University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	**Robotics Foundations**			
Coursework Number	**1**			
Deadline	**Time:**	**4:30 pm**	**Date:**	**16 March 2023**
% Contribution to final course mark	**20**		**This exercise should take at most these many hours:**	**20 (individual)**
Solo or Group ✓	**Solo**	✓	**Group**	✓
Submission Instructions	**Via Moodle**			
Who Will Mark This? ✓	**Lecturer ✓**	**Tutor ✓**	**Other**	
Feedback Type? ✓	**Written ✓**	**Oral**	**Both**	
Individual or Generic? ✓	**Generic**	**Individual ✓**	**Both**	
Other Feedback Notes				
Please Note: This Coursework cannot be Re-Done				

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below. The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

(i) in respect of work submitted not more than five working days after the deadline
   a. the work will be assessed in the usual way;
   b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.

(ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

The penalty for non-adherence to submission Instructions is 2 bands

Marking Criteria
See Page 4

Contents

Introduction

The coursework is based on what you have been working on in the labs. Your project should be developed using Python, ROS and associated technologies, including Gazebo and RViz. Your solution should draw on the skills you have built up so far. The aim is to develop a solution to enable Baxter to play chess, see Figure 1. The tasks you must solve for this coursework are specified as follows:

- Baxter must be able to pick and place chess pieces.

- Your ROS solution must locate spawned chess pieces and autonomously set up the chessboard state.
    - Tip: use Gazebo to TF ROS node from lab 4 as a starting point; **note that you should not develop a vision-based solution.**

- Baxter should be able to set up the chessboard successfully (with at least 5 chess pieces, see section "Chessboard startup states").
    - Baxter must move each chess piece one at a time; therefore, each piece should spawn one at a time.

- You must define a chess game given your team's chosen chessboard state and make Baxter play at least 4 hardcoded moves.
    - Moves should be valid moves; you can generate valid moves by setting up the chessboard and playing a game at https://nextchessmove.com/.
    - In the online game at the URL above, you can set up the chessboard by dragging chess pieces out of the board and to a given square. After this, click on calculate next move and click on the suggested move URL link to perform the move. You can repeat this process to get valid moves!

Tips: We suggest working out first how to pick and place chess pieces at the same location, then setting up the chessboard and, finally, integrating all modules. You can use code from the labs. Also, remember to define the table in RViz as in Lab 4 or 5!

NOTE: Remember to save your RF workspace as noted in the "*Lab Briefings and Environment Setup*" document!

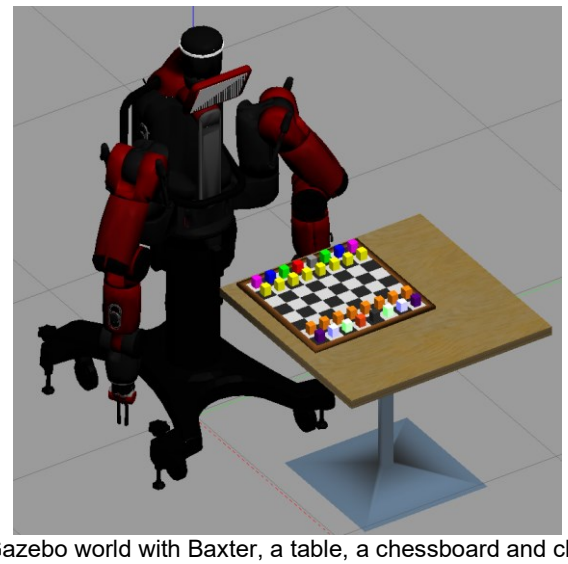

Figure 1. Gazebo world with Baxter, a table, a chessboard and chess pieces.

Your team and how to work in a team

You are free to work on this coursework as a Team. For this, you can form teams of up to 4 students – **groups of 5 or more will not be allowed; there is no minimum, and you can do your project individually**. You do not need to register your team; you will let us know who your team was during your submission (see the *What should be submitted* section).

If you are working in a team, you should consider how the tasks associated with developing your team's solution should be divided. **Try to ensure that activities are assigned so that every team member can always be involved, as we will test your understanding of the whole coursework** – the coding part accounts for 2 marks only. Here is a possible breakdown of responsibilities for a four-person team (this is just an example, and you should feel free to organise things differently if you prefer):

1. Pick and place
2. Object localisation and parsing
3. Integration of modules – high-level planning and design, e.g. message passing or ROS services, between ROS nodes.
4. Play a choreographed game

If a teammate disappears, is unresponsive, does not contribute, etc., feel free to complete the coding aspect of the coursework. We will not mediate team problems; this is because each team member will need to answer 3 questions individually, where we will assess your understanding and active involvement in the development of the solution. These 3 questions account for 18 marks. **If you actively participate in developing the solution, you should be capable of answering all questions!** 😊

Materials

For this assessed exercise, ensure you have the virtual machine's latest version. If you are not sure, please ask us in the lab! The latest virtual machine version has all the essential ROS packages you will need for this exercise, located at **`~/Desktop/RFLabs/`**.

