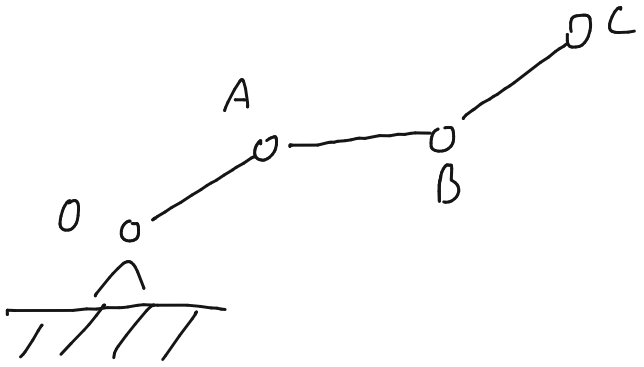
Unit 1:

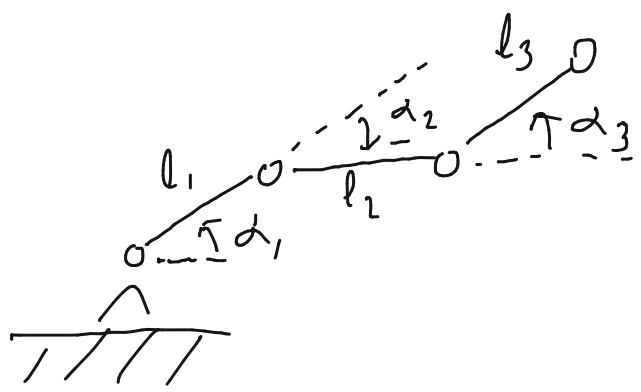
Kinematics: Where in the world are pieces of the system?

How are they moving?

(Unit 2: Dynamics will cover why they move)

Basic Kinematics Problem: "Serial chain" robot arm





- Robot arm is a collection of rigid links: OA, AB, and BC
- Each link has a length l_1, l_2, or l_3
- We know the angle from the ground to link OA, and from each link to the next
- Want to find the positions $\vec{r}_A$, $\vec{r}_B$, and $\vec{r}_C$

Approach: Build up positions of points on chain as

1. The location of a point at the end of a link is the sum of the link vectors up to and including that link

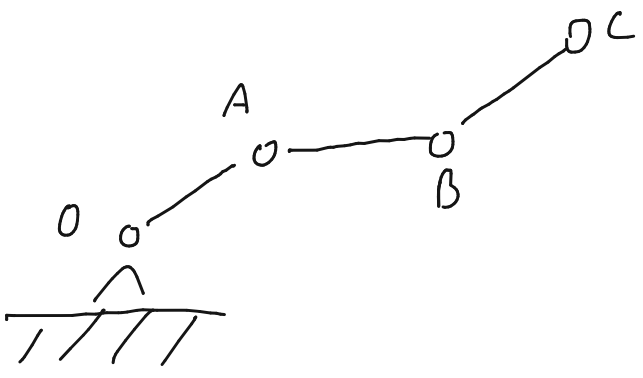
2. Each link vector is defined <u>locally</u> by a vector in a frame associated with that link, and <u>globally</u> by using the orientation of that frame to rotate the vector into the global frame

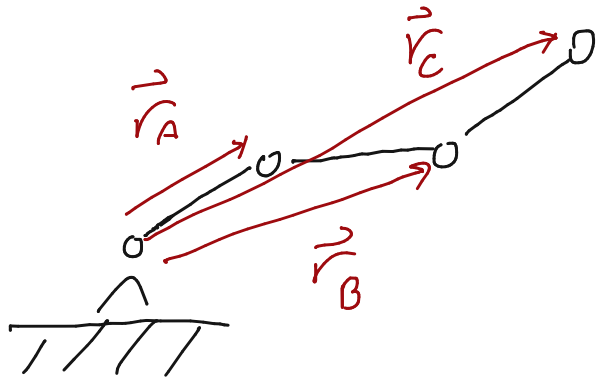
3. The orientation of each link's frame is the combination of the joint angles between the ground and that link

