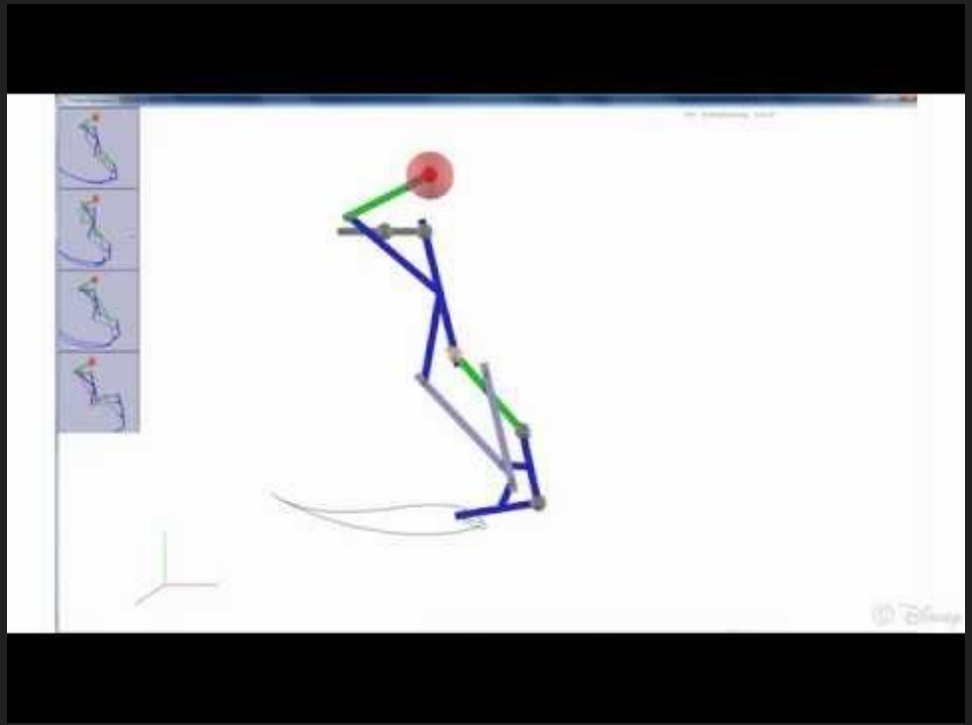
Kinematics of Mechanisms

Tutorial A2


MECHANICAL MARVELS:
CLOCKWORK
DREAMS

Linkages



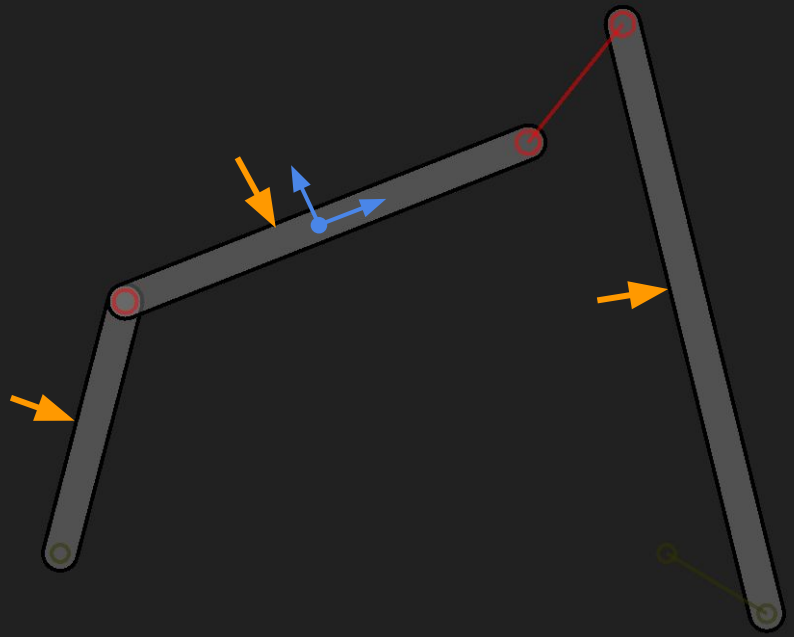
Linkages - Rigid bodies

Rigid bodies. We will model the links as rigid bodies. The position $x_{p,i}$ and rotation angle θ_i define the state $x_i = \{x_b, \theta_i\}$ of the rigid body i.

