CSC376H5F - Fundamentals of Robotics

Introduction to Gazebo – CSC376 – Fall 2020

Installation and Setup

Eigen library

Eigen is a C++ library that to efficiently perform matrix and vector operations.

- Install it via the commend line (Ubuntu)

```
$ sudo apt-get install libeigen3-dev
```

- Check if the installation is successful by checking the version of Eigen installed

```
$ pkg-config --modversion eigen3
```

- Install it via command line (Mac) : `http://macappstore.org/eigen/`

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
install/master/install)" < /dev/null 2> /dev/null
$ brew install eigen
```

You can find the documentation here `http://eigen.tuxfamily.org/dox/`. Make sure to get yourself familiar with Eigen, as it will be heavily used during the assignments.

Gazebo

Gazebo is a simulator that allows for realistic visualization and manipulation of robots. This simulator will be used throughout this course and the assignments in order to consolidate your understanding of certain concepts. Gazebo is available freely from here: `http://gazebosim.org/` We summarize the installation instructions in the following. The Gazebo website provides additional instructions: `http://gazebosim.org/tutorials?cat=install`

- We highly recommend using either Linux or Mac (or a Virtual Machine running Ubuntu when using Windows)
- Install Gazebo via the command line (Ubuntu)

```
$ sudo apt install curl
$ curl -sSL http://get.gazebosim.org | sh
```

- Install Gazebo via the command line (Mac)

```
$ curl -ssL http://get.gazebosim.org | sh
```

- If the above installation on Mac does not work, Gazebo should be built from source following this link: `https://github.com/osrf/gazebo_tutorials/blob/master/install_from_source/tutorial_default.md#build-and-install-gazebo`

- Gazebo installation instructions using Windows can be found here: `http://gazebosim.org/tutorials?tut=install_on_windows&cat=install`
- Run the program directly from the command line

```
$ gazebo
```

Simulation Framework and Panda robot

The Panda manipulator from Franka Emika is a state-of-the-art robotic arm. Due to the implementation of torque sensor in each of the seven links, it enables high safety protocols for human interaction. Throughout this course we will use this manipulator in the Gazebo simulator to emphasize certain discussed concepts. Furthermore, you will write code to model and manipulate this simulated robot in the assignments.

To set up the simulation framework used in this course, including the Gazebo simulator as well as a simulated Panda robot, follow these steps:

- Download and unzip the folder provided on Quercus alongside this practical
- Copy the `panda_arm` folder into `/usr/share/gazebo-11/models` so that the Franka Panda is available on Gazebo.
- Run the following commands in your terminal from the folder

