RoVi Project

Assoc. Prof. Dirk Kraft, kraft@mmmi.sdu.dk
Assoc. Prof. Aljaz Kramberger, alk@mmmi.sdu.dk

Document Version: 1

Robotics and Computer Vision 2025
University of Southern Denmark (SDU)

1 Background

Pick and Place tasks are one of the most frequent kind of tasks executed by robots. In typical production setups the pick and place locations are well known in advance and do not change, meaning, the parts are placed in specialised holders, preventing them to change pose, although a disturbing force may occur. In the majority of the cases the part holders add additional complexity and increase the cost of the application. In recent years there is a strive to make production setups flexible to allow for quick changeover between products. Therefore there is a need for vision based localisation solutions in order to overcome the limitations of fixed holders and to make applications more flexible.

2 Overview

In this project you are asked to design such a pick and place solution (in a simulated environment). The task can be decomposed into three major component, the robotics parts (see Section 3), the vision part (see Section 4) and the combination of the two (see Section 5).

3 Robotics

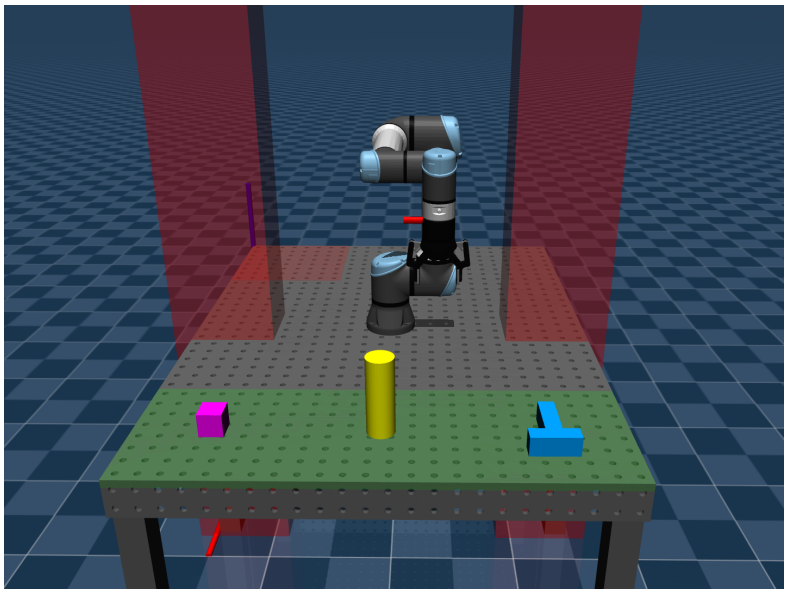

Figure 1 Pick (green) and place (red) locations.

3.1 Robot motion planning

In order to execute the pick and place action, you are asked to calculate the robot trajectories for this specific task. To do so, use / implement the following algorithms:

- Point to Point interpolator - with trapezoidal velocity profile, you should select at least 6 points to interpolate between in order to fulfill the pick and place task (use the provided workcell from exercise 6).
- Use the RRT algorithm to plan the trajectory. Use the provided workcell with obstacles (from exercise 6).

Implement the two approaches and test their performance with all 3 provided objects. The place location can be the same for all. For the pick operations, select your preferred configuration based on the object (from the top or from the side). Both algorithms should be used for all 3 objects in the scene.

3.2 Report structure

The content of the report should be **MAX** 5 pages long - excluding the front page and references if applicable. The report should have the following sections:

3.2.1 Introduction

1/5 page MAX
A short introduction to the problem of the assignment.

3.2.2 Methods

Short description of the two trajectory generation methods should be provided here, you should focus on:

- Differences and similarities when comparing the two approaches.
- Explain how the methods work based on the position, velocity, and acceleration profile of the generated trajectories
- Give a simple pseudo algorithm outlining the implementation of the two methods.

Size **MAX 1 PAGE**

3.2.3 Results and Discussion

- Evaluate the implemented algorithms and present the results in a graphical form - time series plots of the generated trajectories.
- Combine the plots of both methods on a single plot, which shows the trajectories by DOF.
- Calculate the direct kinematics (DK) for the two executions and plot them as a separate plot by DOF.
- Calculate the velocity and acceleration profiles for both cases and plot them on a single plot by DOF.
- Discuss the results, focus on comparing the performance of the two methods.

Size **MAX 2.5 PAGE**

3.2.4 Conclusion

Evaluate the performance of the two algorithms, propose what can be improved, Size **MAX 1/2 PAGE**

4 Vision

4.1 Overview

In the vision part of this project, you are asked to design a pose estimation solution that could be used for a pick-and-place solution (in a simulated environment).

You need to implement the method, potentially optimise the parameters of the method, and then evaluate the performance of the method.

4.1.1 Method

You have to implement a "simulated depth sensor" based pose estimation method. You are provided with "rendered" point clouds and an object model (of the object to pose estimate). Your method should then be able to find the object's pose.

