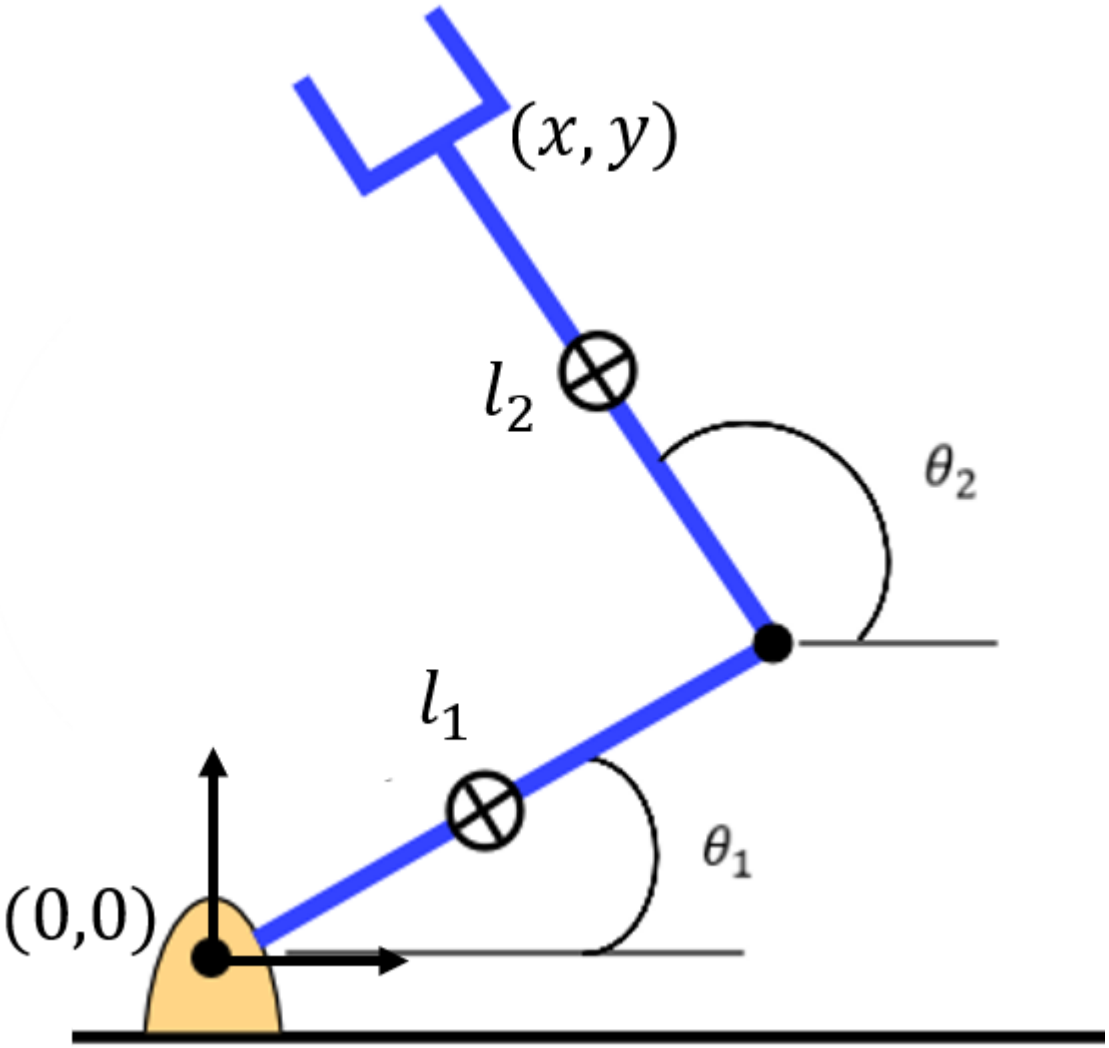
COMS 3251 Spring 2021: Lab 3

YOUR NAME(s): Vicente Farias

YOUR UNI(s): vf2272

Robot Kinematics

This lab will help you to visualize and think about vector spaces in the context of robotics. Think about your arm, starting from the shoulder, going down to your elbow and eventually your hand. The movement of your shoulder and elbow is directly related to the movement of your hand (assuming your wrist is rigid). Let's specify some quantities in the figure of the robot arm below.



