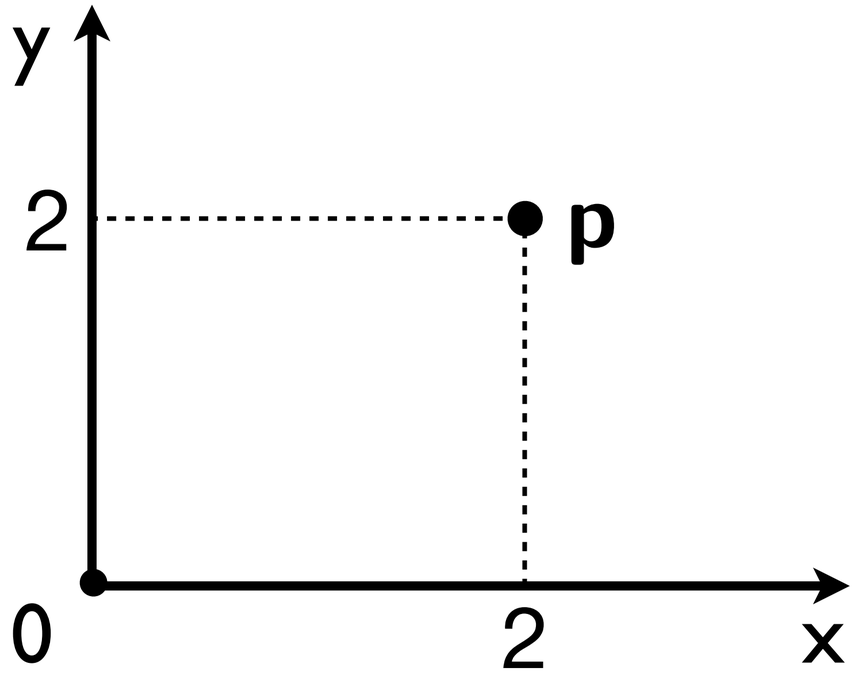
Transforming Frames

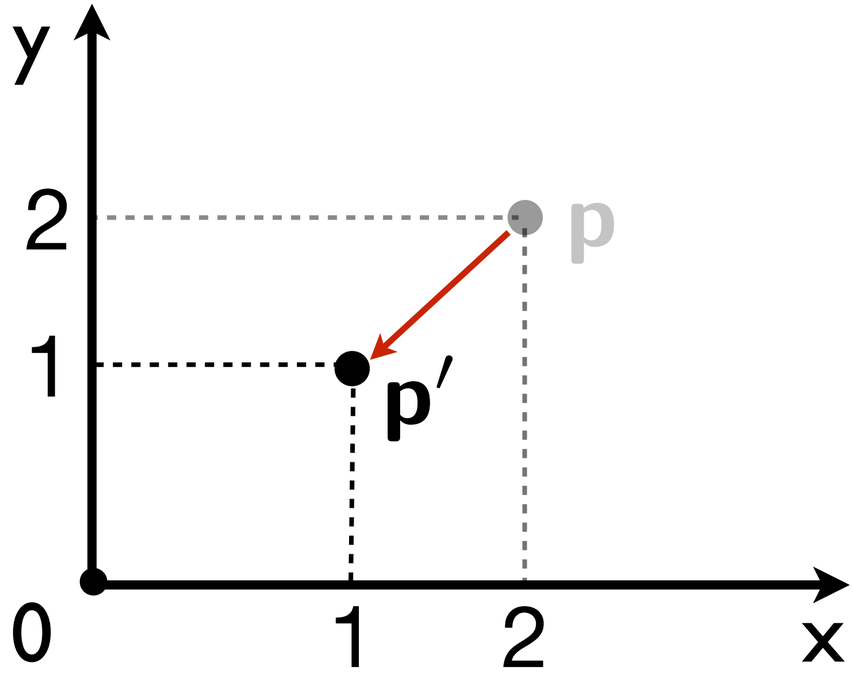
Representing points in different coordinate systems

Coordinate Transformations

- Previous transformations moved points around.
- But, we can also think of transformations as changing the coordinate systems in which points are represented.

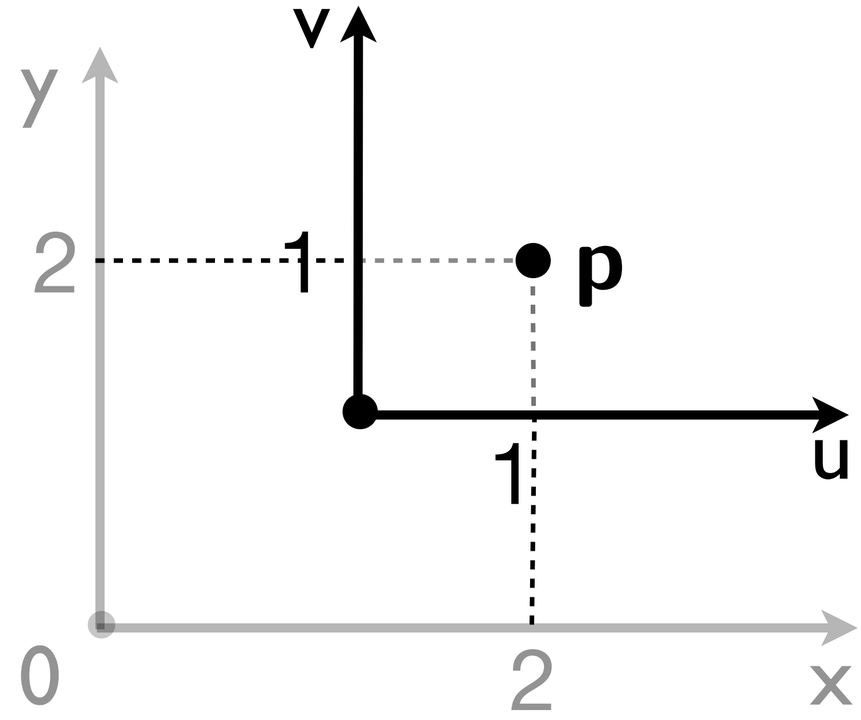


Point **p** in the x-y coordinate system has coordinates $(2,2)^T$.



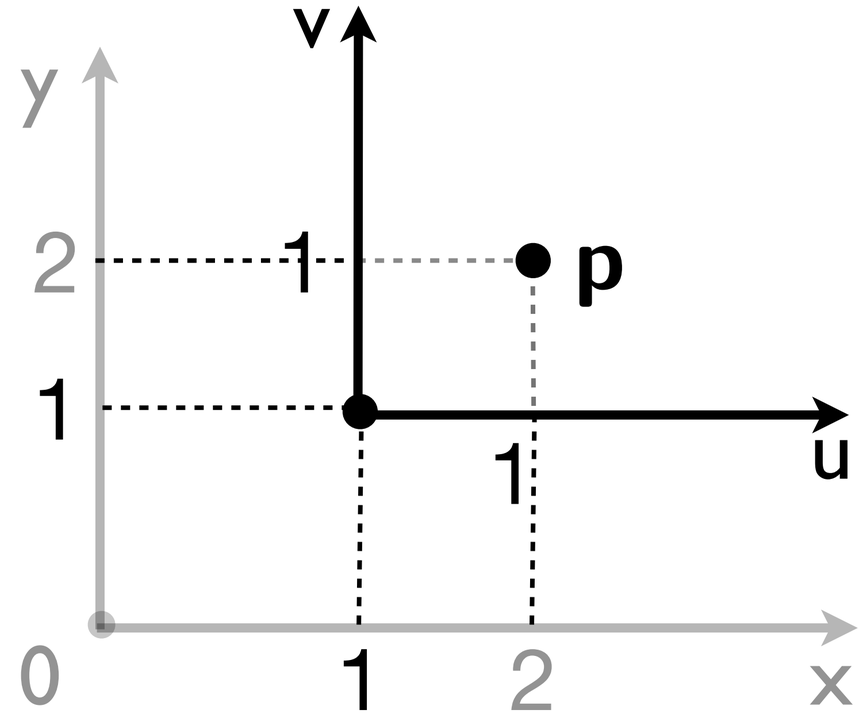
Point **p’** is **p** translated by a vector **t** = (-1,-1).

The coordinates of **p**’ in the x-y coordinate system is $(1,1)^T$.



The same coordinates, i.e., $(1,1)^T$ can also be found if we translate the x-y coordinate system by $-\mathbf{t} = (1,1)$ while keeping **p** fixed.

Note that the transformation changing the representation of **p** from the x-y coordinate system to the u-v coordinate system is the inverse of the transformation performed under the original coordinate system.



The same coordinates, i.e., $(1,1)^T$ can also be found if we translate the x-y coordinate system by **-t** = (1,1) while keeping **p** fixed.

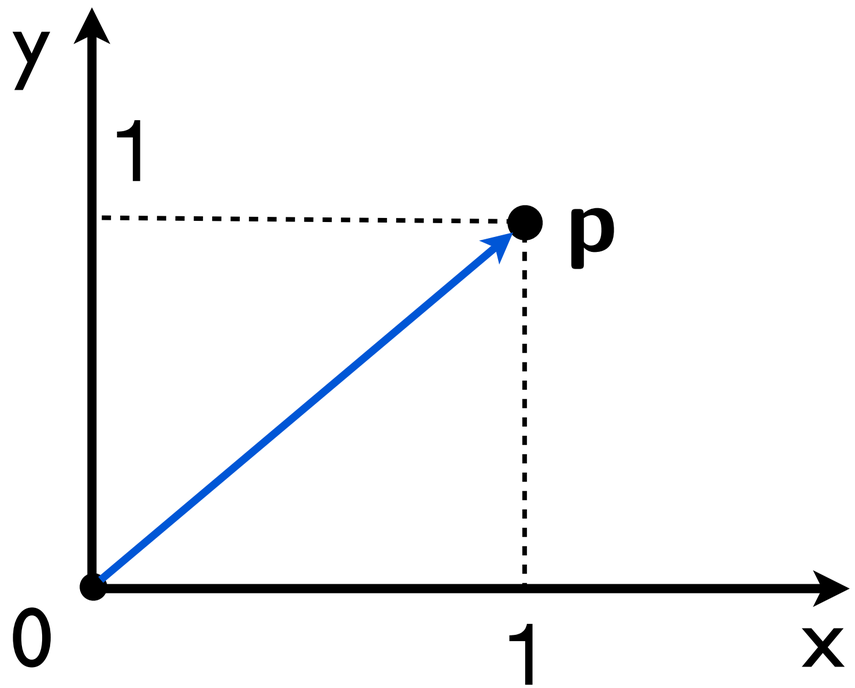
$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

