Optimization, Robotics & Biomechanics (ORB)
Institute of Computer Engineering (ziti)
Heidelberg University

Prof. Dr. Katja Mombaur,
Benjamin Reh,
Silvan Lindner

Robotics 1 (WS 2018/2019)

Exercise Sheet 8

Presentation during exercises in calendar week 2

In this sheet we will use inverse kinematics to have our robot juggle a ball on a pinpong paddle for Christmas. Although the second assignment builds ontop of the first it is possible to solve it without having a solution for the first.

Exercise 8.1 – Reflecting the ball on a paddle

This exercise bases on the "bouncing ball" assignment from a previous sheet. Friction and wind are set to zero, as we want to keep the total energy in the system constant.

a) Download and compile the provided code. For this assignment run `./paddle_only` and load the results into meshup with `meshup paddle_only.lua motion-paddle_only.txt`. You should see the ball dropping down through the paddle.

b) As we saw in the previous sheet, to make the ball bounce we can invert one component of the ball's velocity vector $\vec{v}$. In our case, the normal axis for the paddle points in z-direction. Design a reflection matrix R to swap the sign of the z-component

$$\vec{v}_{\text{after}} = R \cdot \vec{v}_{\text{before}} \tag{1}$$

The given code has a `paddle.cpp` containing the `Paddle` class that will be used though the whole exercise. Enter the matrix R into function `Paddle::reflect(Eigen::Vector3d v)`.

As a result you will see the ball being reflected on the paddle while slowly wandering of the paddle.

c) In this step we want to implement the criteria to decide when the ball hits the paddle and must be reflected. This is decided in funciton `Paddle::canHitBall(...)`. Three criteria must met:

- The ball's z-component of the position must be < 0. That is already implemented.
- The ball's z-component of the velocity must be < 0. This is important in order not to reflect the ball twice. (How can this actually happen?)
- The ball's projected (x, y) position should be on the paddle, so the ball does not miss the paddle. It is up to you whether you also want to take the ball's radius into account.
  Hint: Take an inspiration from cylinder coordinates.

You should now see the ball bouncing off the paddle and then dropping when it misses the paddle.

d) Up until now the paddle lies in the coordinate origin and its normal points in z-direction. Using the member variable `Paddle::position` you can move the paddle around in space. Adjust the `Paddle::canHitBall(...)` function to internally use a vector in paddle-frame (e.g. $\vec{d}$) instead of world-frame.

`Paddle::orientation` is a 3×3 matrix that contains the rotation matrix O from the world frame to the paddle's local frame:

$$\vec{v}_{\text{local}} = O \cdot \vec{v} \tag{2}$$

Tilt the paddle by about $\frac{\pi}{10}$ by setting this rotation matrix in the `main()`-function.
Hint: You can use the angle-axis representation using Eigen's `AngleAxisd(...)` function from the previous sheets to comfortly define the orientation matrix around for example the x axis.

Rotate $\vec{d}$ and the ball velocity in `Paddle::canHitBall(...)` into the local frame using the orientation matrix from eq. (2).

In `Paddle::reflect(Eigen::Vector3d v)` the reflected velocity has to be first transformed into the local frame, then the reflection matrix has to be applied. At last the result has to be transformed back into the world frame. Set up the correct return value for the new, reflected velocity.
Reminder: For rotations $A^T = A^{-1}$ and in general $AB \neq BA$.

You should now be able to place the paddle in an arbitrary positon with an arbitrary orientation and have a correct reflection of the ball as shown in fig. 1. Of course, if the ball hits from the backside (dark side) it will just fall through but that is not to be bothered.

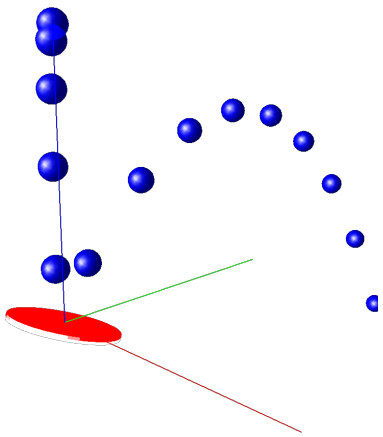

Figure 1: Ball reflecting on a tilted paddle

Exercise 8.2 – Tracing the ball

In this assignment we will set up the inverse kinematics of the Kuka robot to follow the trajectory of the ball. We will use the `Paddle` class from the previous assignment. If you had problems with the implementation, you can use the original one supplied with this exercise. You will not see a correct reflection but tracing the ball trajectory will work anyway.

