pivot_chain

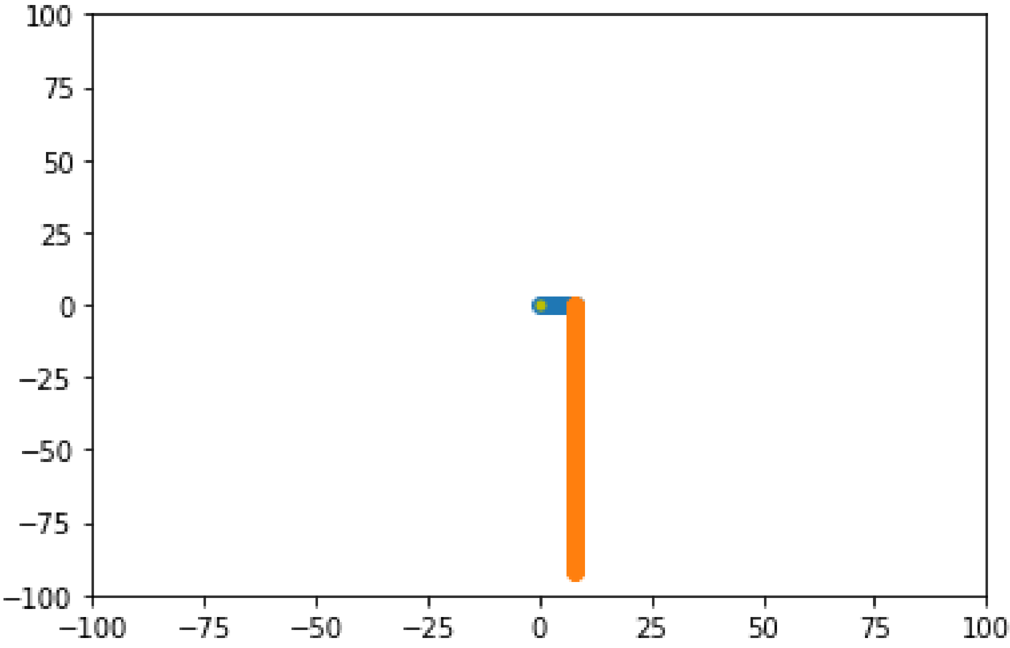
February 22, 2018

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

        n = 100
        straight_line = np.zeros((n+1,2), dtype=int)
        for i in range(n+1):
            straight_line[i][0] = i


        def draw(path, last_pivot_node=n):
            x = path[:,0]
            y = path[:,1]
            plt.plot(x[:last_pivot_node+1], y[:last_pivot_node+1], '-o')
            plt.plot(x[last_pivot_node:], y[last_pivot_node:], '-8')
            plt.plot(0, 0, 'y.') # yellow dot for (0.0)
            plt.xlim(-n,n)
            plt.ylim(-n,n)
            # plt.savefig(img_filename.format(t))
            plt.show()


        draw(straight_line)
```

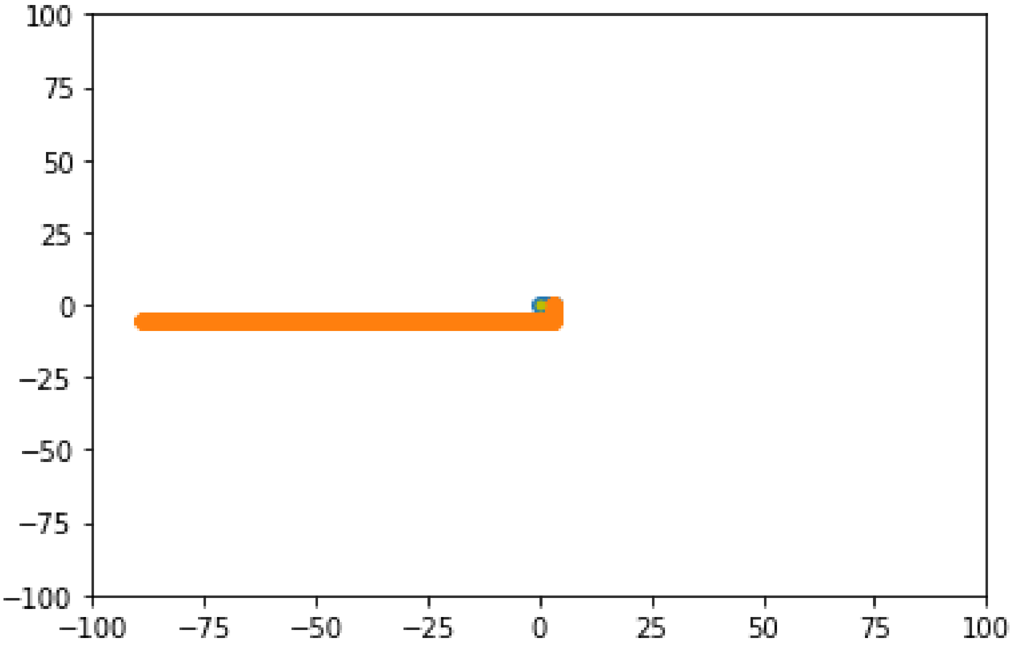
```
<matplotlib.figure.Figure at 0x7f389814e250>
```

```
In [2]: # example of implementation of the rotation by 90 degrees
        def rotate_90_after_node(old_path, k):
            new_path = old_path.copy()
            node = old_path[k]
            for j in range(k,n+1): # apply rotation to every node after k
                new_path[j] = node + np.array([[0,1],[-1,0]]).dot(old_path[j] - node)
            return new_path
```

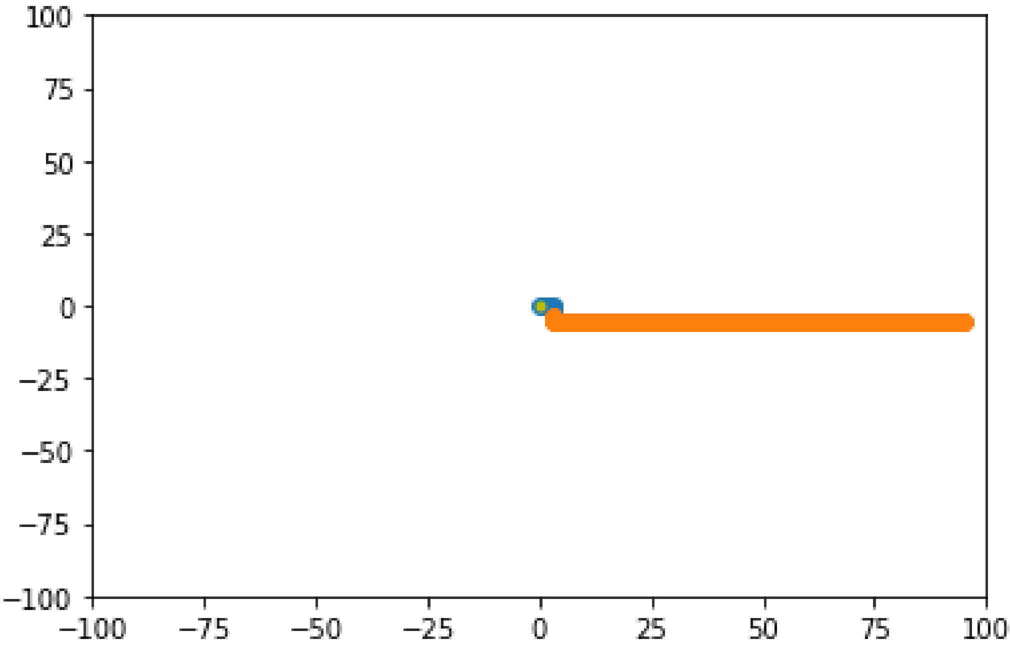
```
In [3]: path_1 = rotate_90_after_node(straight_line, 8)
        draw(path_1, 8)
```



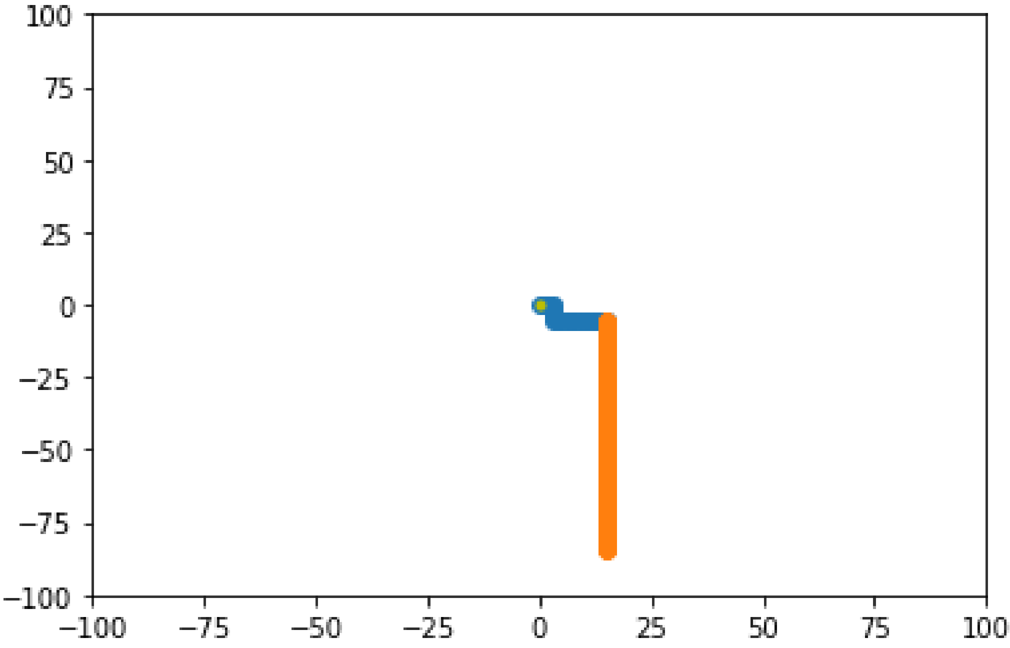
```
In [4]: path_2 = rotate_90_after_node(path_1, 3)
        draw(path_2, 3)
```

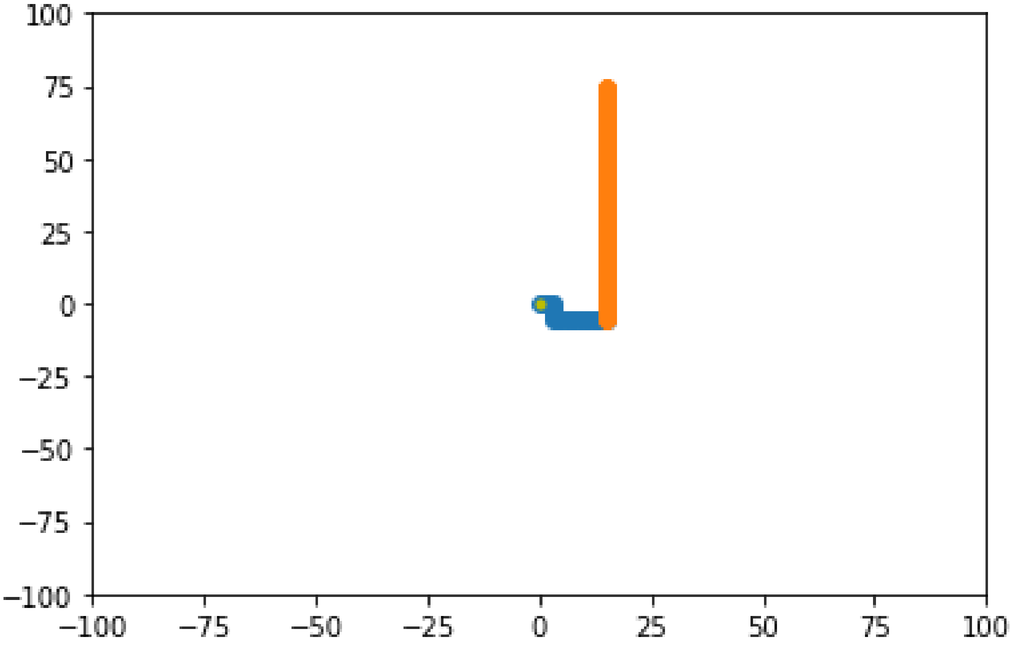