---

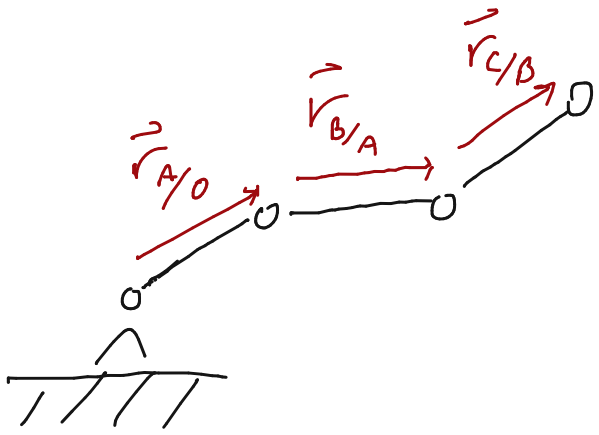
1. Location of each point

The location of each point can be taken as a vector $\vec{r}$ from the origin to that point:

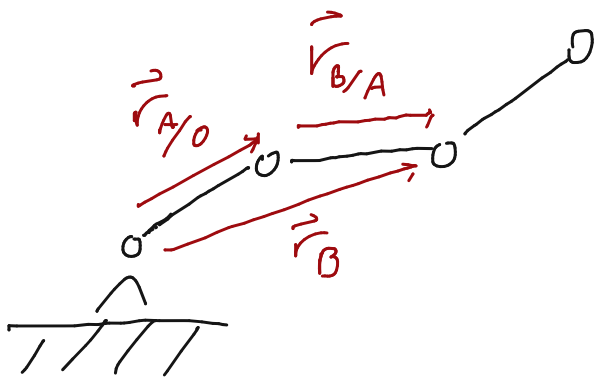




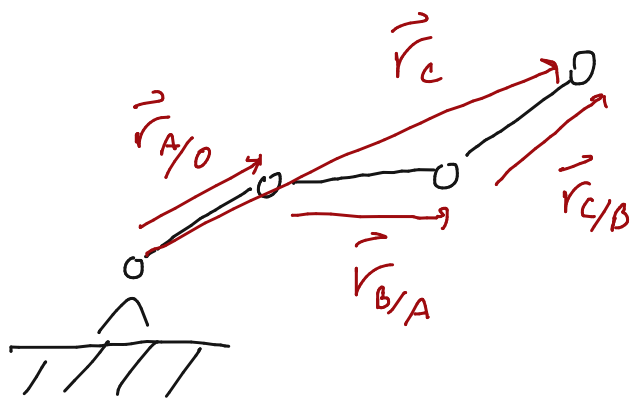
These vectors can be constructed from the individual link vectors:



link vectors



$\vec{r}_B = \vec{r}_{A/O} + \vec{r}_{B/O}$



$\vec{r}_C = \vec{r}_{A/O} + \vec{r}_{B/A} + \vec{r}_{C/B}$

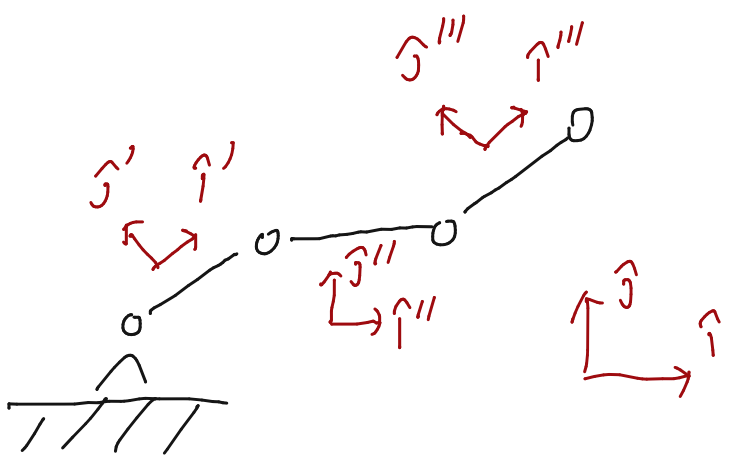
Notation: $\vec{r}_{A/B}$ ← Point, ← relative to, ← reference

$\vec{r}_{C/A} = \vec{r}_{B/A} + \vec{r}_{C/B}$ | summing vectors multiplies subscript

$\vec{r}_C \equiv \vec{r}_{C/O}$ | if reference is not given, it is the origin

2. Link frames

Each link has an attached frame, and its geometry is defined by a vector in this frame. For now, lets say that the frame's $\hat{\imath}$ axis is aligned with the link.