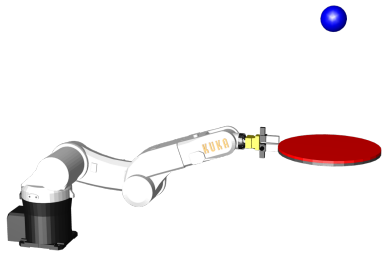

Figure 2: Kuka robot underneath the ball

a) Run `./kuka_trace` and load the results into meshup with `meshup kuka_paddle.lua animation_kuka_trace.csv`. You should see the ball dropping down through the paddle.

b) We need to set the paddle's position and orientation using the forward kinematics function for the kuka model including the paddle.

```
int paddle_id = model.GetBodyId("Paddle");

paddle.position = CalcBodyToBaseCoordinates(model, q,
   paddle_id,Vector3d(0,0,0),true);
paddle.orientation = CalcBodyWorldOrientation (model, q,
   paddle_id,false);
```

Now the ball will bounce off the static paddle. (If you did not do the previous assignment then you will see nothing new in this step.)

c) Now we want to use the inverse kinematic of the model to trace the ball's trajectory in the xy-plane. The z-component shall be fixed to $z = 0.5$. Since the paddle's normal should point upwards, we also fix the orientation to the identity matrix.

We have to set up two contraints in the following manner:

```
InverseKinematicsConstraintSet cs;
cs.AddPointConstraint (paddle_id,Vector3d(0,0,0),
   desiredPosition);
cs.AddOrientationConstraint (paddle_id, desiredOrientation);
```

Now the paddle should follow the ball exactly until the ball gets out of reach where it will fall off the paddle.

d) As we can see the work space is rather limited if we fix all three rotational DOFs of the paddle. It would be desirable to allow for a motion around the z-axis.

This can be achieved by replacing the orientation constraint by a second position constraint: We define a virtual point in local z-direction above the center of the paddle. In the local system of the paddle this could be for example $(0, 0, 1)^T$. This virtual point should be matched in form of a position constraint keeping the normal always pointing upwards. The range of motion should now increase.

Exercise 8.3 – More on Jacobians

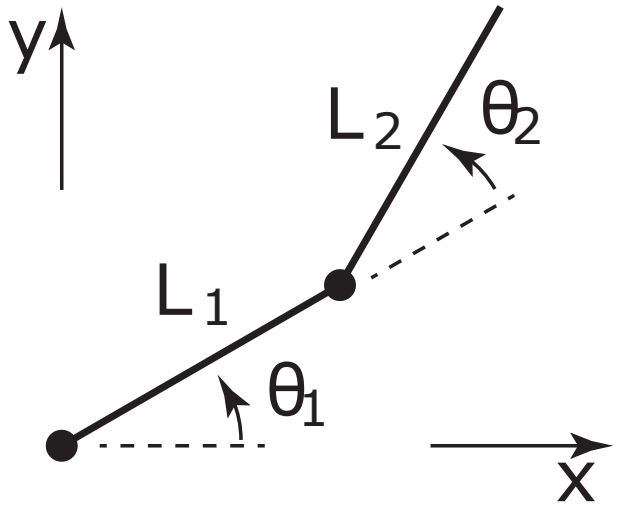

Figure 3: The 2R robot arm.

Consider the 2 segment arm given in figure 3. Let $L_1 = L_2 = 1$. From the lecture you know, that the Jacobian of this system is

$$J(\theta_1, \theta_2) = \begin{bmatrix} -\sin(\theta_1) - \sin(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) \\ \cos(\theta_1) + \cos(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) \end{bmatrix} \tag{3}$$

Assume the robot configuration $\theta_1 = 0°$, $\theta_2 = 90°$.

- Suppose we wish to make the end-effector move with a linear velocity $\dot{x} = 10$, $\dot{y} = 0$. What are the required input joint velocities $\dot{\theta}_1$ and $\dot{\theta}_2$?
- Suppose that the maximum allowable torque for each joint motor are

  $$\|\tau_1\|_2 \leq 10, \quad \|\tau_2\|_2 \leq 20.$$

  What is the maximum force that can be applied by the tip of the end-effector?
  Hint: The relation between joint-torques τ and forces at the end-effector γ_e at a static equilibrium is given by $\tau = J(q)^T \gamma_e$. (Derivation sketch: The power at the joints can be expressed in terms of the power at the end-effector as $\tau^T \dot{q} = \gamma_e^T v_e$. Substituting the identity $v_e = J(q)\dot{q}$, we obtain the given relation.)