$\mathbf{p}_{uv}$ $\mathbf{p}_{xy}$

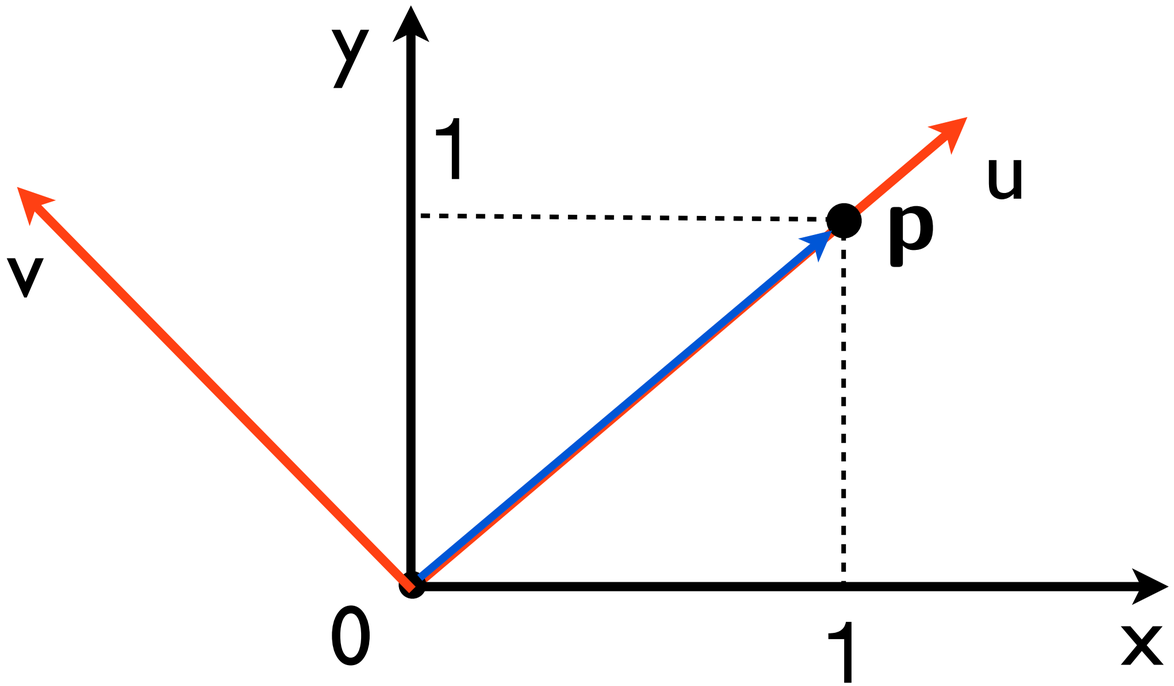
$$\begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

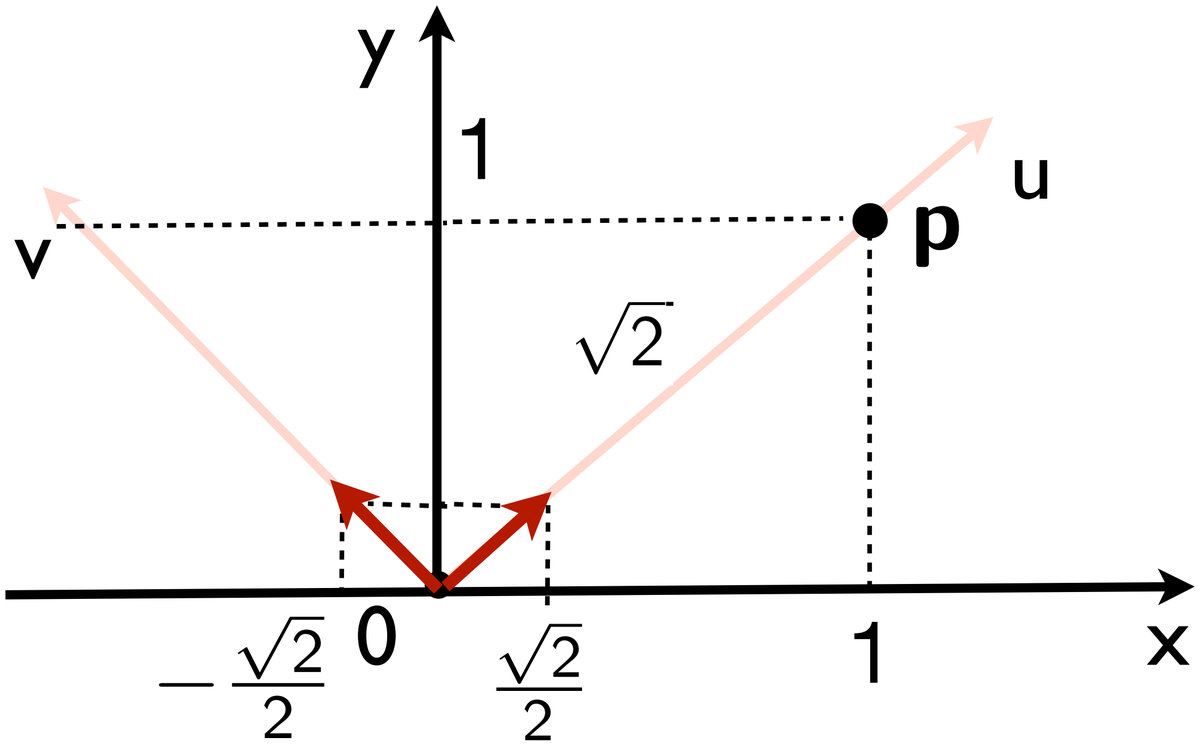
$\mathbf{p}_{xy}$ $\mathbf{p}_{uv}$

Consider the point $\mathbf{p} = (1,1)^T$ represented in the x-y coordinate system.



Point **p** can also be represented with respect to the u-v coordinate system. In this case **p** = $(\sqrt{2},0)^T$

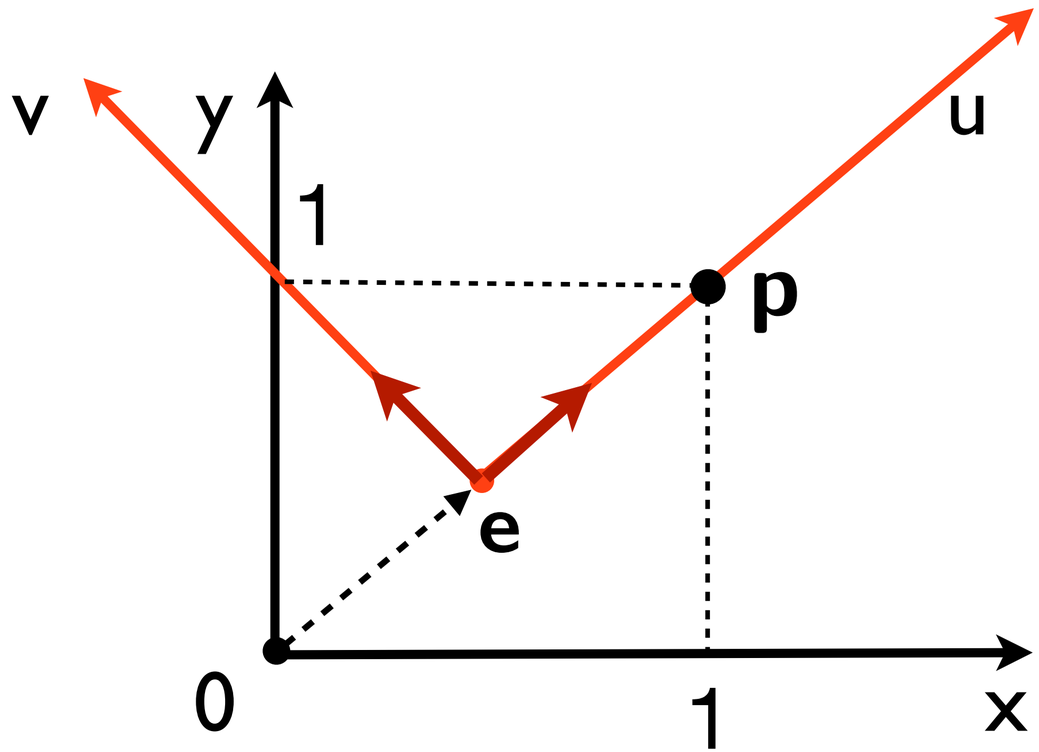




Basis vectors **u** and **v** have the following representations in the canonical (i.e., x-y) coordinate system):

$$\mathbf{u} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$$

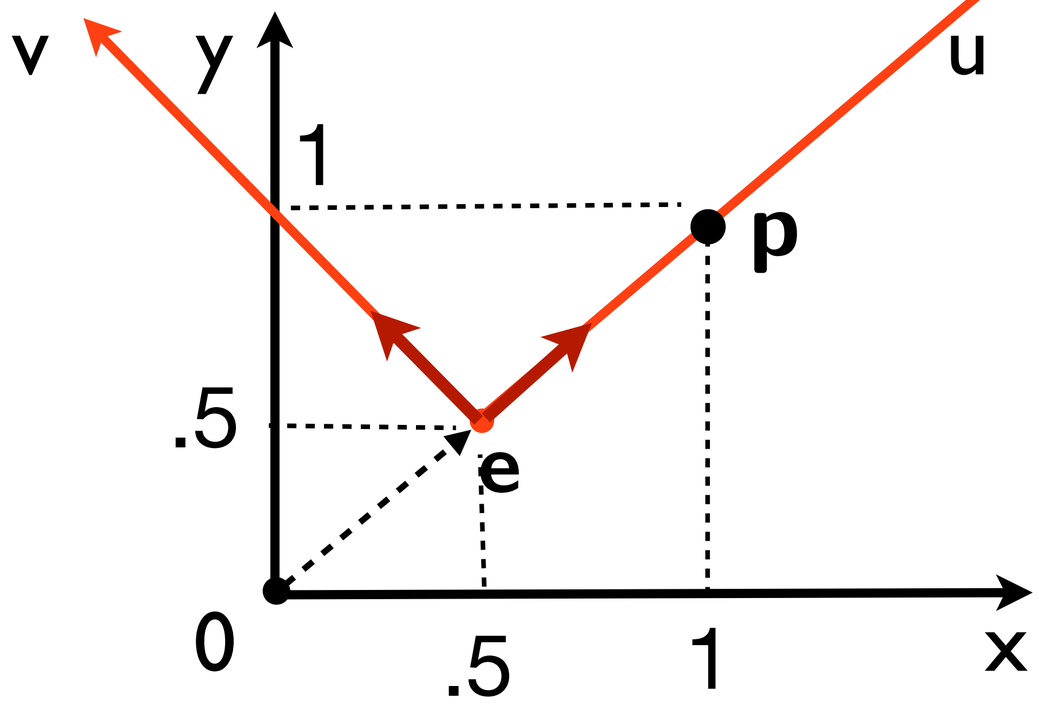
Let’s consider the case where the u-v coordinate system is at some point **e** different from **0.**



In homogeneous coordinates:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

Let's consider the case where the u-v coordinate system is at some point **e** different from **0.**



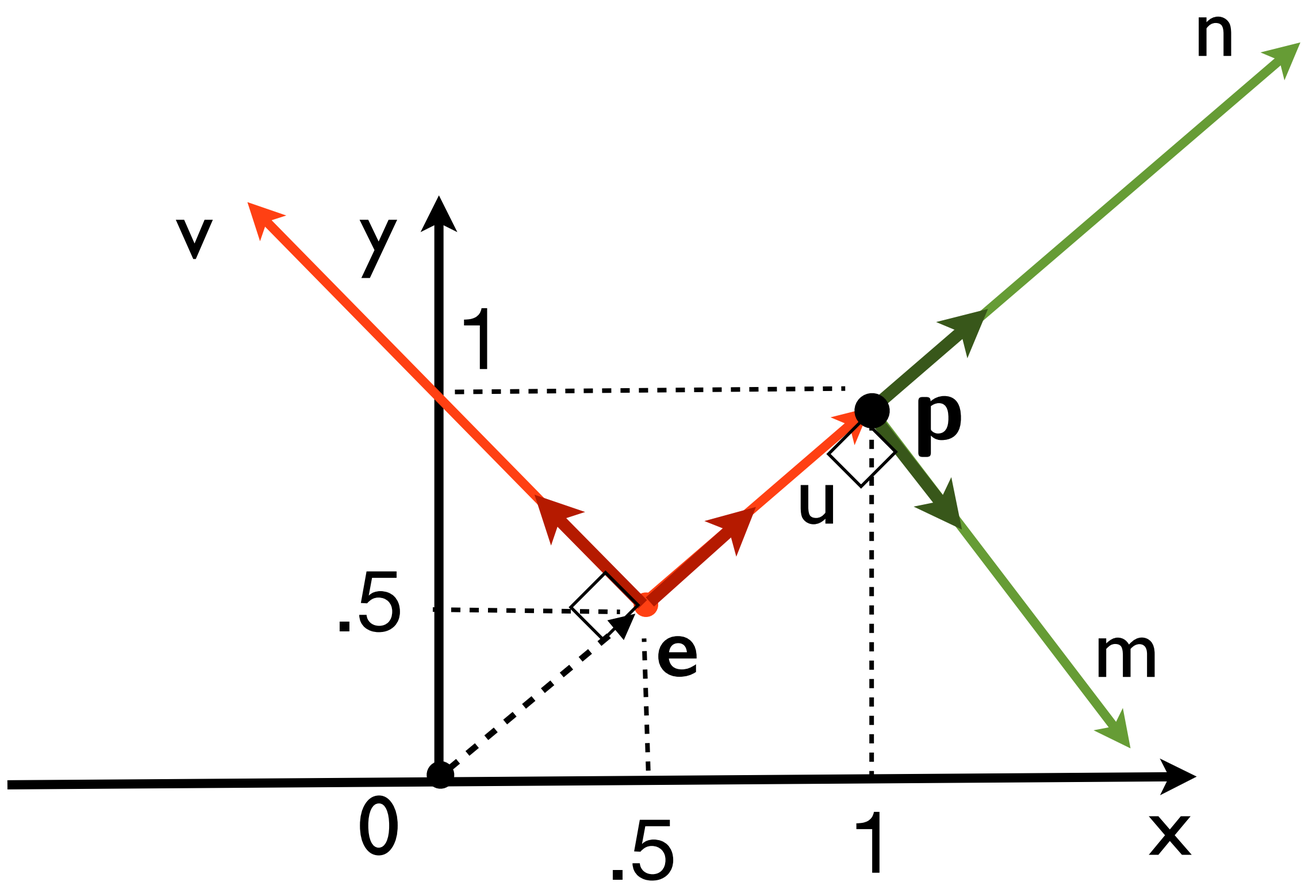
In homogeneous coordinates:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