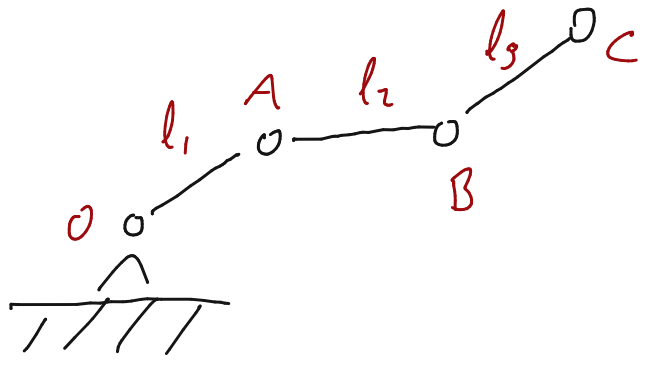


The link vectors are then of length ℓ_k in their respective frames

$$\vec{r}_{A/O} = \begin{bmatrix} \ell_1 \\ 0 \end{bmatrix}'$$

$$\vec{r}_{B/A} = \begin{bmatrix} \ell_2 \\ 0 \end{bmatrix}''$$

$$\vec{r}_{C/B} = \begin{bmatrix} \ell_3 \\ 0 \end{bmatrix}'''$$



These link vectors are expressed in different frames, so we cannot add them directly. Instead, we must first bring them all into a shared frame.

Continued ...

2, continued

Bringing vectors from the local frames into the global frame means finding the r_x and r_y values that correspond to known r_x' and r_y' values in expressions such as

$$\vec{r}_{A/O} = \begin{bmatrix} r_x' \\ r_y' \end{bmatrix}_{A/O}' = \begin{bmatrix} r_x \\ r_y \end{bmatrix}_{A/O}$$

no prime indicates that coefficients should be interpreted as being in global frame

coefficients in global frame

coefficients in local frame

prime indicates that coefficients should be interpreted as being in local frame

To find these values, let's look at what these vector expressions mean. The column representation of the global expression

$$\vec{r}_{A/O} = \begin{bmatrix} r_x \\ r_y \end{bmatrix}$$

is shorthand for saying that $\vec{r}_{A/O}$ is a weighted sum of the global $\hat{\imath}$ and $\hat{\jmath}$ vectors,

$$\vec{r}_{A/O} = r_x \hat{\imath} + r_y \hat{\jmath},$$

which can be written as the "dot product" of the basis vectors and the coefficients,

$$\vec{r}_{A/O} = \begin{bmatrix} \hat{\imath} & \hat{\jmath} \end{bmatrix} \begin{bmatrix} r_x \\ r_y \end{bmatrix}.$$

The local column expression is similarly shorthand for a weighted sum that can be written as a basis-coefficient "dot product",

$$\vec{r}_{A/O} = r_x' \hat{\imath}' + r_y' \hat{\jmath}' = \begin{bmatrix} \hat{\imath}' & \hat{\jmath}' \end{bmatrix} \begin{bmatrix} r_x' \\ r_y' \end{bmatrix}$$

continued...

2, continued

Combining our expanded expressions for the global and local forms of the link vectors gives us

$$\begin{bmatrix} \hat{\imath} & \hat{\jmath} \end{bmatrix} \begin{bmatrix} r_x \\ r_y \end{bmatrix} = \begin{bmatrix} \hat{\imath}' & \hat{\jmath}' \end{bmatrix} \begin{bmatrix} r_x' \\ r_y' \end{bmatrix}.$$

The basis vectors for the global space are by definition $\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, so the left-hand side of the equation resolves simply to

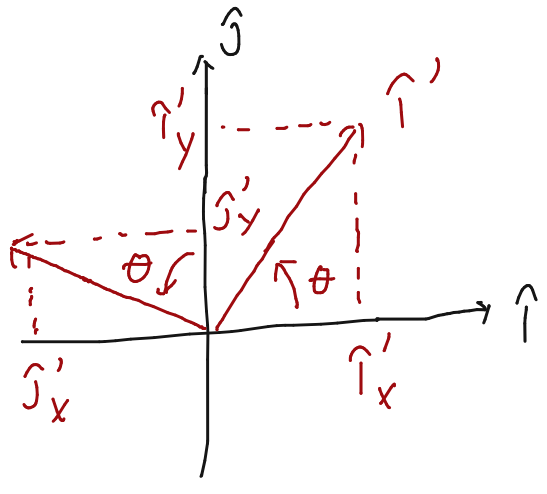
$$\begin{matrix} \hat{\imath} & \hat{\jmath} \end{matrix}$$
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r_x \\ r_y \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \end{bmatrix}.$$