You can also find a ZIP file in Moodle or **`~/Desktop/RFLabs/coursework`** (double-click on "Get RF labs" to get the latest update) that will help you develop your solution. The ZIP file contains two ROS nodes and a folder. Specifically, these are: -

- **spawn_chessboard.py** will set up the environment, i.e. 1 wooden table and the chessboard. The script will also set up chess pieces configurations in the parameter server.
- **delete_chess_game.py** will delete spawned chess pieces, the chessboard and the table from Gazebo. This node will help you to re-initialise the state of the board.
- A **models** folder which you should put inside your ROS package solution. This folder contains Gazebo 3D models of the chess pieces and chessboard used in the above scripts.

spawn_chessboard.py and **delete_chess_game.py** ROS nodes have some variables set in the parameter server. These are:

- **board_setup** – sets up the initial state of the game
- **list_pieces** – is a list that contains concatenated unique name chess pieces, e.g. "rnbqkpRNBQKP", each is:
    - r – black rook
    - n – black knight
    - b – black bishop
    - q – black queen
    - p – black pawn
    - then, upper case names for white pieces.

- **piece_target_position_map** – contains 3D positions for each square in the chessboard
- **piece_names** – is a list of those chess pieces defined in **board_setup**
- **pieces_xml** – is a python dictionary with file paths to Gazebo models, i.e. SDF files

Please note that the above variables are defined in the parameter server when **spawn_chessboard.py** is executed via **rosrun**. You can use these parameter server variables for your solution, allowing you to make your solution invariant to the initial chess game state. You can see an example of how to use these parameter server variables in **delete_chess_game.py**.

Chessboard startup states

You can change the initial game state in **spawn_chessboard.py**, line 51. This notation is a simplified version of the FEN notation used in chess games (https://www.chessprogramming.org/Forsyth-Edwards_Notation). For example, for a standard chess game including all pieces in their usual starting position, you could define **board_setup** as follows (see Figure 2a for an example in Gazebo):

```
board_setup = ['rnbqkbnr', 'pppppppp', '', '', '', '', 'PPPPPPPP', 'RNBQKBNR'.
```

To define a simplified initial game state (currently the default state for your project), you should add an asterisk, i.e. *, to the piece you do not want to include in each row. This is only valid for those rows in the chessboard that have pieces. For empty rows in the chessboard, you only need to add single quotation marks for each row that should be empty. The default chess game state is as follows (see Figure 2b):

```
board_setup = ['r******r', '', 'k*******', '', '', '*******K', '', 'R******R']
```

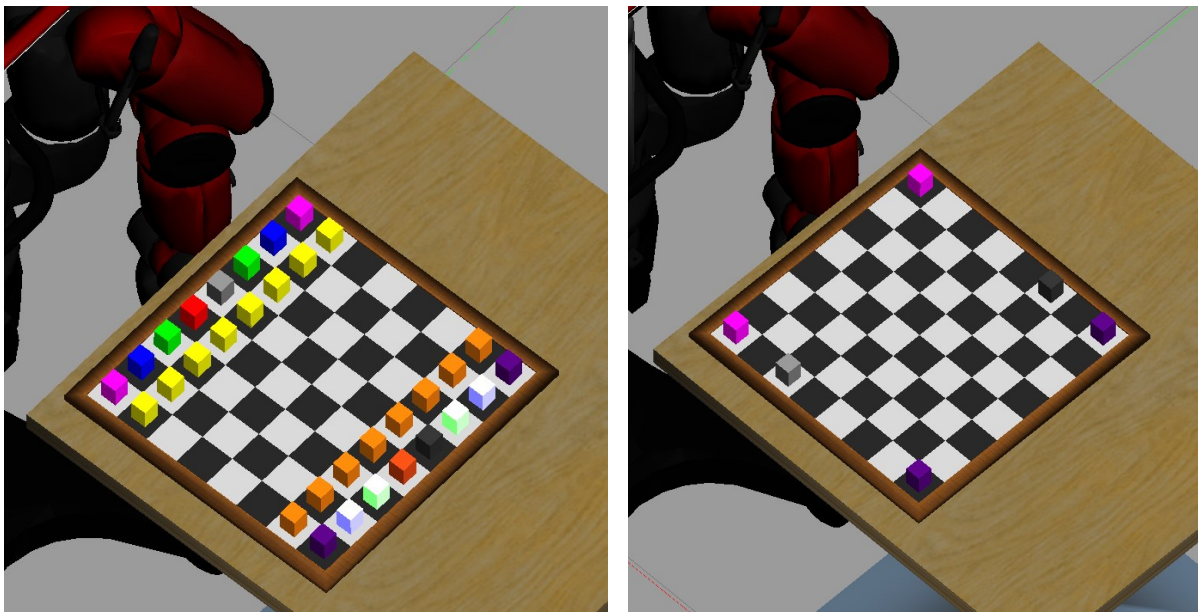

Figure 2. (left) standard chess game with all pieces; (right) simplified chess game.

