# ch11a_line_and_fillet_path_manager

April 1, 2022

## 0.1 Problem 0: Path manager implementation

Implement the following and ensure the unit tests pass. * chap11/path_manager_utilities.py: * inHalfSpace(...) * chap11/line_manager.py: * construct_line(...) * line_manager(...) * chap11/fillet_manager.py: * construct_fillet_line(...) * construct_fillet_circle(...) * fillet_manager(...)

### 0.1.1 Hints on implementation

- np.linalg.norm(...) will be very useful for computing the norm of a vector
- Make sure to set the airspeed on your paths. The airspeed can be found using path.airspeed = get_airspeed(waypoints, ptr)
- Problems 1 and 2 help guide you to accounting for singularities
- The path manager does not clear after use, so you will see the previous paths on the display if you run all of them sequentially.

### 0.1.2 Note on the unit tests

There will be no unit tests for the line_manager(...) and fillet_manager(...) functions. Furthermore, there are several scenarios where dividing by zero becomes an issue. As these are singularities that can be handled in various ways, there is not a correct answer. Thus, these cases may not all arise in the unit tests, but they will be tested in the problems 1 and 2.

[1]:

```python
import mav_sim.parameters.planner_parameters as PLAN
import numpy as np
from mav_sim.chap3.mav_dynamics import DynamicState
from mav_sim.chap11.run_sim import run_sim
from mav_sim.message_types.msg_sim_params import MsgSimParams
from mav_sim.message_types.msg_waypoints import MsgWaypoints

from mav_sim.chap11.waypoint_viewer import WaypointViewer
from mav_sim.chap3.data_viewer import DataViewer
from mav_sim.tools.display_figures import display_data_view, display_mav_view

# The viewers need to be initialized once due to restart issues with qtgraph
if 'path_view' not in globals():
    print("Initializing waypoint viewer")
    global waypoint_view
    waypoint_view = WaypointViewer()
```

```
if 'data_view' not in globals():
    print("Initializing data_view")
    global data_view
    data_view = DataViewer()

# Initialize the simulation parameters
sim_params_default = MsgSimParams(end_time=130., video_name="cha11.avi") # Sim␣
 ↪ending in 10 seconds
state = DynamicState()

# Function for running simulation and displaying results
def run_sim_and_display(waypoints: MsgWaypoints, sim_params: MsgSimParams =␣
 ↪sim_params_default):
    global waypoint_view
    global data_view
    data_view.reset(sim_params.start_time)
    (waypoint_view, data_view) = run_sim(sim=sim_params, waypoints=waypoints,␣
 ↪init_state=state, waypoint_view=waypoint_view, data_view=data_view)
    display_data_view(data_view)
    display_mav_view(waypoint_view)
```

```
136.62760706610337
Initializing waypoint viewer
Initializing data_view
```