Numerical example:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0.5 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

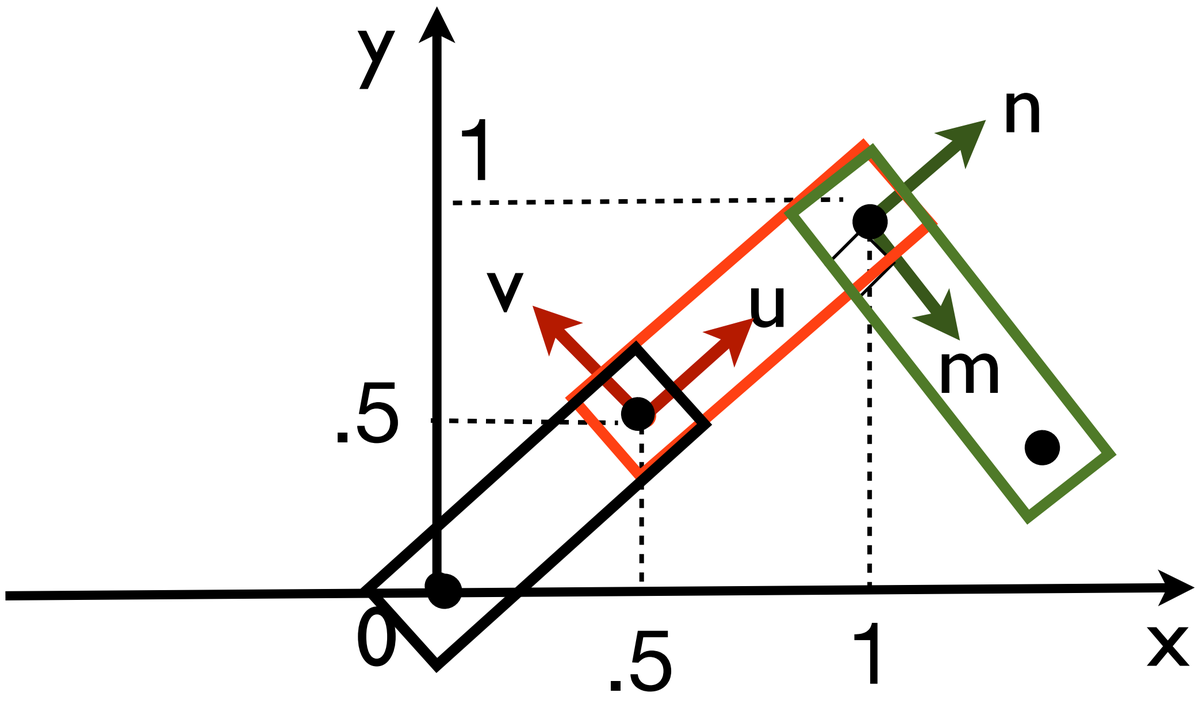
Exercise 1: convert from (m,n) to (u,v) and from (u,v) to (x,y).



$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

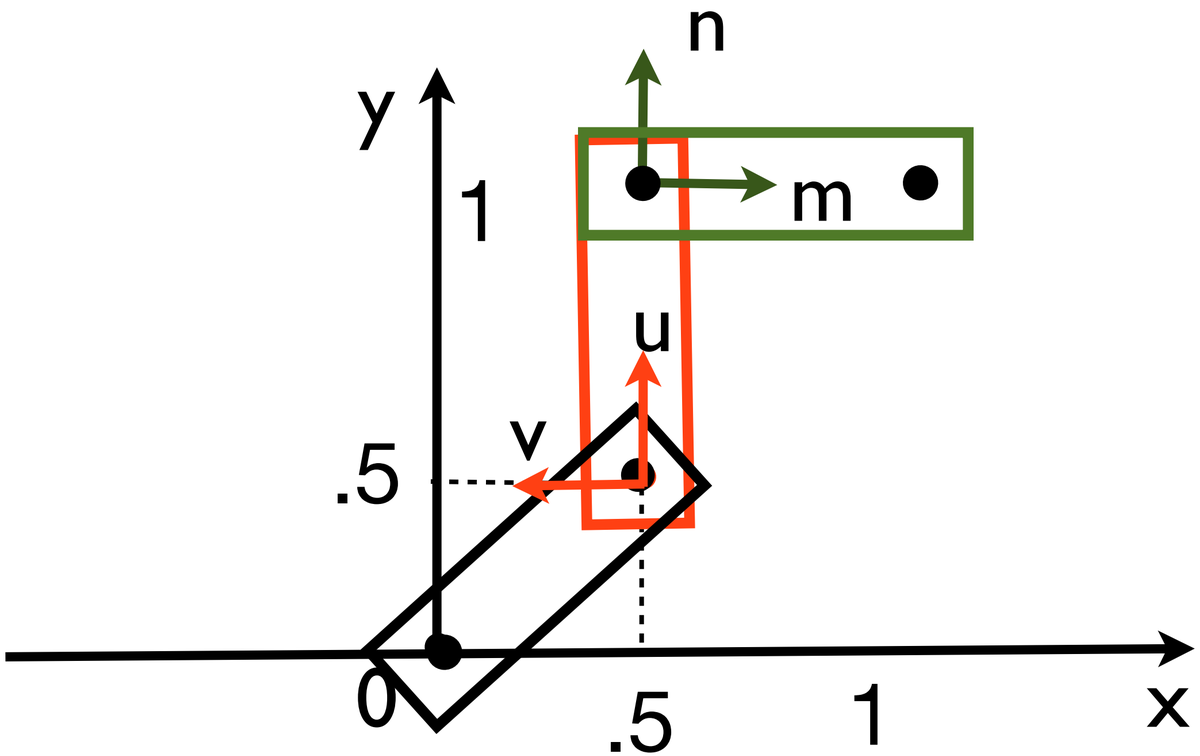
Exercise 2: show the sequence of transformations needed to draw the kinematic chain.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$



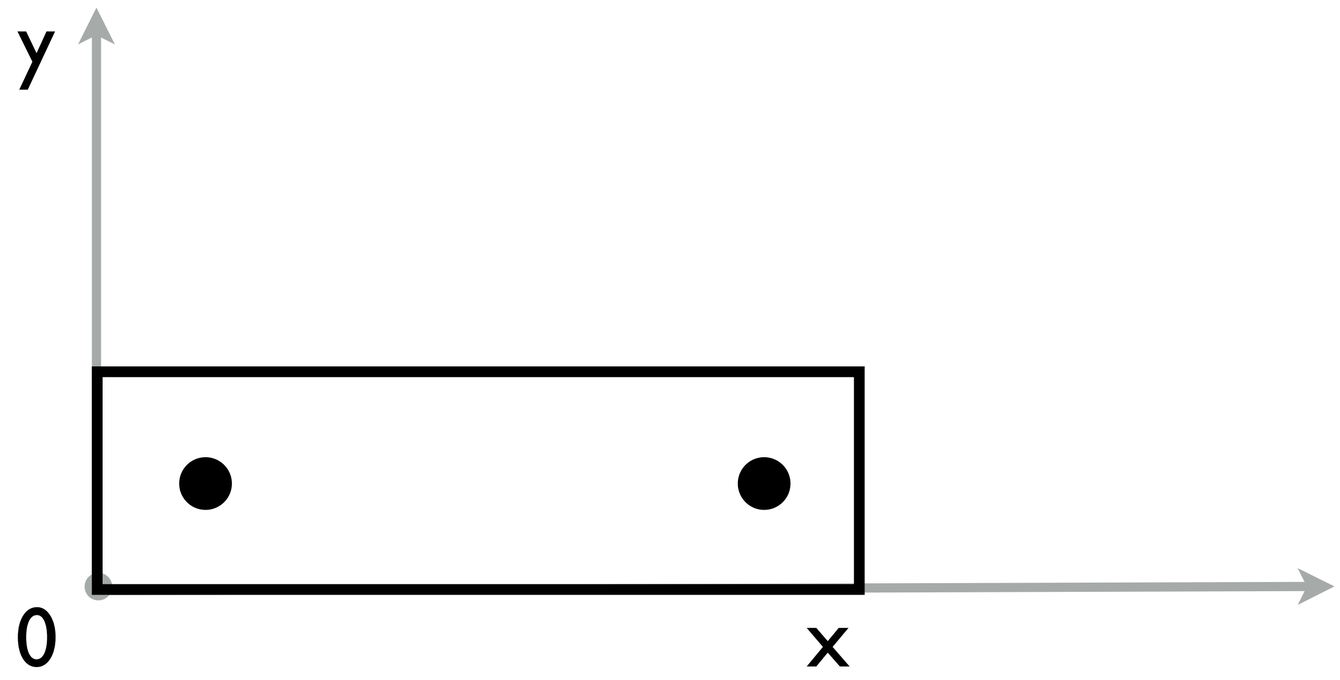
Hint: All parts are drawn using the x-y coordinate system. But, you will need to transform the parts using change of coordinates. These transformations will form a chain of transformations.

Exercise 3: show the sequence of transformations needed to draw the kinematic chain.