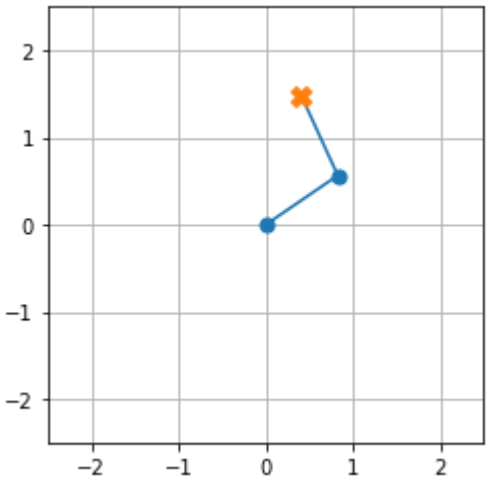
The base $(0, 0)$ of the arm above can be thought of as the "shoulder", and its angle from the horizontal is denoted θ_1. The "elbow" is a distance l_1 away from the base, and its angle from the horizontal is denoted θ_2. Finally, the end-effector or "hand" is located a distance of l_2 away from the elbow, and we denote its position as (x, y).

Visualization

The following code computes the kinematics for the arm, assuming $l_1 = l_2 = 1$. Given the angles θ_1 and θ_2, a figure is drawn showing what the entirety of the arm looks like. The two joints are shown as dots, the two links as line segments, and the end effector as a X. As usual, the angles are provided in radians and treated as modulo 2π. Try showing different combinations of joint angles to get a sense for the configurations the arm can be in.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

l1 = 1
l2 = 1

def RR_position(th):
  return np.array([[l1*np.cos(th[0]) + l2*np.cos(th[1])],
                   [l1*np.sin(th[0]) + l2*np.sin(th[1])]])

def draw_RR(th):
  plt.plot([0, l1*np.cos(th[0]), l1*np.cos(th[0]) + l2*np.cos(th[1])],
           [0, l1*np.sin(th[0]), l1*np.sin(th[0]) + l2*np.sin(th[1])])
  plt.scatter([0, l1*np.cos(th[0])], [0, l1*np.sin(th[0])], zorder=3, s=50)
  plt.scatter([l1*np.cos(th[0]) + l2*np.cos(th[1])],
              [l1*np.sin(th[0]) + l2*np.sin(th[1])],
              marker='X', zorder=3, s=100)
  plt.grid()
  plt.axis('scaled')
  plt.xlim([-2.5,2.5])
  plt.ylim([-2.5,2.5])
```

In [2]:
```
theta = np.array([0.6, 2.0])
draw_RR(theta)
```



Jacobian Matrix

It can be shown through geometry and calculus that there is a *linear* mapping between the angular velocities of the shoulder and elbow (ω_1, ω_2) and the translational velocities of the hand (v_x, v_y). This mapping is given by the **Jacobian** matrix and looks like the following:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = J(\theta_1, \theta_2) \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} -l_1 \sin\theta_1 & -l_2 \sin\theta_2 \\ l_1 \cos\theta_1 & l_2 \cos\theta_2 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}$$

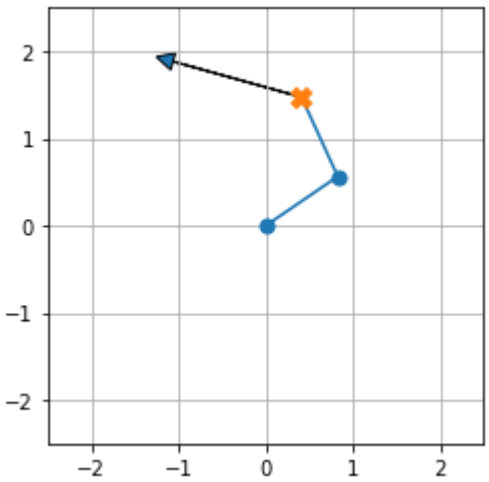
As you can see, the Jacobian implements a linear function that maps from $\mathbb{R}^2$ to $\mathbb{R}^2$ (to be precise, the domain and codomain are tangent spaces of $\mathbb{T}^2$ and $\mathbb{R}^2$, but they can be pretty much be treated as $\mathbb{R}^2$).

Note that this matrix depends on θ_1 and θ_2. As the arm's configuration changes, the nature of the function mapping will change as well. The following code include a function that computes and returns the Jacobian matrix for given values of θ_1 and θ_2, as well as a function to display the velocities (v_x, v_y) as a vector at the end effector, given joint velocities (ω_1, ω_2).

In [3]:
```
def RR_jacobian(th):
  return np.array([[-l1*np.sin(th[0]), -l2*np.sin(th[1])],
                   [l1*np.cos(th[0]), l2*np.cos(th[1])]])