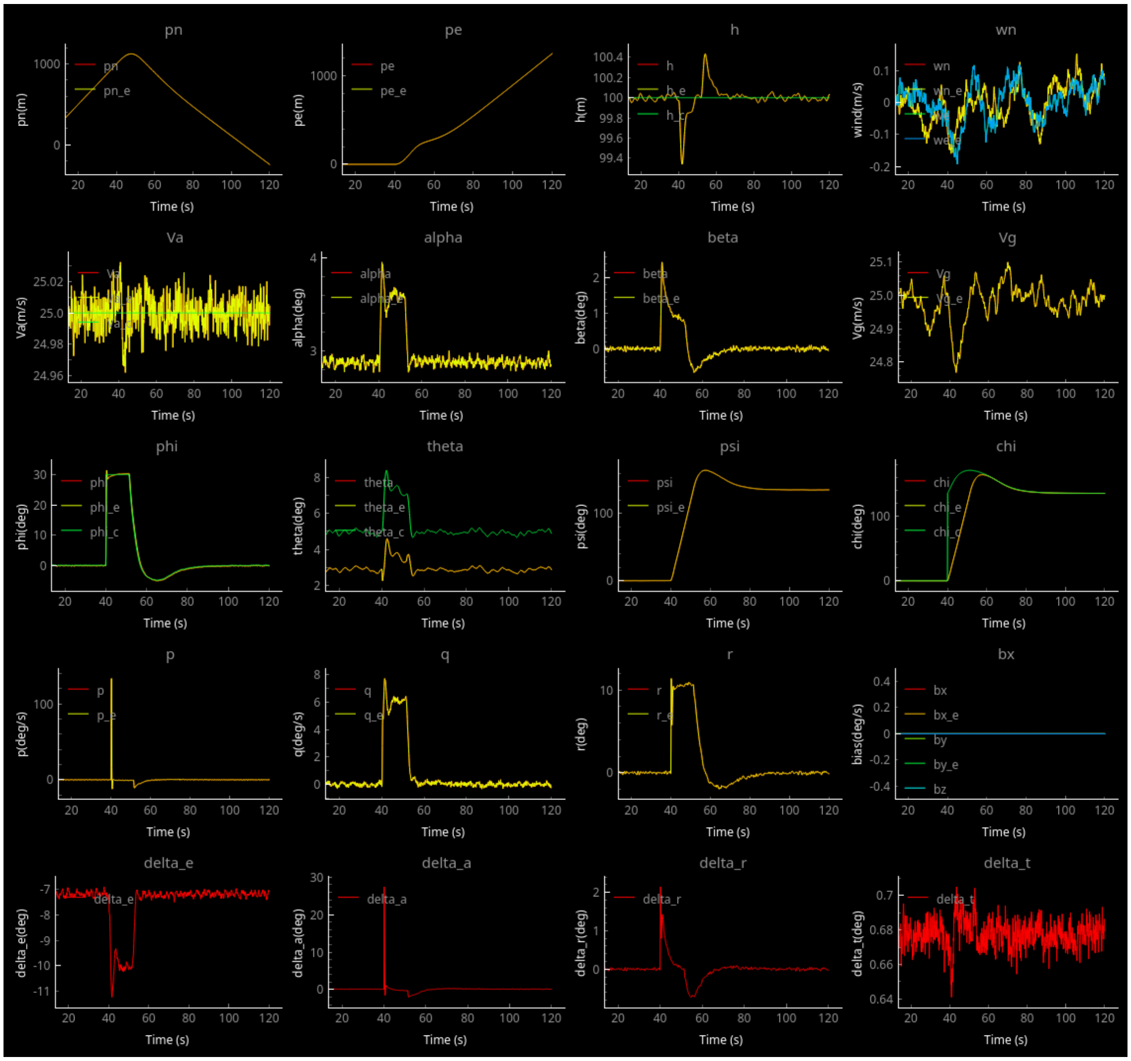
## 0.2 Problem 1 - Straight line fillet paths