The world coordinates, p_w, of a point p_l, that is in local coordinates of a rigid body i, can be thus computed as:

$$p_w(x_i) = x_{p,i} + R(\theta_i)p_l \qquad (7)$$

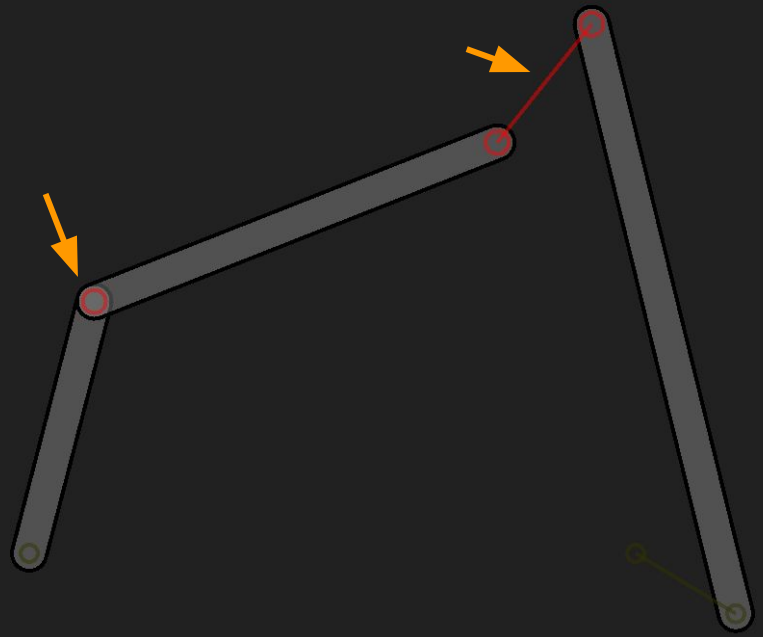
where $R(\theta)$ is the rotation matrix corresponding to the rotation angle θ.



Linkages - Joints

Hinge joints. A hinge joint connects two rigid bodies i and j at attachement locations p_i and p_j, which are in local coordinates of the corresponding rigid bodies, i and j. For a hinge joint to be valid, we want the world coordinates of both attachement points to be equal:

