~/Classes/CS225A/OpenSai/cs225a/homework/hw0/hw0.cpp

```cpp
// some standard library includes
#include <math.h>
#define _USE_MATH_DEFINES

#include <iostream>
#include <fstream>
#include <mutex>
#include <string>
#include <thread>

// sai main libraries includes
#include "SaiModel.h"

// sai utilities from sai-common
#include "timer/LoopTimer.h"
#include "redis/RedisClient.h"

// redis keys
#include "redis_keys.h"

// for handling ctrl+c and interruptions properly
#include <signal.h>
bool runloop = true;
void sighandler(int) { runloop = false; }

// namespaces for compactness of code
using namespace std;
using namespace Eigen;

// config file names and object names
const string robot_file = "${CS225A_URDF_FOLDER}/rprbot.urdf";

int main() {
    SaiModel::URDF_FOLDERS["CS225A_URDF_FOLDER"] = string(CS225A_URDF_FOLDER);

    // set up signal handler
    signal(SIGABRT, &sighandler);
    signal(SIGTERM, &sighandler);
    signal(SIGINT, &sighandler);

    // Make sure redis-server is running at localhost with default port 6379
    // start redis client
    auto redis_client = SaiCommon::RedisClient();
    redis_client.connect();

    // load robots
```

```cpp
    auto robot = new SaiModel::SaiModel(robot_file, true);

    /*
    These are mathematical vectors from the library Eigen, you can read up on the documentation online.
    You can input your joint information and read sensor data C++ style "<<" or ">>". Make sure you only
    expect to read or are writing #D.O.F. number of values.
    */
    int dof = robot->dof();
    Eigen::VectorXd robot_q = Eigen::VectorXd::Zero(dof);
    Eigen::VectorXd robot_dq = Eigen::VectorXd::Zero(dof);
    robot_q << 0.0, 0.6, M_PI/3; // Joint 1,2,3 Coordinates (radians, meters, radians)
    robot_dq << 1.0, 0.0, 0.0; // Joint 1,2,3 Velocities (radians/sec, meters/sec, radians/sec), not used here
    robot->setQ(robot_q);
    robot->setDq(robot_dq);

    /*
    Here we use our redis set method to serialize an 'Eigen' vector into a specific Redis Key
    Changing set to get populates the 'Eigen' vector given
    This key is then read by the physics integrator or visualizer to update the system
    */
    redis_client.setEigen(JOINT_ANGLES_KEY, robot->q());
    redis_client.setEigen(JOINT_VELOCITIES_KEY, robot->dq());

    /*
    Update model calculates and updates robot kinematics model information
    (calculate current jacobian, mass matrix, etc..)
    Values taken from robot->q() will be updated to currently set _q values
    */
    robot->updateModel();

    cout << endl << endl;

    // operational space
    const string ee_link_name = "link2"; // Link of the "Task" or "End Effector"

    // Empty default values
    Vector3d ee_pos_in_link = Vector3d(0.0, 0.0, 0.0); // Position of Task Frame in relation to Link Frame (When using custom E.E. attachment, etc..)
    Vector3d ee_position = Vector3d::Zero(); // 3d vector of zeros to fill with the end effector position
    MatrixXd ee_jacobian(3, dof); // Empty Jacobian Matrix sized to right size
    VectorXd g(dof); // Empty Gravity Vector
```

```
87
88      // Examples how to update and get position, Jacobians, gravity vectors
89      ee_position = robot->position(ee_link_name, ee_pos_in_link); // get end-
    effector's position, and write into ee_position
90      cout << "End effector position w.r.t. ground :" << endl;
91      cout << ee_position.transpose() << endl << endl;
92
93      ee_jacobian = robot->Jv(ee_link_name, ee_pos_in_link); // get jacobian, and
    write into ee_jacobian
94      cout << "Printing Jacobian and Mass matrix : " << endl;
95      cout << ee_jacobian << endl; // Print Jacobian
96      cout << robot->M() << endl << endl; // Print Mass Matrix, you can index
    into this variable (and all 'Eigen' types)!
97
98      g = robot->jointGravityVector(); // get gravity vector, and write into g
99      cout << "Printing gravity : " << endl;
100     cout << endl << g.transpose() << endl << endl;
101
102     /*
103     Retrieve multiple values of the gravity or M with a for loop of calling
    robot->SetQ(robot_q),
104     setting redis keys for display update if needed and don't forget robot-
    >updateModel()!