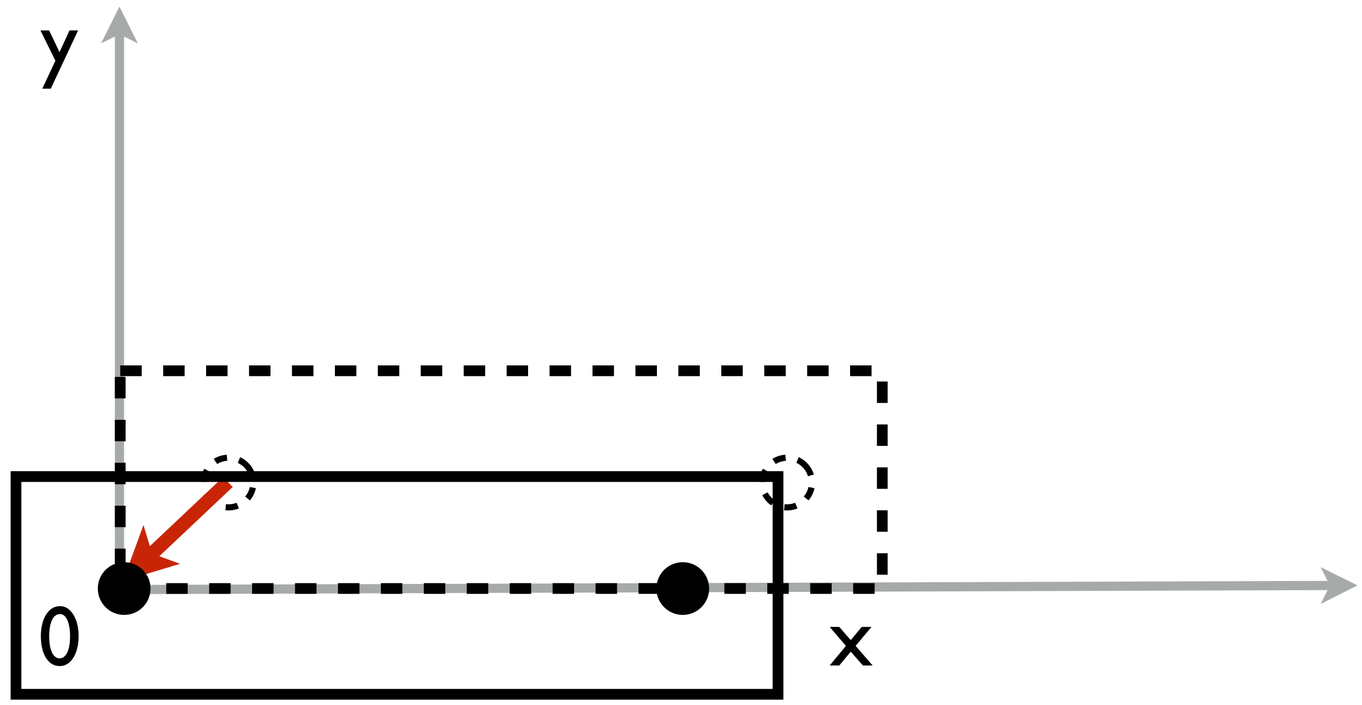


$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}.$$

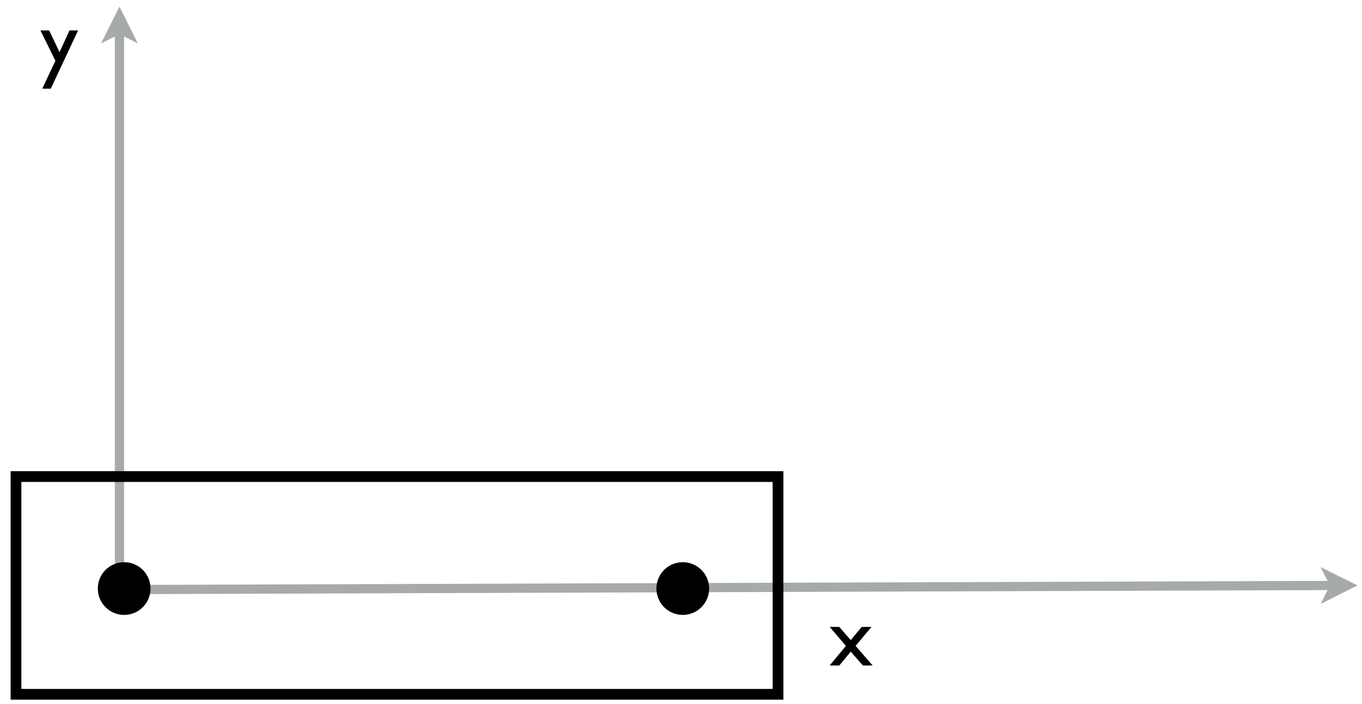
Drawing Kinematic Chains



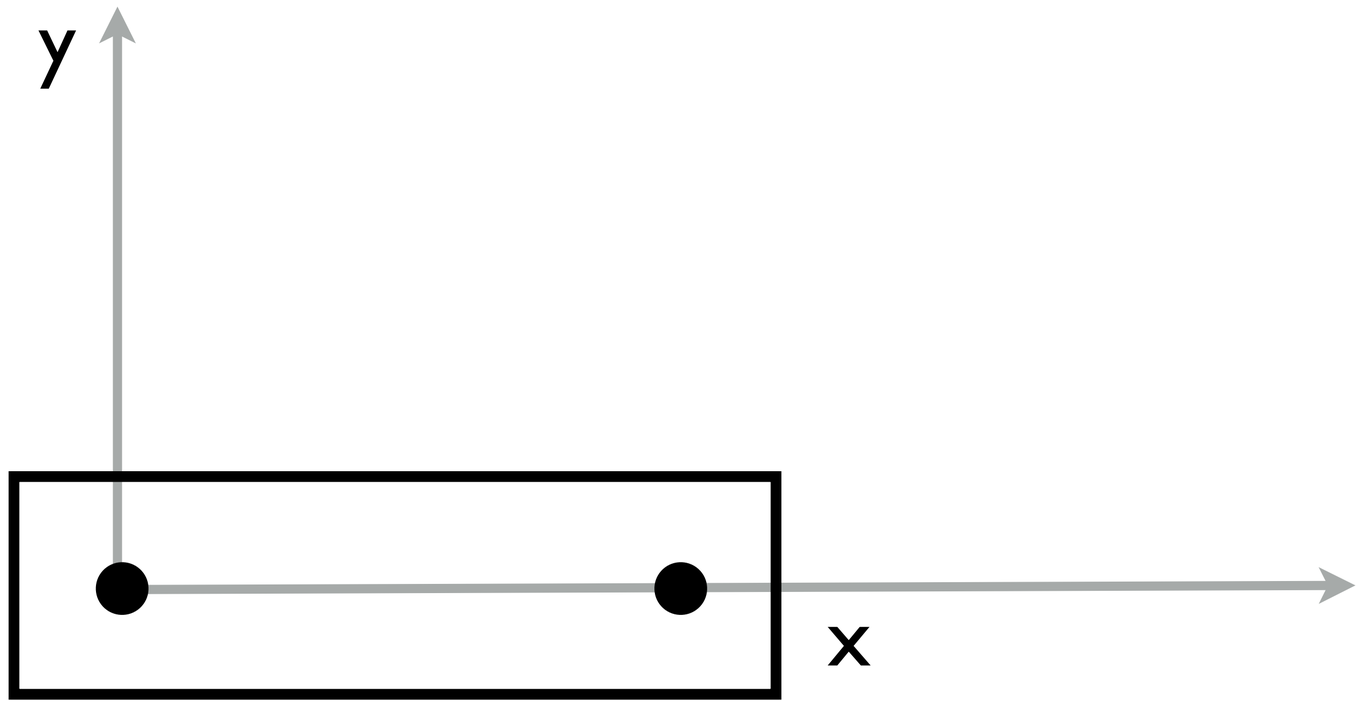
- Draw a rectangle with two joint points at location (0,0)
- We want the left-joint point to connect to the origin.



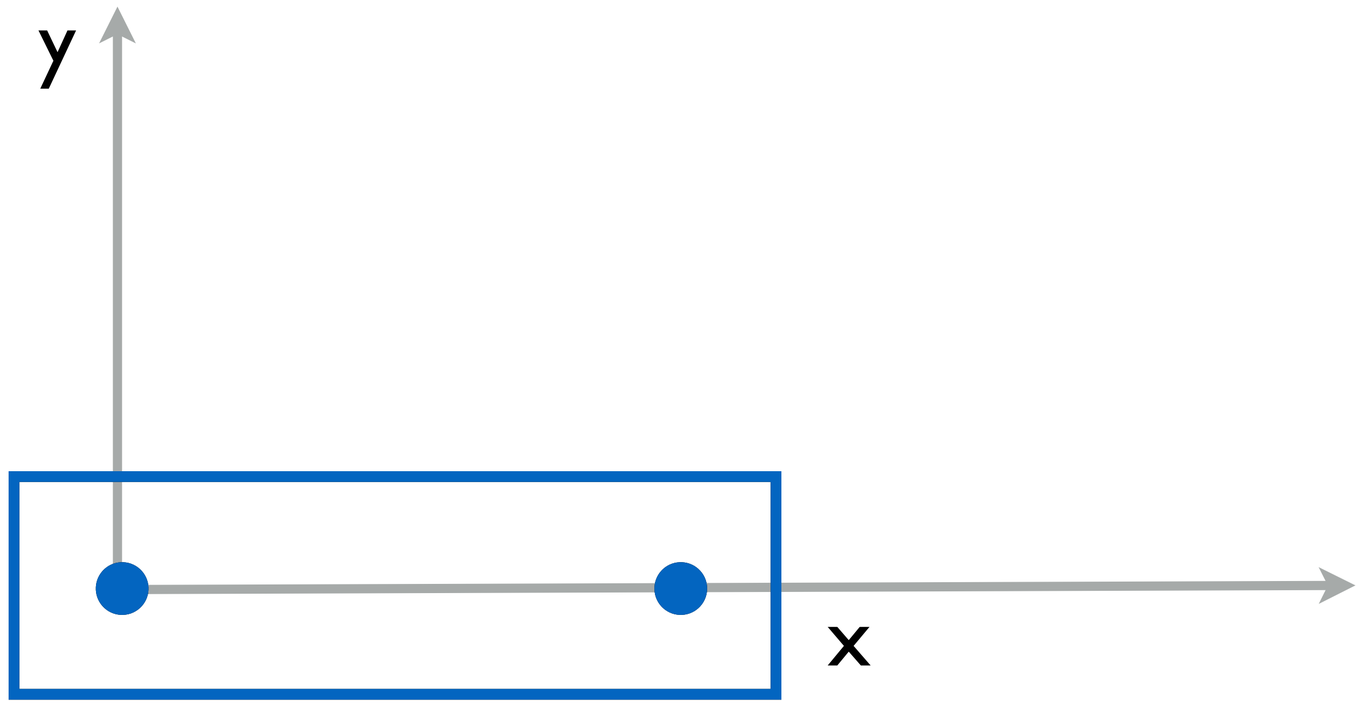
- Draw a rectangle with two joint points at location (0,0)
- We want the left-joint point to connect to the origin.
- To do that, we need to translate the whole figure so that the left-joint point connects to the origin