def draw_RR_velocity(theta, omega):
  EE = RR_position(theta)
  J = RR_jacobian(theta)
  v = J @ omega
  print('J =', J)
  print('v = ', v)
  draw_RR(theta)
  plt.arrow(EE[0,0], EE[1,0], v[0], v[1], head_width=0.2, head_length=0.2)
```

In [4]:
```
theta = np.array([0.6, 2.0])
omega = np.array([1,1])
draw_RR_velocity(theta, omega)
```

```
J = [[-0.56464247 -0.90929743]
 [ 0.82533561 -0.41614684]]
v =  [-1.4739399   0.40918878]
```



Exercises

Since the Jacobian is a matrix implementing a linear function, we can talk about the vector spaces associated with it, as well as their dimensionalities. You will be asked to think about interpretations of the column and null spaces, which will change as the Jacobian matrix itself changes. Don't hesitate to ask for help or guidance on the mathematical ideas if you are still unclear on them.

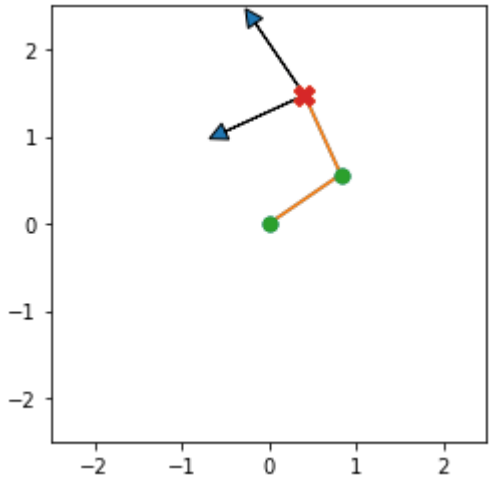
The exercises in this lab will be more conceptual and require more written responses than coding solutions than the previous labs. We encourage you to talk through your ideas with your lab partner and use the provided functions to visualize scenarios as much as you can. While there are a number of different parts, each should only require two or three sentences or lines of code to complete.

PART 1 (15 points)

Suppose the arm is in the configuration $(\theta_1, \theta_2) = (0.6, 2.0)$ (as in the example). Come up with two sets of joint velocities (ω_1, ω_2) so that one resultant end-effector velocity pair (v_x, v_y) is equal to the first column of J and the other is equal to the second column of J. Plot both end-effector velocities on the same plot by calling `draw_RR_velocity` twice, once with each joint velocity pair.

In [5]:
```
theta = np.array([0.6, 2.0])