If we can find expressions for $\hat{\imath}'$ and $\hat{\jmath}'$ as they appear in the global frame, we can then solve for r_x and r_y as

$$\begin{bmatrix} r_x \\ r_y \end{bmatrix} = \begin{bmatrix} \hat{\imath}' & \hat{\jmath}' \end{bmatrix} \begin{bmatrix} r_x' \\ r_y' \end{bmatrix}.$$

3. Frame orientations

The basis vectors $\hat{\imath}'$ and $\hat{\jmath}'$ of a frame rotated relative to the global frame by an angle θ are



$$\hat{\imath}' = \begin{bmatrix} \hat{\imath}'_x \\ \hat{\imath}'_y \end{bmatrix} = \begin{bmatrix} c\theta \\ s\theta \end{bmatrix}$$

$$\hat{\jmath}' = \begin{bmatrix} \hat{\jmath}'_x \\ \hat{\jmath}'_y \end{bmatrix} = \begin{bmatrix} -s\theta \\ c\theta \end{bmatrix}$$

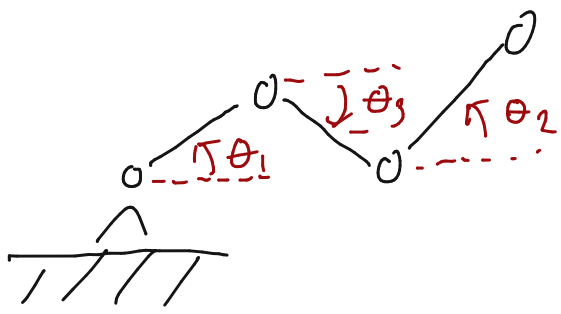
The matrix formed by combining these basis vectors is called the <u>rotation matrix</u> $R(\theta)$ associated with the primed frame,

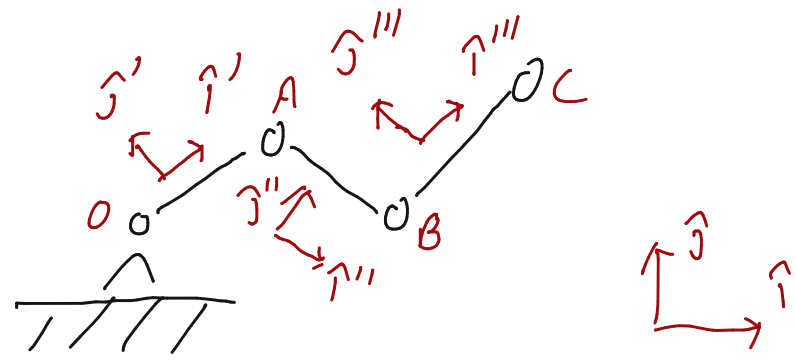
$$\begin{bmatrix} \hat{\imath}' & \hat{\jmath}' \end{bmatrix} = \begin{bmatrix} \hat{\imath}'_x & \hat{\jmath}'_x \\ \hat{\imath}'_y & \hat{\jmath}'_y \end{bmatrix} = \underbrace{\begin{bmatrix} c\theta & -s\theta \\ s\theta & c\theta \end{bmatrix}}_{R(\theta)}$$

Continued...

3, continued

If we know the orientations θ_k of the links,





then the rotation matrices for the link frames, as seen in the global frame, are

$$R_{OA} = \begin{bmatrix} \hat{\imath}' & \hat{\jmath}' \end{bmatrix} = \begin{bmatrix} c\theta_1 & -s\theta_1 \\ s\theta_1 & c\theta_1 \end{bmatrix}$$

$$R_{AB} = \begin{bmatrix} \hat{\imath}'' & \hat{\jmath}'' \end{bmatrix} = \begin{bmatrix} c\theta_2 & -s\theta_2 \\ s\theta_2 & c\theta_2 \end{bmatrix}$$

$$R_{BC} = \begin{bmatrix} \hat{\imath}''' & \hat{\jmath}''' \end{bmatrix} = \begin{bmatrix} c\theta_3 & -s\theta_3 \\ s\theta_3 & c\theta_3 \end{bmatrix}$$

