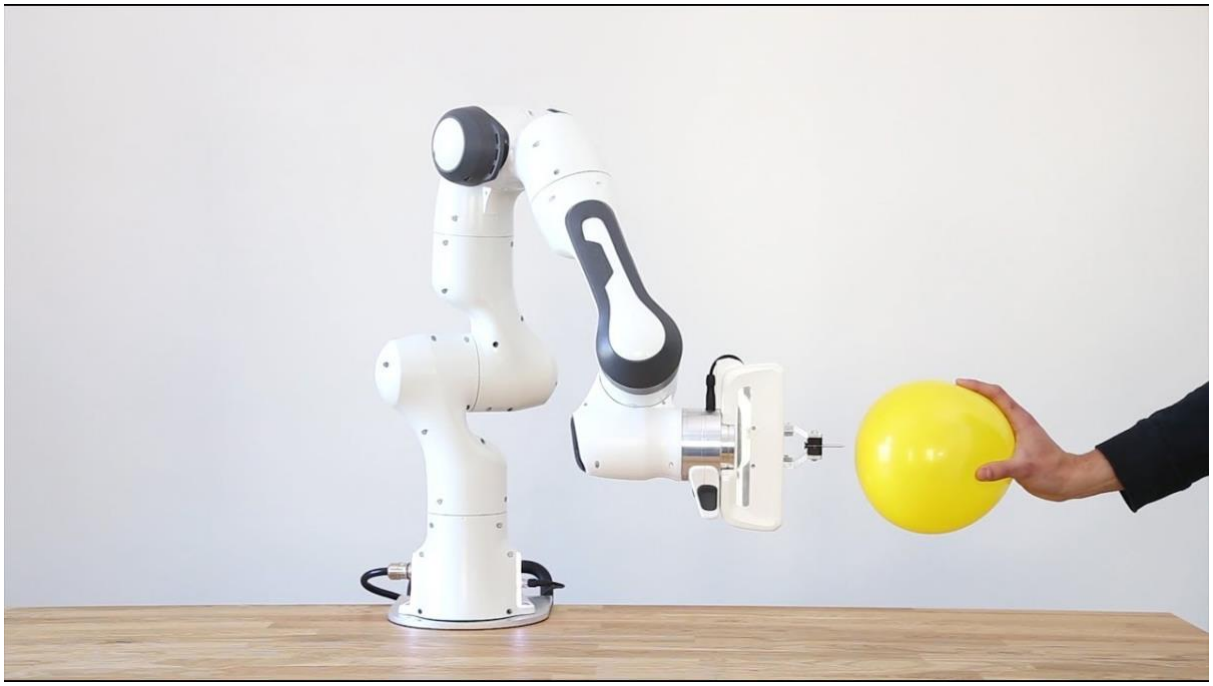
COMP0129: Robotic Sensing, Manipulation and Interaction

CourseWork Assignments

Author: Dimitrios Kanoulas



CourseWork 3 [45%]

Due Date: Wednesday, March. 31, 2021 (UK, 15:59)
Topics: Autonomous Pick and Place Grasping Task, using Visual (RGB-D) and Touch (Bumper) Sensing
In this coursework, you will implement an autonomous pick-and-place task, using originally RGB-D sensing to detect an object and bumper sensors to place it correctly to various tables, by moving around objects that may already be there.

Code:
> sudo apt-get install ros-melodic-pcl-conversions

(We should replace the moveit_tutorials package of Lab 3)
> cd ws_comp0129/src
> rm -rf moveit_tutorials
> git clone -b melodic-devel https://github.com/RPL-CS-UCL/moveit_tutorials.git

> git clone https://github.com/COMP0129-S21/comp0129_s21_cw3.git

Code:
1. *Feel free to change the methods' input/output argument.*
2. *Add helper functions/methods when needed; don't repeat code.*
3. *You will need to fix all comments, even in the defined methods in the header file, to fit to 80 characters length, adding the input/output/returned value comments, etc.*
4. *Do not add new dependencies to the package.xml or CMakeLists*

Compile:
> cd ../..
> catkin build

Run:
> source devel/setup.bash
> roslaunch cw3 cw3.launch

Submission Guidelines:

Folder name: Create a folder named **comp0129-s21-cw3-teamX** (X is your team number).

