Team Name: NamingSoHard
Component Design
3/26/2022

Legend:

1) Main components:
    a) #include <Eigen>
       Eigen library that contains all functions for matrix multiplication.
    b) #include <ros/ros.h>
       ROS/RVIZ will be used to visualize the generated point cloud.
2) Custom class “Robot” read from URDF file.
   http://wiki.ros.org/urdf

```
class Robot() {
    read_urdf(filepath);
    linked_list<Joint> joints;
    forward_kinematics(joints, joint_angles);
    getWS();
    unordered_map point_cloud;
    save2pcd();
}
```

a) **Robot::read_urdf():**
   This function parses a URDF file and stores all the relevant information in a linked list data structure. For example, given a UR3e arm, 6 revolute joints would be defined with their respective parameters as defined by Robot::Joint().

```
Robot::Joint() {
    joint_type enum {
        revolute,
        prismatic
    };

    string frame_name;
    string parent_frame;
    Eigen::Vector3d origin_xyz; // offset from parent frame
    Eigen::Vector3d origin_rpy;
    Eigen::Vector3d axis;       // rotational axis
    double joint_state;         // joint angle
    double[2] joint_limit;      // lower and upper limit
```

```
    print();                    // debug purposes
}
```

b) **Robot::forward_kinematics():**
The forward_kinematics function will sweep all n-joints of the serial manipulator and compute the end-effector pose using denavit-hartenberg (DH) parameters. The end-effector poses will be saved to a hash map such that the pose can be queried given n-joint angles.

c) **Robot::getWS()**
This member function will call the **forward_kinematics** function via a multi-threaded server, hence, inserting into the hash map in a parallel manner.

d) **Robot::save2pcd()**
This member function will take the hash map of end-effector poses given joint angles and store the data as a point cloud file that can be visualized in many open-source softwares.

RVIZ visualizer:
Work as an independent code package, takes in a saved ".PCD" point cloud file, and visualizes it using RVIZ. Implement RVIZ's built-in functions to visualize a robot using URDF files and STL models.

Unit Testing

a) **Robot::read_urdf():**
Information from a urdf file can be read and imported into a joint object. The members of the object can then be compared to the values listed in the original urdf file. If all the values match, then the file was read correctly, and the test passes.
- Test correct urdf file.
- Test urdf that's not tree structured.
- Test urdf without joint-limit information.

b) **Robot::forward_kinematics():**
The forward kinematics package will be tested against a the Peter Corke's Robotics toolbox for Matlab (https://petercorke.com/toolboxes/robotics-toolbox/).

c) **Robot::getWS()**
Make a simplified robot arm that only has two joints and generate the pointcloud. Since there are only two joints, there will be one link and the workspace will be a circle. Confirm this by finding the norm of every point location and checking that it matches the linkage length.

d) **Robot::save2pcd()**

Make a simplified robot arm with two joints and write the output to a pcd file. Compare the magnitudes of the coordinates of the output to the length of the robot arm. If they are all the same value, the test passes.

Task Allocation

Tasks	Daniel	Kirtan	Maheer	William	Troy
Implement Robot::Joint()					
Robot::read_urdf()					
Robot::forward_kinematics()					
Robot::getWS()					
Robot::save2pcd()					
Implement RVIZ					
Unit Testing (a)					
Unit Testing (b)					
Unit Testing (c)					
Unit Testing (d)					