The pipeline here would typically be (you decide on which steps to do or not to do and which concrete choices to make here):

- Load point cloud (We are doing that for you.)
- Filter/Segment point cloud
- Create object point cloud (We are doing that for you, you might want to adjust parameters.)
- Use global pose estimation to find the object pose
- Use local pose estimation to refine the object pose
- Combine results into a single final pose estimate

4.1.2 Setup

You will be provided with a set of scene pcd files. These are created by randomly placing the object in question (a rubber ducky) in different locations within the pickup area (see Fig. 1). Your method should be able to handle all possible poses (within the pickup area).

4.1.3 Evaluation

Once you have implemented and setup your methods in a reasonable way it is time to evaluate the performance of the method. The idea here is to evaluate the method's performance for different scenes and under different conditions.

The performance here can typically be measured in two ways:

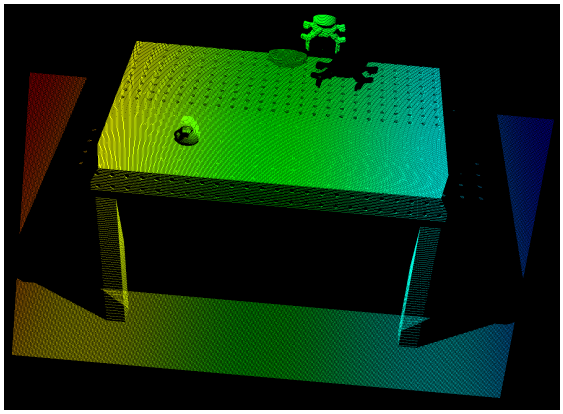

Figure 2 Sample pointcloud.

- Do you get a reasonable pose estimate at all or is the result not even close to the real object position? You want to measure this as a binary outcome and then summarise it as a percentage of cases the method works. (You could now make a manual decision for each pose estimate here if you feel the outcome if reasonable or not. Another more practical approach might be to define bounds on the pose error and to decide based on these.)

- The pose error when the pose estimate works (is close to the real pose). Here you want to filter out the cases where you above decided that the pose is not good and then you want to calculate the distance to the real pose. Typically you have to do this independently for position and orientation.

Many of the processes here are deterministic if the input is deterministic (which it is in our case because of the nature of the simulator/rendering). You therefore need to add some indeterminism to the process. Typically this is done in the form of noise. The noise here can be added as:

- Additive Gaussian noise added to all the computed points.

We provide you with some functionality for this.

4.2 Actual work

4.2.1 Provided items

- We provide 280 different "pre-rendered" scene pointclouds and their corresponding ground truth. They are located in the Scenes folder. They can look like in Fig. 2. The file Scenes.pdf on ItsLearning will tell you which are relevant for you.

- An object file ("duck.stl") located in the Scenes folder.

- A template for implementing your pose estimation code. In "VisionProject/Code/do_po.py".

- A script for testing your pose estimation code on a single scene in "VisionProject/Code/trial_run.py".

- A script for running pose estimation for all your cases in "VisionProject/Code/run_all.py".
- Scripts for visualising the results in "VisionProject/Code/visu1.py" and "VisionProject/Code/visu2.py"

4.2.2 Steps to take

We provide you with a lot of infrastructure for this project. Therefore it is important that you do things in the right way.

The steps you should take to tackle this exercise are:

- Update the "VisionProject/Code/settings.py" file with the indexes relevant for you (from the Scenes.pdf file on ItsLearning).
- Try to run the "VisionProject/Code/trial_run.py" script. If you get errors you might need to adjust the paths in the "VisionProject/Code/settings.py" file. The reported pose estimation errors will be really bad, but that is OK for now.
- Try to read and understand the "VisionProject/Code/trial_run.py" script.
- Implement the "def do_pose_estimation(scene_pointcloud, object_pointcloud):" function in "VisionProject/Code/do_po.py". Using the "VisionProject/Code/trial_run.py" script to do the testing. You might need to temporarily add debug information to your implementation to see what is going wrong. You will have to run your pose estimation on 55 scenes for the final evaluation, so spending some time on speed optimisation might pay off here.
- Once you are happy with the performance of your pose estimation function you can run the script "VisionProject/Code/run_all.py" to apply it to all relevant scenes. This will create a set of files in the "VisionProject/results" folder.
- Afterwards you can run the "VisionProject/Code/visu1.py" and "VisionProject/Code/visu2.py" scripts to turn these results into graphs placed in the "VisionProject/graphs" folder.
- Now you are ready to package your graphs and code (as described below) for hand in.

The "run_all.py" script runs 5 pose estimations for

1. the first scene mentioned in "indexes" in "settings.py" with all "noise_levels" (also from "settings.py"), and
2. all the scenes mentioned in "indexes" in "settings.py" without noise added.

The performance of the first one will be shown in the "2XNoiseXXX.png" graphs while the second one will be shown in the "1XScenesXXX.png" graphs.