Note that chess pieces are colour coded. We defined this colour definition to allow you to get visual feedback only.

What should be submitted

You should ensure that your ROS node/package addresses the following requirements:

Deployment

- A ROS package that groups the nodes your team used for this coursework
  - Here, you should create a new ROS package for your solution and make sure you can run **`catkin_make`** and use **`rosrun`**.

- Your ROS package can be deployed on the marker's virtual box
  - However, the marker should make any **`chmod`** operations appropriately - sometimes permissions are lost while compressing folders.

Functionality

- The ROS nodes allow Baxter to set up a chess game, and Baxter can perform at least 4 moves of your choice.

- Your ROS node must cover the task specification defined in the Introduction section. Specifically, we will be looking for the following:
  - The robot should be able to pick and place chess pieces (1 ROS node)
  - Make use of MoveIt! for Motion Planning
  - Being able to locate chess pieces such as different chessboard states can be defined (1 ROS node)
  - Service to spawn chess pieces on a pre-defined location. (1 ROS service)
  - Baxter must perform up to 4 valid moves as part of the game – these can be pre-defined for a given initial chessboard state.

- Video demonstration of your solution. For this, use your mobile phone video recording capabilities to record your computer's screen directly to save resources on the PC 😊
  - One video per team can be shared among the team members
  - Failing to submit as specified will result in losing 1 mark in question 1 in the following section.

Reflective Analysis (aka Lessons Learned)

Individually, you should submit your team's (or individual) code and video submission and answer 3 questions that consist of a reflective analysis of your team's (or individual) solution. For this, you will have to access the "Assessed Coursework 2023" quiz activity on the course's Moodle page. Below you can find the questions in the quiz:

1. Write down your teammates' student IDs and upload your code and video demonstration (2 marks).
    - In here, you will have a text field where you need to input your teammates' IDs and a drag & drop field where you can upload a zip file containing the source code and video demo.

2. Explain how your team (or you, if an individual submission) approached the problem (2 marks) and the assumptions taken (2 marks) (max. 200 words, 4 marks)

3. What did you struggle with the most during the coursework? Be specific and answer this question (2 marks), also elaborate on "why it was difficult" (4 marks), and give at 2 examples of how you would solve it (2 marks per example). (max. 400 words, 8 marks)

4. In Lab 5, you used the Canny edge detector and image moments to detect an object and its orientation for a pick-and-place task. How would you use deep learning to develop a vision module for this coursework? Assume you have an image database of chess pieces. (max. 300 words, 6 marks)

We recommend using a word processor to complete the above questions and copy/paste your answers to Moodle. Code should not be included in your answers; it is a reflective analysis rather than a coding exercise.

Tips:

- You **should** use your team's (or individual) video submission to support your statements.
- Note that each question has a maximum word requirement. If your answer is just above this maximum (i.e. 10% more), you will not be penalised as long as you have addressed the question!
- Make sure to acknowledge your team in question 1; otherwise, your submission will be counted as plagiarism and will be reported.
- Last question follows the format and style for the exam!

How to submit

You should submit via the quiz activity in Moodle:

- A ZIP or 7z file of your ROS package and video demonstration.
    - Note that the video demonstration should be within your ZIP or 7z file.

- **Individual:** The answers to the above questions.

You will be required to complete a Declaration of Originality for your submission. If you have used any external sources, be sure to acknowledge them in your answers. You have unlimited attempts, and non will be graded automatically. **We will only grade and give feedback on your last submission.**

Appendix 1: Marking scheme

The following marking scheme is intended to give a broad indication of how marks will be allocated. For details, please refer to this handout's "What should be submitted" section.

Category	Marks
Deployment ROS package is well structured (e.g. source code in src, services in srv, etc.) – http://wiki.ros.org/Packages#Common_Files_and_Directories	0.5
Functionality ROS nodes enable Baxter to play chess, and Baxter can perform at least 5 moves (marker can run the solution)	1
Code The code contains helper functions/classes (if required) and is readable and commented	0.5
Reflective Analysis Questions Answers should reflect a clear understanding of what the student has developed. See questions 2 to 4 in the “Reflective Analysis” section.	18
Total	20