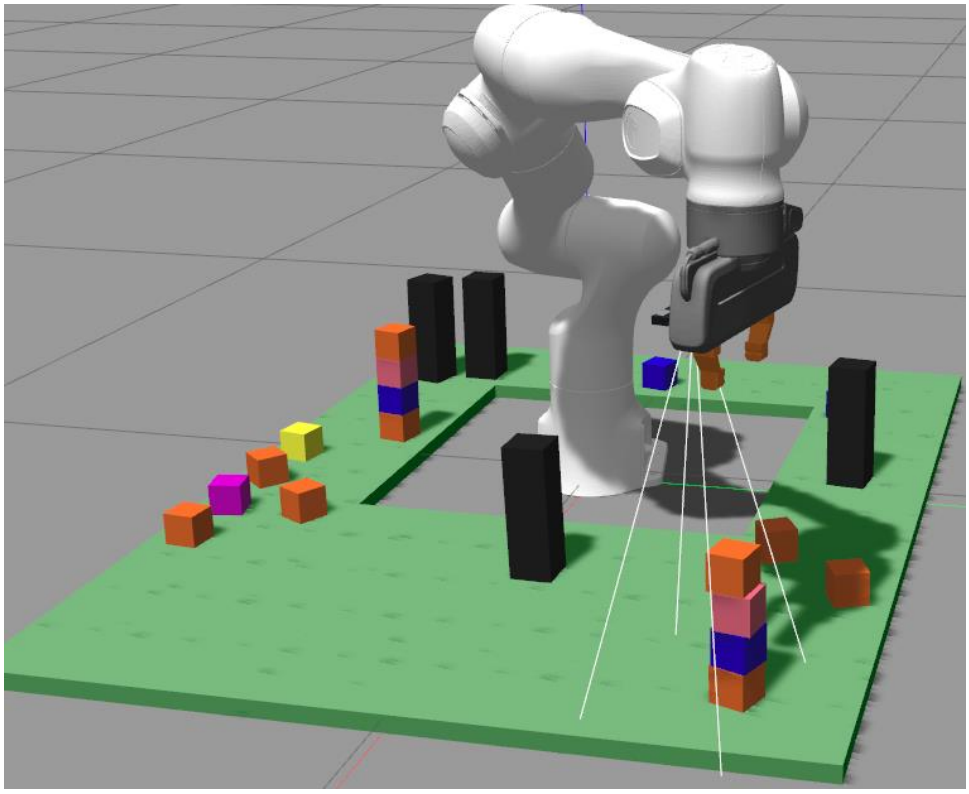
COMP0129: Robot Sensing, Manipulation, and Interaction

Coursework 3: Make a Pile of Objects!

Due Date: 30 Mar 2022 at 16:00 (UK time)

Worth: 45% of your final grade



General Goal

The goal of this coursework is to combine the knowledge you gained during CW1 and CW2, be able to generate a solution to a pick-and-place problem on your own, and design/present your method and experiments that validate fully your system.

Part 1 – Coding [Grade: 50%]

Coding Setup

For the coursework you will need the **comp0129_s22_robot** repository (as we did in CW1 and Labs). Here is a recap:

```
> git clone https://github.com/COMP0129-S22/comp0129_s22_robot --recurse-submodules
```

The coursework is set up in a package called **cw3_world_spawner**. This needs to be added to the **comp0129_s22_robot** folder on your machine. We will **add this as a submodule**:

```
> cd comp0129_s22_robot/src

> git submodule add https://github.com/COMP0129-S22/cw3_world_spawner.git
```

The next step is to **create the package which will contain your solution**. Each team will submit one folder called **cw3_team_X** where **X** is your team number. For example, team 4 would submit cw3_team_4, team 11 would submit cw3_team_11:

```
> catkin_create_pkg cw3_team_<team number> roscpp rospy cw3_world_spawner
```

Your solution package should be run using a launch file called **run_solution.launch**. We will make this file now in a folder called launch:

```
> cd cw3_team_<team number>

> mkdir launch

> touch launch/run_solution.launch
```

Now, open **run_solution.launch** in your favourite text editor and paste in the following:

```
<launch>
    <!-- launch with a delay to allow gazebo to load, feel free to edit -->
    <arg name="launch_delay" value="5.0"/>

    <!-- load panda model and gazebo parameters -->
    <include file="$(find panda_description)/launch/description.launch"/>

    <!-- start the coursework world spawner with a delay -->
    <include file="$(find cw3_world_spawner)/launch/world_spawner.launch">
        <arg name="launch_delay" value="$(arg launch_delay)"/>
    </include>

    <!-- add your own nodes to launch below here -->
 </launch>
```

Now, you can **run** the coursework with:

```
> cd ~/comp0129_s22_robot

> catkin build

> source devel/setup.bash

> roslaunch cw3_team_<team number> run_solution.launch
```

Coding Information

1. The only directory you will submit is **cw3_team_<team number>**. All your solution code must be inside this directory.