We expect that your implemented method succeeds for each scene without noise in at least one of the five runs. You can have a look at the produced "10-ScenesSuccessrate.png" graph. It should be 20% or above for all five scenes. If you feel this is not possible please contact Dirk significantly before the deadline.

A performance of less than 20% for one of your scenes will impact your grade negatively. Performance of more than 20% will not impact your grade (so don't spend time optimizing if you already have a 20% performance for each of your scenes).

5 Integration

In the integration part your task is to create a fully integrated solution that:

I0 Moves the duck to a new random position

I1 Pose estimates the duck from the pick area

I2 Grasps the duck

I3 Places the duck in the place area

5.1 I0

We provide functionality to place the duck at a randomised position in the pick area (see exercises/cv_demo.py).

5.2 I1

For pose estimating the duck you need to get the point cloud (use the function get_pointcloud from cam.py to create the point cloud and save it in a file on disk.

MuJuCo is a simulation / visualization tool, therefore it uses a different notation for camera coordinates and rotation. We have corrected this in the above function for creating a point cloud. But that also means that the camera position that mujoco has the camera at does not align with that information. We therefore provide a function to give you the corresponding camera pose (get_camera_pose_cv in cam.py), you will need that information to transfer your camera centric pose estimate into mujoco world coordinates.

5.3 I2

For the robot grasping, use the Point to Point interpolator - with trapezoidal velocity profile, for witch some of the poses are defined by you and others e.g., grasping pose and approach should come from the pose estimation algorithm.

5.4 I3

Place the duck in the place area which is defined in the workspace of the robot (same as in 3.1), use the Point to Point interpolator for this part of the integration as well.

5.5 Process

Please implement a process that performs the above steps (automatically).

We then ask you to run the process 3 times (with different random positions) and record videos for the 3 executions.

6 Formalities

Deadline for the project is XXX, XX^{rd} , 9am. The hand-in is individual and will be handled through ItsLearning. Mails will not be accepted as hand-in.

6.1 Robotics

The handin for the robotics side of the project consists of a report and code. The formal requirements for this part are the following:

- The report must contain the parts mentioned in Sections 3.2.3 in addition to some general report parts.
- Report clearly explaining the algorithms used; the motivation behind the choices and the tests performed. The report is to be between 2000 and 6000 words.
- A zip file containing the source code developed in the project and a plain text-file explaining the content of the files, how to compile and how to run the tests.

6.2 Vision

For the vision side your hand-in consists of the 12 images created by the visualisation scripts and your code. These components need to be placed in a **zip file** that unpacks **exactly** (nothing more, nothing less) as shown in Fig. 3. (Except replace Dirk's name with yours.) (Top folder handin-DirkKraft, subfolders Code and graphs, ...). The zip file should be named **Vision.zip**. Dirk's version is attached on ItsLearning if you are in doubt about the structure. (When looking at Dirk's graphs be aware that it is optimised for speed not performance, maybe not a good idea.)

6.3 Integration part

You should have recorded 3 videos for the integration part. Please place these 3 videos together with your code (the code used for the integrated execution of this task) into a **zip file** called Integration.zip.

6.4 File to hand in

You should now have 3 zip files:

- Robotics.zip

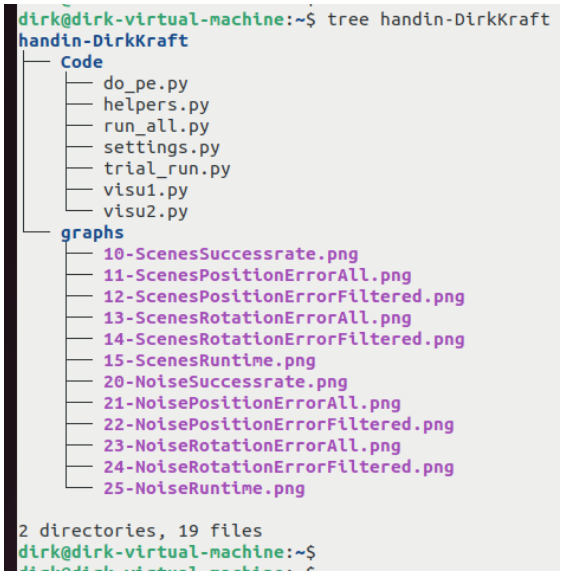

Figure 3 File structure.

- Vision.zip
- Integration.zip

Please place these 3 files (and only these 3 files) into another **zip file** called YourName.zip (in Dirk's case it would be DirkKraft.zip). This is the only file you should hand in.

6.5 Plagiarism

The project is to be solved individually. Individuals are NOT allowed to copy code (or results) from each other! It is completely OK to discuss with each other and inspire each other. In case copying is found both individuals will fail the course and be reported to the university officials! (This suggests that putting your code/results/... in a public place, e.g., github, is not a good idea.) If code snippets from the exercise solutions on ItsLearning (or other public locations) are used, make sure they are properly referenced in your code.