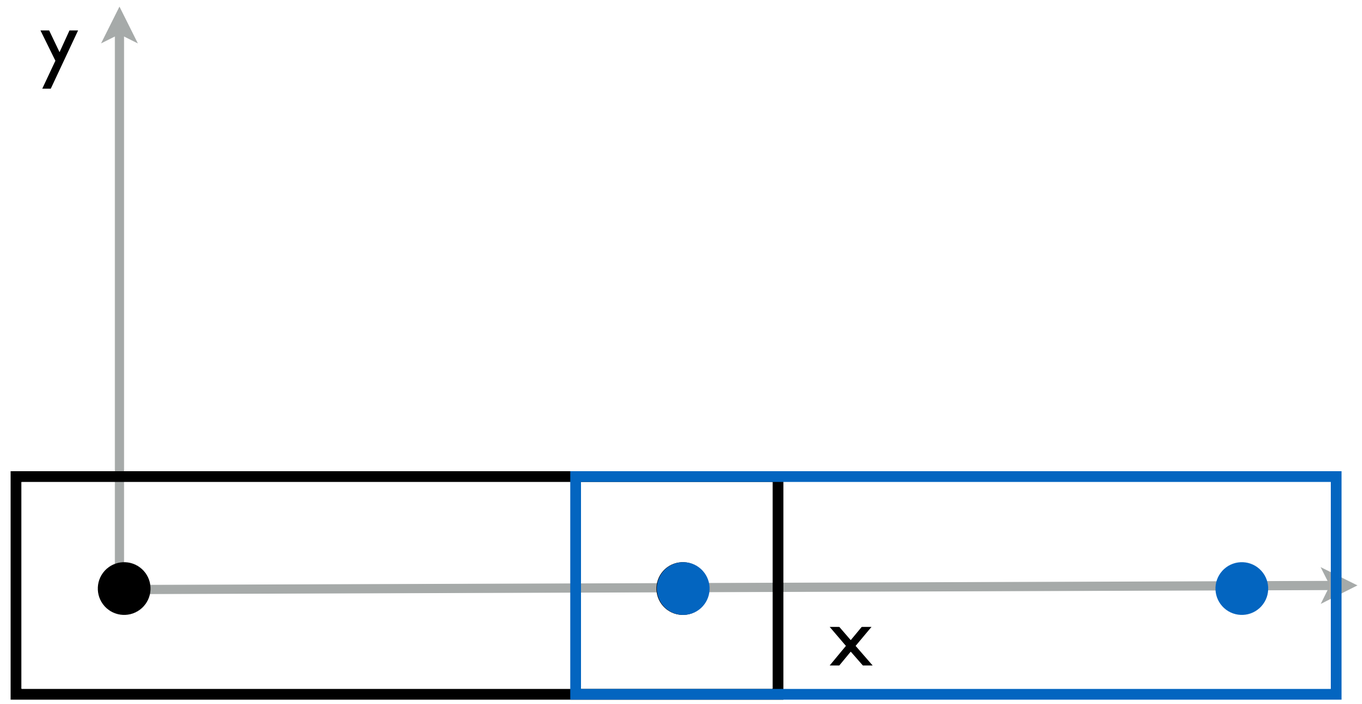
- Now that the basic part is in the correct place, we can draw another part that is connected to this one.
- The procedure is similar. The only difference is that we will need to translate the new part to the location of the right-joint point.



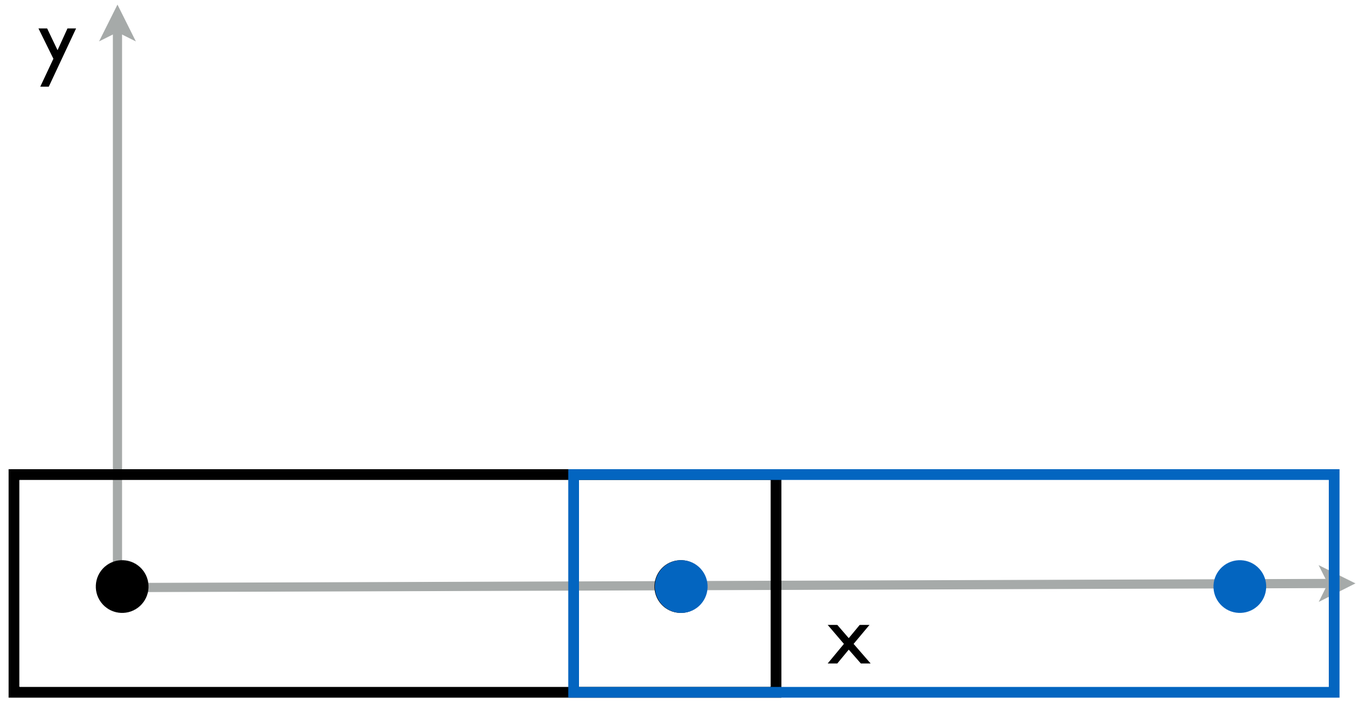
- We can think of this procedure as drawing the part at the initial location (i.e., on top of the first part) and then translating it to the new position.



- We can think of this procedure as drawing the part at the initial location (i.e., on top of the first part) and then translating it to the new position.



- We can think of this procedure as drawing the part at the initial location (i.e., on top of the first part) and then translating it to the new position.
- The translation transformation is by the length between the two joint points.



- Once we write the function to draw the basic parts, we can transform the chain in a recursive manner to obtain various kinematic configurations.
- This recursive process is called *forward kinematics*.

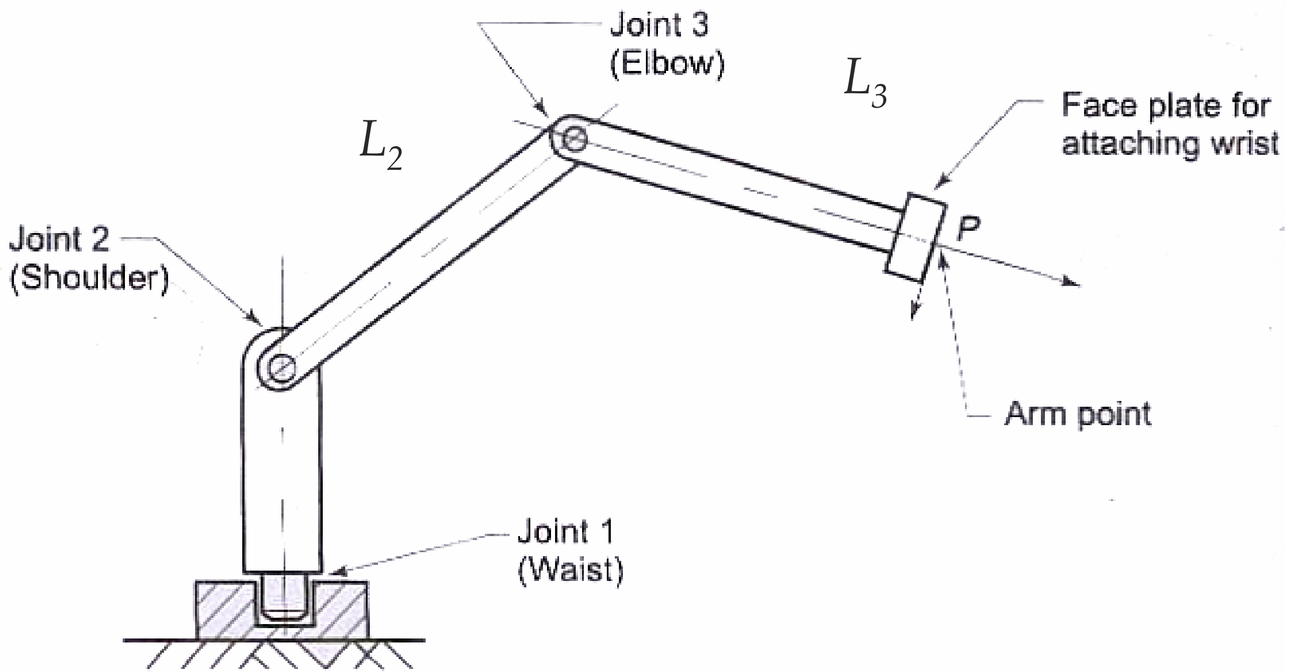
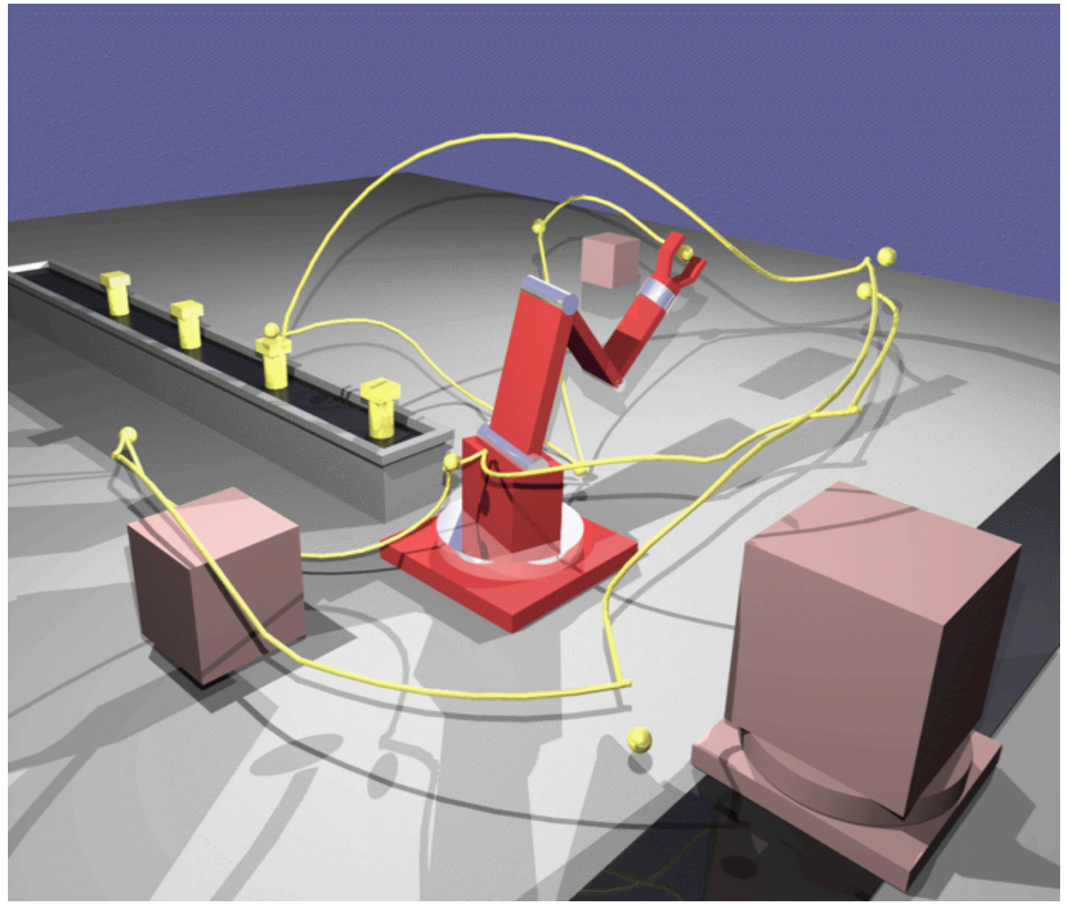