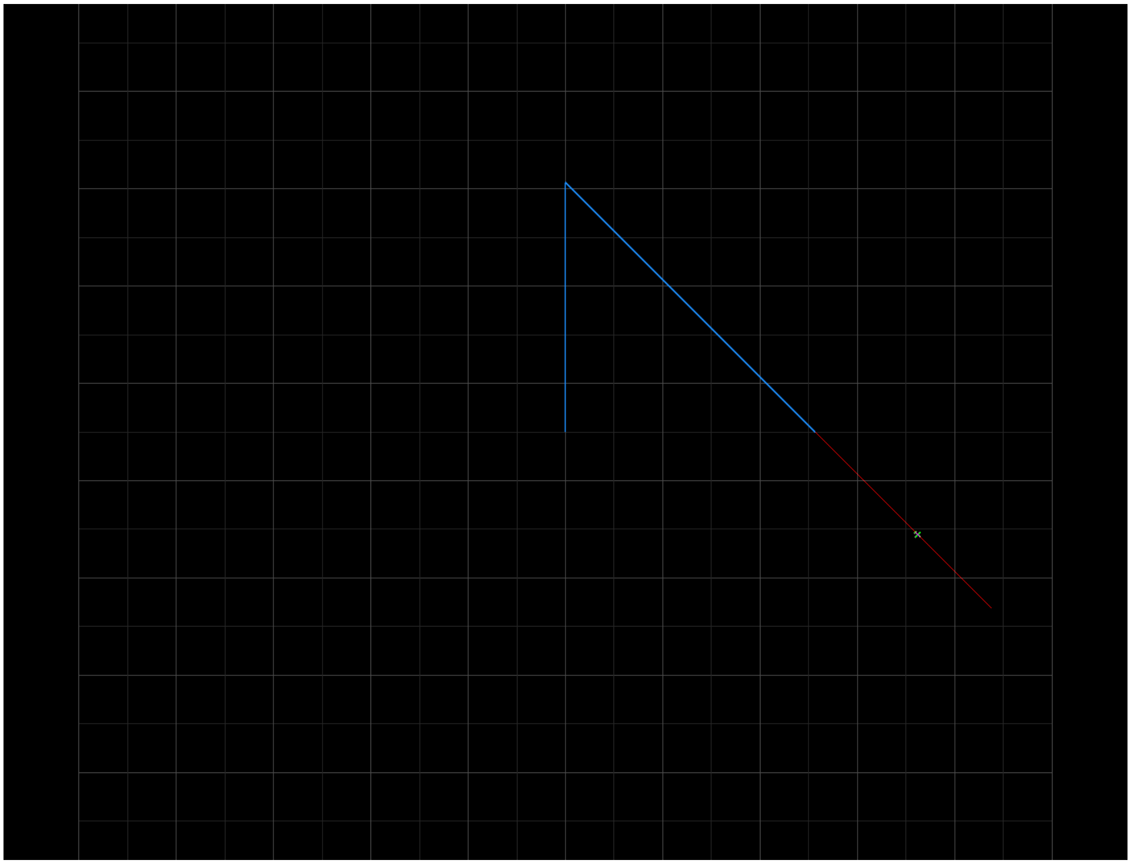
There are two scenarios where the fillet path equations have a singularity. The first is the most common: the waypoints cannot form a straight line. This is very problematic because many paths have three waypoints that are colinear. In fact, the default motion for the UAV once it has reached the final waypoint is to proceed in a straight line.

Modify `construct_fillet_line` and/or `construct_fillet_circle` so that the fillet path manager will work for the code below. Make sure that the unit tests still pass.