```
In [5]: def symmetry_y_axis_after_node(old_path, k):
            new_path = old_path.copy()
            node = old_path[k]
            for j in range(k,n+1): # apply to any node after k
                new_path[j] = node + np.array([[-1,0],[0,1]]).dot(old_path[j] - node)
            return new_path


        path_3 = symmetry_y_axis_after_node(path_2, 7)
        draw(path_3, 7)
```



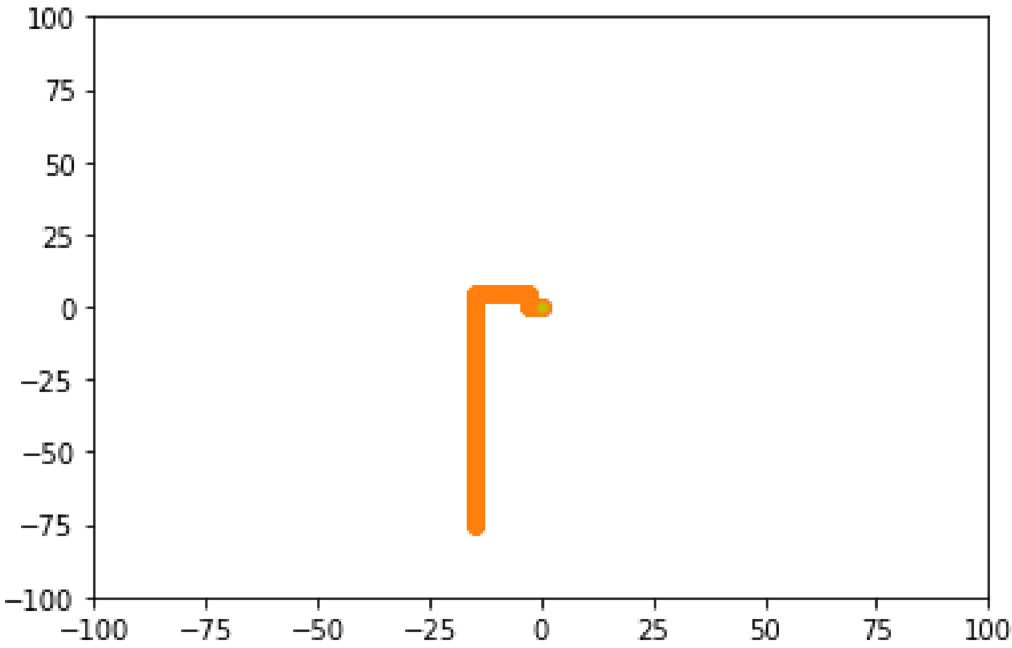
```
In [6]: def symmetry_x_axis_after_node(old_path, k):
            new_path = old_path.copy()
            node = old_path[k]
            for j in range(k, n+1):
                new_path[j] = node + np.array([[1, 0], [0, -1]]).dot(old_path[j] - node)
            return new_path

In [7]: path_4 = rotate_90_after_node(path_3, 20)
        draw(path_4, 20)
        path_4 = symmetry_x_axis_after_node(path_4, 20)
        draw(path_4, 20)
```





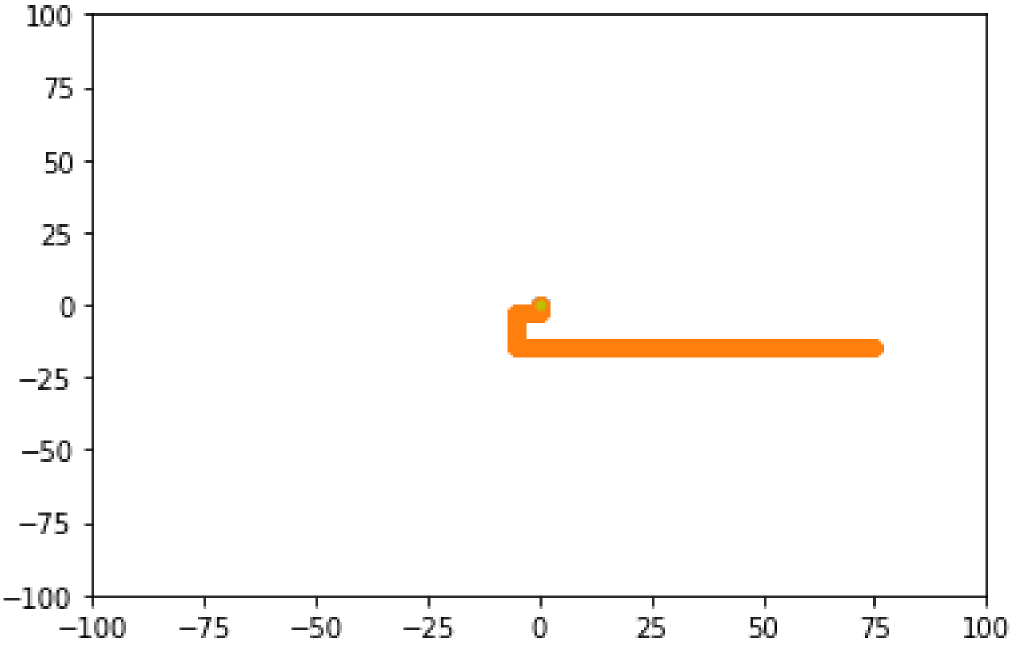
```
In [8]: def rotate_180_after_node(old_path, k):
            new_path = old_path.copy()
            node = old_path[k]
            for j in range(k, n+1):
                new_path[j] = node + np.array([[-1, 0], [0, -1]]).dot(old_path[j] - node)
            return new_path
```

```
In [9]: path_5 = rotate_180_after_node(path_4, 0)
        draw(path_5, 0)
```



