Computational Aspects of Fabrication, Spring 2015

Assignment 4: Linkage Design

Due April 22nd at 11:59pm.

For this assignment you will design some cool linkages, one of which will be physically manufactured. Provided with this assignment is a fully functional Matlab code for simulating planar linkages with revolute joints.

In this assignment you will be responsible for:

1. Modifying the code to allow oscillatory driver input (instead of rotational)
2. Designing several linkages
3. Fabricating one of your designs

The remainder of this document is organized as follows:

1. Getting Started
2. Starter Code and Implementation Notes
3. Your Tasks
4. Submission Instructions

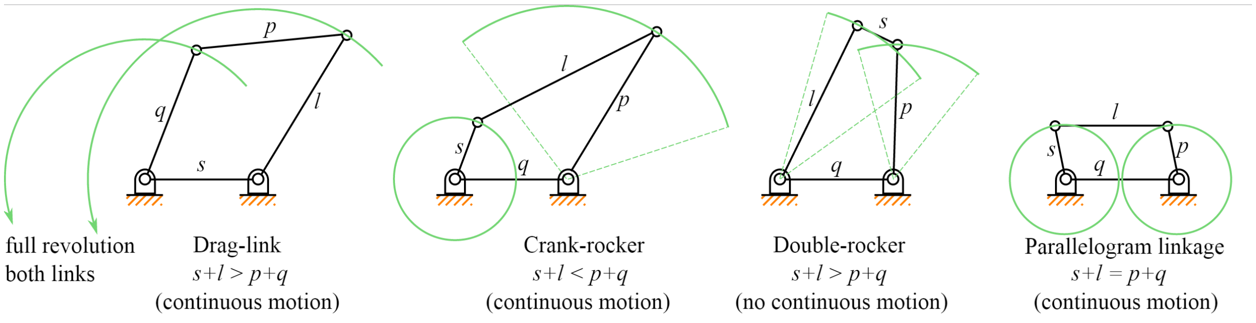


Figure 1: Four-bar linkages [wikipedia]. s = smallest link length; l = longest link length; p, q = other two lengths.

1 Getting Started

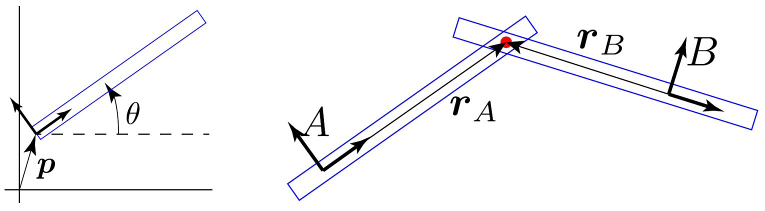
1.1 Running the Code

The provided code is standalone and does not require any external libraries. Here we outline how to run the code from Matlab's command window.

1. Extract `linkage.m` to `<WORKINGDIR>`.
2. Open Matlab and go to the command window.
3. `>> cd <WORKINGDIR>`
4. `>> linkage(0)`

Matlab will then open a figure window that shows the simulation of the sample linkage. The pin constraint between the right and top links is randomized, so if you run the code multiple times, you'll get a slightly different mechanism each time. The argument "`0`" specifies the scene to run, which in this case is a crank-rocker mechanism. Each new linkage mechanism that you create should have a new scene number.

2 Starter Code and Implementation Notes



(a) Link **(b)** Pin

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.

2.1 How to Create Scenes

To create a scene, we need a list of links and a list of constraints between the links. This is done inside the `switch` statement at the top of the code. The sample four-bar linkage scene is defined in `case 0`.

Link: A link is modeled as a rigid body, and in 2D, it has three degrees of freedom: $\theta \in \mathbb{R}$ and $\boldsymbol{p} \in \mathbb{R}^2$ (Fig. 2a). 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

$$\boldsymbol{x} = R(\theta)\,\boldsymbol{r} + \boldsymbol{p}, \tag{1}$$

where R is the rotation matrix given by

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}. \tag{2}$$

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

```
links(1).angle = 0;
links(1).pos = [0;0];
links(1).verts = [0,-0.1;2,-0.1;2,0.1;0,0.1]';
```

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. In this assignment, we simply assign to each link a list of vertices expressed in the link's local coordinates and transform these vertices into world coordinates whenever we move the link. Note that the mesh is for display only—the simulation would work just fine even if we don't have a mesh. In the example above, the vertices are

$$\begin{pmatrix} 0 \\ -0.1 \end{pmatrix}, \quad \begin{pmatrix} 2 \\ -0.1 \end{pmatrix}, \quad \begin{pmatrix} 2 \\ 0.1 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ 0.1 \end{pmatrix},$$

which defines a rectangle with the link's coordinate frame defined at the left end, as in Fig. 2a. These vectors are stored column wise in the `links(1).verts` matrix.

