SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.017 Graphics and Visualization

Assignment 5 – Linkage Design

Handout date: 2022.07.22

Submission deadline: 2022.08.01, 11:59 pm

Late submissions are not accepted

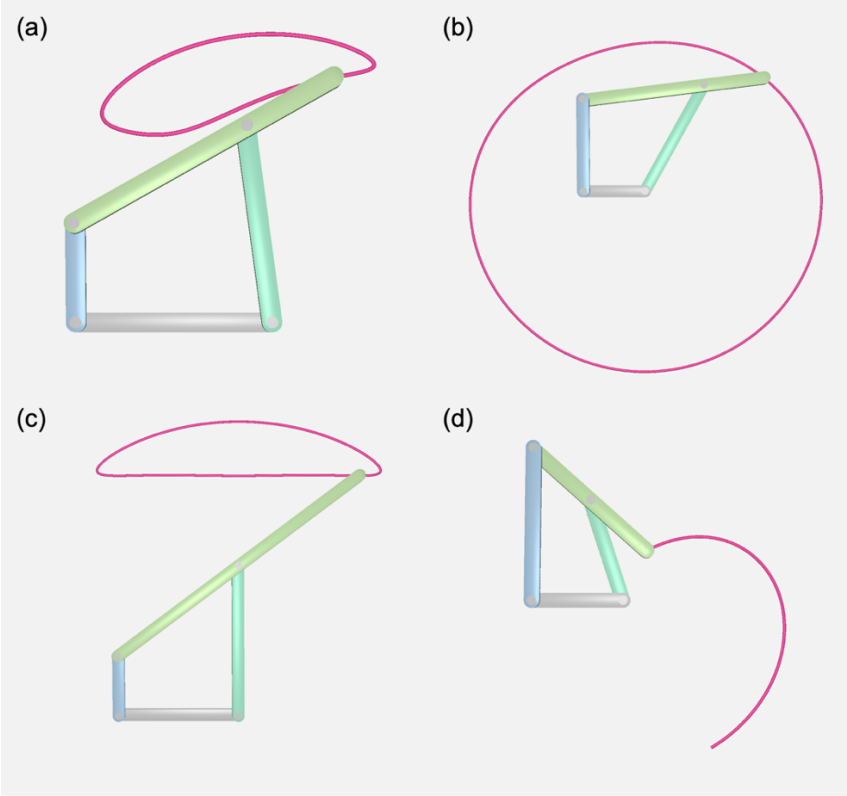


Figure 1. Expected program output of four different 4-bar linkages: (a) Crank-rocker; (b) Drag-link; (c) Hoecken; and (d) Double-rocker.

A linkage is an assembly of bodies connected to manage forces and movement. In this assignment, you will design some cool linkages. We'll be only looking at kinematics and ignore forces, and furthermore, we'll limit ourselves to 2D 4-bar linkage with revolute joints.

Provided with this assignment is a C++ framework for modeling and simulating linkages with the following functionalities:

- Press number key '1'-'4' to switch different types of linkage
    - Press KEY = '1' to show Crank-rocker linkage
    - Press KEY = '2' to show Drag-link linkage
    - Press KEY = '3' to show Hoecken linkage
    - Press KEY = '4' to show Double-rocker linkage
- Press 'space' key to start/stop the motion

- Press ‘a’ for slowing down the motion
- Press ‘d’ for speeding up the motion
- Press ‘c’ to show/hide the trajectory curve of the end-effector
- Left mouse drag to rotate the linkage
- Shift + left mouse drag to translate the linkage
- Scrolling mouse to scale the linkage

Please compile the code with ‘x64’ on Windows platform. Building the code in ‘release’ mode can improve running efficiency.