```
In [10]: def rotate_270_after_node(old_path, k):
             new_path = old_path.copy()
             node = old_path[k]
             for j in range(k, n+1):
                 new_path[j] = node + np.array([[0, -1], [1, 0]]).dot(old_path[j] - node)
             return new_path
```

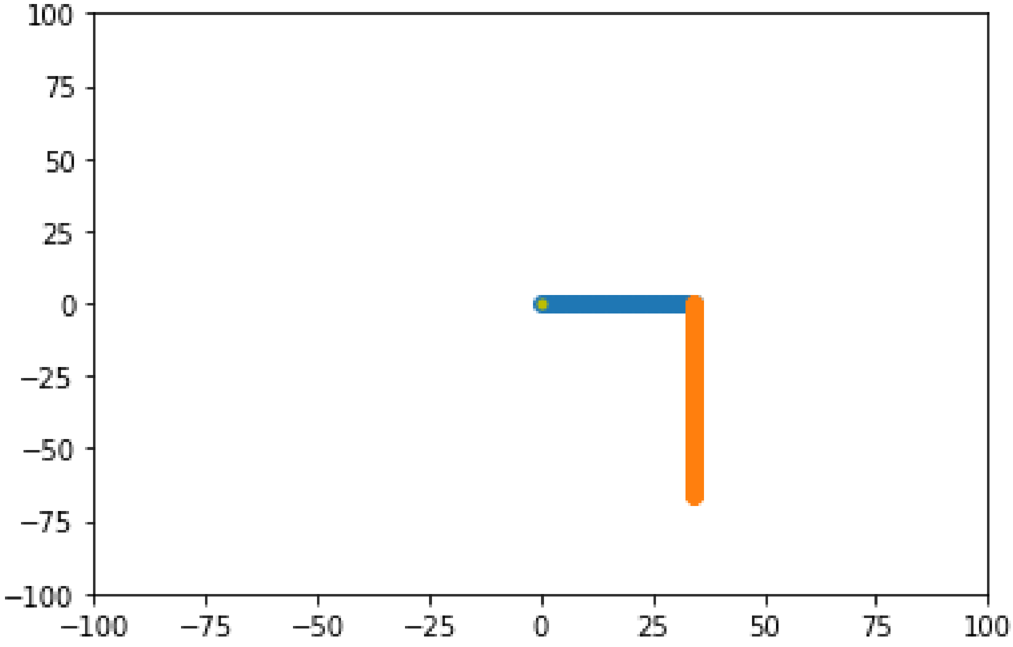
```
In [11]: path_6 = rotate_270_after_node(path_5, 0)
         draw(path_6, 0)
```

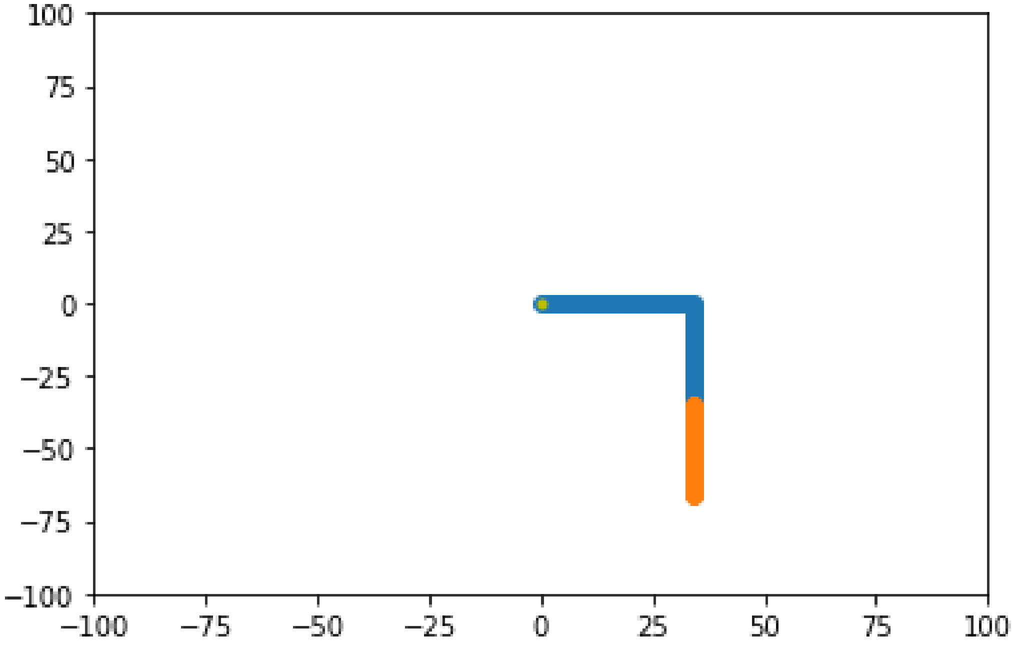


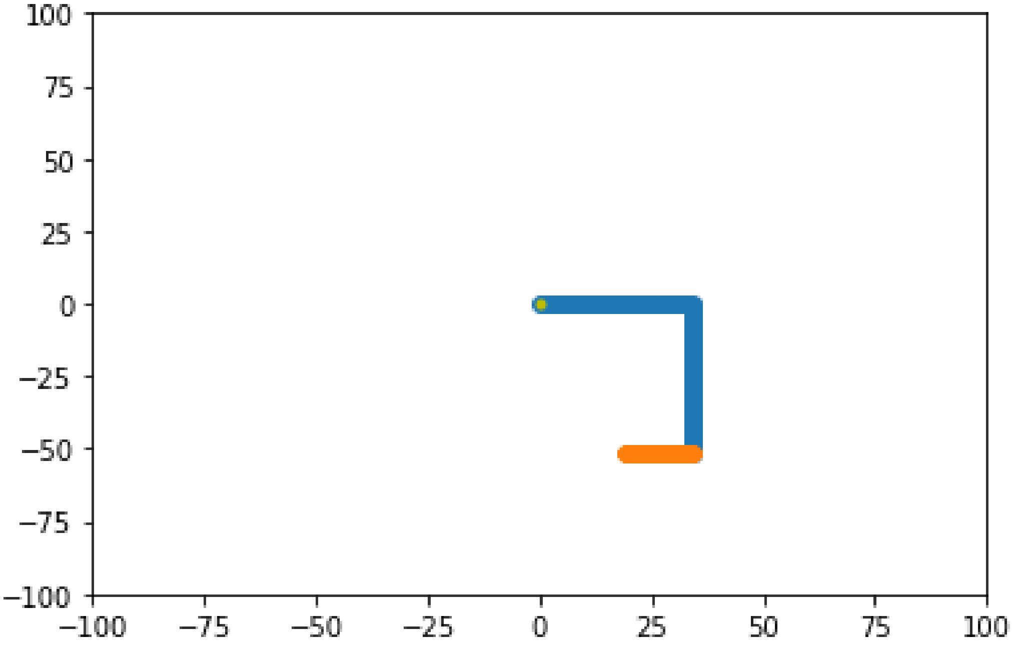
```
In [12]: def is_valid(path): # validate the path where the matrix is applied to all nodes afte
             positions = []
             for j in range(n+1):
                 positions.append(tuple(path[j]))
                 if j < n:
                     if not np.linalg.norm(path[j] - path[j+1]) == 1.0:
                         return False # the path is not continuous
             if len(positions) != len(set(positions)):
                 return False # some positoins are visited several times by the path, it is no
             return True
```

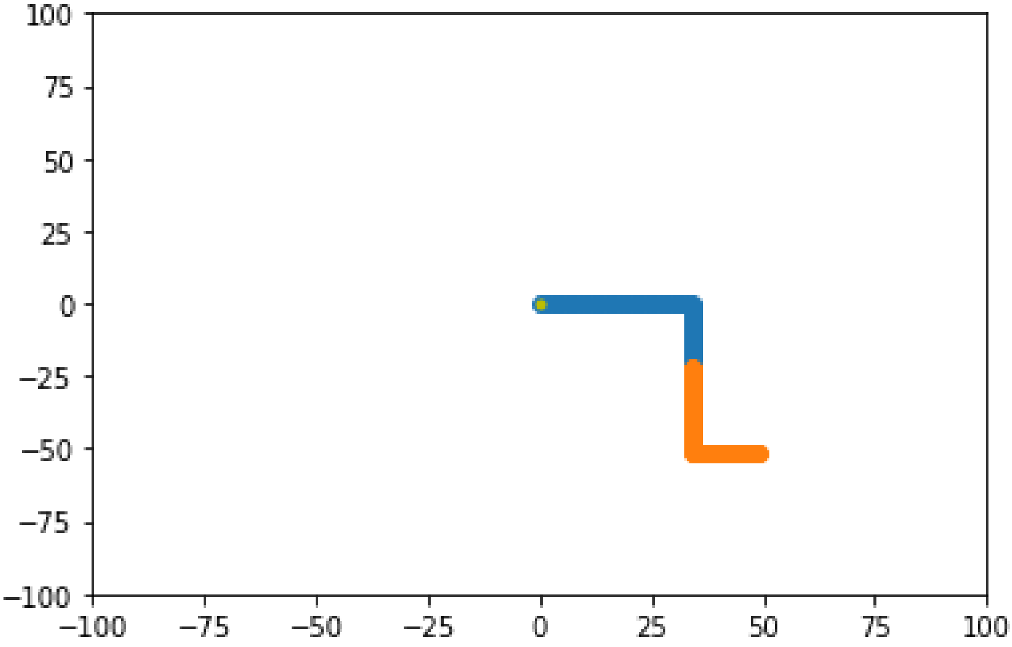
```
In [13]: # Apply a random transformation uniformly
         current_path = straight_line # starting from a straight line

