2: Global 3D Point Cloud Alignment

Pedro Miraldo and Rodrigo Ventura
Instituto Superior Técnico, University of Lisbon

(Version 1.1, October 15, 2021)

Introductory Notes:

- Any kind of sharing code outside your group is considered plagiarism;
- Developing your code in any open software development tool is considered sharing code;
- You can use GitHub. Make sure you have private projects and remove them afterward;
- If you get caught in any plagiarism, either by copying the code/ideas or sharing them with others, you will not be graded; and
- The scripts and other supporting materials produced by the instructors cannot be made public!

Introduction

A good perception sensor is of most importance for any intelligent agent. More and more, autonomous agent rely of 3D sensors, such as 3D LiDAR or RGB-D cameras. While a conventional camera gives us color data, which is important for detection and classification tasks, 3D sensors have been widely used for 3D computer computer vision problems, such as agent's localization and mapping of the environment. A proof of the efficiency and robustness's of these kinds of sensors is that most of the in development autonomous driving cars rely significantly on 3D sensing for navigation and pedestrian detection and avoidance. While 2D cameras give us a dense representation of the environment, making it easy to get point to point correspondences, 3D sensors such as LiDAR gave us depth information. This simplifies significantly the localization problem making it the preferable sensor for autonomous vehicles for this task.

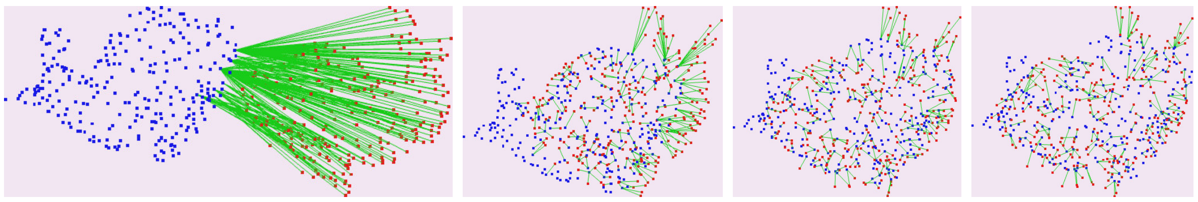

Figure 1: We show a sequence of 4 images of the alignment using the method in assignment #1. The last one corresponds to the final computed alignment, i.e., the one obtained from the main loop stopping criteria. Thus, although the method converged to some rotation and translation, we can see that they do not correspond to the correct alignment.

1 Problem Statement and Solution

The solution to the 3D alignment presented in the previous assignment has some limitations. One of them is the way it gets stuck in local minima. See the example in Fig. 1, where we ran a solution to the first assignment. Most of the times, this happens because we are trying to estimate larger rotations. The method derived in assignment #1 only works for very small transformations (i.e., $\mathbf{R} = \mathbf{I}$ and $\mathbf{t} = \mathbf{0}$). It is only considered a refinement technique. In fact, there is no analytical solution to obtain the true alignment of two-point clouds from general rotations. In this assignment, students will have to present a software agent that should consider the particularities of the problem and can use search strategies for solving the alignment for general three degrees of freedom rotations.

2 Objective

This assignment aims to solve the previous section's problem using **uninformed** search strategies. Namely, breath-first search, depth-first search, or uniform cost search. **The use of informed search is not allowed**. This includes defining:

- A state representation;
- The operators;
- The goal condition; and
- The search strategy to be used,

allowing an appropriate search method to find the (optimal) solution. The implementation should be done in Python version 3.x. No extra modules besides the Python Standard Library and NumPy are allowed. The search algorithm implementations are the ones from the GitHub repository of the course textbook, namely the module search.py available from `https://github.com/aimacode/aima-python`. The problem should be implemented as a Python class with the name `align_3d_search_problem`, which derives from the abstract class search `problem`, and that defines the following methods (not necessarily all):

`actions(s)` Returns a list (or a generator) of operators applicable to state `s`;

`result(s, a)` Returns the state resulting from applying action `a` to state `s`;

`goal_test(s)` Returns True if state s is a goal state, and False otherwise;

`path_cost(c, s1, a, s2)` Returns the path cost of state `s2`, reached from state `s1` by applying action `a`, knowing that the path cost of `s1` is `c`;

The choice of state and action representations and the searching algorithm is entirely up to the students choice, with the following restrictions:

- The use of implemented algorithms in the `search.py` module is mandatory;
- Only algorithms taught in the theoretical classes are allowed; and
- No changes in the `search.py` algorithms are allowed. (You will not be uploading this file.)

In addition to the class, `align_3d_search_problem`, the students will have to define a function named `compute_alignment(.)` which the called by the evaluator for computing the solution. This function should instantiate an object from type `align_3d_search_problem` and compute the solution. It returns a `Tuple`. (see the details in the code submission template.)