Link

A link is modeled as a rigid body. In 2D, a link has three degrees of freedom: $\theta \in R$ and $\boldsymbol{p} \in R^2$ (Figure 2(a)). θ specifies the orientation of the link with respect to the positive x-axis, and $\boldsymbol{p}$ specifies the position of the center of rotation. Given a point, $\boldsymbol{r}$, in the link’s local coordinates, the world coordinates of that point can be computed as

$$\boldsymbol{x} = R(\theta)\,\boldsymbol{r} + \boldsymbol{p}$$

where R is the rotation matrix given by

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

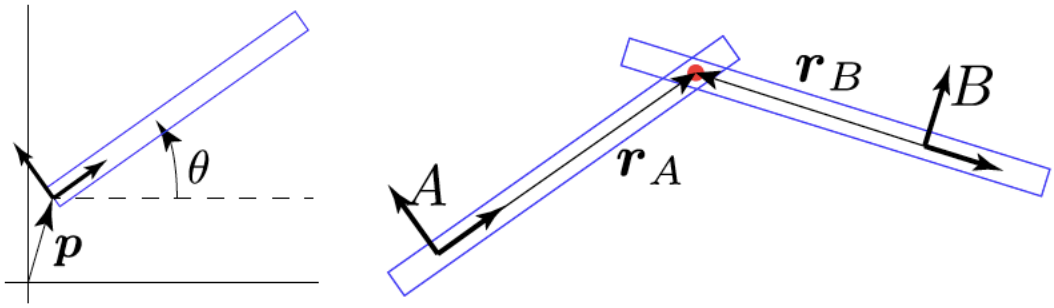


Figure 2. (a) A link defined by its orientation, θ, and position, $\boldsymbol{p}$. (b) A pin between two links. $\boldsymbol{r}_A$ and $\boldsymbol{r}_B$ define where the pin location is with respect to the links.

For example, the following code creates a horizontal link at the origin.

```
rotateAngle.push_back(0);
transPosition.emplace_back(0, 0);
```

The kinematics of a link is fully expressed using θ and $\boldsymbol{p}$. In order to draw it on screen, however, we need to give it a mesh. And push this mesh into visualization vector.

```
CreateLinkMesh(verL, norL, triL, length, width, thickness);
verListVec.push_back(verL);
norListVec.push_back(norL);
triListVec.push_back(triL);
```

Where ‘`length`’, ‘`width`’ and ‘`thickness`’ are geometry parameters of this link. Besides, we need set a depth in z-axis for this mesh to show on screen.

```
depth.push_back(0);
```

Additionally, we specify that one of the links is grounded, meaning that it cannot move, and another link is the driver, meaning that its orientation and position are specified procedurally. For example, `groundLinkID = 0`; `driverLinkID = 1`; specifies that link-0 is grounded, and link-1 is the driver. The number of links are 4: `linksNum = 4`.

Pin

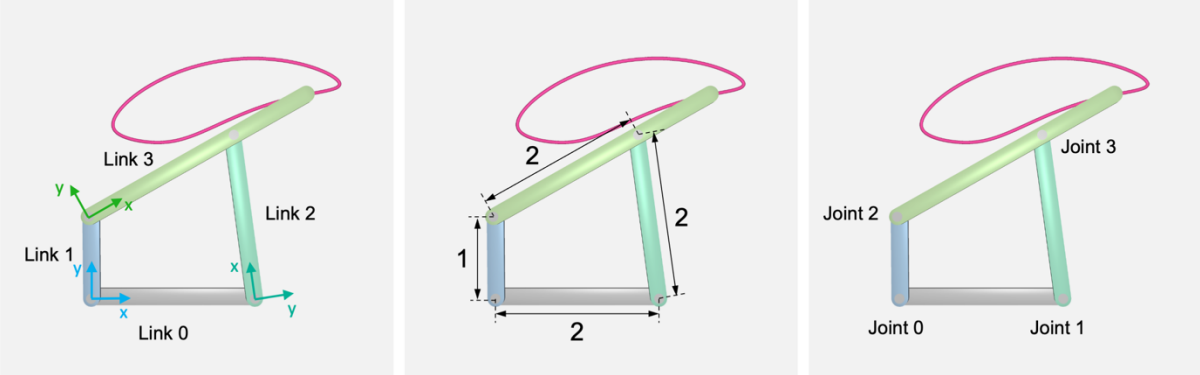
A pin constrains two links with a revolute joint. It stores the indices of the two links to constrain (say A and B) as well as where on the two links the pin constraint is located: $\boldsymbol{r}_A$ and $\boldsymbol{r}_B$ (Figure 2(b)). The constraint should then make sure that, if $\boldsymbol{r}_A$ and $\boldsymbol{r}_B$ are transformed to world space, their locations match.