```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:csc376-assignment-framework_path/build
$ export GAZEBO_RESOURCE_PATH=/usr/share/gazebo-11:
/usr/share/gazebo_models:csc376-assignment-framework_path
$ gazebo --verbose ../csc376_assigment.world
```

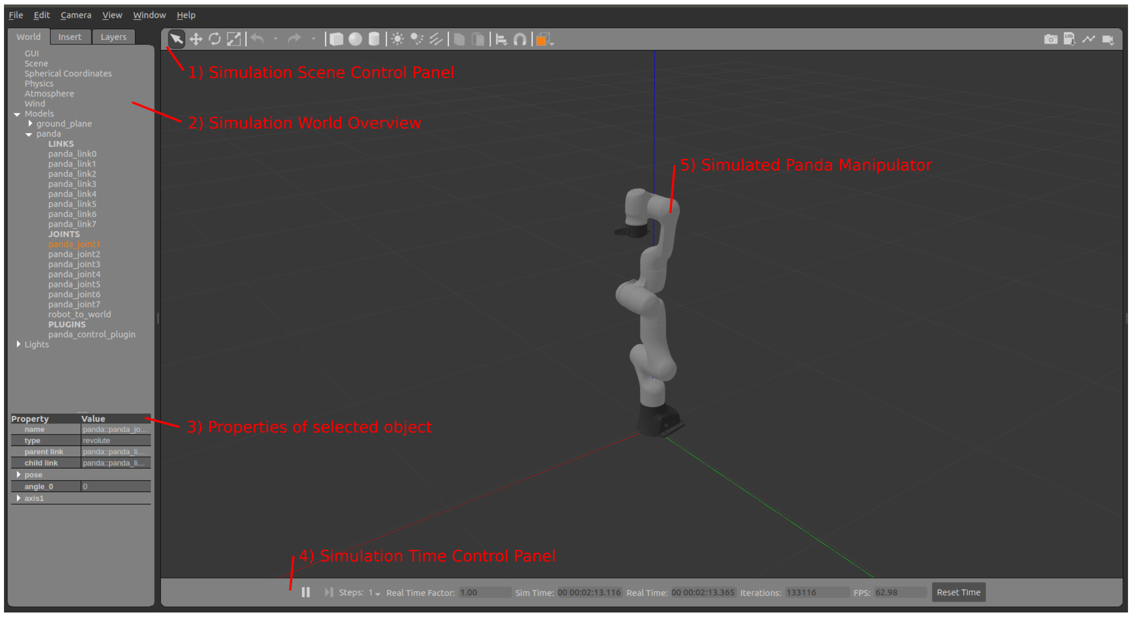
- The last command starts the gazebo simulator with a predefined setup file and you should now be able to see the visualization of the robot
- Note that `csc376-assignment-framework_path` needs to be replaced with the location of your unzipped folder
- The two exported path variables are local to this specific instance of the terminal and need to be repeated for any new terminal that should run the simulation environment
- In another terminal, you can run the executable ./csc376-assignment in the build folder to manipulate the simulation environment

Simulation Framework Introduction

Gazebo Simulation Environment

Fig. 1 shows an overview of the Gazebo simulation environment. The most relevant parts are discussed in the following.

1) Simulation Scene Control Panel: Tools to navigate the simulation scene (move, rotate, etc.)
2) Simulation World Overview: Overview of the simulated world, that is defined in a `*.world` file that is loaded when starting Gazebo
3) Properties of selected object: The main properties of the currently selected object from the world scene (e.g. the current values or limits of a selected joint)



Property	Value
name	panda::panda_jo...
type	revolute
parent link	panda::panda_li...
child link	panda::panda_li...
pose	
angle_0	0
axis1	

Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:02:13.116 Real Time: 00 00:02:13.365 Iterations: 133116 FPS: 62.98

Figure 1: Overview of the Gazebo simulation environment

4) Simulation Time Control Panel: Tools to control the time of the robot simulation (start, stop, step forward, etc.)

5) Simulated Panda Manipulator: Model of the simulated robot

C++ Code Framework