Code:
In **comp0129-s21-cw3-teamX**, place your:
- **comp0129_s21_cw3** folder
- **moveit_tutorials** folder
- **panda_moveit_config** folder
without any devel, build, logs folders. Just the source code.

R-Report (PDF and Latex folder): In **comp0129-s21-cw3-teamX**, make a **research-report** folder and place the **PDF** file of the resultant latex (overleaf) and the **.tex, images, etc** folder/files**.** Don't forget to complete the **readme/license** files.

C-Report (PDF and Latex folder): In **comp0129-s21-cw3-teamX**, make a **code-report** folder and place the **PDF** file of the resultant latex (overleaf) and the **.tex, images, etc** folder/files**.** Don't forget to complete the **readme/license** files.

Failure in following the submission guidelines will result in points deduction (additionally to the reductions of points that are described below).

Grading: We will grade the coursework both for quality and correctness (as CW1). This means that apart from a solution producing the right results (correctness), we will heavily grade code and report quality (e.g., helper function, repeated code, hardcoded values, parameter space, ros services, ros debugging, etc.), as taught during the labs.

Load the robot (having sourced and used the new **ws_comp0129** package):
> roslaunch cw3 cw3.launch
(ignore the world to robot_frame warnings)

It has been already implemented (function *addCollisionObjects*) the generation of four collision objects in MoveIt!:

1. A box-table (**table1**) whose center is in position: (x,y,z) = (0.0, 0.5, 0.2) with respect to panda_link0 frame, and with dimensions along (x,y,z): (0.4, 0.2, 0.4).
2. A box-table (**table2**) whose center is in position: (x,y,z) = (-0.5, 0.0, 0.2) with respect to panda_link0 frame, and with dimensions along (x,y,z): (0.2, 0.4, 0.4).
3. A box-table (**table3**) whose center is in position: (x,y,z) = (0.0, -0.5, 0.2) with respect to panda_link0 frame, and with dimensions along (x,y,z): (0.4, 0.2, 0.4).
4. A box-object (**object**) whose original center is in position: (x,y,z) = (-0.5, 0.0, 0.5) with respect to panda_link0 frame, and with dimensions along (x,y,z): (0.02, 0.02, 0.2).

Note: In this coursework, key-press detection is not implemented. Key presses can be added by editing the file ***test_cw3.cpp***.

Question 1 [20%] (Visual-based Object Detection):
We know that the robot base is standing on a planar ground floor and towards the panda_link0 x-axis there is one and only one object (we will call it **cylinder-object**) in front of a perpendicular wall. There are two code files that will run in this coursework: ***cw3.cpp/cw3.h*** and ***cylinder_segment.cpp.***

Starting with the ***cylinder_segment.cpp***:

1. How could you make the cylinder segmentation faster, by applying the correct filtering(s)? Implement it in a function: ***filterEnvironment*** inside the ***cylinder_segment.cpp***. (The filter should be activated when '**f**' is pressed and deactivated when it is pressed again) Call it in the right place inside the ***cloudCB*** function. You can check the framerate with the **FPS_CALC** function (already implemented). In your report, explain the solutions and mention the trade-offs and how much is the time complexity improvement?

2. Assume that we know the rough area that the object lies on, which is (x: -0.05, y: 0.16, z: 0.92) with respect to the *camera_rgb_optical_frame* frame, with an *0.25m* estimation error of the area, in every direction. Can you make the segmentation code faster given these details? How? How much faster is it? Implement it in a function: ***fastFilterEnvironment*** inside the ***cylinder_segment.cpp***. (The filter should be

activated when ‘p’ is pressed and deactivated when it is pressed again) Call it in the right place inside the ***cloudCB*** function.

