TU Berlin robotics WS2020/2021

Author: Jingsheng Lyu

Lab Assignment #0

1. Implement the ***struct*** **Vec2d(in SpringMass.h)** which should represent a 2d vector (x and y).

```cpp
struct Vec2d {
  double x; //position
  double y; //velocity

  // TODO
  Vec2d()
  {
    x = 0;  //init position
    y = 0;  //init velocity
  }
};
```

2. Implement the **constructor** of the **class SpringMass**. Define the necessary **member variables** for storing the initial position and velocity of the object and the position and velocity of the object when the spring is unstretched and in equilibrium. Don't forget to create a variable for the current time.

   2.1. Define the **member variables**

```cpp
  // TODO define your private methods and variables here
  std::vector<Vec2d> traj; //define a vector to describe trajectory

  double position;     //define position
  double velocity;     //define velocity

  double position_eqm; //define position in equilibrium
  double velocity_eqm; //define velocity in equilibrium

  int time;      //define current time;

};

#endif // SpringMass__H__
```

   2.2.

   2.3. Implement the **constructor** of the ***Class SpringMass***

2.4.

```
SpringMass::SpringMass(double pos_init, double vel_init, double pos_eqm, double vel_eqm)
{
    //init
    position = pos_init;
    velocity = vel_init;

    position_eqm = pos_eqm;
    velocity_eqm = vel_eqm;

    time = 0;

    //record state into <vector>traj
    Vec2d state;
    state.x = position;
    state.y = velocity;
    traj.push_back(state);

}
```

3. Implement the method ***SpringMass::step()***. Use the **constants** defined in the SpringMass class. It should perform one **simulation step** of the system according to the equations of motion given above. It should return the last simulated timestep where the time step is an integer.
    3.1. Perform one simulation step
    3.2.

```
// TODO SpringMass simulation step
int SpringMass::step() {

    //Step by step simulation
    velocity = velocity - (SPRING_CONST/MASS) * (position - position_eqm);
    position = position + velocity;
    time = time + 1;

    //record state into <vector>traj
    Vec2d state;
    state.x = position;
    state.y = velocity;
    traj.push_back(state);
    return time;
}
```

4. Implement the method ***SpringMass::getCurrentSimulationTime() const***.
    4.1. Return the current time
    4.2.

```
// TODO SpringMass current simulation time getter
int SpringMass::getCurrentSimulationTime() const {

    return time;  //return the current time
}
```

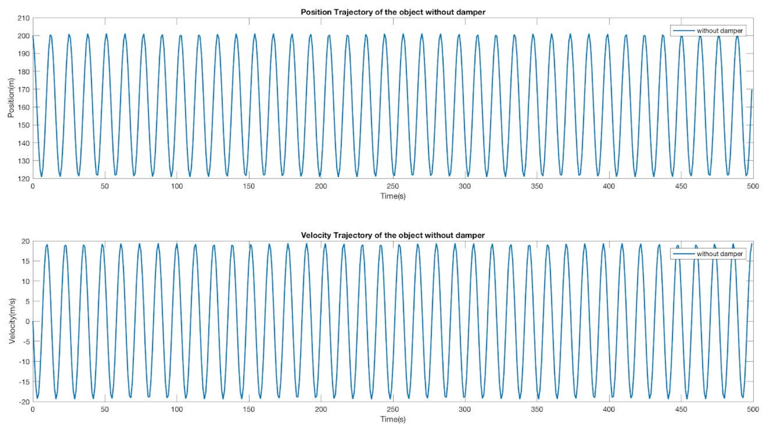
5. Implement the method ***SpringMass::getConfiguration(int t, Vec2d& state) const***. Given a time t, it should **return the state (position, velocity) of the object** at the time. Only times which have already been simulated should be allowed as input (return false if t is invalid, true otherwise).
    5.1. Return the state(position, velocity) of the object at the time

5.2.

```
// TODO SpringMass configuration getter
bool SpringMass::getConfiguration(int t, Vec2d& state) const {
  if (t <= time)
  {
    //take the state from <vector>traj
    Vec2d state_t = traj[t];
    state.x = state_t.x;
    state.y = state_t.y;
    return true;
  }

  else
  {
    return false;
  }

}
```

6. Generate a trajectory with initial position 200, initial velocity 0 and x0=161 for the spring mass system for t going from 0 to 500. Use your favorite plotting tool to visualize the generated data (position and velocity).

6.1.



7. Implement the ***class SpringMassDamper***. We now add a damper to our system.

The equations of motion change to:

$$
\begin{aligned}
\dot{x}(t+1) &= \dot{x}(t) - \frac{b}{m}\dot{x}(t) - \frac{k}{m}(x(t) - x_o) \\
x(t+1) &= x(t) + \dot{x}(t+1)
\end{aligned}
$$

where b is the damping coefficient.

Implement the new ***class*** such that it follows the altered equations of motion. It should be a subclass of the class SpringMass.