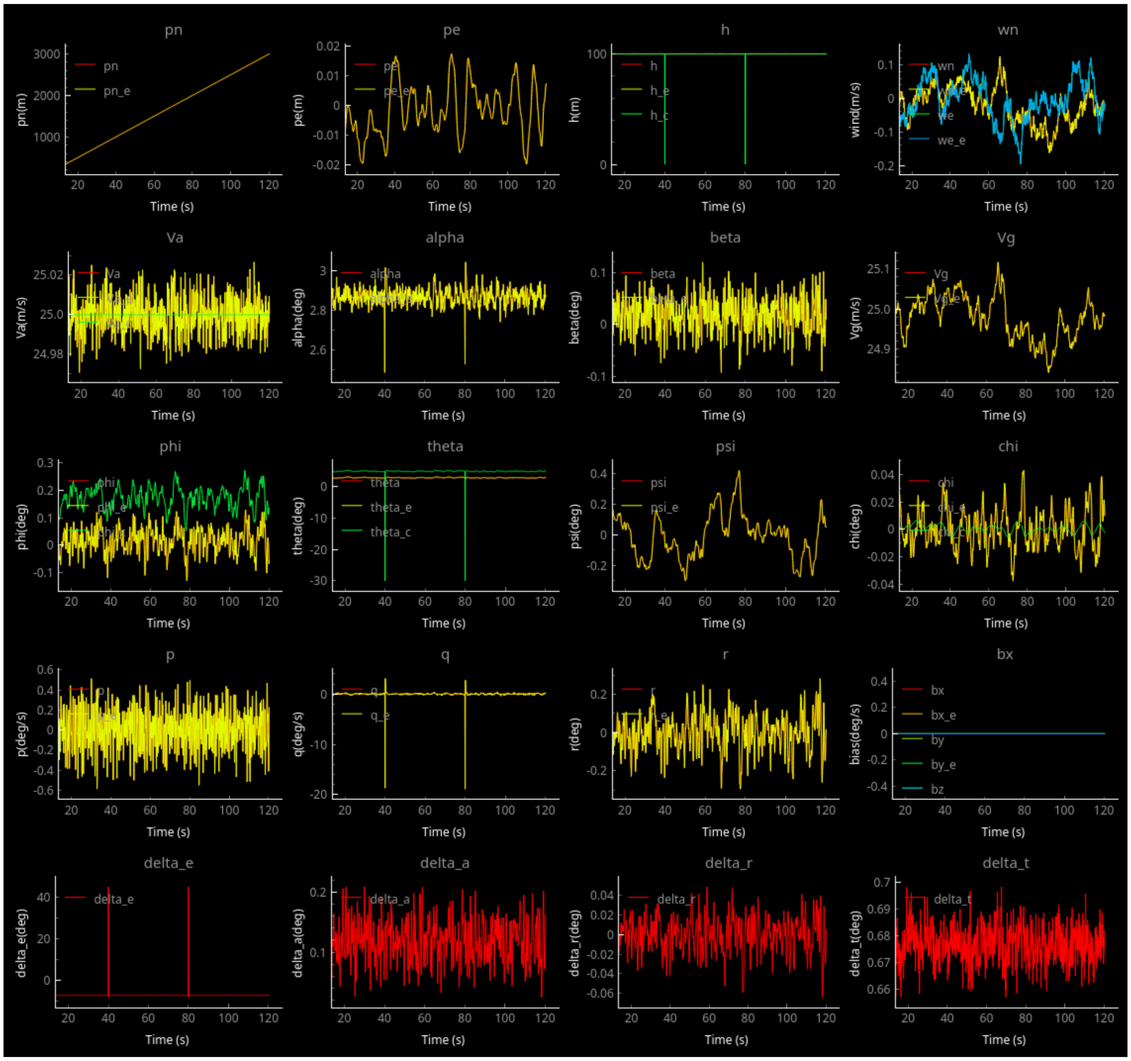
[2]:

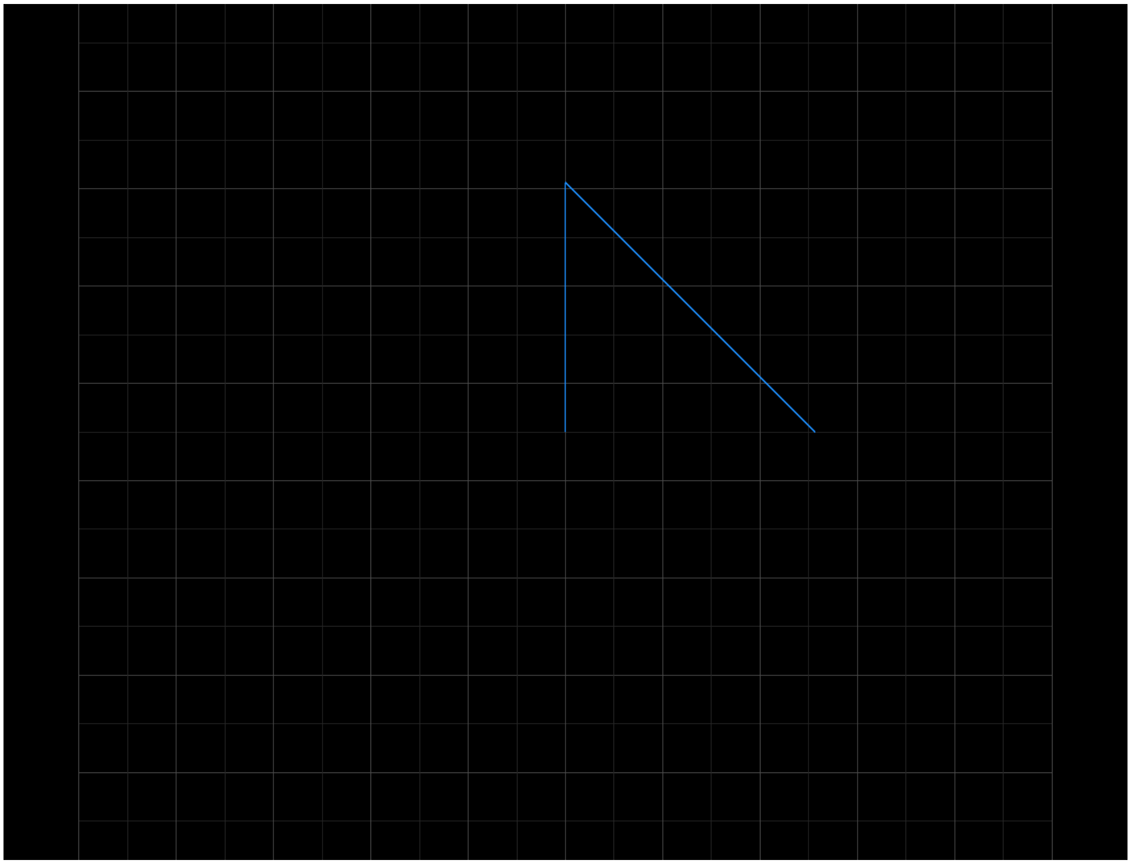
```
# Waypoint definition
waypoints = MsgWaypoints()
waypoints.type = 'straight_line'
Va = PLAN.Va0
waypoints.add(np.array([[0, 0, -100]]).T, Va, np.radians(0), np.inf, 0, 0)
waypoints.add(np.array([[1000, 0, -100]]).T, Va, np.radians(45), np.inf, 0, 0)
waypoints.add(np.array([[0, 1000, -100]]).T, Va, np.radians(45), np.inf, 0, 0)

# Run the simulation
run_sim_and_display(waypoints=waypoints)
```