105     We'll have a logger for you later to dump redis values at whatever rate you
    choose
106     */
107
108     //
    ********************************************************************************
109     // **********************    WRITE YOUR CODE AFTER
    **********************
110     //
    ********************************************************************************
111
112     // ---------------------------  question 2-b ------------------------------
    -
113     ee_pos_in_link = Vector3d(0.0, 0.0, 2.5); // modify this
114
115     // ---------------------------  question 2-c ------------------------------
    -
116     // part i
117     robot_q << 0.0, 0.5, -1 * M_PI / 2; // modify this
118     robot->setQ(robot_q);
119     robot->updateKinematics();
120     ee_position = robot->position(ee_link_name, ee_pos_in_link);
121     cout << "======================================== Q2-c-i" << endl << endl;
122     cout << "End effector position for configuration i\n" <<
    ee_position.transpose() << endl << endl;
123     // part ii
```

```
124     robot_q << 1 * M_PI / 2, 0.5, -1 * M_PI / 2; // modify this
125     robot->setQ(robot_q);
126     robot->updateKinematics();
127     ee_position = robot->position(ee_link_name, ee_pos_in_link);
128     cout << "======================================== Q2-c-ii" << endl <<
    endl;
129     cout << "End effector position for configuration i\n" <<
    ee_position.transpose() << endl << endl;
130 
131     // --------------------------  question 2-d ------------------------------
    -
132     // part i
133     robot_q << 0.0, 0.5, -1 * M_PI / 2; // modify this
134     robot->setQ(robot_q);
135     robot->updateKinematics();
136     ee_jacobian = robot->Jv(ee_link_name, ee_pos_in_link);
137     cout << "======================================== Q2-d-ii" << endl <<
    endl;
138     cout << "Jv for configuration d-i\n" << ee_jacobian << endl << endl;
139     // part ii
140     robot_q << 1 * M_PI / 2, 0.5, -1 * M_PI / 2; // modify this
141     robot->setQ(robot_q);
142     robot->updateKinematics();
143     ee_jacobian = robot->Jv(ee_link_name, ee_pos_in_link);
144     cout << "======================================== Q2-d-ii" << endl <<
    endl;
145     cout << "Jv for configuration d-ii\n" << ee_jacobian << endl << endl;
146 
147     // --------------------------  question 2-e ------------------------------
    -
148     // part i
149     ofstream file_2e_i;
150     file_2e_i.open("../../homework/hw0/q2-e-i.txt");
151     robot_q << 0.0, 0.5, -1 * M_PI / 2; // modify this
152     robot->setQ(robot_q);
153     robot->updateModel();
154     file_2e_i << robot->M()(0) << "\t" << robot->M()(4) << "\t" << robot->M()
    (8) << "\n"; // modify this
155     int n_steps = 250;
156     for(int i=0 ; i < n_steps ; i++)
157     {
158         robot_q << 0.0, 0.5, -1 * M_PI / 2 + (i*M_PI)/n_steps;
159         robot->setQ(robot_q);
160         robot->updateModel();
161         file_2e_i << robot->M()(0) << "\t" << robot->M()(4) << "\t" << robot-
    >M()(8) << "\n";
162 
163     }
164     file_2e_i.close();
```

```cpp

    // part ii
    ofstream file_2e_ii;
    file_2e_ii.open("../../homework/hw0/q2-e-ii.txt");
    robot_q << 0.0, 0.0, 0.0; // modify this
    robot->setQ(robot_q);
    robot->updateModel();
    file_2e_ii << robot->M()(0) << "\t" << robot->M()(4) << "\t" << robot->M()(8) << "\n"; // modify this
    n_steps = 250;
    for(int i=0 ; i < n_steps ; i++)
    {
        robot_q << 0.0, i*2/n_steps, 0;
        robot->setQ(robot_q);
        robot->updateModel();
        file_2e_ii << robot->M()(0) << "\t" << robot->M()(4) << "\t" << robot->M()(8) << "\n";
    }
    file_2e_ii.close();

    // ---------------------------  question 2-f -------------------------------
    // part i
    ofstream file_2f_i;
    file_2f_i.open("../../homework/hw0/q2-f-i.txt");
    robot_q << 0.0, 0.5, -1 * M_PI / 2; // modify this
    robot->setQ(robot_q);
    robot->updateModel();
    g = robot->jointGravityVector();
    file_2f_i << g.transpose() << "\n";
    n_steps = 250;
    for(int i=0 ; i < n_steps ; i++)
    {
        robot_q << 0.0, 0.5, -1 * M_PI / 2 + (i*M_PI)/n_steps;
        robot->setQ(robot_q);
        robot->updateModel();
        g = robot->jointGravityVector();
        file_2f_i << g.transpose() << "\n";
    }
    file_2f_i.close();

    // part ii
    ofstream file_2f_ii;
    file_2f_ii.open("../../homework/hw0/q2-f-ii.txt");
    robot_q << 0.0, 0.0, 0.0; // modify this
    robot->setQ(robot_q);
    robot->updateModel();
```

```cpp
    g = robot->jointGravityVector();
    file_2f_ii << g.transpose() << "\n";
    n_steps = 250;
    for(int i=0 ; i < n_steps ; i++)
    {
        robot_q << 0.0, i*2/n_steps, 0;
        robot->setQ(robot_q);
        robot->updateModel();
        g = robot->jointGravityVector();
        file_2f_ii << g.transpose() << "\n";
    }
    file_2f_ii.close();

    // -----------------  question 2-g : extra credit-------------------------
    // extra credit
    VectorXd grav_bis = VectorXd::Zero(4);

    // write your code

    return 0;
}

```