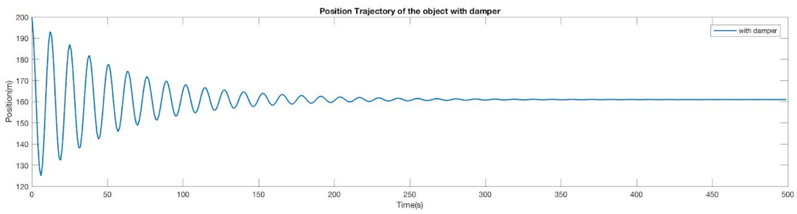
7.1. Define the new variable and new methode ***(SpringMassDamper.h)***

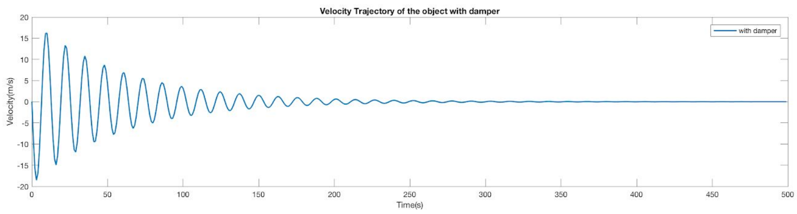
7.2. Define the new class with damper

7.3.

```cpp
#include "SpringDamperMass.h"

// TODO
// Define your methods here

int SpringDamperMass::step()
{
    //Step by step on equation
    velocity = velocity - (damping_coeff/MASS) * velocity - (SPRING_CONST/MASS) * (position - position_eqm);
    position = position + velocity;
    time = time + 1;

    //record state into traj
    Vec2d state;
    state.x = position;
    state.y = velocity;
    traj.push_back(state);

    return time;
}
```

8. Generate a trajectory with initial position 200, initial velocity 0, x0=161 and b=1 for the spring mass damper system for t going from 0 to 500. Use your favorite plotting tool to visualize the generated data (position and velocity).





8.1.

9. Preliminary

9.1. Test by terminal

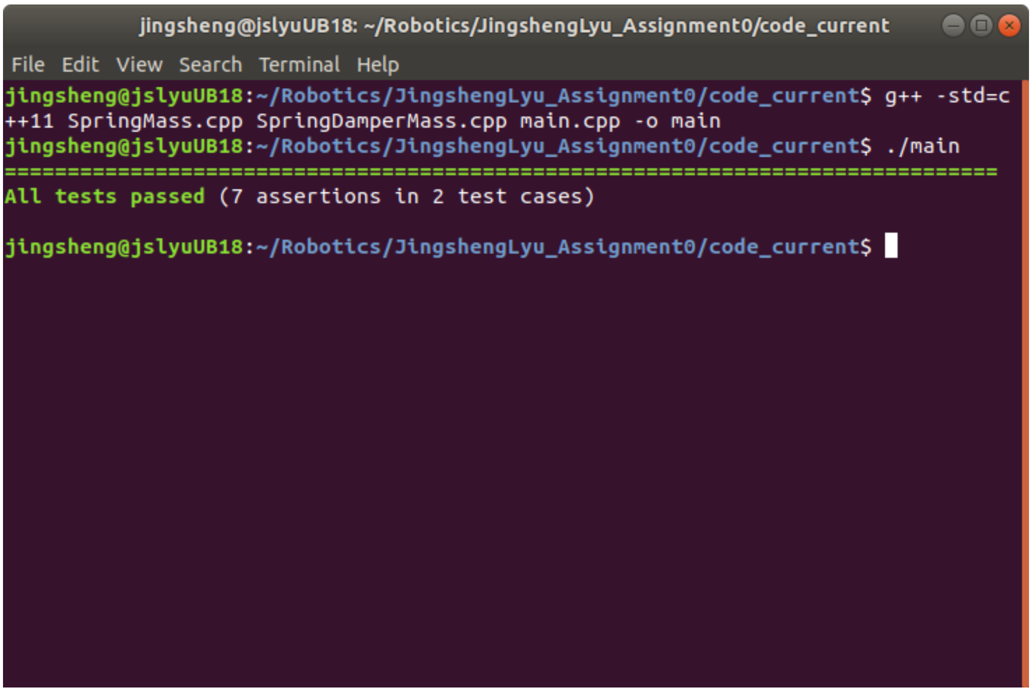
9.1.1. $ cd folder_with_code

9.1.2. $ g++ -std=c++11 SpringMass.cpp SpringDamperMass.cpp main.cpp -o main

9.1.3. $ ./main

9.2. Result

9.3.


jingsheng@jslyuUB18: ~/Robotics/JingshengLyu_Assignment0/code_current
File Edit View Search Terminal Help
jingsheng@jslyuUB18:~/Robotics/JingshengLyu_Assignment0/code_current$ g++ -std=c++11 SpringMass.cpp SpringDamperMass.cpp main.cpp -o main
jingsheng@jslyuUB18:~/Robotics/JingshengLyu_Assignment0/code_current$ ./main
===
All tests passed (7 assertions in 2 test cases)
jingsheng@jslyuUB18:~/Robotics/JingshengLyu_Assignment0/code_current$