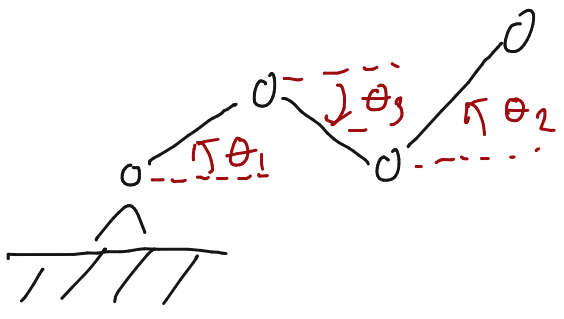
[Note that in the figure, we have marked the arrow for θ_3 to indicate that θ_3 has a negative value as currently displayed (and therefore is defined with the same counter-clockwise positive convention as θ_1 and θ_3. See the separate note on joint angle conventions for more discussion on this point.)

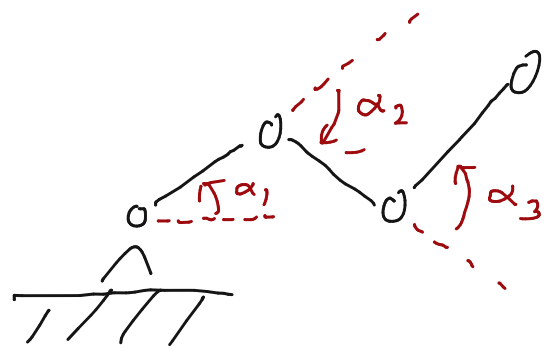
For most systems, however, we cannot directly measure or control the orientations θ of the links. Instead, we have direct access only to the joint angles between the links, and must build the link orientations from these angles.

continued...

3, continued

For a planar mechanism, the orientations of the links can be found by summing the joint angles between the ground and that link,





$$\theta_1 = \alpha_1$$

$$\theta_2 = \alpha_1 + \alpha_2$$

$$\theta_3 = \alpha_1 + \alpha_2 + \alpha_3,$$

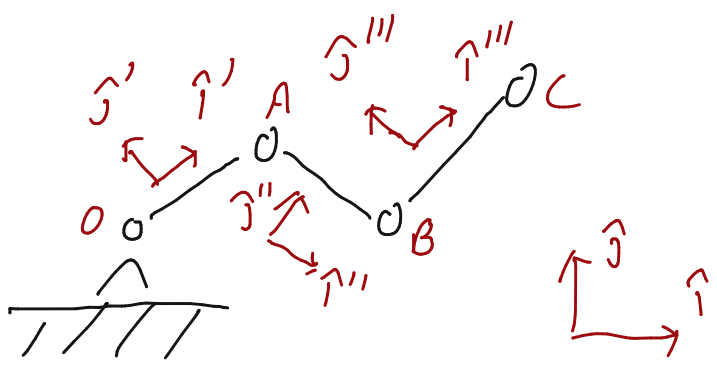
and these orientations can be used to construct the rotation matrices for each link. This approach does not, however, work for systems that move in three dimensional space, because rotations around different axes do not add (not even as vectors).

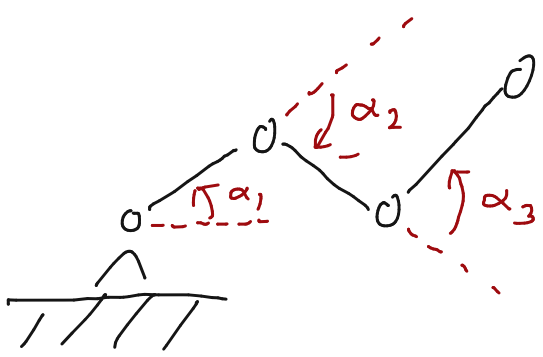
An alternative approach that <u>does</u> generalize to spatial motion is to use the joint angles to construct rotation matrices for the orientation of each link relative to the previous link, and then <u>multiply</u> these rotation matrices together.

continued...