2. For your solution, all necessary nodes must be launched from **run_solution.launch**. During our evaluation, no other files from that directory will be launched. We will only run:

   ```
   > roslaunch cw3_team_<team number> run_solution.launch
   ```

3. You are allowed to add new dependencies to ROS **standard** libraries. These include sensor_msgs, geometry_msgs, moveit, cv2, pcl. If you are unsure post on the discussion forum or contact the TA team for more information. To simplify using MoveIt! and PCL, we recommend you use C++.

4. During development you are free (and probably you will need to, to check all different world variances) to edit the **cw3_world_spawner package.** We recommend you look at ***scripts/coursework3_world_spawner.py*** as well as the service definitions in the ***srv*** folder. Even though you will need to tweak those to test your system in various scenarios (reported

in your research report), you will not submit your version of the **cw3_world_spawner** package. So, **any changes you make to it will not be submitted**. We will mark your work with our own, clean version of **cw3_world_spawner**.

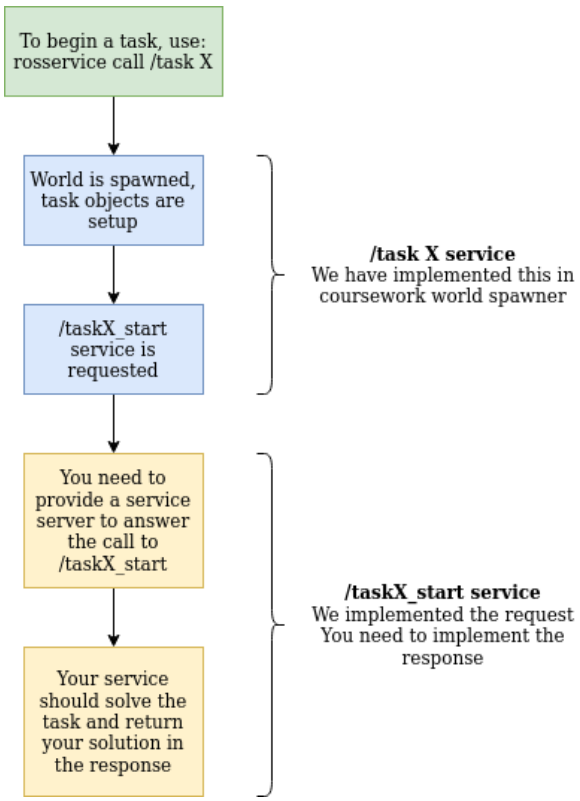
Coding Overview

The objective of this coursework is to perform pick-and-place tasks in Gazebo, using MoveIt! to move the robot and PCL to detect object positions and colours. The coursework splits into **three tasks**, detailed later in this document.

To start each task, you will call the service '**rosservice call /task X**', where **X** is the number of the task. When this runs, it will automatically set up the task environment in Gazebo and the service '**/taskX_start**' will be requested. Each task takes the form of a service request that you need to respond to. The request message contains the information needed to solve the task, for example the position of an object to pick up. Your job is to receive the task request, solve the task, and fill in the solution in the task response message.

Each task's service request and response arguments are defined in ***coursework3_world_spawner/srv***. Thus, you will need to implement methods that receive those requests and fill in the response arguments as specified in the task descriptions later in this document.

In summary, solving each of the three tasks will look like this:



Evaluation

Your submission will be evaluated on:

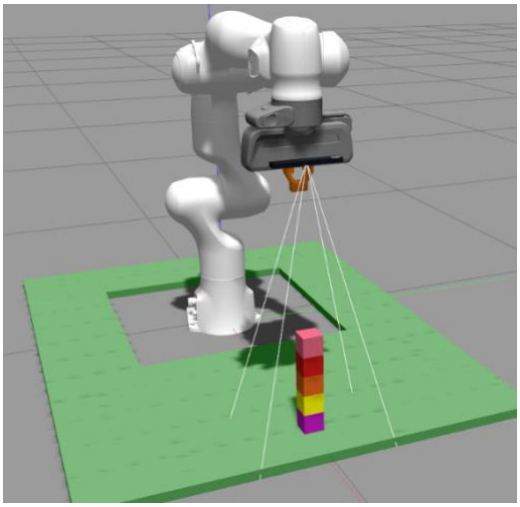
- the **correctness** of your outputs and behaviour of the robot - the robot should not collide into the ground, obstacles, or itself.
- the **performance** of your code – the solutions should not take too long, the design choices made should be reasonable and efficient.
- the **clarity and presentation** of your code - repetitive code should be structured into functions, comments should be used well, a README containing information about your code, authors, how it should be built and run, etc. If we fail to build, we will follow whatever instructions you have provided in the README and **will not attempt to fix anything major that is broken.**