The following code creates a pin between link-0 and link-1 as shown in Figure 2(b).

```
jointPairLinkID.emplace_back(0,1);
jointPairLocalPos1.emplace_back(4,0);
jointPairLocalPos2.emplace_back(-4,0);
```

The `jointPairLinkID` specifies that the pin is between links 0 and 1, and the `jointPairLocalPos1` and `jointPairLocalPos2` specifies that

$$\boldsymbol{r}_A = \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \boldsymbol{r}_B = \begin{pmatrix} -4 \\ 0 \end{pmatrix},$$



(a) (b) (c)

Figure 3. Crank-rocker linkage. (a) Link IDs and the body frame of each link. Note that that link 0 and link 1 share the same body frame (in blue). (b) Distance between joints. (c) Joint IDs.

Below is the code to specify the information of the four joints in the crank-rocker linkage. You should understand the meaning of the numbers in the code based on the above explanation of pin joint as well as illustrations in Figure 3.

```
// Joints
jointPairLinkID.emplace_back(0,1);
jointPairLocalPos1.emplace_back(0,0);
jointPairLocalPos2.emplace_back(0,0);

jointPairLinkID.emplace_back(0,2);
jointPairLocalPos1.emplace_back(2,0);
jointPairLocalPos2.emplace_back(0,0);

jointPairLinkID.emplace_back(1,3);
```

```
jointPairLocalPos1.emplace_back(1,0);
jointPairLocalPos2.emplace_back(0,0);

jointPairLinkID.emplace_back(2,3);
jointPairLocalPos1.emplace_back(2,0);
jointPairLocalPos2.emplace_back(2,0);
```

Particle

To accentuate the interesting motions of linkages, we can add a tracer particle (i.e., the end-effector point) to a link. The pink curve in Figure 1 shows the trajectory of the particle for a whole motion period of the linkage. The following code creates a particle.

```
particleLinkID = 3;
particleLinkPos = Vector3d (3, 0, 0);
```

The `particleLinkID` specifies the parent link, and the `particleLinkPos` specifies where on this parent link the particle is located (expressed in the parent link's coordinates).

Linkages

There are several types of 4-bar linkages. In this assignment, we focus on four types of 4-bar linkages:

1. Crank-rocker linkage. The driver motion trajectory is a circle. The output motion trajectory is an open or closed curve.
2. Drag-link linkage. The driver motion trajectory is a circle. The output motion trajectory is also a circle.
3. Hoecken linkage. This is a crank-rocker mechanism that produces a nearly straight line; see https://en.wikipedia.org/wiki/Hoecken_linkage and Figure 1(c).
4. Double-rocker linkage. The driver motion must be oscillatory, not rotational.

We have implemented the crank-rocker linkage in the provided code framework, and **your task is to initialize the other three linkages by providing the joint information in Linkages.cpp to achieve the expected results in Figure 1(b-d).**

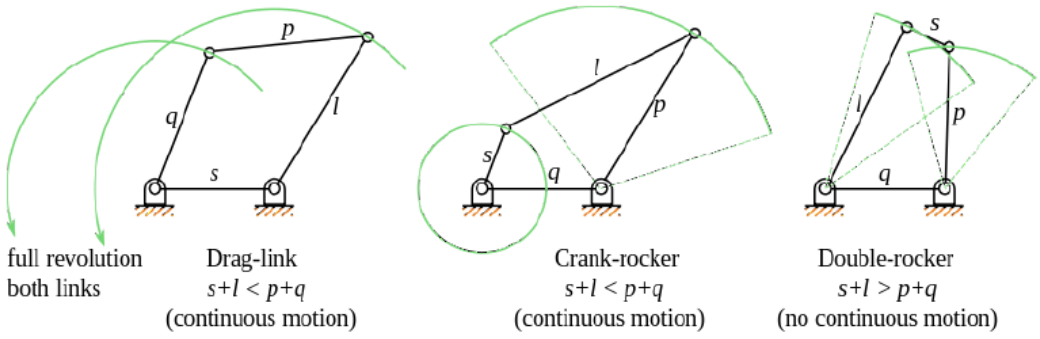


Figure 4. Four-bar linkages. s = smallest link length; l = longest link length; p, q = other two lengths.