$$c_h(x_i, x_j) = p_w(x_i) - p_w(x_j) = 0 \qquad (1)$$



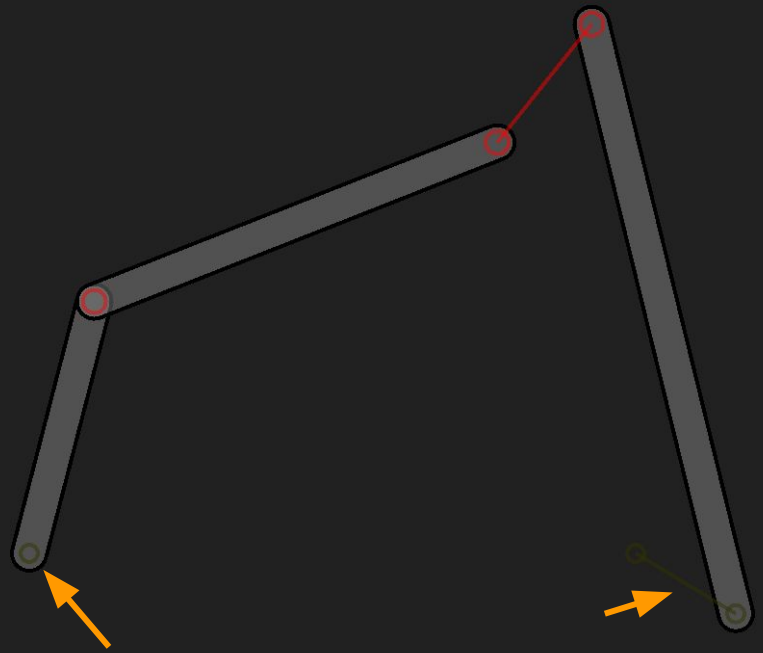
Linkages - Joints

Fixed joints. Fixed joints ensure that a point p_l in local coordinates of a rigid body i has the world coordinates of the point p_f:

$$c_f(x_i) = p_f - p_w(x_i) \tag{2}$$

Fixed angle joints force the relative angle of two rigid bodies i and j to be at a target angle θ_t:

$$c_a(x_i) = \theta_i - \theta_t \tag{3}$$



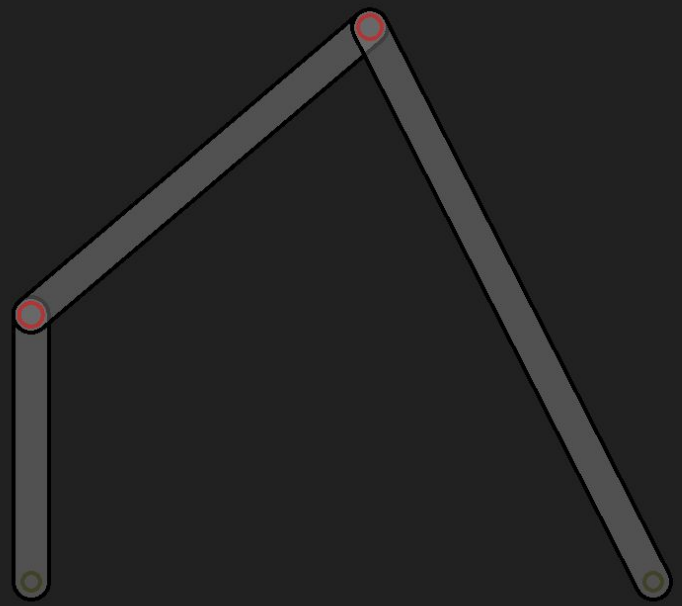
Linkages - Simulation

$$x = \text{argmin}_{\tilde{x}} \frac{1}{2} c(\tilde{x})^T c(\tilde{x}) = \text{argmin}_{\tilde{x}} f(\tilde{x})$$

Vector with states of all rigid bodies

Vector with constraints of all joints

Solve using a numerical optimization method (e.g. Newton's method)



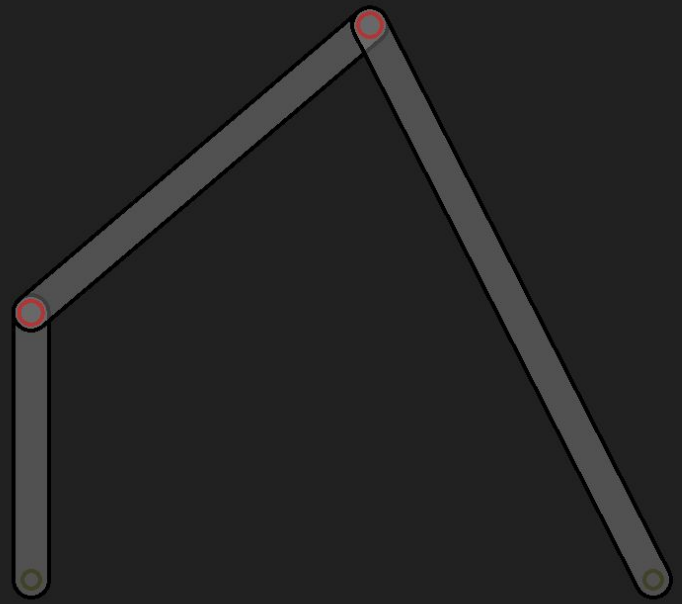
Linkages - Gauss-Newton approximation

Need to compute derivatives:

$$\begin{aligned} \nabla f(x) &= J(x)^T c(x) \\ \nabla^2 f(x) &= J(x)^T J(x) + \sum_{i=0}^{m-1} c_i(x) \nabla^2 c_i(x) \end{aligned} \tag{6}$$