The use of the method derived in the previous assignment is optional. However, if the students want to use these functionalities, a module `get_compute()` (with no arguments) must be called. Notice the `registration` class in `registration.py` (will be available soon) was changed slightly. Now students do not have access to the function `compute(.)`. It is now private. You can call `get_compute()` which is a wrapper to `__compute(.)` without input arguments.

A submission template is sent at the end of the assignment.

3 Evaluation

The deliverable for this assignment is made through DEEC Moodle, with the submission of two python files, called `solution.py`, and `search_solution.py`. The former corresponds to the implementation of the functionalities of assignment #1 (changes are allowed, not infringing stipulations imposed in that assignment). The latter implements the modules mentioned above. Instructions for this platform are available on the course web page. Finally, the grade is computed in the following way:

- 50% from the public tests;
- 50% from the private tests; and
- -10% from the code structure and readability.

Deadline: **23h59, 22-October-2020**. Projects submitted after the deadline will not be considered for evaluation.

4 Run an example

We will provide the students with a script to run the public tests (available at Fenix, project section) for at-home tests and evaluations, with a visualization tool for validation. Students need to have installed `NumPy` and `vtk` libraries. The file `run_example_nr2.py` contains a simple routine to use the previously mentioned classes.

There will be 8 public tests available. To run the scripts, in a terminal call:

```
$ python run_example_nr2 key
```

where key is a string: `PUB1`, `PUB2`, ..., `PUB8` calling for the respective test.

Note:

- Supporting scripts will be made available soon.
- We will provide the public tests in the week of the submission deadline.

5 Submission template:

```python
from typing import Tuple
from numpy import array
import search

# you can use the class registration_iasd
# from your solution.py (previous assignment)
from solution import registration_iasd

# Choose what you think it is the best data structure
# for representing actions.
Action = None

# Choose what you think it is the best data structure
# for representing states.
State = None


class align_3d_search_problem(search.Problem):

    def __init__(
            self,
            scan1: array((...,3)),
            scan2: array((...,3)),
            ) -> None:
        """Function that instantiate your class.
        You CAN change the content of this __init__ if you want.

        :param scan1: input point cloud from scan 1
        :type scan1: np.array
        :param scan2: input point cloud from scan 2
        :type scan2: np.array
        """

        # Creates an initial state.
        # You may want to change this to something representing
        # your initial state.
        self.initial = None

        return


    def actions(
            self,
            state: State
            ) -> Tuple[Action, ...]:
        """Returns the actions that can be executed in the given state.

        :param state: Abstract representation of your state
        :type state: State
        :return: Tuple with all possible actions
        :rtype: Tuple
        """

        pass


    def result(
            self,
            state: State,
            action: Action
            ) -> State:
        """Returns the state that results from executing the given
        action in the given state. The action must be one of
        self.actions(state).

        :param state: Abstract representation of your state
        :type state: [type]
        :param action: An action
        :type action: [type]
        :return: A new state
        :rtype: State
        """

        pass


    def goal_test(
            self,
            state: State
            ) -> bool:
        """Returns True if the state is a goal. The default method compares the
        state to self.goal or checks for state in self.goal if it is a
        list, as specified in the constructor. Override this method if
        checking against a single self.goal is not enough.

        :param state: gets as input the state
        :type state: State
        :return: returns true or false, whether it represents a node state or not
        :rtype: bool
        """

        pass


    def path_cost(
            self,
            c,
            state1: State,
            action: Action,
            state2: State
            ) -> float:
        """Returns the cost of a solution path that arrives at state2 from
        state1 via action, assuming cost c to get up to state1. If the problem
        is such that the path doesn't matter, this function will only look at
        state2. If the path does matter, it will consider c and maybe state1
        and action. The default method costs 1 for every step in the path.

        :param c: cost to get to the state1
        :type c: [type]
        :param state1: parent node
        :type state1: State
        :param action: action that changes the state from state1 to state2
        :type action: Action
        :param state2: state2
        :type state2: State
        :return: [description]
        :rtype: float
        """

        pass


def compute_alignment(
        scan1: array((...,3)),
        scan2: array((...,3)),
        ) -> Tuple[bool, array, array, int]:
    """Function that returns the solution.
    You can use any UN-INFORMED SEARCH strategy we study in the
    theoretical classes.

    :param scan1: first scan of size (..., 3)
    :type scan1: array
    :param scan2: second scan of size (..., 3)
    :type scan2: array
    :return: outputs a tuple with: 1) true or false depending on
        whether the method is able to get a solution; 2) rotation parameters
        (numpy array with dimension (3,3)); 3) translation parameters
        (numpy array with dimension (3,)); and 4) the depth of the obtained
        solution in the proposes search tree.
    :rtype: Tuple[bool, array, array, int]
    """

    pass
```