Figure from slide by Asanga Ratnaweera



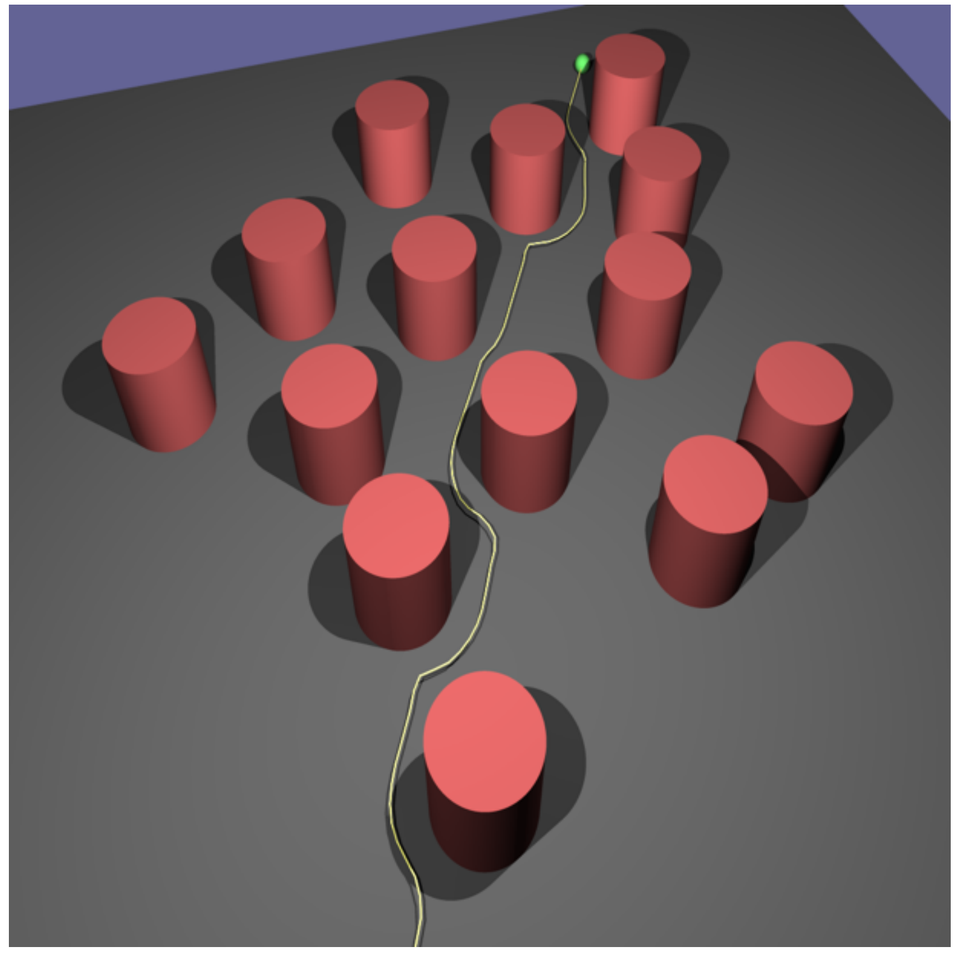
Breen, 1997

- Similar reasoning applies to 3-D kinematic chains.

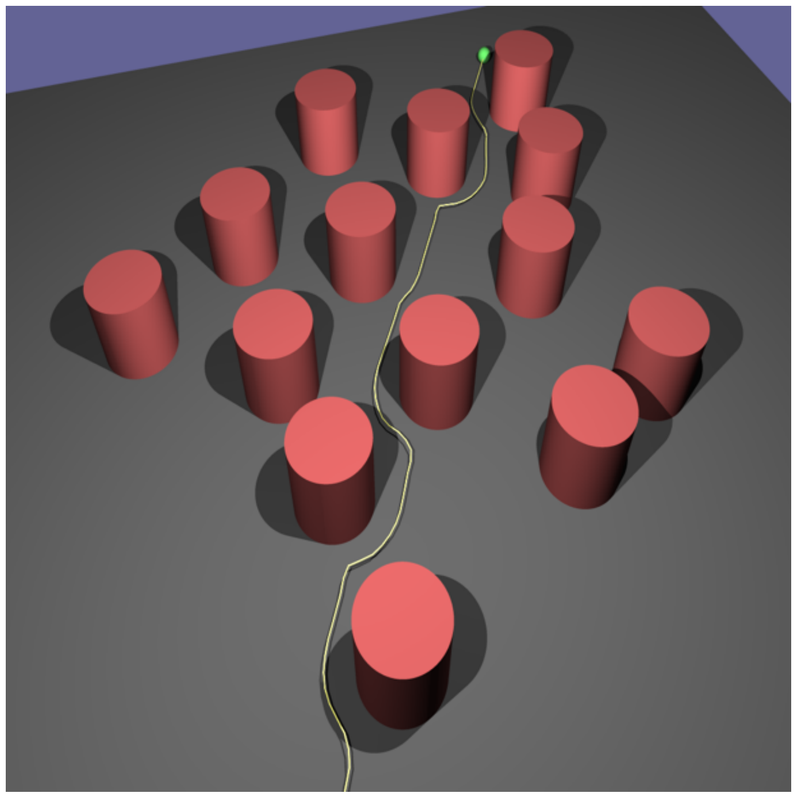
Energy Minimization for Animation

Animation by minimizing cost functions

- Goal-oriented motion.
- We can add constraints. These constraints change the topography of the cost functions.
- Animation becomes a task of defining a function
- A disadvantage is that animator surrenders control over details to the algorithm.



Example 1



$$C_{\text{PathPlan}}(\mathbf{x}) = \|\mathbf{x} - \mathbf{g}\| + \sum_{i=1}^{n} \mathcal{F}(\|\mathbf{x} - \mathbf{o}_i\|)$$

Where:

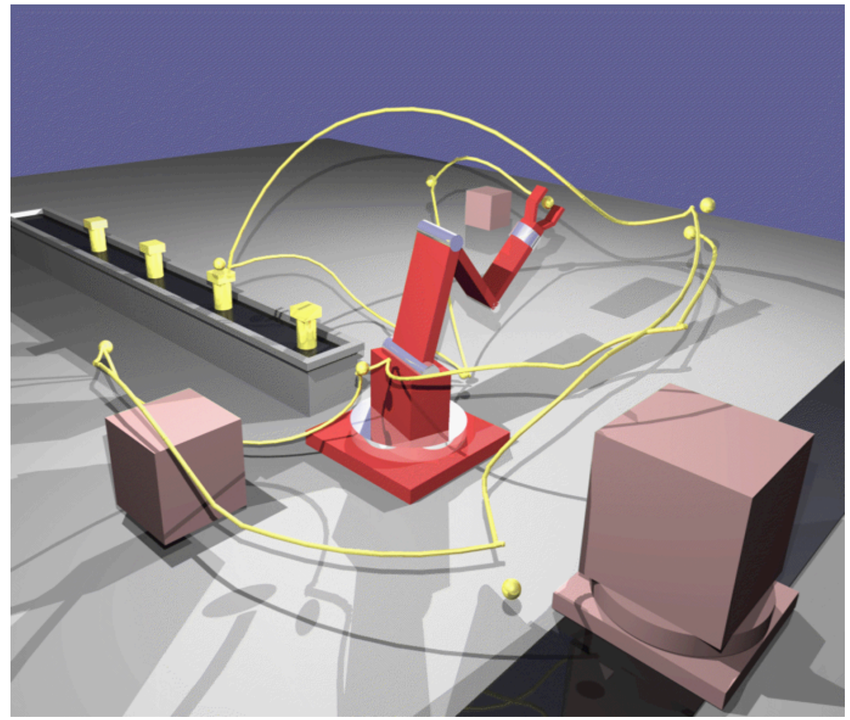
$\mathbf{x}$: Current location of the animated object

$\mathbf{g}$: Goal location

$\mathbf{o}_i$: Location of object i

$\mathcal{F}$: Penalty field for collision avoidance

Example 2



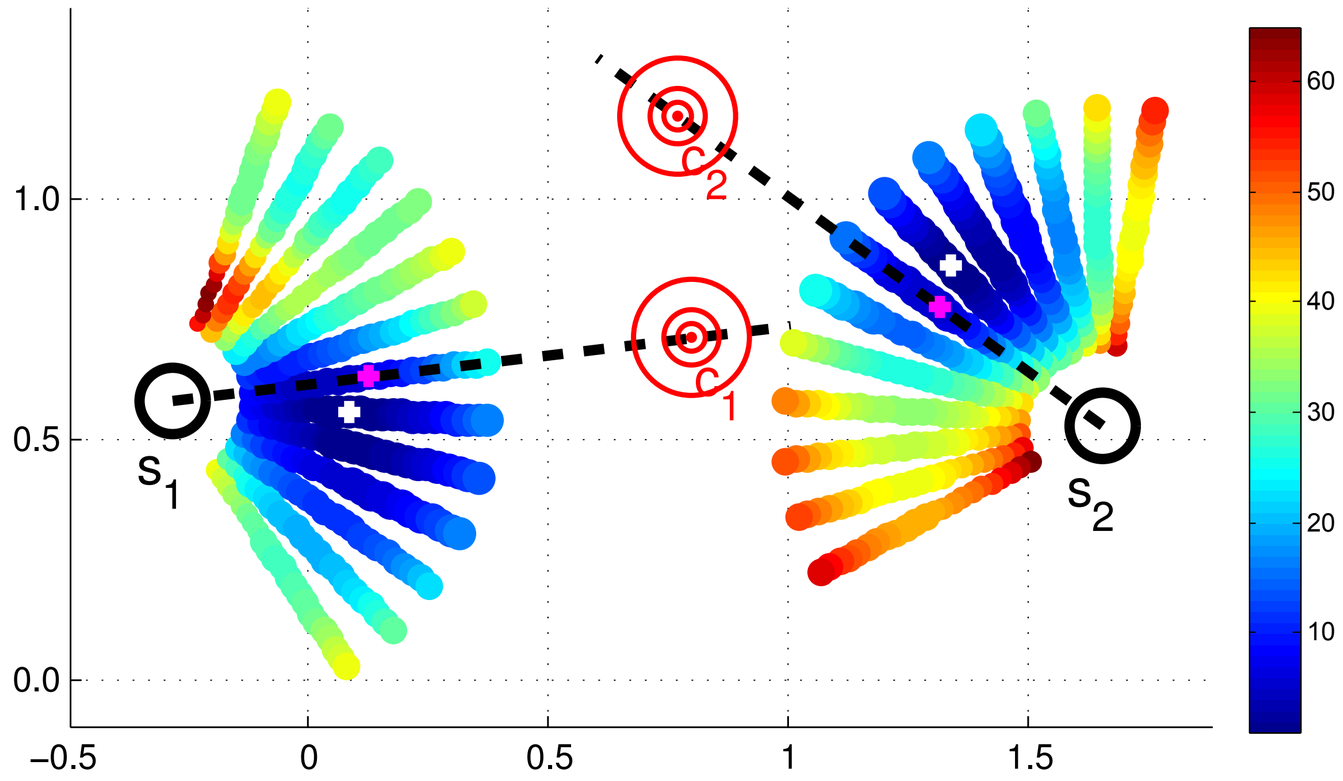
$$C_{\text{Reach}}\left(\phi_0,\phi_1,\phi_2\right) = \left\| P_{\text{tip}}\left(\phi_0,\phi_1,\phi_2\right) - \mathbf{g} \right\| + \sum_{i=1}^{n} \mathcal{F}\left(\left\| P_{\text{tip}}\left(\phi_0,\phi_1,\phi_2\right) - \mathbf{o}_i \right\|\right) + \sum_{i=0}^{2} \text{limit}\left(\phi_j\right).$$

Social-force model

D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995.