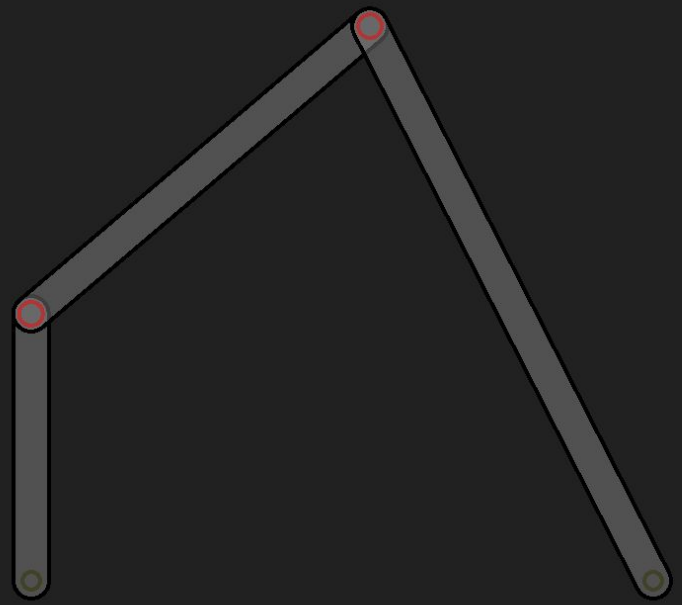
Jacobian: dC / dx

Very small if close to constraint satisfaction



Linkages - Jacobian

Let’s look at it a bit closer ...

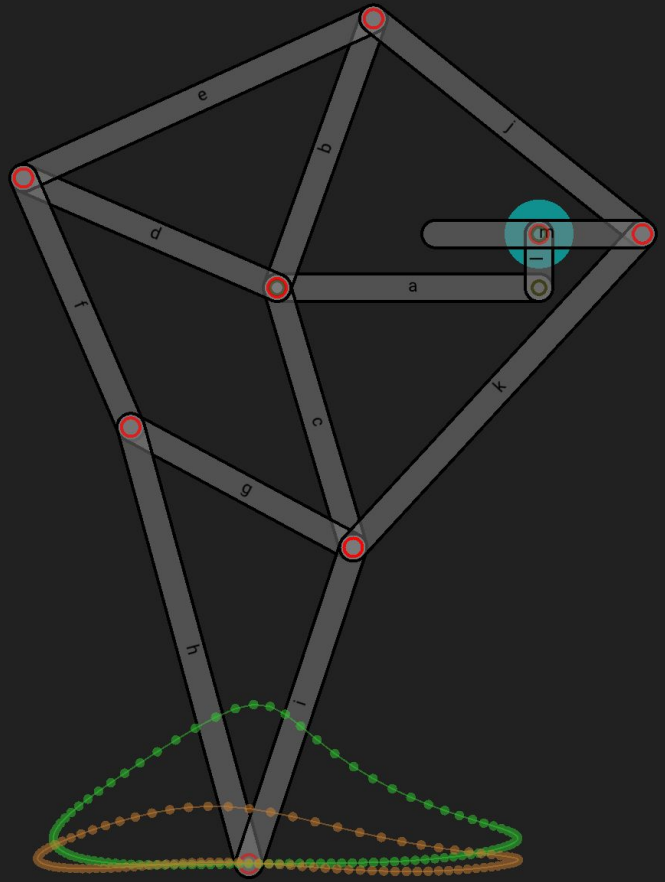


Design optimization of linkages

Goal: Find dimensions of all links/rigid bodies (design parameters), such that the end-effector trajectory matches a target trajectory.