Note that when creating linkages, the links do not need to be precisely positioned, since they will snap to configurations that satisfy all the constraints that you specify, as long as those constraints are satisfiable.

Simulation

After the links creation process finishes, we get links mesh, pins and the particle. The simulator then takes this information and does the following:

```
while simulating
    1. Procedurally set the driver angle
    2. Solve for linkage orientations and positions
    3. Update particle positions
    4. Draw scene
end
```

In Step 1, we manually specify the target angle of the driver link. As long as the mechanism has a single degree of freedom, the orientations and the positions of the rest of links can be solved for in Step 2 using nonlinear least squares. In Step 3, we compute the world positions of the particles given the newly updated link configurations. Finally, in Step 4, we draw the scene on the screen.

The meat of the simulator is in Step 2. The optimization variables in the nonlinear least squares problem are the orientations, θ, and positions, $\boldsymbol{p}$, of all the links, including grounded and driver links. We'll use $\boldsymbol{q}_i = [\theta_i, \boldsymbol{p}_i]^T$ to denote the configuration of the i-th link. We are looking for $\boldsymbol{q}_0, \boldsymbol{q}_1 \ldots, \boldsymbol{q}_n$ so that all of the constraints are satisfied. There are two types of constraints implemented so far: prescribed and pin. In both cases, we'll express the constraint in the form $c(\boldsymbol{q}) = 0$.

Prescribed: Grounded and driver links have their configurations manually specified. We can express this as a constraint on the orientation θ and the position $\boldsymbol{p}$ of the link.

$$c_\theta(\boldsymbol{q}) = \theta - \theta_{target}, \quad c_p(\boldsymbol{q}) = \boldsymbol{p} - \boldsymbol{p}_{target}.$$

For grounded links, the target orientation and position are fixed throughout the simulation, whereas for driver links, they are specified procedurally.

Constraints: A pin constrains two links so that they share a common point. Let the two links be denoted by A and B. We can express the constraint as

$$c_{pin}(\boldsymbol{q}; \boldsymbol{r}) = \boldsymbol{x}_A - \boldsymbol{x}_B = (R(\theta_A)\boldsymbol{r}_A + \boldsymbol{p}_A) - (R(\theta_B)\boldsymbol{r}_B + \boldsymbol{p}_B).$$

We are looking for orientations and positions of the two links such that this constraint function evaluates to zero. The orientations and positions of the two links, θ and $\boldsymbol{p}$, are the variables of this constraint function, whereas the locations of the pin joints expressed in local coordinates, $\boldsymbol{r}_A$ and $\boldsymbol{r}_B$, are the parameters of this constraint function.

Let c be the concatenation of all the constraints. We're looking for the orientations and positions of all the joints (θ_i and $\boldsymbol{p}_i$) such that the constraint violations are minimized:

$$\min_q \frac{1}{2} c^T c.$$

We can solve this using the Levenberg-Marquardt solver; see LMsolver.cpp.

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Grading

Each part of this assignment is weighted as follows:

- Drag-link mechanism: 33%
- Hoecken linkage: 33%
- Double-rocker mechanism: 34%

Submission

A .zip compressed file renamed to AssignmentN_Name_I.zip, where N is the number of the current assignment, Name is your first name, and I is the number of your student ID. It should contain only:

- The **source code** project folder (the entire thing).
- A **readme.txt file** containing a description of how you solved each part of the exercise (use the same numbers and titles) and whatever problems you encountered. If you know there are bugs in your code, please provide a list of them, and describe what do you think caused it if possible. This is very important as we can give you partial credit if you help us understand your code better.
- A couple of **screenshots** clearly showing that you can design and simulate different types of linkages mechanism.

Upload your zipped assignment to e-dimension. Late submissions receive 0 points!