Social-force model

$$d_{ij}^2(t, \tilde{\mathbf{v}}_i) = ||\mathbf{p}_i + t\tilde{\mathbf{v}}_i - \mathbf{p}_j - t\mathbf{v}_j||^2$$



- Colors denote energies for different velocities.
- White dots mark the minima

Reference: http://vision.cse.psu.edu/courses/Tracking/vlpr12/PellegriniNeverWalkAlone.pdf

Gradient Descent Search

Gradient Descent

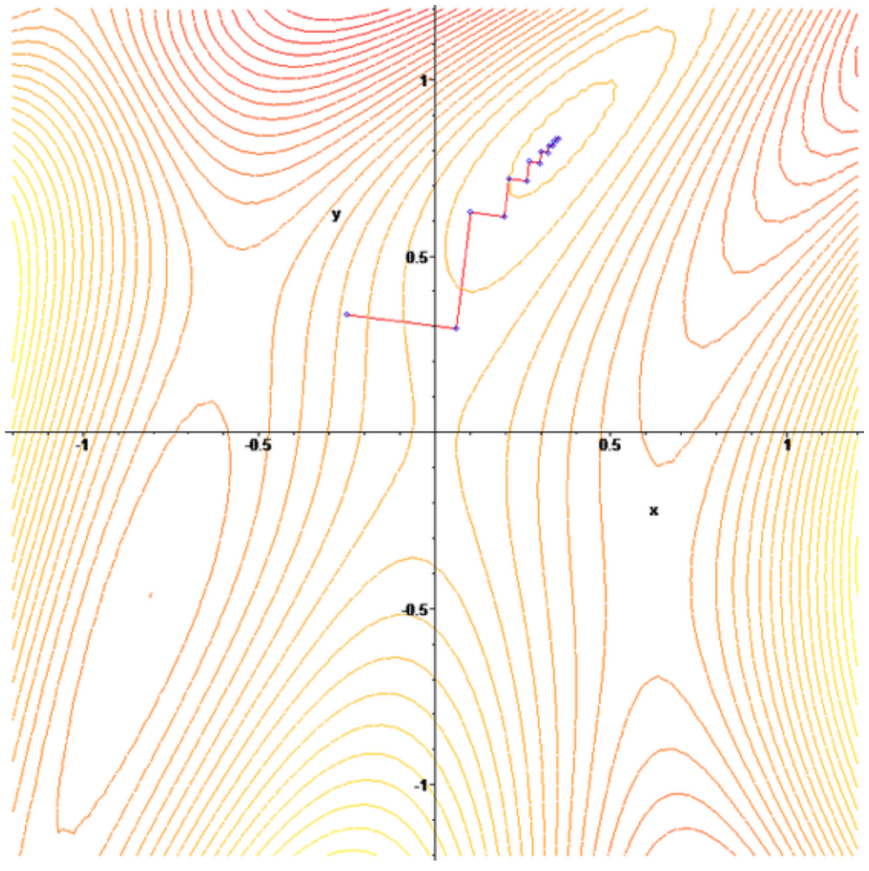
- The gradient-descent algorithm searches for the minimum of a function.
- In our case, we want a point at the end of our kinematic chain to move towards a target location in space.
- As a result, we will have a function of the joint angles and the distance from the end of the chain to the target.

Gradient-descent algorithm

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}), \quad \text{for } n \geq 0.$$

- Example of the search for the minimum starting from an initial location and walking in the direction of the negative gradient.

$$F(x,y) = \sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)\cos(2x + 1 - e^y)$$



http://en.wikipedia.org/wiki/Gradient_descent

Gradient-descent algorithm

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}), \quad \text{for } n \geq 0.$$

A computational example

[edit]

The gradient descent algorithm is applied to find a local minimum of the function $f(x)=x^4-3x^3+2$, with derivative $f'(x)=4x^3-9x^2$. Here is an implementation in the Python programming language.

```
# From calculation, we expect that the local minimum occurs at x=9/4

x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size
precision = 0.00001

def f_prime(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_prime(x_old)
print "Local minimum occurs at ", x_new
```

http://en.wikipedia.org/wiki/Gradient_descent

Gradient-descent algorithm

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}), \quad \text{for } n \geq 0.$$

A computational example

[edit]

The gradient descent algorithm is applied to find a local minimum of the function $f(x)=x^4-3x^3+2$, with derivative $f'(x)=4x^3-9x^2$. Here is an implementation in the Python programming language.

```
# From calculation, we expect that the local minimum occurs at x=9/4

x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size
precision = 0.00001

def f_prime(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_prime(x_old)
print "Local minimum occurs at ", x_new
```

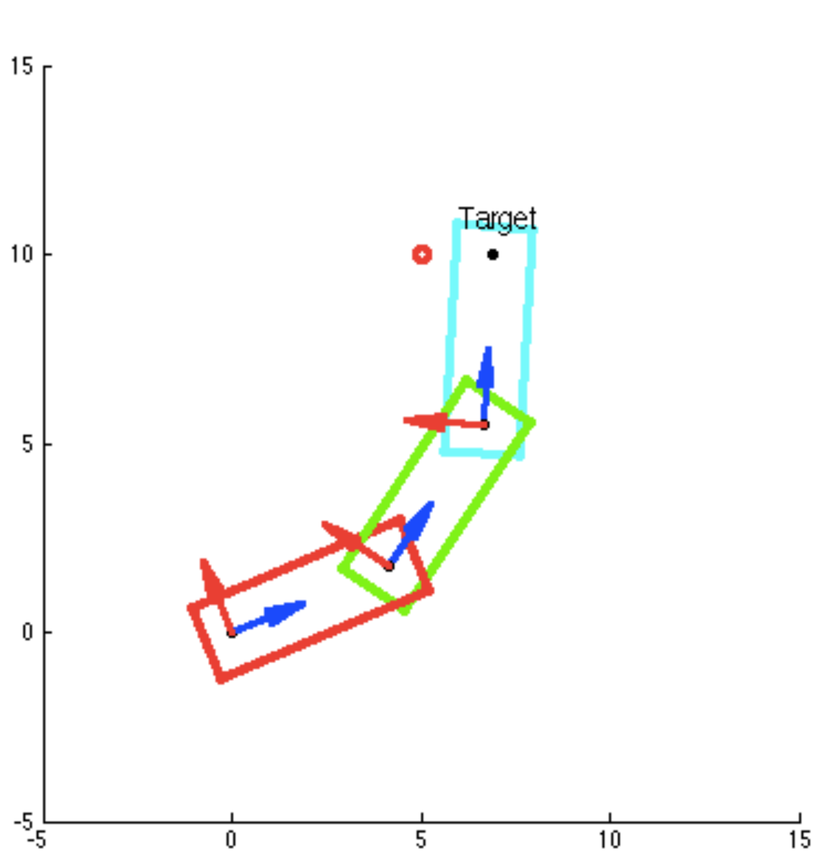
The above piece of code has to be modified with regard to step size according to the system at hand and convergence can be made faster by using an adaptive step size. In the above case the step size is not adaptive. It stays at 0.01 in all the directions which can sometimes cause the method to fail by diverging from the minimum.

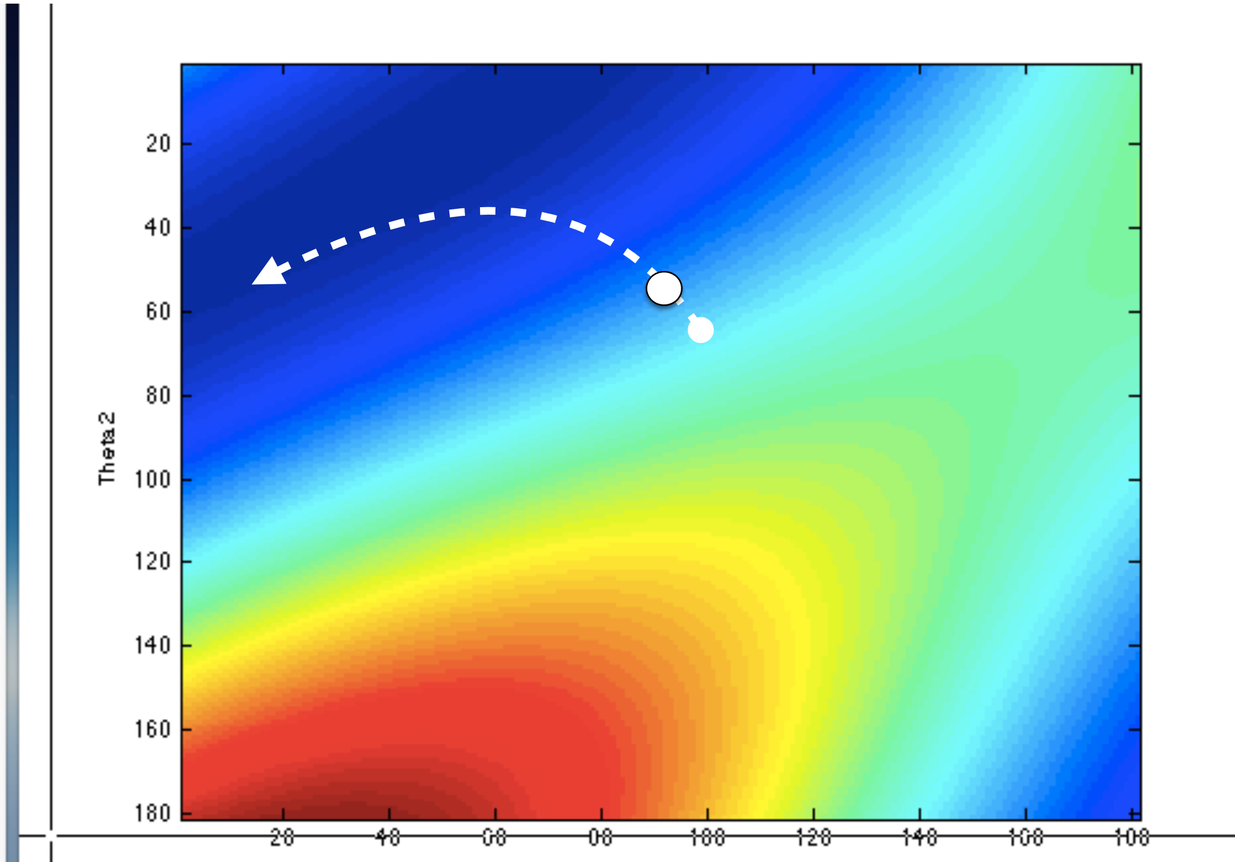
http://en.wikipedia.org/wiki/Gradient_descent

Calculating the gradient of the kinematic chain

- To implement the gradient-descent algorithm as described in the previous slides, we need to calculate the gradient of the function.
- Calculating the gradient of the function involving the kinematic chain is possible but we can try to avoid that by using a brute-force search of the configuration space.
- This method is illustrated in the next slides.