# FILL IN: first set of joint velocity vectors
omega_a = np.array([1.0, 0.0])

# FILL IN: second set of joint velocity vectors
omega_b = np.array([0.0, 1.0])

draw_RR_velocity(theta, omega_a)
draw_RR_velocity(theta, omega_b)
```

```
J = [[-0.56464247 -0.90929743]
 [ 0.82533561 -0.41614684]]
v =  [-0.56464247  0.82533561]
J = [[-0.56464247 -0.90929743]
 [ 0.82533561 -0.41614684]]
v =  [-0.90929743 -0.41614684]
```



Answer each question below with one to sentences each.

1. What space do the two end-effector velocity vectors serve as a basis for?
2. What is the set of all velocities that is achievable by the end-effector of the robot?
3. What are the rank and nullity of the Jacobian matrix?

ENTER YOUR RESPONSES HERE

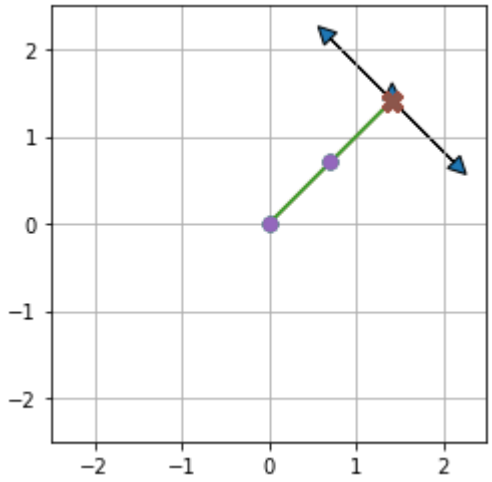
1. The two end-effector velocity vectors serve as a basis for $\mathbb{R}^2$.
2. The set of all velocities achievable by the end-effector is $\mathbf{v} \in \mathbb{R}^2$.
3. The rank of the Jacobian is 2, and its nullity is 0.

PART 2 (10 points)

Come up with a configuration (θ_1, θ_2) such that the columns of J are linearly dependent. Then come up with a basis vector for the null space of J. The end-effector velocities due to each component of the null space basis vector will be plotted separately below.

In [9]:
```
# FILL IN: joint angles you came up with
theta = np.array([np.pi / 4, np.pi / 4])

# FILL IN: basis vector for null J
ns = np.array([1.0, -1.0])

draw_RR_velocity(theta, np.array([ns[0],0]))
draw_RR_velocity(theta, np.array([0,ns[1]]))
```