3, continued

The orientation of each link relative to the previous link can be constructed as





$$R_{\frac{OA}{ground}} = [\hat{i}' \; \hat{j}'] = R(\alpha_1) = \begin{bmatrix} c\alpha_1 & -s\alpha_1 \\ s\alpha_1 & c\alpha_1 \end{bmatrix}$$

$$R_{\frac{AB}{OA}} = [\hat{i}'' \; \hat{j}'']' = R(\alpha_2) = \begin{bmatrix} c\alpha_2 & -s\alpha_2 \\ s\alpha_2 & c\alpha_2 \end{bmatrix}$$

$$R_{\frac{BC}{AB}} = [\hat{i}''' \; \hat{j}''']'' = R(\alpha_3) = \begin{bmatrix} c\alpha_3 & -s\alpha_3 \\ s\alpha_3 & c\alpha_3 \end{bmatrix}$$

Note that each rotation matrix describes how the basis vectors for one frame appear from the perspective of the previous frame. To find the orientation of a frame with respect to the world, we multiply these matrices together, e.g.,

$$R_{AB} = R_{\frac{AB}{ground}} = R_{\frac{OA}{ground}} R_{\frac{AB}{OA}} = \begin{bmatrix} c\alpha_1 & -s\alpha_1 \\ s\alpha_1 & c\alpha_1 \end{bmatrix} \begin{bmatrix} c\alpha_2 & -s\alpha_2 \\ s\alpha_2 & c\alpha_2 \end{bmatrix}$$

$$= \begin{bmatrix} c\alpha_1 c\alpha_2 - s\alpha_1 s\alpha_2 & -(c\alpha_1 s\alpha_2 + s\alpha_1 c\alpha_2) \\ s\alpha_1 c\alpha_2 + c\alpha_1 s\alpha_2 & c\alpha_1 c\alpha_2 - s\alpha_1 s\alpha_2 \end{bmatrix}$$

$$= \begin{bmatrix} c(\alpha_1+\alpha_2) & -s(\alpha_1+\alpha_2) \\ s(\alpha_1+\alpha_2) & c(\alpha_1+\alpha_2) \end{bmatrix},$$

which gives the same result as if we were to sum the joint angles and then build a rotation matrix from the sum.

How does this work in practice? The preceding list started from the goal (finding the positions of the ends of the links), and worked down to finding the rotation matrices associated with each joint. When actually solving for these quantities, we typically start at the other end and work upwards.

A typical computer program for evaluating the kinematics will start by taking in:

`link_vectors` ← the link vectors in their local frames
(e.g., $\vec{r}'_{A/O}$, $\vec{r}''_{B/A}$, $\vec{r}'''_{C/B}$)

`joint_angles` ← the joint angles
(e.g., $\alpha_1, \alpha_2, \alpha_3$)

and returning the locations of the ends of the links.

It will make use of several subfunctions that abstract the key steps in the kinematics process:

`R_planar` takes in an angle and returns the corresponding rotation matrix, e.g.,

$$R\_planar(\alpha) \longrightarrow \begin{bmatrix} c\alpha & -s\alpha \\ s\alpha & c\alpha \end{bmatrix}$$

`planar_rotation_set` takes in a vector of joint angles and returns a set of rotation matrices, e.g.

$$planar\_rotation\_set([\alpha_1, \alpha_2, \alpha_3]) \rightarrow \{R(\alpha_1), R(\alpha_2), R(\alpha_3)\}$$

Continued ...

Subfunctions in a kinematics program, continued:

`rotation_set_cumulative_product`

takes in a set of rotation matrices, and returns their cumulative product, e.g.,

`rotation_set_cumulative_product` $(\{R_1, R_2, R_3\})$

$\rightarrow \{R_1, R_1R_2, R_1R_2R_3\}$

`vector_set_rotate` take a set of vectors given in local coordinates and a set of rotation matrices for the local frames, and multiply each matrix into its corresponding vector, e.g.,

`vector_set_rotate` $(\{V_1, V_2, V_3\}, \{R_1, R_2, R_3\})$

$\rightarrow \{R_1V_1, R_2V_2, R_3V_3\}$

`vector_set_sum` takes a set of vectors given in a shared coordinate system and returns their cumulative sum, e.g.,

`vector_set_sum` $(\{V_1, V_2, V_3\})$

$\rightarrow \{V_1, V_1+V_2, V_1+V_2+V_3\}$

For programs like these, it would be a good idea to become familiar with *cell arrays* in Matlab (or their equivalent in other languages), so that you can work with sets of vectors or matrices. Other useful things are *for loops* to iterate over these cell arrays*, and the *length* or *numel* commands so that your code can adapt to sets of different size

* You can also use *arrayfun* or *cellfun* instead of a loop, but these are harder to learn, and their benefits don't show up at the scale of problems we consider in this course.