For all these tasks, you are not allowed to:

- **tweak internal robot parameters** for speed and acceleration,
- **pre-define the location of the spawned models** through any means other than provided by the service calls or using the RGB-D input topics (*for testing your system you will need to tweak the location of the objects in the spawner in coursework_world_spawner/scripts/world_spawner.py but notice we will use our own spawner to evaluate your coursework – we suggest tweaking the parameters and spawn_trial_course() of each Task*).

Submission

1. Submit your folder **cw3_team_<team number>**, as a zip file.
2. **README** file: make sure you include licenses, authors, how to build and run the package, and importantly include for each task below how much time (*rough number of hours*) and the percentage (*rough amount of contribution to the solution*) each student of the team worked on the tasks. We expect roughly equal amount of contribution by each student. *If we notice that there is not an equal amount of contribution by some student(s), then the individual grade of the student(s) that contributed less will be decreased accordingly.*



Task 1: Detect and Localize the Stack of Objects [Grade: 20%]

***Grade:* 10%-Correctness; 5%-Style; 5%-Structure/Efficiency**

The environment for this task is represented by a grid of green tiles on the ground. We consider also the '***graspable objects***' of our world to be cubes of size [width, length, height] = **[0.04m, 0.04m, 0.04m],** spawned in **any rotation** in the world frame**.** Their colour varies and is defined by the combination of their **red (r)**, **green (g)**, or **blue (b)** values. Their colour can be anything, except **black**, i.e., (r, g, b) >= (0.1, 0.1 0.1).

A **stack of several graspable objects** is placed **somewhere around** the robot, but not necessary within a reachable distance. The stack might have **from 3 to 5 objects** and **with any stack rotation,** however all objects will always have the **same** orientation as each other.

The **task** is to:

- **localize** the position and orientation of the stack of objects in the world frame
- **detect** the number of objects in the stack
- provide a **list of the colour sequence** of each object in the pile starting from the bottom.

Notice that we do not care about keeping the original pile of objects untouched during the task. So, feel free to remove objects from it, if needed (just explain your approach in the report).

To **begin the task**, just call:

```
> rosservice call /task 1
```

This will automatically run the service **/task1_start**, passing **no inputs**. Using the camera topics starting with '**/r200/**', you need to detect and localise the stack of objects.

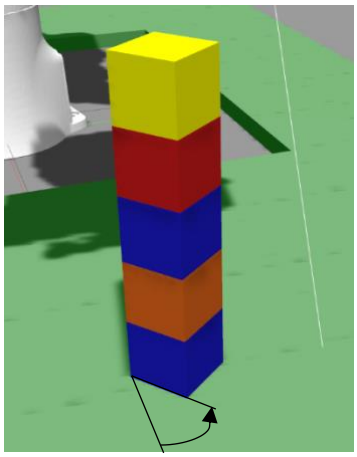
When you are finished, you should return the service response with:

std_msgs/ColorRGBA[] ***stack_colours***

geometry_msgs/Point ***stack_point***

float32 ***stack_rotation***

where the stack_colours is a vector of RGBA messages which defines what the colour order of the stack is. We will **ignore** A (alpha), you need to set RGB. The first element in the vector is colour of the **bottom** cube in the stack, the next should be for the object above it and so on. The number of elements in the stack_colours vector indicates how many objects you found in the stack. The stack point gives the **x and y position** of the stack, and the stack rotation gives the angle in **radians** that every cube in the stack is rotated about the **vertical** axis.



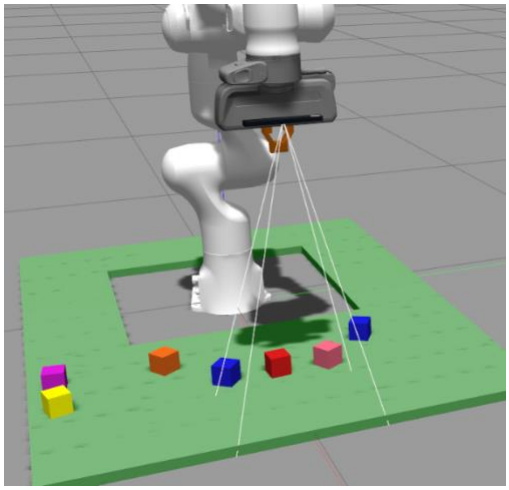
stack_colours – from bottom to top eg (blue, orange, blue, red, yellow)

stack_point – (x,y) position of the centre of the bottom of the stack

stack_rotation – angle of stack in radians from x-axis

Trial: Trial scenarios can be edited in file cw3_world_spawner/scripts/world_spawner.py