Forward design: User changes link lengths, and sees effect on end-effector trajectory immediately.

Design optimization: Algorithm finds optimal design parameters.



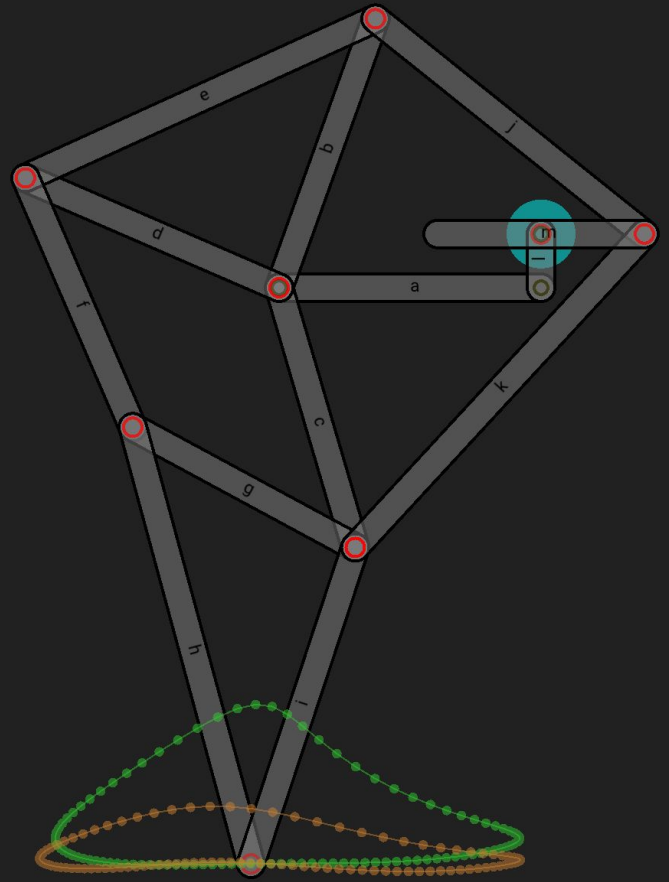
Design optimization of linkages

Trajectory given design parameters ***p***

Target trajectory

$$f_t(p) = \min_{\tilde{p}} ||s(p) - s_t||^2$$

In this assignment:
solve using random search!



Code - Rigid bodies

Rigid bodies. We will model the links as rigid bodies. The position $x_{p,i}$ and rotation angle θ_i define the state $x_i = \{x_b, \theta_i\}$ of the rigid body i.

The world coordinates, p_w, of a point p_l, that is in local coordinates of a rigid body i, can be thus computed as:

$$p_w(x_i) = x_{p,i} + R(\theta_i)p_l \qquad (7)$$

where $R(\theta)$ is the rotation matrix corresponding to the rotation angle θ.

```
class RigidBody {
public:
    RigidBody(int dofIdx, double length, std::string name = "", double
dofIdx(dofIdx), length(length), width(width), name(name) { ...}

    // Returns the world coordinates of a local point `p` given the glo
    // `x` contains the states of all rigid bodies. The member `dofIdx`
    // at which index in `x` the DOFs for this rigid body start.
    Vector2d pWorld(const VectorXd &x, const Vector2d &p) const { ...

    // Returns the Jacobian `dp_world / d_xi` given the global state `x
    // local point `p`.
    Matrix<double, 2, 3> dpWorld_dx(const VectorXd &x, const Vector2d &

    // Returns the position of this rigid body given the global state `
    inline const Vector2d pos(const VectorXd &x) const { ...}

    // Returns the rotation angle of this rigid body given the global s
    inline const double &theta(const VectorXd &x) const { ...}
    // Returns the rotation angle of this rigid body given the global s
    inline double &theta(VectorXd &x) const { ...}

public:
    int dofIdx;
    double length, width;
    std::string name;
};
```