pn
pn(m)
1000
0
pn
pn_e
pe
pe(m)
1000
0
pe
pe_e
h
h(m)
100.4
100.2
100
99.8
99.6
99.4
h
h_e
h_c
wn
wind(m/s)
0.1
0
-0.1
-0.2
wn
wn_e
we
we_e
Va
Va(m/s)
25.02
25.0
24.98
24.96
Va
Va_e
alpha
alpha(deg)
4
3
alpha
alpha_e
beta
beta(deg)
2
1
0
beta
beta_e
Vg
Vg(m/s)
25.1
25.0
24.9
24.8
Vg
Vg_e
phi
phi(deg)
30
20
10
0
phi
phi_e
phi_c
theta
theta(deg)
8
6
4
2
theta
theta_e
theta_c
psi
psi(deg)
100
0
psi
psi_e
chi
chi(deg)
100
0
chi
chi_e
chi_c
p
p(deg/s)
100
0
p
p_e
q
q(deg/s)
8
6
4
2
0
q
q_e
r
r(deg)
10
0
r
r_e
bx
bias(deg/s)
0.4
0.2
0
-0.2
-0.4
bx
bx_e
by
by_e
bz
delta_e
delta_e(deg)
-7
-8
-9
-10
-11
delta_e
delta_a
delta_a(deg)
30
20
10
0
delta_a
delta_r
delta_r(deg)
2
1
0
delta_r
delta_t
delta_t(deg)
0.7
0.68
0.66
0.64
delta_t
20 40 60 80 100 120
Time (s)



[3]:
```
# Waypoint definition
waypoints = MsgWaypoints()
waypoints.type = 'fillet'
Va = PLAN.Va0
waypoints.add(np.array([[0, 0, -100]]).T, Va, np.radians(0), np.inf, 0, 0)
waypoints.add(np.array([[1000, 0, -100]]).T, Va, np.radians(45), np.inf, 0, 0)
waypoints.add(np.array([[2000, 0, -100]]).T, Va, np.radians(45), np.inf, 0, 0)

# Run the simulation
run_sim_and_display(waypoints=waypoints)
```