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, `grounded = 1; driver = 2;` specifies that `links(1)` is grounded, and `links(2)` is the driver.

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 $\boldsymbol{r}_B$ (Fig. 2b). The constraint should then make sure that, if $\boldsymbol{r}_A$ and $\boldsymbol{r}_B$ are transformed to world space (using Eq. 1), their locations match. (If you're curious how this is actually implemented, see Sec. 2.2.)

The following code creates a pin between `links(1)` and `links(2)` as shown in Fig. 2b.

```
pins(1).links = [1,2];
pins(1).pts = [4,0;-4,0]';
```

The `links` field specifies that the pin is between links 1 and 2, and the `pts` field specifies that

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

Particle: To accentuate the interesting motions of linkages, we can add tracer particles to the links (like the green lines in Fig. 1.) The following code creates a particle.

```
particles(1).link = 4;
particles(1).pt = [0.5;0.1];
```

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

Hint: When creating scenes, 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.

2.2 How the Simulation Works (optional reading)

After the scene creation process finishes, we have a list of links, pins, and particles. 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 $q_i = [\theta_i, \boldsymbol{p}_i]^T$ to denote the configuration of the i^{th} link. We are looking for $q_1, \ldots, q_n$ 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 $c(q) = 0$.

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.

$$c_\theta(q) = \theta - \theta_{\text{target}}, \quad c_p(q) = \boldsymbol{p} - \boldsymbol{p}_{\text{target}}. \tag{3}$$

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

Pin: 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

$$\begin{aligned} c_{\text{pin}}(q; \boldsymbol{r}) &= \boldsymbol{x}_A - \boldsymbol{x}_B \\ &= (R(\theta_A)\,\boldsymbol{r}_A + \boldsymbol{p}_A) - (R(\theta_B)\,\boldsymbol{r}_B + \boldsymbol{p}_B). \end{aligned} \tag{4}$$

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:

$$\underset{q}{\text{minimize}} \quad \frac{1}{2} c^T c. \tag{5}$$

We can solve this easily in Matlab using the function `lsqnonlin`. What we need to provide is a function that evaluates the vector-valued function c and its Jacobian, $J = \partial c / \partial q$. For grounded and driver constraints, the Jacobian is just the identity: $J_\theta = 1, J_p = I$. For pin constraints, the Jacobian is

$$\begin{aligned} J &= \begin{pmatrix} \partial c/\partial\theta_A & \partial c/\partial\boldsymbol{p}_A & \partial c/\partial\theta_B & \partial c/\partial\boldsymbol{p}_B \end{pmatrix} \\ &= \begin{pmatrix} R'(\theta_A)\,\boldsymbol{r}_A & I & -R'(\theta_B)\,\boldsymbol{r}_B & -I \end{pmatrix} \end{aligned} \tag{6}$$

where

$$R'(\theta) = \begin{pmatrix} -\sin\theta & -\cos\theta \\ \cos\theta & -\sin\theta \end{pmatrix}. \tag{7}$$

3 Your Tasks

1. The first task is to modify the driver so that it can also generate oscillatory motion in addition to rotational motion. Currently, the piece of code that produces rotational motion is

```
...
dt = 0.01;
angVel = 2*pi;
while t < T
  links(driver).angleTarget = links(driver).angleTarget + dt*angVel;
  ...
end
```

   This integrates the target angle with a constant angular velocity (2π radians per second), so that the driver rotates at a constant rate. Your task here is to modify this code so that you can specify a range that the driver oscillates between. For example, if the range is $[-\pi/3, \pi]$, then the driver should go back and forth between $-\pi/3$ and π at some set frequency. You may not need `angVel` any more, since the angular velocity will no longer be constant.

2. Design a family of linkages. The sample code (`linkage(0)`) produces a crank-rocker mechanism, since the shortest link, s, is the crank (driver). Starting from this sample code, you will create four new linkages as detailed also in the matlab source code file: a Chebyshev linkage, a Peaucellier-Lipkin linkage, a Klann linkage, and the linkage used for one of the legs of this animated LEGO horse: `https://www.youtube.com/watch?v=TN1qxbxKAAw`.

3. Fabricate one of the linkage structures using 3D Printing or laser cutting. Please go to `http://ideate.andrew.cmu.edu/process/laser/rabbit/` for information on using the laser cutters if this is the fabrication method you choose.

4 Submission Instructions

Please provide a report with your submission (PDF). The report should include images of all the mechanisms that you designed. Document also the physical prototypes. How close did your design get to the target motion, especially for the last linkage? You should also hand in the matlab code for the new linkage designs.