Linkages - Joints

Hinge joints. A hinge joint connects two rigid bodies i and j at attachement locations p_i and p_j, which are in local coordinates of the corresponding rigid bodies, i and j. For a hinge joint to be valid, we want the world coordinates of both attachement points to be equal:

$$c_h(x_i, x_j) = p_w(x_i) - p_w(x_j) = 0 \qquad (1)$$

```
class HingeJoint {
public:

    HingeJoint(int rb0Idx, int rb1Idx, Vector2d
        : rbIdx({rb0Idx, rb1Idx}), local({local0

    VectorXd computeConstraints(const VectorXd &

    MatrixXd computeJacobian(const VectorXd x, c

public:
    std::array<int, 2> rbIdx;
    std::array<Vector2d, 2> local;
};
```

Linkages - Joints

Fixed joints. Fixed joints ensure that a point p_l in local coordinates of a rigid body i has the world coordinates of the point p_f:

$$c_f(x_i) = p_f - p_w(x_i) \tag{2}$$

Fixed angle joints force the relative angle of two rigid bodies i and j to be at a target angle θ_t:

$$c_a(x_i) = \theta_i - \theta_t \tag{3}$$

```
class FixedJoint
{
public:
    VectorXd computeConstraints(const VectorXd &x, const Rigi

    MatrixXd computeJacobian(const VectorXd x, const RigidBod

public:
    Vector2d pos;       // world coordinates
    int rbIdx;          // index of rigid body
    Vector2d localPos;  // position in rigid body coordinates
};

class FixedAngleJoint
{
public:
    VectorXd computeConstraints(const VectorXd &x, const Rigi

    MatrixXd computeJacobian(const VectorXd x, const RigidBod

public:
    int rbIdx;
    mutable double angle;
};
```

Linkages - Simulation

$$x = \text{argmin}_{\tilde{x}} \frac{1}{2} c(\tilde{x})^T c(\tilde{x}) = \text{argmin}_{\tilde{x}} f(\tilde{x})$$

Vector with states of all rigid bodies

Vector with constraints of all joints

Solve using a numerical optimization method (e.g. Newton's method)

```
class KinematicEnergy : public ObjectiveFunction    ⚠'KinematicEn
{
public:
    virtual double evaluate(const VectorXd& x) const { ...}

    virtual void addGradientTo(const VectorXd& x, VectorXd& grad)

    virtual void addHessianEntriesTo(const VectorXd& x, std::vect
const { ...}

    VectorXd computeConstraints(const VectorXd &x) const { ...}

    MatrixXd computeJacobian(const VectorXd &x) const { ...}

    int getNumConstraints() const { ...}

public:
    std::vector<RigidBody> rigidbodies;
    std::vector<HingeJoint> hingeJoints;
    std::vector<FixedJoint> fixed;
    std::vector<FixedAngleJoint> fixedAngle;
    bool fixedAngleEnabled = false;
};
```

The Eigen library

Documentation: http://eigen.tuxfamily.org/dox/

```
Matrix<double, N, M> A;                    // matrix of size NxM
Matrix<double, -1, -1> B(N, M);            // dynamically sized matrix
double y = A(i,j);                         // element at (i,j)
auto C = A.block<n, m>(p, q);              // submatrix at A(p,q), size NxM
B.block<n,m>(i,j) = A.block<n, m>(p, q);// can also write to block-matrix!
A.setZero();                               // sets all elements to zero
auto D = A.transpose() * B;                // operator overloading
                                           // for common operations
```

→ Code Review

starter code:
github.com/computational-robotics-lab/comp-fab-a2
post issues there!

Questions

- Questions about assignments on corresponding issues page: https://github.com/computational-robotics-lab/comp-fab-a2/issues

- Other questions: in your personal repo, or moritzge@inf.ethz.ch