Visualizing the system

When writing computer programs to analyze engineering systems, it is very useful to include tools for visualizing the output of the code, so that you can verify that the code is doing what you think it is doing. A simple way to do this for the planar arm is

1. Add a zero vector to the beginning of the link endpoints,

```
link_end_set = [{[0;0]}, link_end_set]
```

2. Convert the cell array of vectors into a standard array (matrix) with each row one component of all the endpoints,

```
link_ends = cell2mat(link_end_set)
```

3. Plot these endpoints,

```
plot(link_ends(1,:), link_ends(2,:), '-o')
```

More advanced visualization that will make later steps easier:

A: Create your plot elements explicitly, instead of relying on 'plot'

```
f = figure(317)          ensure Figure 317 is open
clf(f,'reset')           clear the figure
ax = axes('Parent', f)   create axes in the figure
axis(ax,'equal')         make sure the axes have an equal
                           aspect ratio
```

these commands can be wrapped into a function `create_figure` that returns the figure handle `f` and axis handle `ax`, which other functions can use to draw items into that axis

continued

A, continued.

Write a function draw_links that takes in an axis handle and the link ends, plots the link ends using the line function, and returns a handle l to the line that can be used to modify it. The core code for this function is of the form

```
l = line('Xdata', link_ends(1,:), 'Ydata', link_ends(2,:),
        'Parent', ax, 'Linestyle', '-', 'Marker', 'o')
```

B. Plot each link as a separate line element. This requires modifying your previous code in the following ways:

I. Write a function build_links that augments each link vector (in local coordinates) with a zero vector representing the base of the link. This code should look something like:

```
link_set = cell(size(link_vectors))
for idx = 1:numel(link_set)
    link_set{idx} = [zeros(size(link_vectors{idx})), link_vectors{idx}]
end
```

↑ beginning of link

↑ end of link

II. Rotate the link vectors using vector_set_rotate. This should "just work", as multiplying a rotation matrix into a matrix with multiple columns is the same as multiplying the matrix into the columns individually

continued

B, continued

III. Write a function place_links that puts the links in the world. It should add the endpoints of the previous links to the base and end points of these links. (Note that the most convenient way to do this is to calculate the endpoints by applying vector_set_sum to the endpoints as before, then adding these results to the rotated vectors in link_set. To add a vector v_{add} to every column in a matrix M:

$$v_{sum} = v_{add} + M$$

(this only works in matlab newer than 2016)

IV. Write a function draw_link_set that makes an individual line element for each link:

```
l = cell(size(link_set))
for idx = 1:numel(link_set)
    l{idx} = line('Xdata', link_set{idx}(1,:), 'Ydata', link_set{idx}(2,:),
                  'Parent', ax, 'Linestyle', '-', 'Marker', 'o')
```

(you will want your actual code to take in an axis handle and pass it to line as a 'Parent')

Prismatic Links.

Some mechanisms have pistons or telescoping segments. These components allow links to change length without rotating, and are called "prismatic links". To allow our programs to handle prismatic links, we need to add or modify the following functions:

`Vector_set_extend` New function:

Take a set of link vectors and a set of lengths by which to extend the links, and add these lengths to the vectors

`build_links` should be modified to include multiple pieces. For example, we could draw a prismatic link as a pair of lines $\frac{3}{4}$ the base length of the link, flanking a line $\frac{1}{2}$ the base length that represents the moving portion. Lines of code that will help here include:

`v_unit_normal = [-v(2); v(1)]/norm(v)` get a unit vector perpendicular to v

`link = [v1, v2, NaN(size(v1)), v3, v4]`

Put vectors into a matrix, separated by NaN values. The NaNs will break up the lines in the drawing commands.

Continued...

Prismatic links build_links, continued.

The two lines making up the proximal portion of the link are

$$L_1 = \left[\begin{bmatrix}0\\0\end{bmatrix} \quad \tfrac{3}{4}\,\text{link\_vector}\right] + \left(V\_\text{unit\_normal} * \text{norm}(\text{link\_vector})/20\right)$$

$$L_2 = \left[\begin{bmatrix}0\\0\end{bmatrix} \quad \tfrac{3}{4}\,\text{link\_vector}\right] - \left(V\_\text{unit\_normal} * \text{norm}(\text{link\_vector})/20\right)$$

↑
note sign difference

and the distal portion of the link is

$$L_3 = \left[\begin{bmatrix}0\\0\end{bmatrix} \quad \tfrac{1}{2}\,\text{link\_vector}\right] + \text{link\_vector}/2 + \text{link\_extension}$$