         for t in range(0,1000):
             random_node = random.choice(range(n+1))
             random_transformation = random.choice([symmetry_y_axis_after_node, rotate_90_afte
             #print random_node, "was picked at step", t
             new_path = random_transformation(current_path, random_node)
             if is_valid(new_path):
                 #print "valid path!"
                 current_path = new_path
                 if t < 20:
                     draw(current_path, random_node)
             else:
```

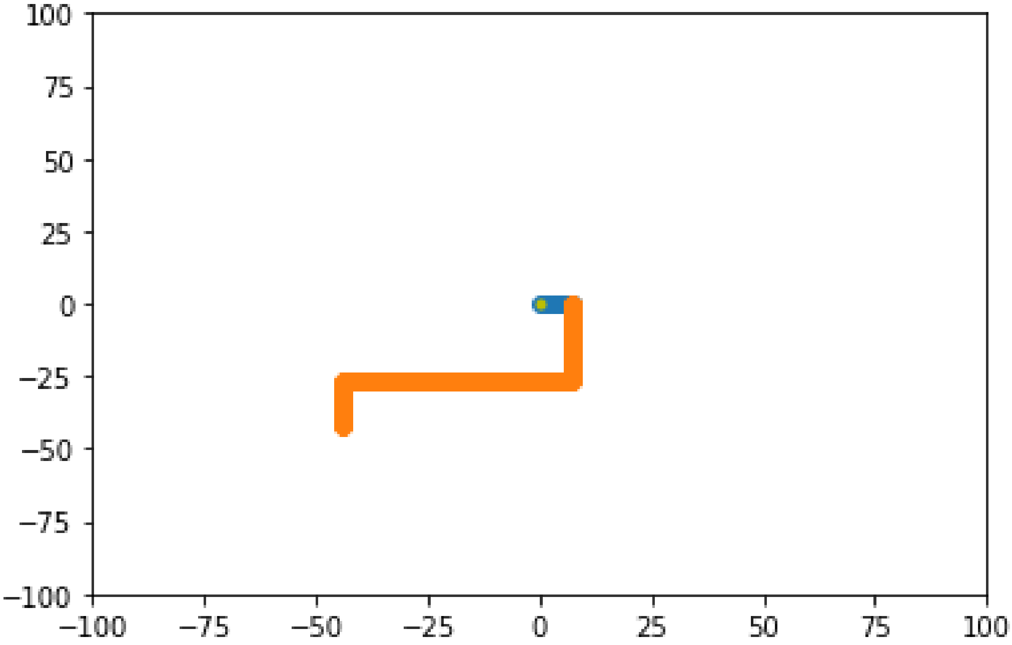
```
            #print "invalid path"
            pass
```

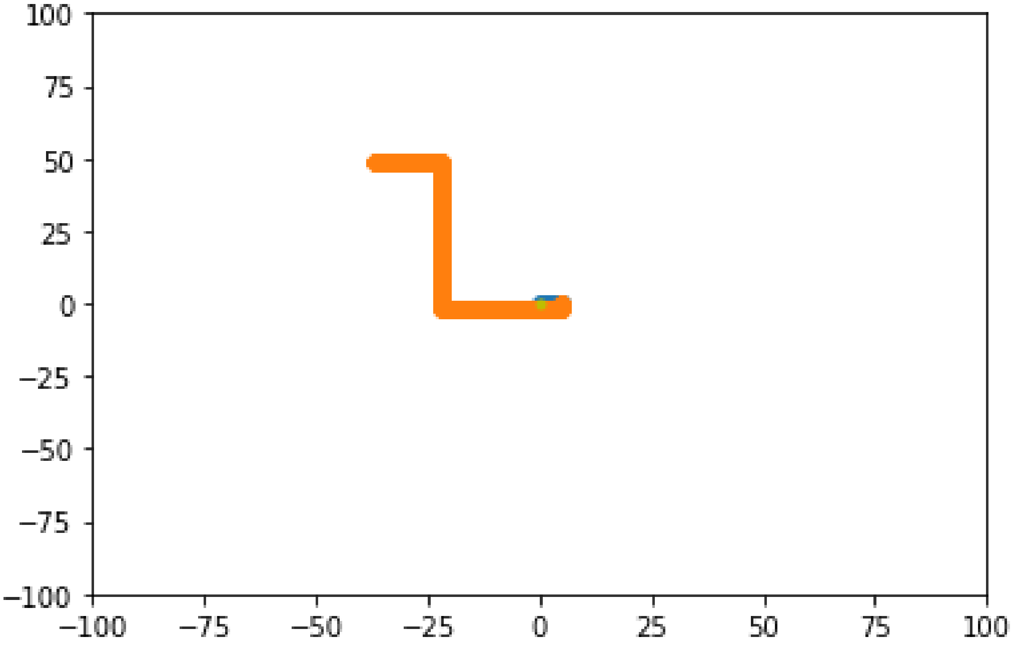


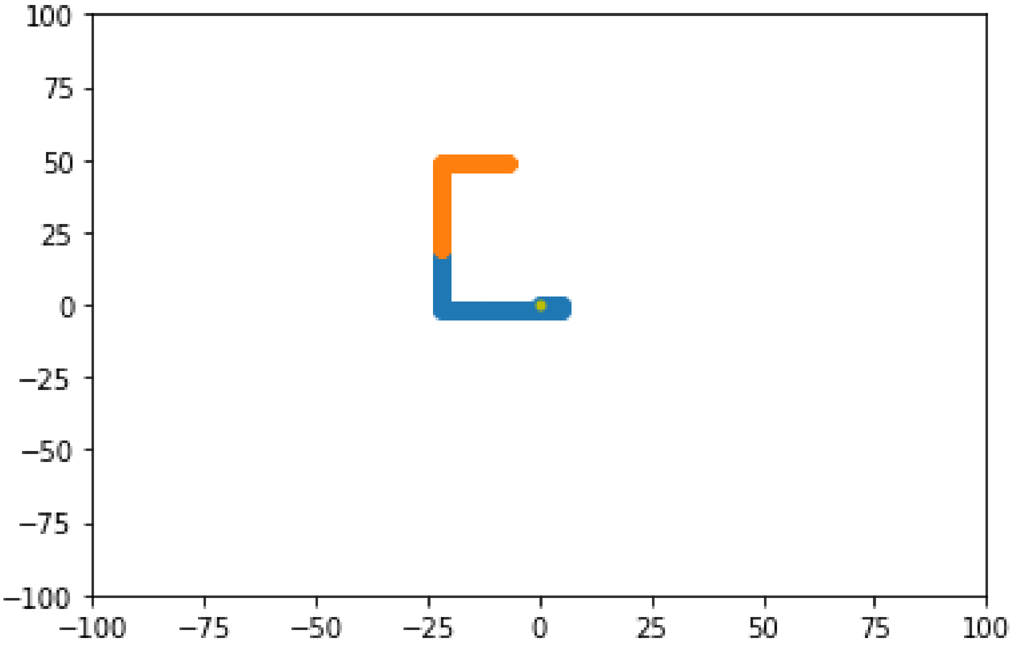


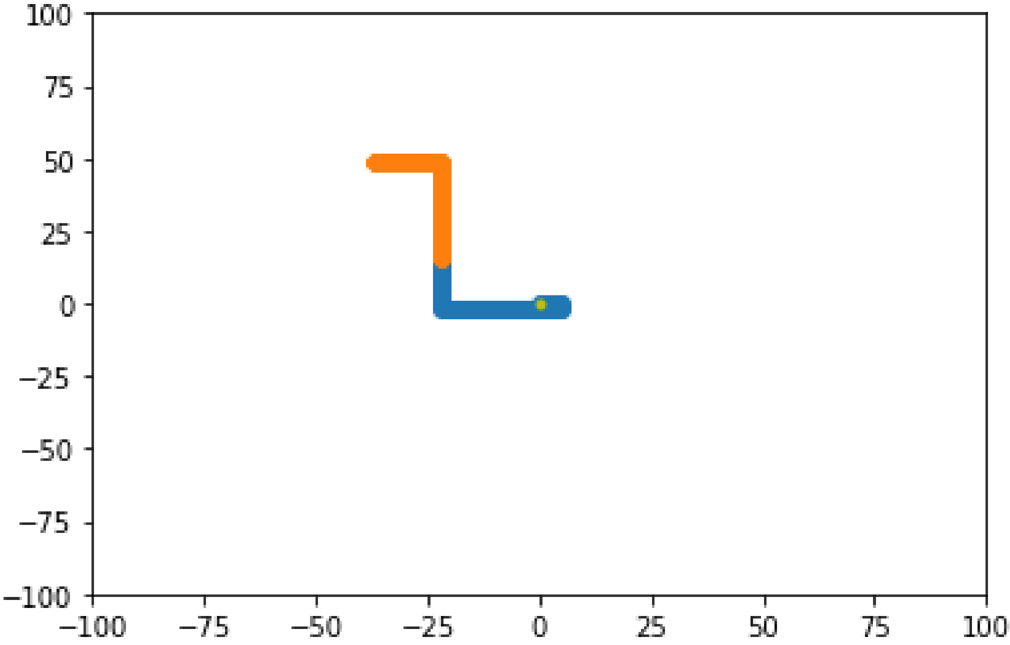

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