Fig. 2 shows an overview of the C++ framework to interface the Gazebo simulation. It consists of two parts: The Gazebo simulation itself and the code to manipulate it. When starting the Gazebo simulator, a predefined scene is loaded, specified using a `*.world` file. In this file, the robot manipulator is described. Furthermore, a plugin is loaded (Panda Model Plugin), that allows to interact with the simulated robot. Communication with this loaded plugin works through the internal gazebo topic infrastructure. You are provided with an already written class (Gazebo Controller) that handles all the interfacing of the Gazebo simulator and offers all necessary functionalities for the practicals and assignments. This class is used in the main function of the provided code framework. Furthermore, a second class (Panda Robot Model) is defined. For now, this class is empty, as it will be your job to implement certain functionalities of it during the assignments and to verify your implementation with the simulated robot.

The provided functions of the Gazebo Controller are shown in Fig.3. While this list is quite exhaustive, we will focus on only a few during this practical. Make sure to read the descriptions of each function and try to understand what each function does and what the inputs and outputs are.

For this practical, we will code some simple example interactions with the simulated robot. A basic framework to start is provided in `main.cpp` as shown in Fig. 4.

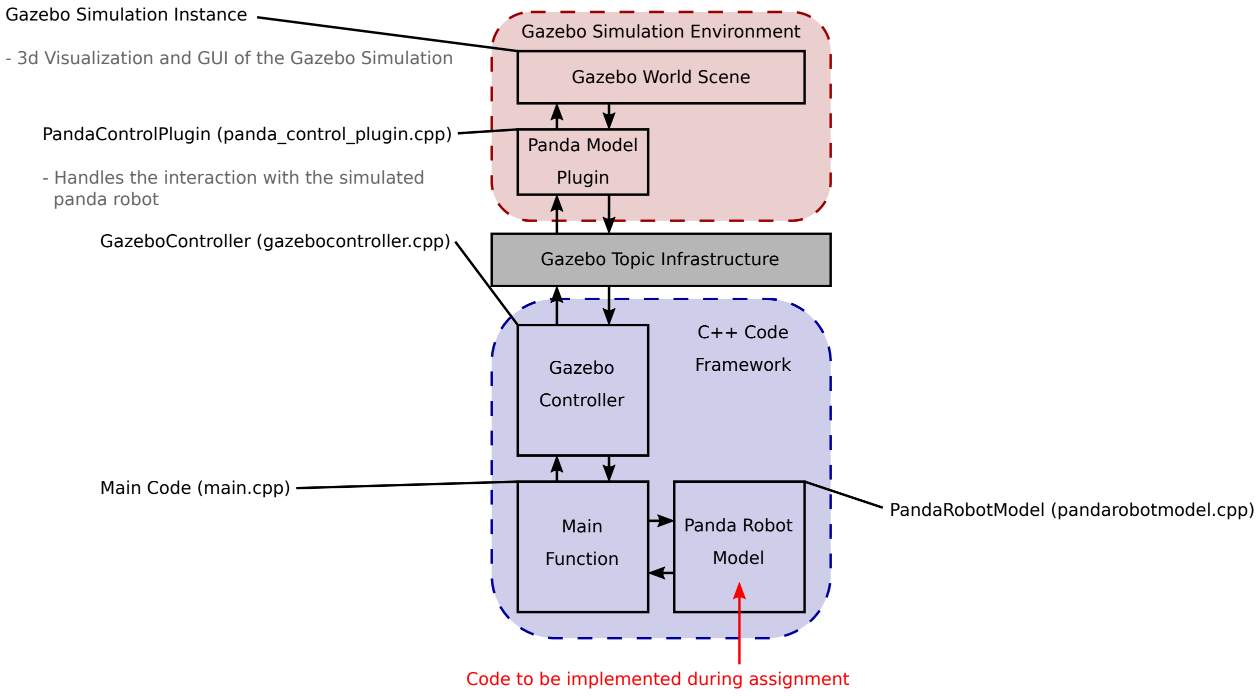

Gazebo Simulation Instance
- 3d Visualization and GUI of the Gazebo Simulation
PandaControlPlugin (panda_control_plugin.cpp)
- Handles the interaction with the simulated panda robot
GazeboController (gazebocontroller.cpp)
Main Code (main.cpp)
PandaRobotModel (pandarobotmodel.cpp)
Gazebo Simulation Environment
Gazebo World Scene
Panda Model Plugin
Gazebo Topic Infrastructure
C++ Code Framework
Gazebo Controller
Main Function
Panda Robot Model
Code to be implemented during assignment

Figure 2: Overview of the provided C++ framework to interface the Gazebo simulation

Coding Workflow

Generally, you want to start the Gazebo simulator once using the specified `*.world` file once using this command (and making sure you are in the build folder):