Validation: The unseen testing environment may contain more objects (graspable and non-graspable), that might occlude the pile of objects. During testing we will evaluate on multiple scenarios. Think about the edge cases – different initial positions of the robot, locations of the objects or obstacles, object colours, etc.! A failure is deemed if the service is not properly received or responded to, a collision occurs, or a timeout is reached before completing the task. The timeout will be set to a duration which is more than sufficient to complete the task.



Task 2: Create a Stack of Objects [Grade: 20%]

***Grade:* 10%-Correctness; 5%-Style; 5%-Structure/Efficiency**

Given the setup in Task 1, this Task 2 is about replicating the stack of objects by picking graspable objects around the robot and placing them in a pile at a given ground position.

To begin the task, just call:

```
> rosservice call /task 2
```

This will automatically run the service **/task2_start**, passing as input:

std_msgs/ColorRGBA[] ***stack_colours***

geometry_msgs/Point ***stack_point***

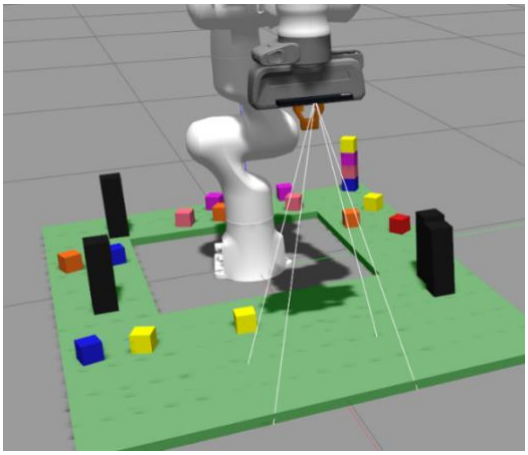
float32 ***stack_rotation***

where the stack_colours is a vector of RGBA messages which defines what the colour order of the stack is. The first element in the vector is colour of the **bottom** cube in the stack, the next should be above it and so on. The stack point gives the **x and y position** of the stack, and the stack rotation gives the angle in **radians** that **every** cube in the stack should be rotated about the **vertical** axis.

The objects can be spawned with **any** rotation about the **vertical** axis. Once the task is completed, return the **empty** *ServiceResponse*.

Trial: Trial scenarios can be edited in file cw3_world_spawner/scripts/world_spawner.py

Validation: The unseen testing environment may contain more objects (graspable and non-graspable), that might occlude each other. Think about the edge cases – different initial positions of the robot, locations of the objects/obstacles, object colours, etc.! A failure is deemed if the service is not properly received or responded to, a collision occurs, or a timeout is reached before completing the task. The timeout will be set to a duration which is more than sufficient to complete the task.



Task 3: Replicate the Stack of Objects with obstacles [Grade: 10%]

***Grade:** 5%-Correctness; 2.5%-Style; 2.5%-Structure/Efficiency*

Task 3 is a combination of the first two tasks. A stack of objects is spawned which you must find. Then, using cubes on the ground, build an **identical stack**. There are also '***non-graspable objects***' (we will call them obstacles) of size **[0.05m, 0.05m, 0.16m]**, of **black** colour (r, g, b) = (0.1, 0.1, 0.1), spread around.

To begin the task, just call:

```
> rosservice call /task 3
```

This will automatically run the service **/task3_start**, passing an input:

geometry_msgs/Point ***stack_point***

where the stack point gives the **x and y position** of the stack **you** will create.

Your stack must be a copy of the target stack. There will only ever be **one** target stack for you to find. You must copy the **number** of objects, their **colour** order, and their **rotation** about the vertical axis. As with the other tasks, all cubes in the stack will always have the same rotation (aligned with each other). The objects can be spawned with **any** rotation about the **vertical** axis. The obstacles can be spawned in any position but will always have sufficient space around them to allow grasping. Once the task is completed, return the **empty** *ServiceResponse*.