pn
pn(m)
pn
pn_e
pe
pe(m)
pe
pe_e
h
h(m)
h
h_e
h_c
wn
wind(m/s)
wn
wn_e
we
we_e
Va
Va(m/s)
Va
alpha
alpha(deg)
alpha
beta
beta(deg)
beta
Vg
Vg(m/s)
Vg
Vg_e
phi
phi(deg)
phi
phi_e
phi_c
theta
theta(deg)
theta
theta_e
theta_c
psi
psi(deg)
psi
psi_e
chi
chi(deg)
chi
chi_e
chi_c
p
p(deg/s)
p
q
q(deg/s)
q
q_e
r
r(deg)
r
r_e
bx
bias(deg/s)
bx
bx_e
by
by_e
bz
delta_e
delta_e(deg)
delta_e
delta_a
delta_a(deg)
delta_a
delta_r
delta_r(deg)
delta_r
delta_t
delta_t(deg)
delta_t
Time (s)



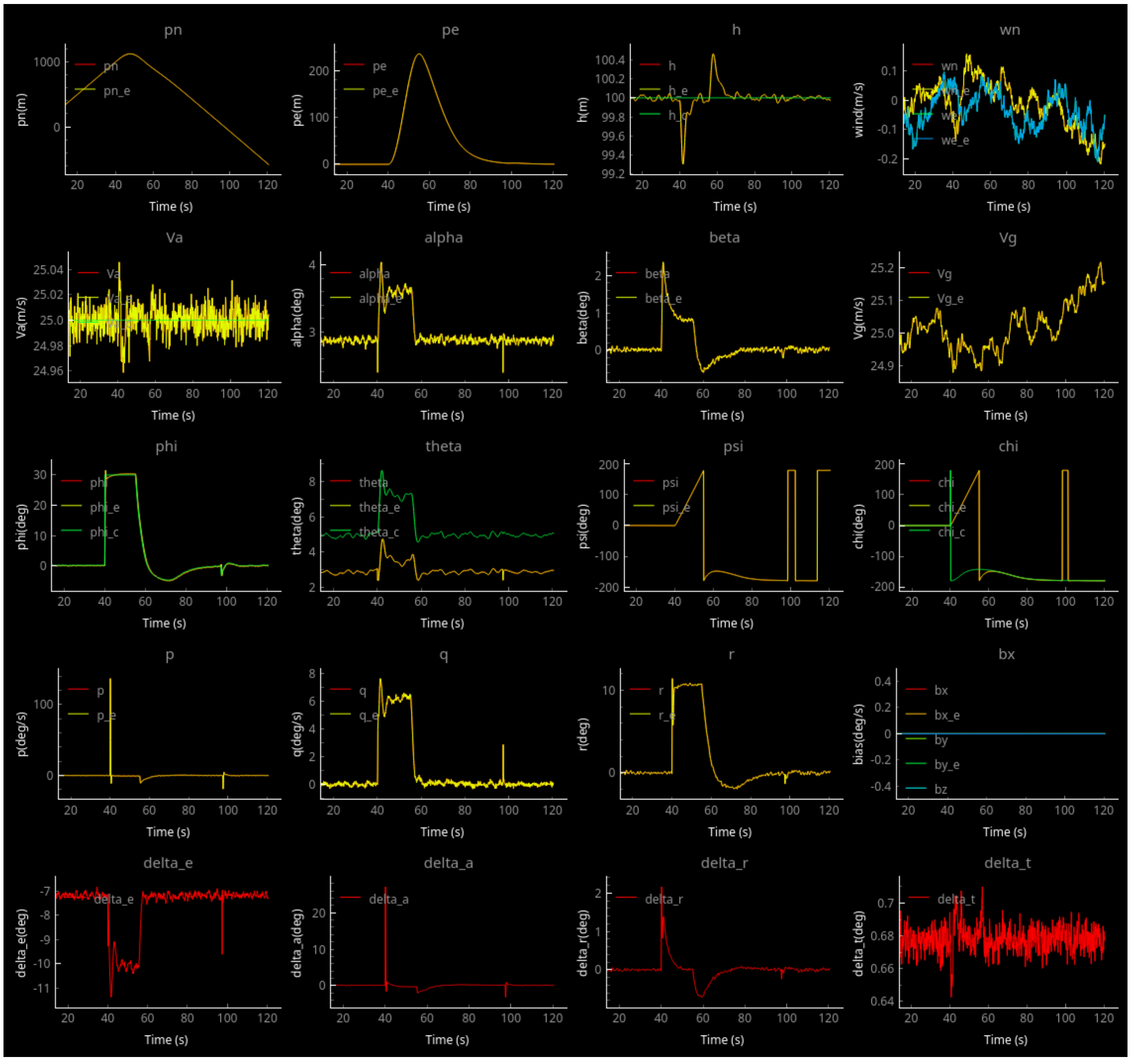
## 0.3 Problem 2 - Paths that fold back

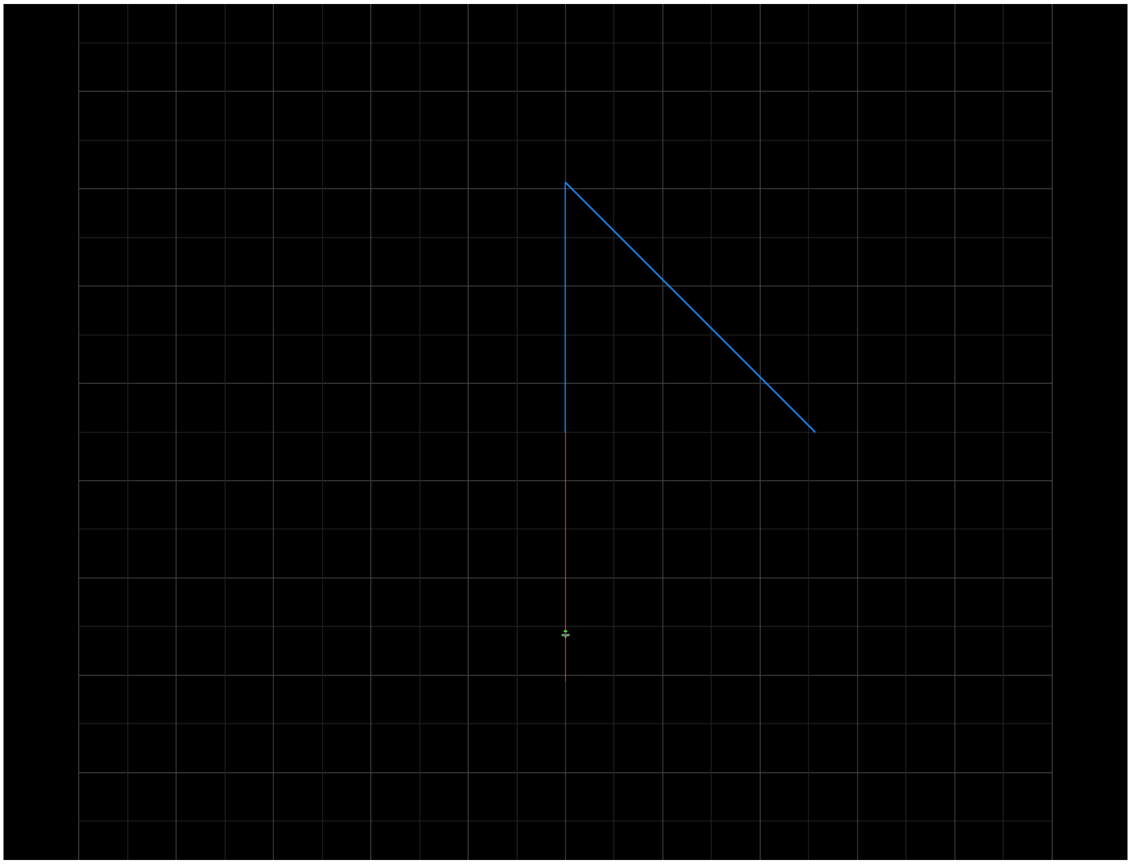
The second scenario with singularities actually occurs in both straight-line and fillet paths. The case occurs when the path folds back onto itself. While not as common of an occurance, it must be dealt with regardless.