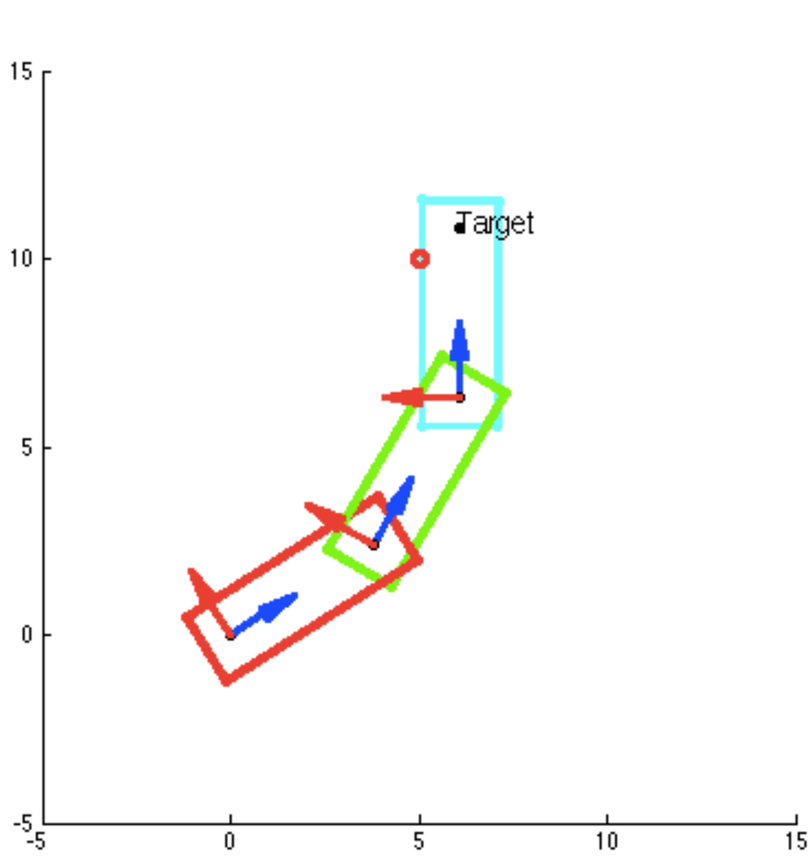
Every location in the configuration space is a solution to the kinematic chain's motion.

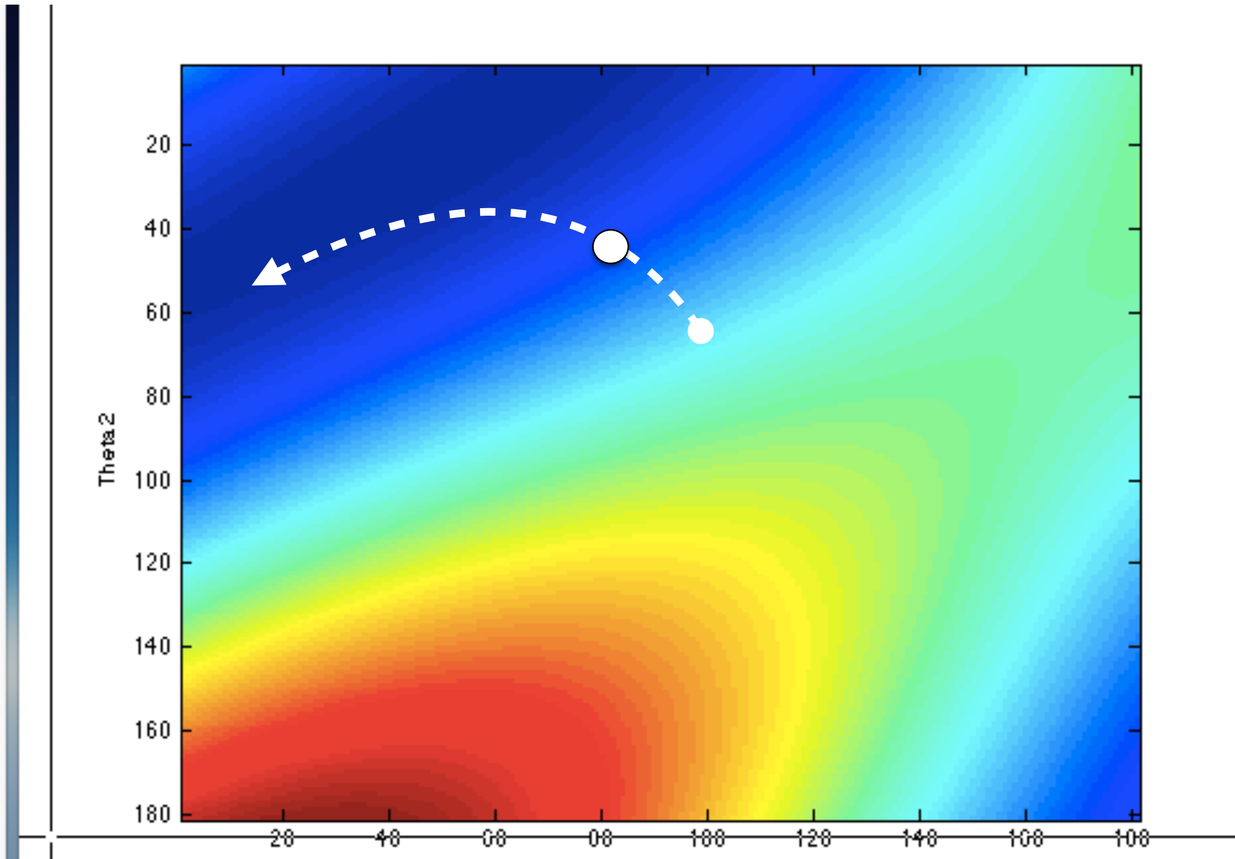




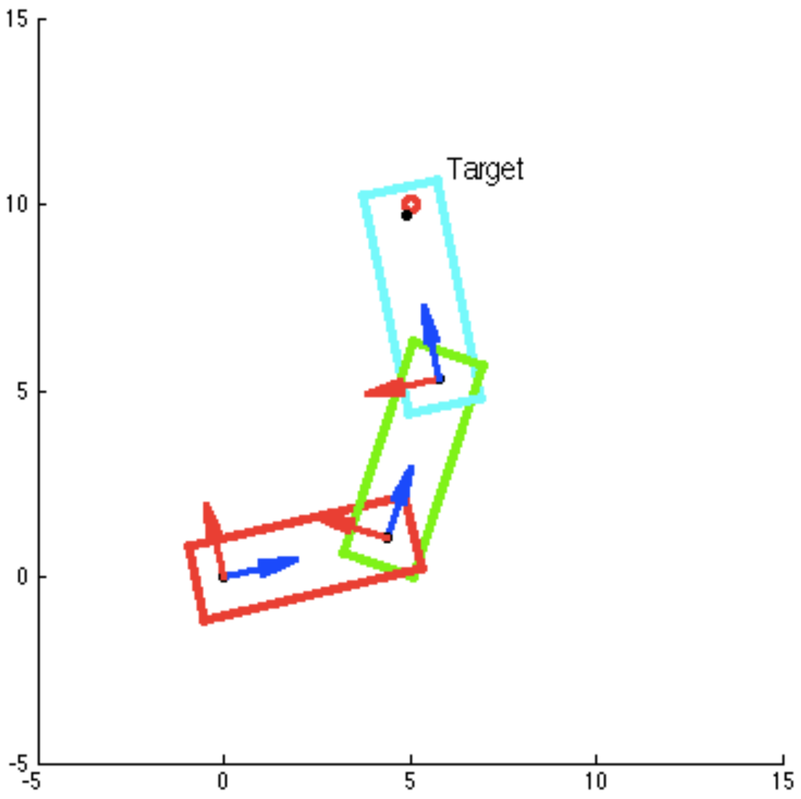
As we move through any path in the configuration space we can draw the kinematic chain for that configuration. The minimum in the configuration space provides the angles that will make the chain meet the target point.

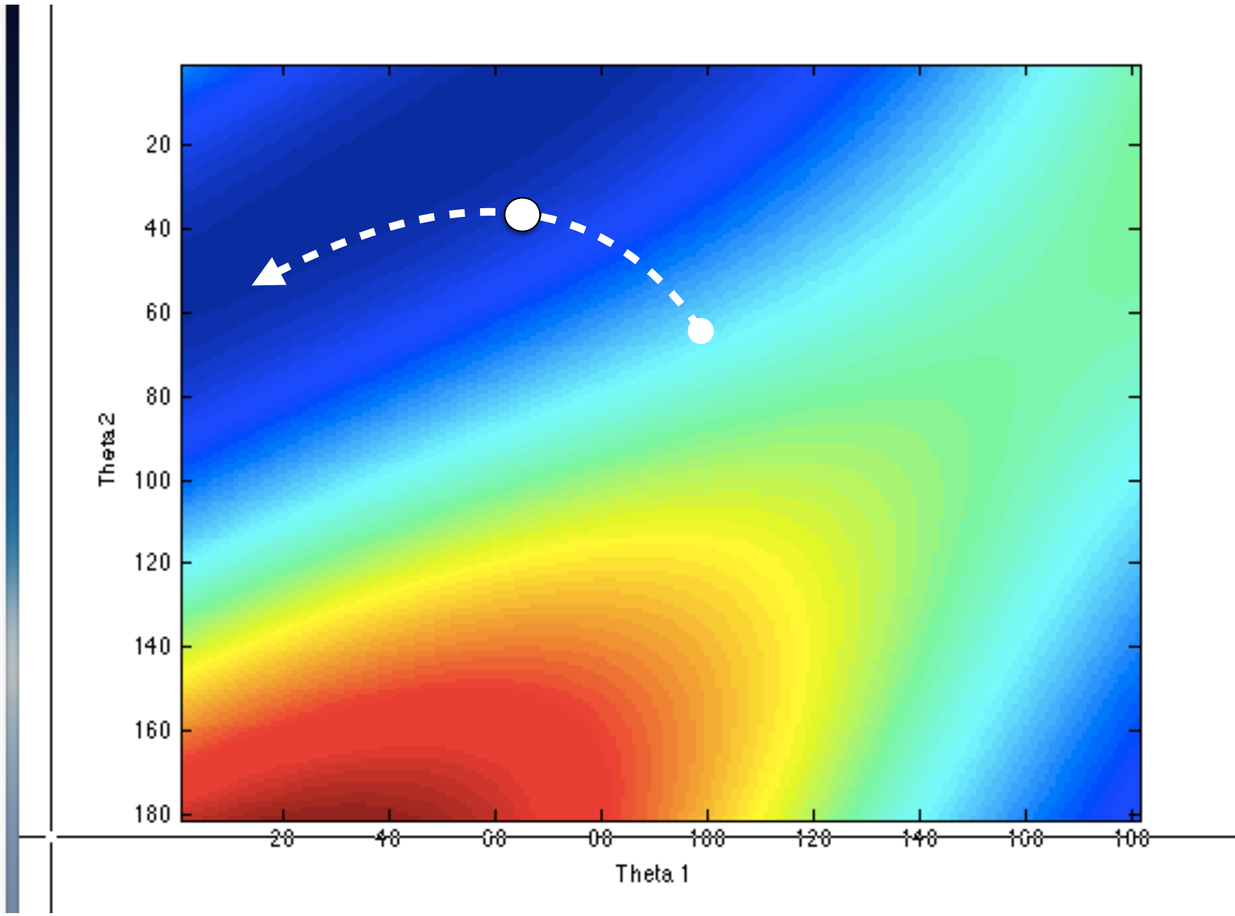
Every location in the configuration space is a solution to the kinematic chain's motion.





Every location in the configuration space is a solution to the kinematic chain's motion.





Every location in the configuration space is a solution to the kinematic chain's motion.