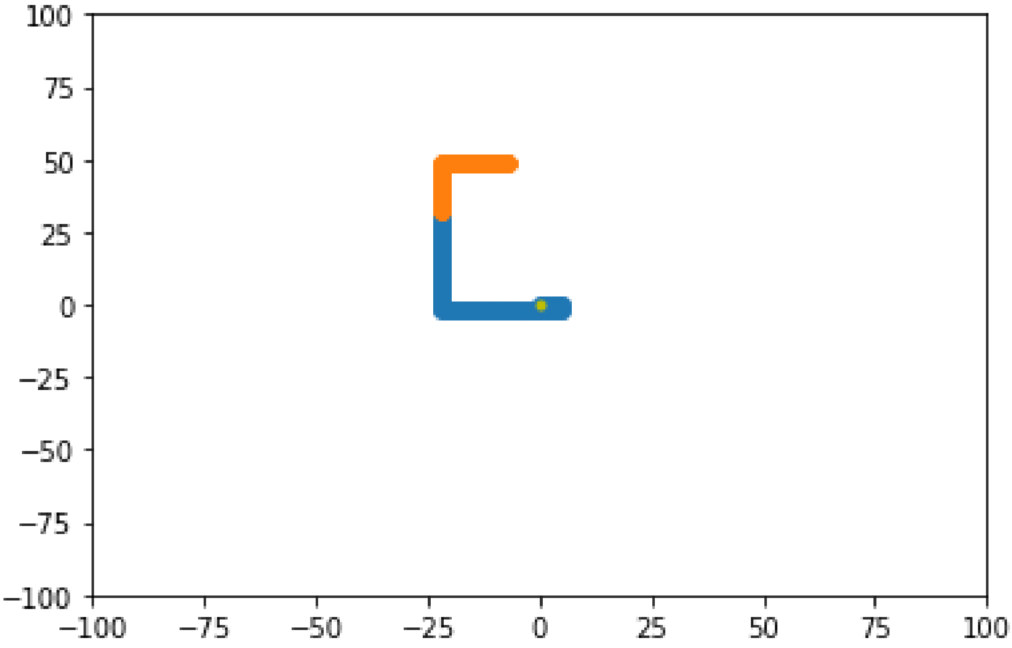

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

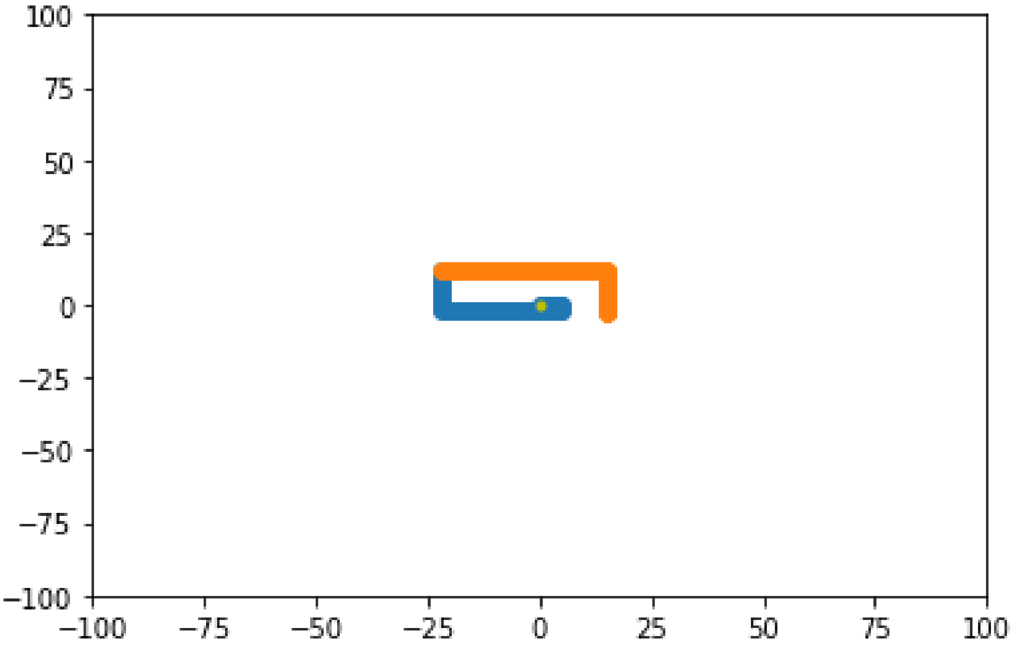

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

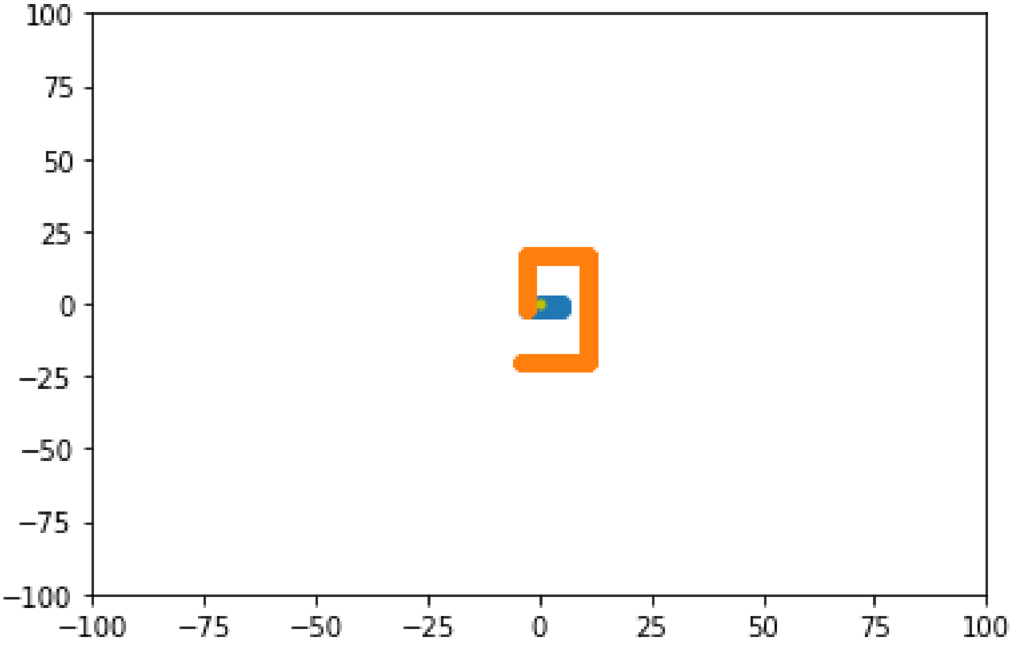

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

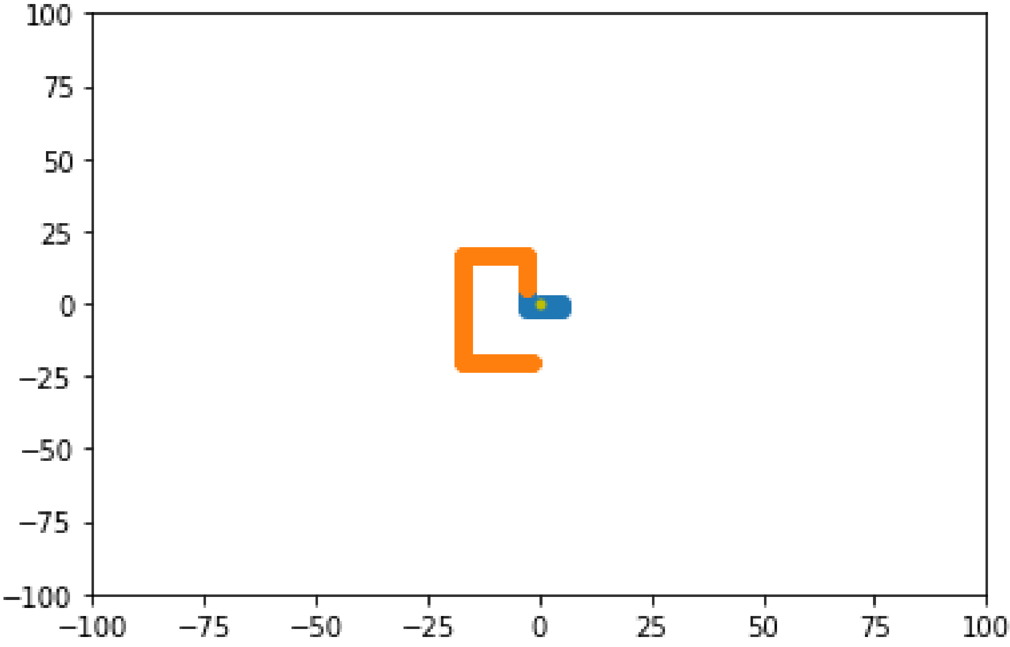

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

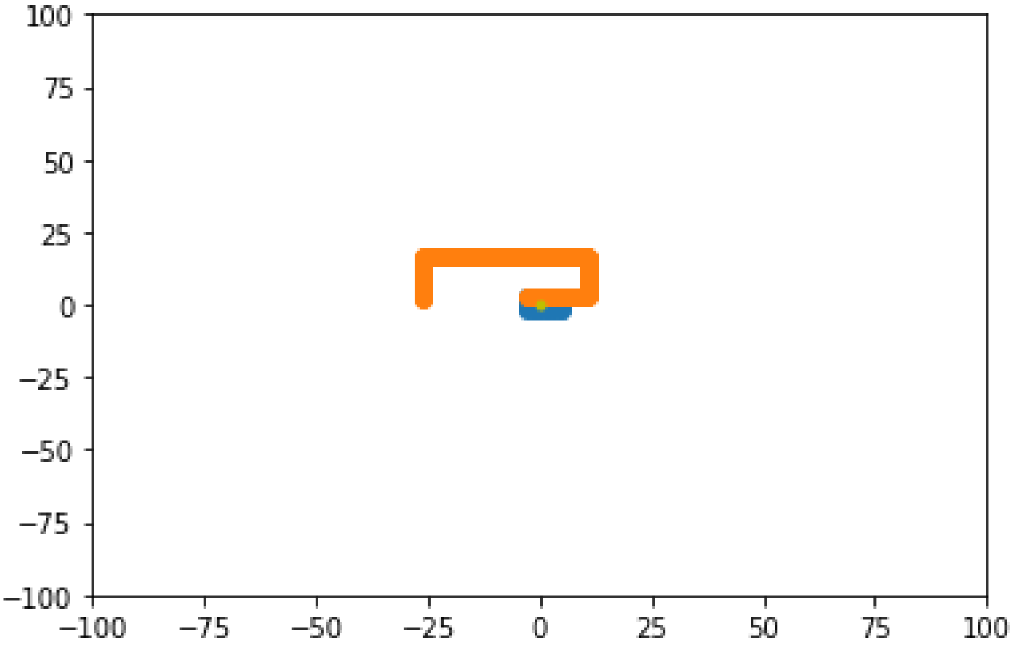

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

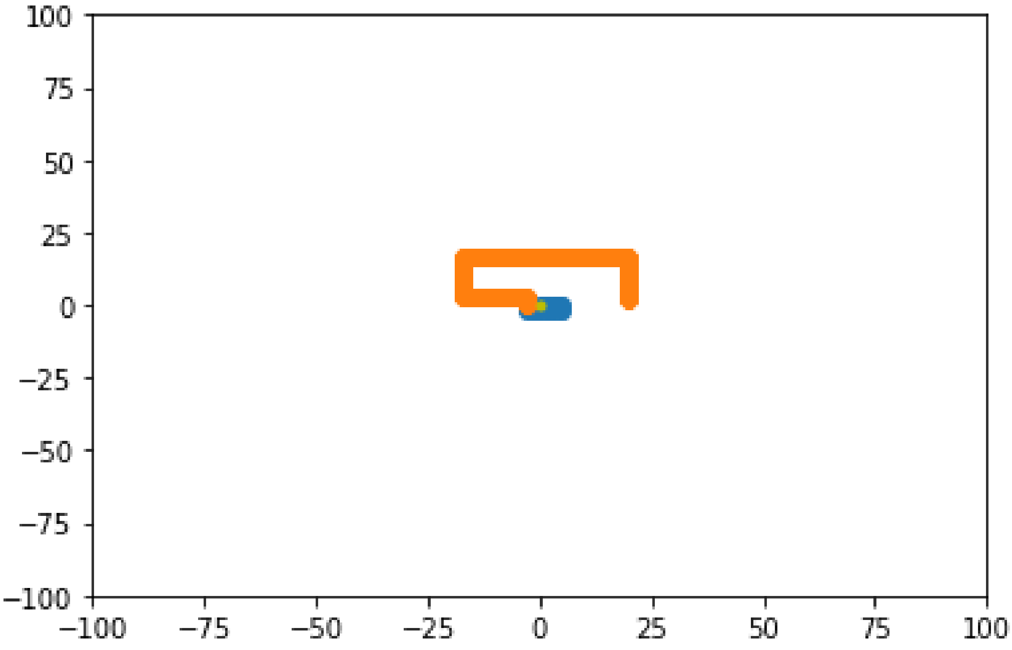

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

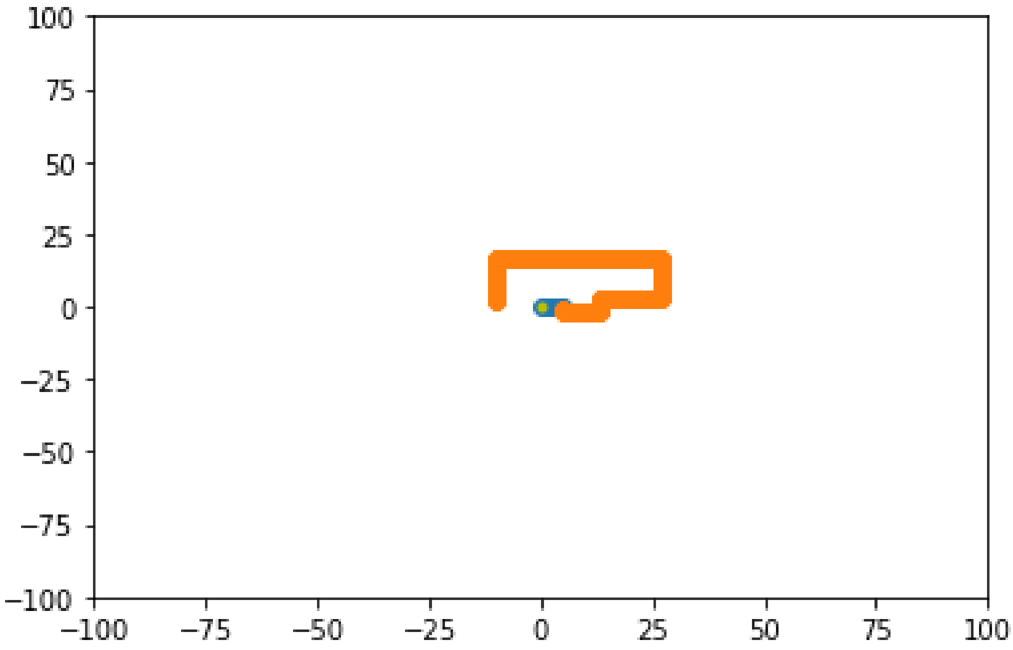

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100



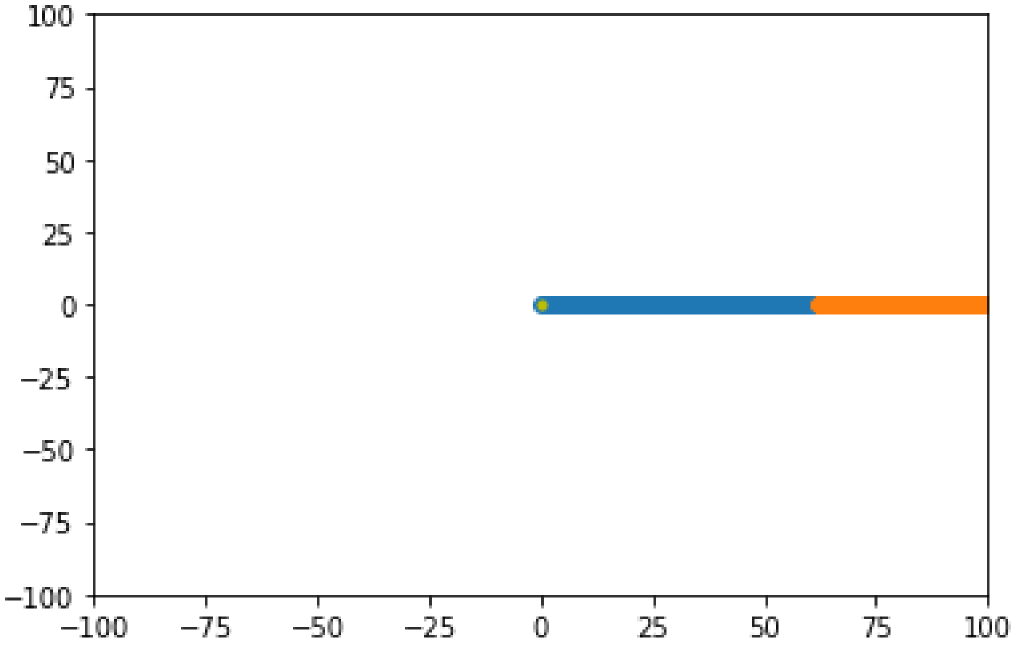
```
In [14]: # Quantity described in part (a) of problem 3.
         def quantity_a(path):
             return np.linalg.norm(path[n])