```
J = [[-0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
v =  [-0.70710678  0.70710678]
J = [[-0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
v =  [ 0.70710678 -0.70710678]
J = [[-0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
v =  [0. 0.]
```



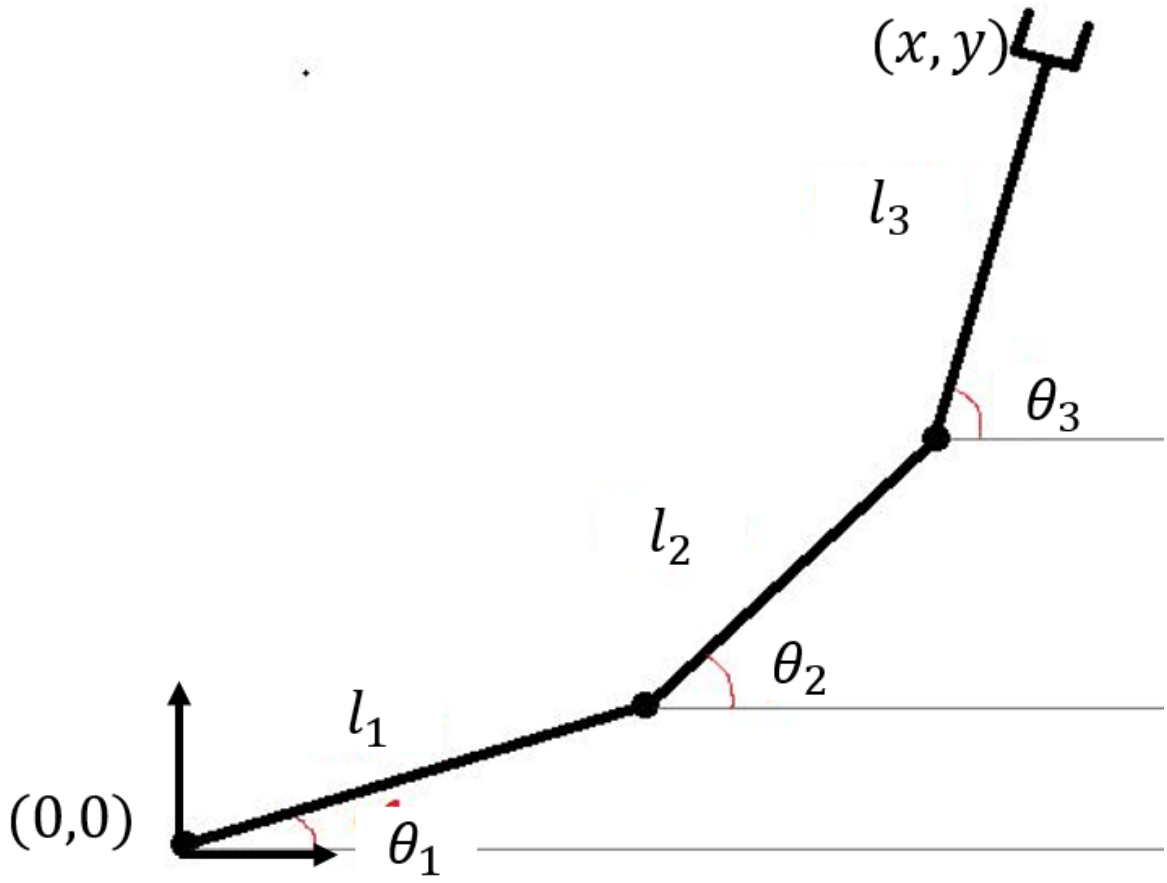
Explain why joint velocities in the null space lead to no movement at the end effector. As a hint, notice how the joint velocities derive from the null space vector components in the `draw` commands. Please make reference to the physical operation of the arm instead of mathematical definitions.

ENTER YOUR RESPONSE HERE

The two velocity vectors represent the translational velocity due to the angular motion of each portion of the arm. These joint velocities in the null space result in no net movement of the arm, as the translational velocity imparted by the angular velocity of the lower joint is cancelled out by the translational velocity imparted by the angular velocity of the upper joint.

PART 3 (15 points)

Let's add a third link to the robotic arm.



Now we have three joint inputs, and the mapping relating the joint velocities $(\omega_1, \omega_2, \omega_3)$ to (v_x, v_y) is as follows:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = J(\theta_1, \theta_2, \theta_3) \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} -l_1 \sin\theta_1 & -l_2 \sin\theta_2 & -l_3 \sin\theta_3 \\ l_1 \cos\theta_1 & l_2 \cos\theta_2 & l_3 \cos\theta_3 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

Code implementing the arm's kinematics assuming $l_1 = l_2 = l_3 = 1$ is provided below, as well as a new plotting function.

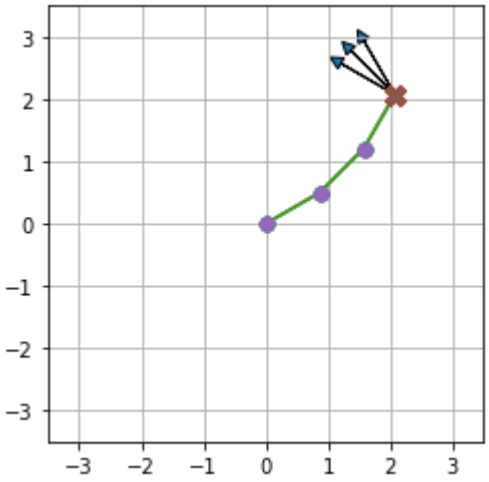
In [6]:
```
l1 = 1
l2 = 1
l3 = 1