3. Create a **cylinder-object** stamped pose message (**geometry_msgs/PoseStamped**) named: **/cylinder_pose** to be published when the cylinder object on the ground is found, with the location of its center and the right orientation. The pose needs to be updated when the object is moved.

** If you cannot solve this question at all, assume that the cylinder object is at position: (x: 0.384, y: 0.035, z: 0.083) with respect to panda_link0 frame. Points will be deducted though.*

Code Report *(inside cw3_code_report.pdf)**:** write a simple report text to explain any of the aforementioned questions. Always mention which functions you wrote to implement the solution (including potentially the lines of code). The grading of the code report will reflect the grade of the corresponding question as described above.*

Question 2 [10%] (Pick-and-Place without Constraint):

Inside the ***cw3.cpp/cw3.h*** code:

1. Place a bumper sensor (publishing std_msgs::Bool messages named /**bumper1,** /**bumper2,** /**bumper3**) in the center of every table in the scene (table1, table2, table3) that will be activated only when an object is located in each table’s center. Notice that table2 has from the beginning an object located at its center.
After the object from the ground is grasped, these three sensors are the only information we have, along with the position of the **cylinder-object**, that you can keep track of (using the ROS topic). We will not have any motion capture or visual sensing available after the first **cylinder-object** grasp.

2. By pressing ‘1’, the robot should receive/listen the pose of the cylinder object (**/cylinder_pose**), grasp it, and place it at the center of table1. The corresponding bumper should be updated.

** Notice that the cylinder is published in a different frame than the robot’s base or the world frame. You will need to transform the cylinder-object pose from one frame to another to get the right representation.*

Code Report *(inside cw3_code_report.pdf)**:** write a simple report text to explain any of the aforementioned questions. Always mention which functions you wrote to implement the solution (including potentially the lines of code). The grading of the code report will reflect the grade of the corresponding question as described above.*

Question 3 [30%] (Pick-and-Place under Constraints):

If the **cylinder-object** is still on the ground. By pressing any of '1', '2', or '3' the robot should read the /**cylinder**_**pose**, and try to place it to the corresponding table: table1, table2, table3. If the selected table has already an object on it (detected from the bumper), you should first move that object to a free table, and then place the cylinder object at its center, updating at the same time the bumpers. Notice that you should keep track of the **cylinder-object**, while for any other object on any table the bumpers are giving enough information to locate them (their size is given in the beginning of the coursework and they stand upside on the table).

After the **cylinder-object** is off the ground on any table, the same functionality as above should hold: by pressing '1', '2', or '3' the robot should move the cylinder object to the corresponding table, by first removing from any other table any object that may be there to a free table.

** If you cannot use the bumper sensing (-10%), then assume that you always know where your objects are, using a global variable.*

Code Report *(inside cw3_code_report.pdf)**:** write a simple report text to explain any of the aforementioned questions. Always mention which functions you wrote to implement the solution (including potentially the lines of code). The grading of the code report will reflect the grade of the corresponding question as described above.*

Research Report (*cw3_research_report.pdf:* **latex-based in PDF format, as your research report) [30%]:** imagine that this coursework was a research project, where you developed a method to complete the task as described above. Write a max **3-pages** double-column; excluding references), including **1)** Title, Authors (2%), **2)** Introduction/Problem Setup/Related Work (7%), **3)** Methodology (5%), **4)** Experiments (8%), and **5)** Conclusion/Future Work (3%). Latex use and language will be also assessed (5%). Do not include any code, but please add contexts such as you method/algorithm and images of your experiments, etc.

Verbal Presentation on April 1st [10%]: all members of the team will need to present the functionality of the code, by running it on their laptops through Zoom and answer questions about the code implementation.