```

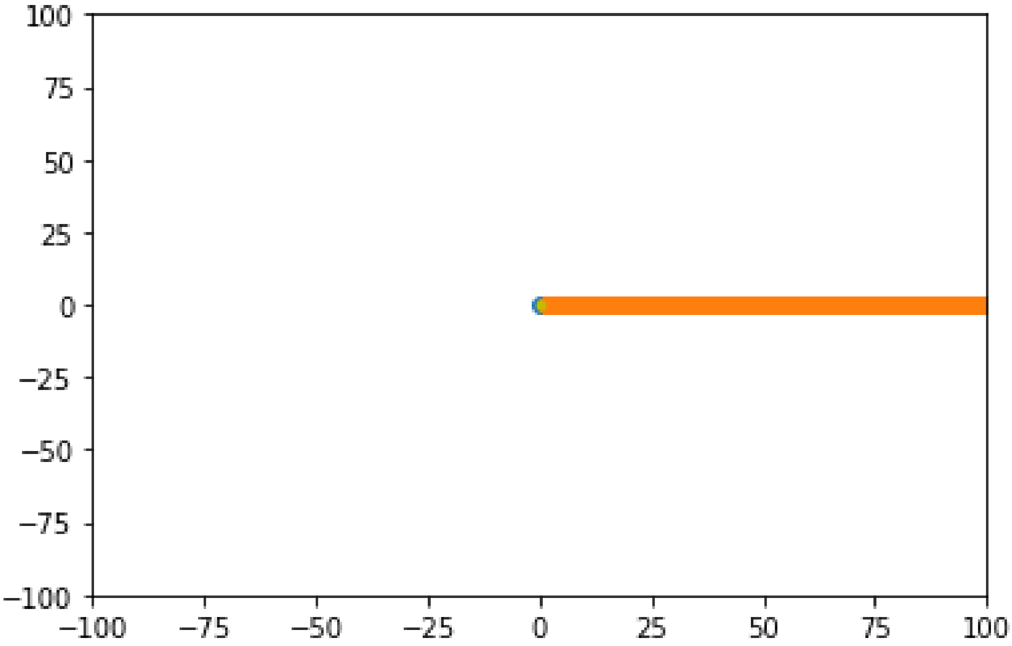
```
In [15]: # Quantity described in part (b) of problem 3.
         def quantity_b(path):
             combos = itertools.combinations(path, 2)
             distances = [np.linalg.norm(c[0] - c[1]) for c in combos]
             return max(distances)
```

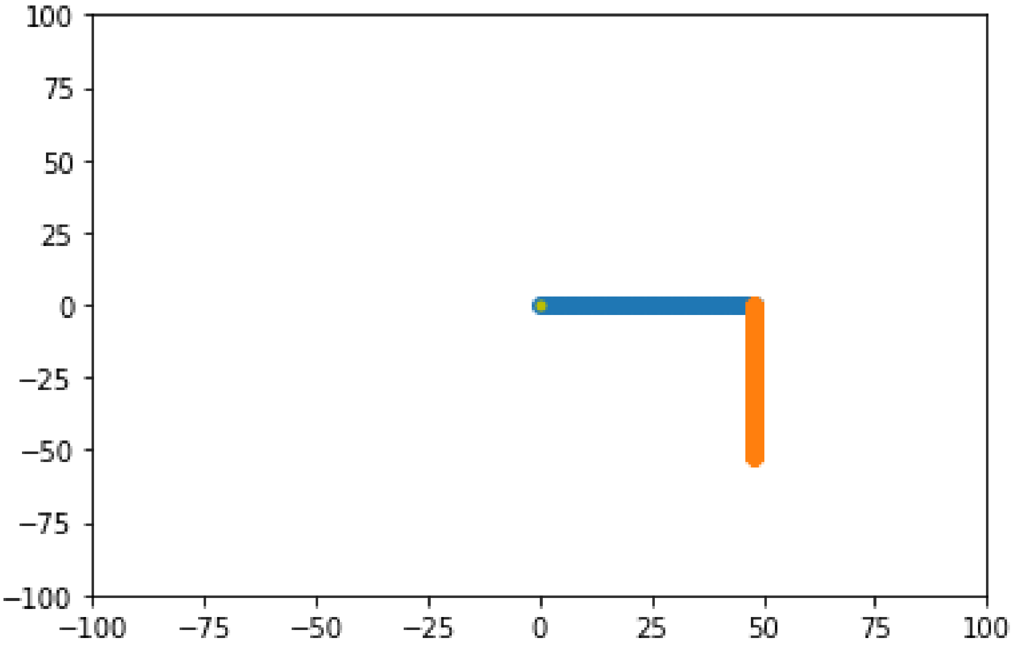
```
In [16]: # Quantity described in part (c) of problem 3.
         def quantity_c(path):
             count = 0
             for j in range(1, n):
                 for i in range(2):
                     if abs(path[j][i] - path[j-1][i]) == 1 and abs(path[j][i] - path[j+1][i])
                         count += 1
             return count
```

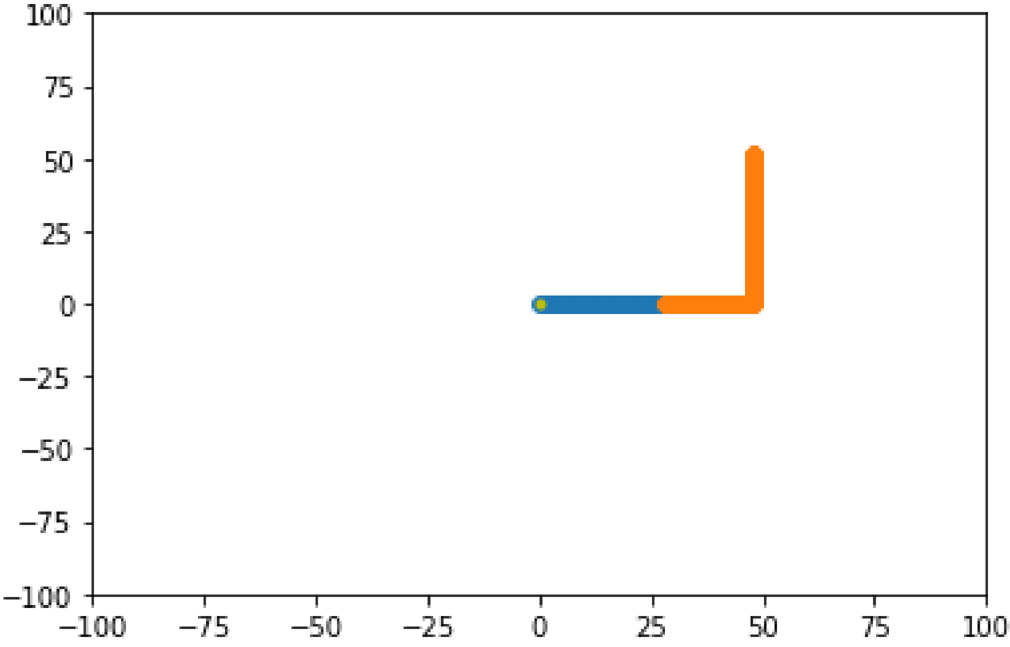
```
In [17]: # Run the pivot chain over the space of self-avoiding paths of length n.