def RRR_position(th):
  return np.array([[l1*np.cos(th[0]) + l2*np.cos(th[1]) + l3*np.cos(th[2])],
                   [l1*np.sin(th[0]) + l2*np.sin(th[1]) + l3*np.sin(th[2])]])

def RRR_jacobian(th):
  return np.array([[-l1*np.sin(th[0]), -l2*np.sin(th[1]), -l3*np.sin(th[2])],
                   [l1*np.cos(th[0]), l2*np.cos(th[1]), l3*np.cos(th[2])],])

def draw_RRR(th):
  plt.plot([0, l1*np.cos(th[0]), l1*np.cos(th[0]) + l2*np.cos(th[1]),
            l1*np.cos(th[0]) + l2*np.cos(th[1]) + l3*np.cos(th[2])],
           [0, l1*np.sin(th[0]), l1*np.sin(th[0]) + l2*np.sin(th[1]),
            l1*np.sin(th[0]) + l2*np.sin(th[1]) + l3*np.sin(th[2])])
```

```
  plt.scatter([0, l1*np.cos(th[0]), l1*np.cos(th[0]) + l2*np.cos(th[1])],
              [0, l1*np.sin(th[0]), l1*np.sin(th[0]) + l2*np.sin(th[1])],
              zorder=3, s=50)
  plt.scatter([l1*np.cos(th[0]) + l2*np.cos(th[1]) + l3*np.cos(th[2])],
              [l1*np.sin(th[0]) + l2*np.sin(th[1]) + l3*np.sin(th[2])],
              marker='X', zorder=3, s=100)
  plt.grid()
  plt.axis('scaled')
  plt.xlim([-3.5,3.5])
  plt.ylim([-3.5,3.5])

def draw_RRR_velocity(theta, omega):
  EE = RRR_position(theta)
  J = RRR_jacobian(theta)
  v = J @ omega
  print('J =', J)
  print('v = ', v)
  draw_RRR(theta)
  plt.arrow(EE[0,0], EE[1,0], v[0], v[1], head_width=0.2, head_length=0.2)
```

Suppose the arm is in the configuration $(\theta_1, \theta_2, \theta_3) = (\frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3})$. As in Part 1, come up with *three* sets of joint velocities $(\omega_1, \omega_2, \omega_3)$ so that the resultant end-effector velocities (v_x, v_y) are equal to the columns of J. Then plot all of the vectors on the same plot.

In [7]:
```
theta = np.array([np.pi/6, np.pi/4, np.pi/3])

# FILL IN: first set of joint velocity vectors
omega_a = np.array([1.0, 0.0, 0.0])

# FILL IN: second set of joint velocity vectors
omega_b = np.array([0.0, 1.0, 0.0])

# FILL IN: second set of joint velocity vectors
omega_c = np.array([0.0, 0.0, 1.0])

draw_RRR_velocity(theta, omega_a)
draw_RRR_velocity(theta, omega_b)
draw_RRR_velocity(theta, omega_c)
```