```
$ gazebo --verbose ../csc376_assigment.world
```

Afterwards, in a second terminal you want to run the program, that is written in the `main.cpp` file using this command (and making sure you are in the build folder):

```
$ ./csc376-assignment
```

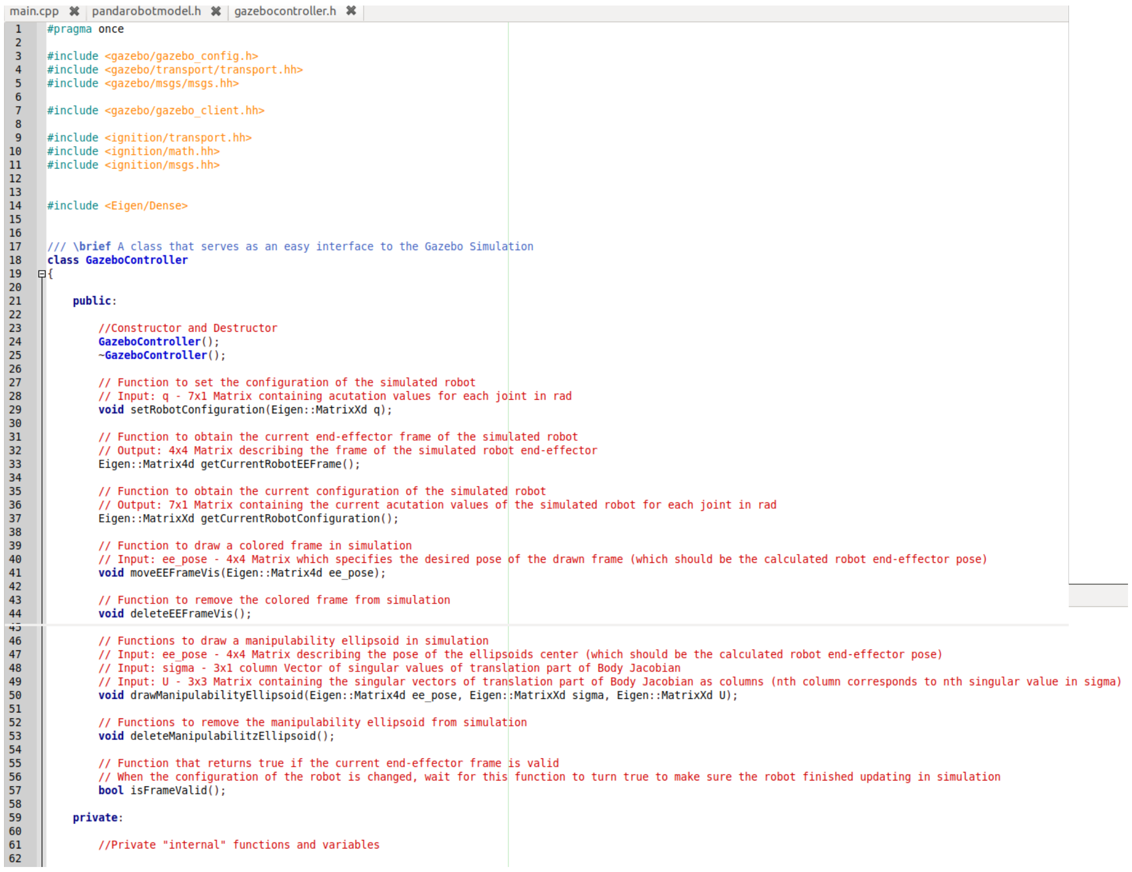
Ideally, after this program is finished, it should reset the simulator to its initial state, allowing you to run the same or any other program again without needing to restart the simulator. When making changes to the code, you need to rebuild the code using these commands in the build folder:

```
$ cmake ..
$ make
```

Since you will most likely only make changes to the C++ code framework and not to the plugins loaded during the Gazebo simulator startup, you do not need to restart the simulation afterwards.

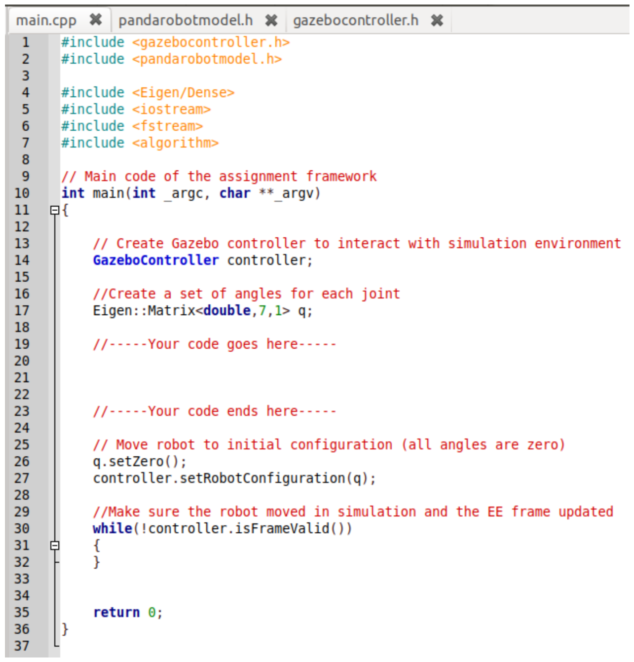
Exercises

1. Install and setup the Gazebo simulation environment together with the provided C++ framework.
2. Using the provided code framework, write a simple program, that takes user inputs from the terminal to move a specific joint of the robot by a defined amount. Make sure to take joint limits into account!
3. Modify the program to allow the user to move any number of joints by specified amounts. Print the resulting end-effector frame (4×4 matrix) of the simulated robot using the provided function.