         T = 10000  # Number of time steps.

         current_path = straight_line
         total_a = total_b = total_c = 0
         for t in range(T):
```

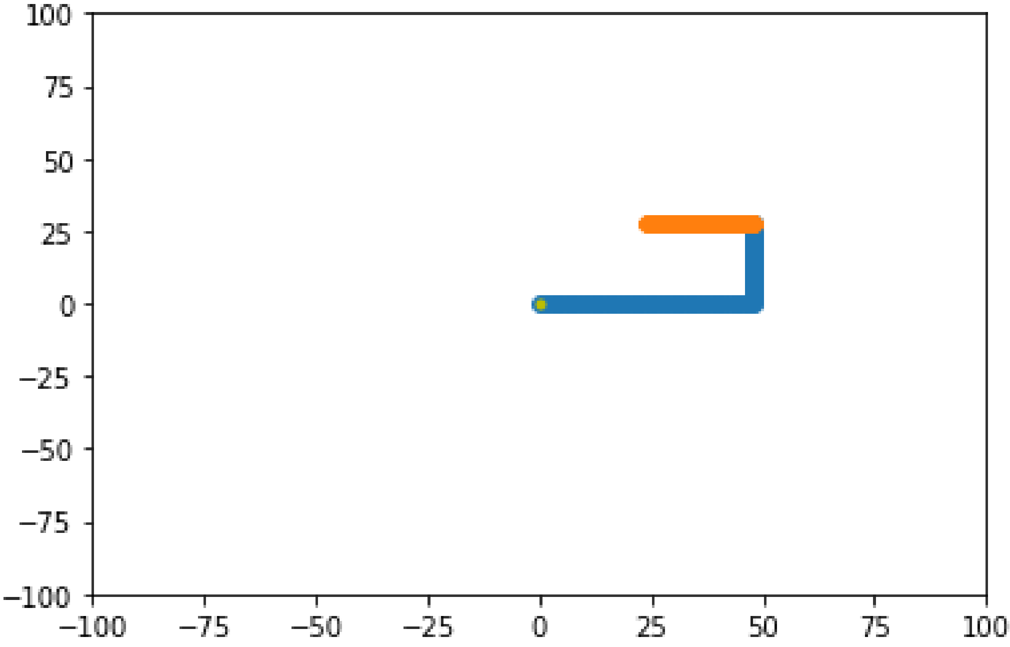
```
# Update the running total of the quantities of interest.
total_a += quantity_a(current_path)
total_b += quantity_b(current_path)
total_c += quantity_c(current_path)