```
J = [[-0.5        -0.70710678 -0.8660254 ]
 [ 0.8660254   0.70710678  0.5       ]]
v =  [-0.5        0.8660254]
J = [[-0.5        -0.70710678 -0.8660254 ]
 [ 0.8660254   0.70710678  0.5       ]]
v =  [-0.70710678  0.70710678]
J = [[-0.5        -0.70710678 -0.8660254 ]
 [ 0.8660254   0.70710678  0.5       ]]
v =  [-0.8660254  0.5      ]
```



Answer each question below with one to sentences each.

1. Are the three end-effector velocity vectors linearly independent?
2. What is the column space of J?
3. Does the addition of the third joint change the end-effector velocity capabilities of the two-link robot manipulator? Why or why not?

ENTER YOUR RESPONSES HERE

1. These vectors are not linearly independent. Because we can choose two columns that are linearly independent, their span would be $\mathbb{R}^2$. Therefore, the third column would not be linearly independent.

2. The column space of J is $\mathbb{R}^2$.

3. Because the column space of J is $\mathbb{R}^2$, no new velocity vectors can be created from the angular velocity of the third joint.

PART 4 (10 points)

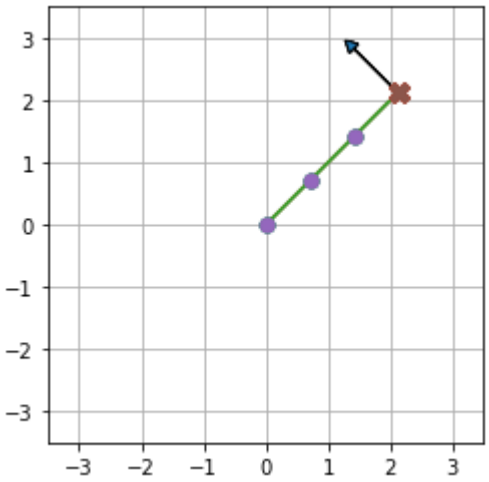
Write code below to find a basis for the null space of the Jacobian matrix in the configuration $(\theta_1, \theta_2, \theta_3) = (\frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3})$. You can use scipy.linalg.null_space. Then come up with a new configuration $(\theta_1, \theta_2, \theta_3)$ so that the nullity of the new Jacobian is larger than in the original configuration, and print the null space basis.

In [11]:
```
from scipy.linalg import null_space

theta = np.array([np.pi/6, np.pi/4, np.pi/3])
# TO DO: compute and print out a basis for the Jacobian null space

a = RRR_jacobian(theta)
ns = null_space(a)

print(ns)
```

```
[[ 0.41768125]
 [-0.80689822]
 [ 0.41768125]]
```

In [16]:
```
# TO DO: come up with a new configuration so that the nullity is larger than before,
#        and compute and print the null space basis
theta = np.array([np.pi / 4, np.pi / 4, np.pi / 4])

a = RRR_jacobian(theta)
ns = null_space(a)

print(ns)

# assume same joint velocity vectors omega_{a,b,c} from Part 3
draw_RRR_velocity(theta, omega_a)
draw_RRR_velocity(theta, omega_b)
draw_RRR_velocity(theta, omega_c)
```

```
[[ 0.81649658  0.        ]
 [-0.40824829 -0.70710678]
 [-0.40824829  0.70710678]]
J = [[-0.70710678 -0.70710678 -0.70710678]
 [ 0.70710678  0.70710678  0.70710678]]
v =  [-0.70710678  0.70710678]
J = [[-0.70710678 -0.70710678 -0.70710678]
 [ 0.70710678  0.70710678  0.70710678]]
v =  [-0.70710678  0.70710678]
J = [[-0.70710678 -0.70710678 -0.70710678]
 [ 0.70710678  0.70710678  0.70710678]]
v =  [-0.70710678  0.70710678]
```



The last cell plots the robot arm in the configuration you came up with, along with the end-effector velocity vectors using the same joint velocities you came up with in Part 3.

1. Does the robot have more or fewer end-effector velocity capabilities when the Jacobian has a higher nullity?

2. Does the robot have more or fewer "useless" joint velocities that lead to zero end-effector movement when the Jacobian has a higher nullity?

ENTER YOUR RESPONSES HERE

1. It would have fewer end-effector velocity capabilities, as more angular velocity combinations result in net zero velocity.

2. It would have more "useless" joint velocities, as more angular velocities, when multiplied by the Jacobian, would result in no translational velocity.