Trial: Trial scenarios can be edited in file cw3_world_spawner/scripts/world_spawner.py

Validation: The unseen testing environment may contain more objects (graspable and non-graspable), that might occlude each other. Knocking over the obstacles will result in penalties. Think about the edge cases – different initial positions of the robot, locations of the objects/obstacles, object colours, etc.! A failure is deemed if the service is not properly received or responded to or a timeout is reached before completing the task. The timeout will be set to a duration which is more than sufficient to complete the task. **Please note**: this task is challenging for MoveIt and validations will be repeated multiple times on non-random scenarios that we know are possible. Remember Gazebo is a physics engine so you can reduce failures with careful pathing, waypoints, pauses to allow joints to settle, collision-avoidance and error-handling.

Part 2 – Research Report [Grade: 50%]

General Goal

In this part you are expected to write a research report of the problem that you solved above (i.e., Task 3 via Task 1 and Task 2). Notice, that even if you are not able to solve all the questions above, you can still write all the parts of the report. The overarching aim of this assignment is to think creatively when solving Part 1 and report your approach and results.

Description

We expect you to write a research report consisting of: title, authors, abstract, introduction, related work, problem statement, proposed technical solution and hypotheses, experimental validation, discussion, and conclusions. A **max 4 double-column pages** is allowed, excluding **references**, which need to be **at least 10**. You will need to use the **IEEE template** (link: https://www.overleaf.com/read/ptsqczcdmjnd) for your report, written in **Latex/Bibtex**. We suggest you use **Overleaf, as in CW2.**

Guidance

1) **Title**:
   a) Be concise, avoiding punctuation, numbers, acronyms, and abbreviations.
   b) Describing the main idea of the paper.
   c) Do not use necessary the title of the coursework; think that this is a research report of a problem you solved.
2) **Authors:**
   a) Include all the authors.
   b) Mention the institution, and emails of the authors.
   c) Include further acknowledgements.
   d) Mention in the list if all authors contributed equally.
3) **Abstract:**
   a) Describe in short, the whole paper outline.
   b) Abstract is like a short introduction (see below).
4) **Introduction:**
   a) What is the motivation for investigating this area, e.g., why is it important, why is it needed?
   b) Define the problem in short.
   c) Did the research community tried to solve it? If yes, how? If no, why?
   d) If the problem is open (even partially), why other people have not solved it?
   e) If the problem is solved, what are the potential issues?
   f) If solving this problem who will benefit?
   g) How do you propose to solve the problem?
   h) How do you propose to evaluate the solution?
   i) Outline the main methodology and validation.

5) **Related Work:**
   a) Describe current approaches, methods and results related to the identified problem, e.g., who are the leading groups and what are they doing?
   b) Provide an assessment of these existing solutions: identify the technical challenges that need to be overcome to achieve progress.
   c) How does your solution compare to the related work?
6) **Problem Statement and Hypotheses:**
   a) Define the problem intuitively and mathematically if possible.
   b) State your hypotheses if any.
   c) Include any hardware and software setup (e.g., simulated) that is used.
7) **Proposed Technical Solution:**
   a) In a concise way, present your solution (for all the problem cases).
   b) Why you believe this solves the problem and if novel, how does it improve the state-of-the-art.
8) **Hypothesis/Evaluation:**
   a) Explain experiments and validations.
   b) Explain the outcomes (completeness).
   c) Provide any measurable or key performance indicators.
9) **Discussion/Conclusions:**
   a) Sum-up the paper, discuss the findings, by also adding some extra potential future directions that the approach will leave open.

Grading Notes

We expect you to follow all the instructions we gave during the course regarding presentation and writing (theory, lab, and research lectures). The grades will reflect your understanding into what we have discussed during those lectures. For examples:

- The grade will reflect the quality of writing (English, understandable context, lack of long sentences, size, etc.).
- The grade will reflect the organization of your thoughts and structure of the paper.
- The grade will reflect the Latex coding (refs, citations, numbers, labels, section, sub-sections, etc.).
- The grade will reflect the correctness and originality of your solution.
- The grade will reflect all aspects described in the assignment, efficient related work, good problem description, evaluation, quality of proposed solution, etc.

In general, the quality of submission should be close to the papers we have read during the Research Week. The following grading scheme will be followed (with minor tweaks). All the following will be equally graded (quality and context) unless specified in brackets:

1) Correct use of LaTeX, and coursework instructions (pages, refs, etc.),

2) Generally correct language (English, type, etc.),

3) Correct structure and template [*half of the others*],

4) Title (correct, concise, descriptive) [*half of the others*],

5) Abstract (size, context),

6) Introduction (size, context)

7) Related Work (size, refs, context vs your proposal)

8) Proposal (size, context, quality of solution)

9) Hypothesis/Experiments (size, context, coverage, quality, novelty)

10) Conclusion (size, context)

11) Originality (problem, solution)