# Choose a vertex and a transformation, both uniformly at random.
k = random.choice(range(n+1))
transformation = random.choice([symmetry_x_axis_after_node,
                                symmetry_y_axis_after_node,
                                rotate_90_after_node,
                                rotate_180_after_node,
                                rotate_270_after_node])

# Update the path.
new_path = transformation(current_path, k)
if is_valid(new_path):
    # If the new path is valid, make it the current path.
    current_path = new_path
    if t < 20:
        draw(current_path, k)
```

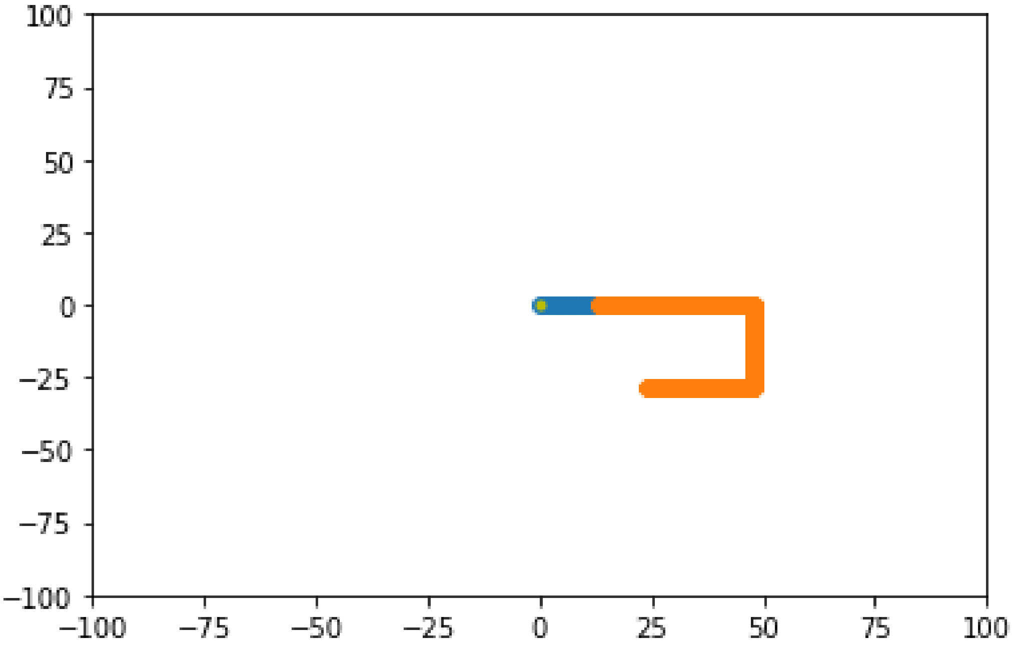


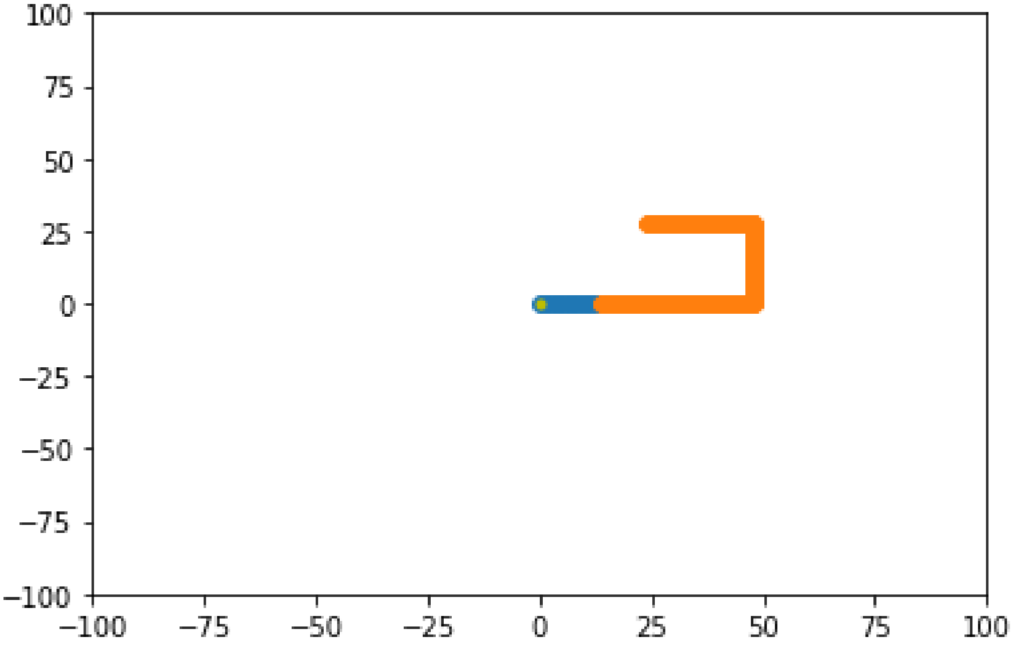

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

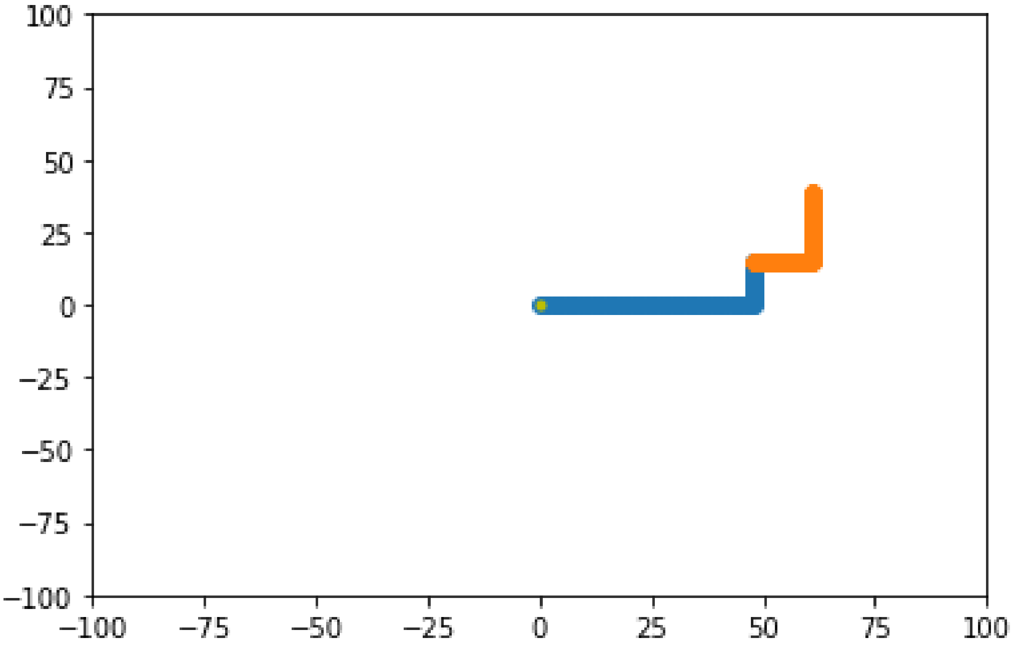

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

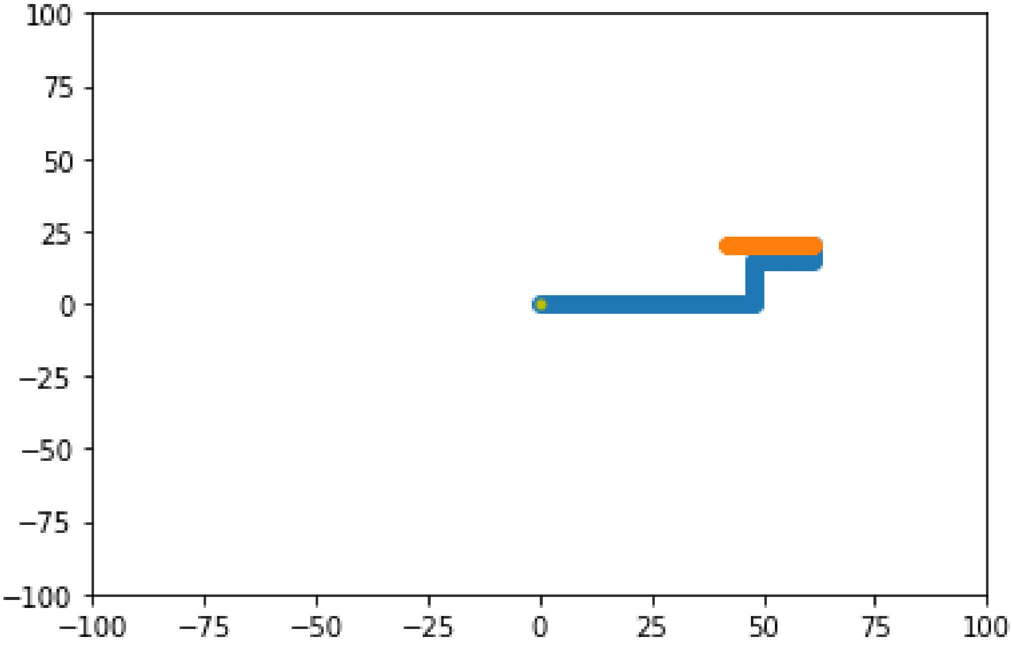

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

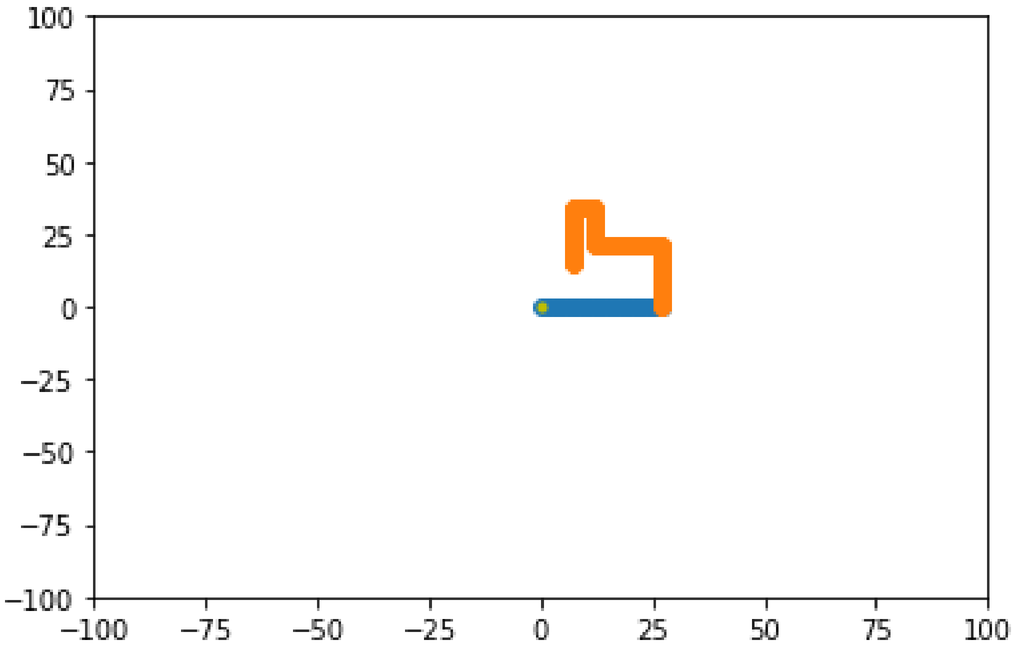

100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100


100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100



```
In [18]: print "Quantity a average:", (float(total_a) / T)
         print "Quantity b average:", (float(total_b) / T)
         print "Quantity c average:", (float(total_c) / T)
```

```
Quantity a average: 26.0402972349
Quantity b average: 32.1169655385
Quantity c average: 41.0481
```