```cpp
#pragma once

#include <gazebo/gazebo_config.h>
#include <gazebo/transport/transport.hh>
#include <gazebo/msgs/msgs.hh>

#include <gazebo/gazebo_client.hh>

#include <ignition/transport.hh>
#include <ignition/math.hh>
#include <ignition/msgs.hh>


#include <Eigen/Dense>


/// \brief A class that serves as an easy interface to the Gazebo Simulation
class GazeboController
{

    public:

        //Constructor and Destructor
        GazeboController();
        ~GazeboController();

        // Function to set the configuration of the simulated robot
        // Input: q - 7x1 Matrix containing acutation values for each joint in rad
        void setRobotConfiguration(Eigen::MatrixXd q);

        // Function to obtain the current end-effector frame of the simulated robot
        // Output: 4x4 Matrix describing the frame of the simulated robot end-effector
        Eigen::Matrix4d getCurrentRobotEEFrame();

        // Function to obtain the current configuration of the simulated robot
        // Output: 7x1 Matrix containing the current acutation values of the simulated robot for each joint in rad
        Eigen::MatrixXd getCurrentRobotConfiguration();

        // Function to draw a colored frame in simulation
        // Input: ee_pose - 4x4 Matrix which specifies the desired pose of the drawn frame (which should be the calculated robot end-effector pose)
        void moveEEFrameVis(Eigen::Matrix4d ee_pose);

        // Function to remove the colored frame from simulation
        void deleteEEFrameVis();

        // Functions to draw a manipulability ellipsoid in simulation
        // Input: ee_pose - 4x4 Matrix describing the pose of the ellipsoids center (which should be the calculated robot end-effector pose)
        // Input: sigma - 3x1 column Vector of singular values of translation part of Body Jacobian
        // Input: U - 3x3 Matrix containing the singular vectors of translation part of Body Jacobian as columns (nth column corresponds to nth singular value in sigma)
        void drawManipulabilityEllipsoid(Eigen::Matrix4d ee_pose, Eigen::MatrixXd sigma, Eigen::MatrixXd U);

        // Functions to remove the manipulability ellipsoid from simulation
        void deleteManipulabilitzEllipsoid();

        // Function that returns true if the current end-effector frame is valid
        // When the configuration of the robot is changed, wait for this function to turn true to make sure the robot finished updating in simulation
        bool isFrameValid();

    private:

        //Private "internal" functions and variables

```

Figure 3: Overview of the functionalities of the GazeboController class of the C++ framework



```cpp
#include <gazebocontroller.h>
#include <pandarobotmodel.h>

#include <Eigen/Dense>
#include <iostream>
#include <fstream>
#include <algorithm>

// Main code of the assignment framework
int main(int _argc, char **_argv)
{

    // Create Gazebo controller to interact with simulation environment
    GazeboController controller;

    //Create a set of angles for each joint
    Eigen::Matrix<double,7,1> q;

    //-----Your code goes here-----



    //-----Your code ends here-----

    // Move robot to initial configuration (all angles are zero)
    q.setZero();
    controller.setRobotConfiguration(q);

    //Make sure the robot moved in simulation and the EE frame updated
    while(!controller.isFrameValid())
    {
    }


    return 0;
}

```

Figure 4: Main function of the C++ framework