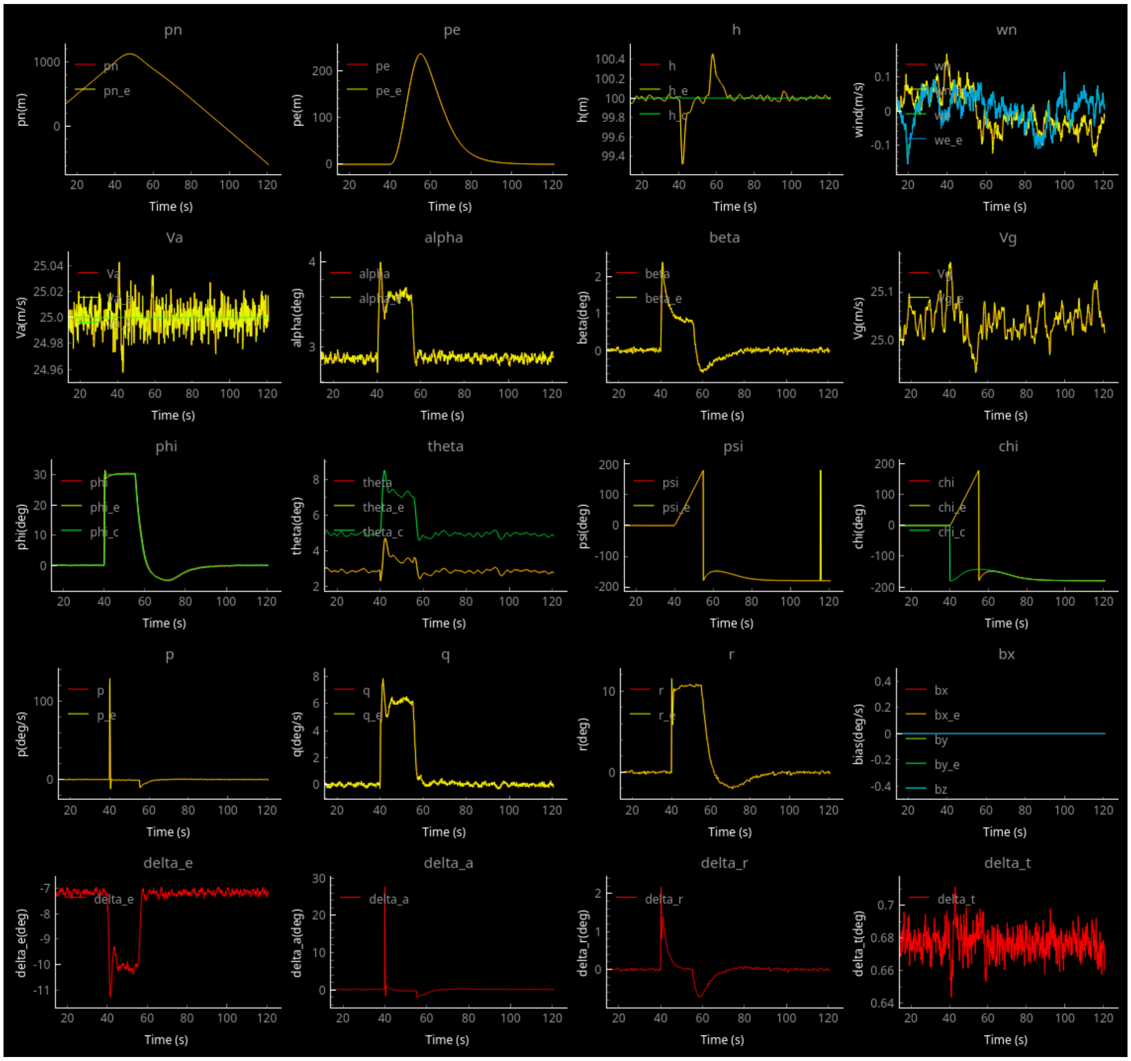
Modify `construct_fillet_line`, `construct_fillet_circle`, and `construct_line` so that the path managers will work for the code below. The switching behaviors are ill-defined in this case. Redesign them so that if a path will fold back onto itself, the UAV will make it all the way to the waypoint before starting back. Make sure that the unit tests still pass.

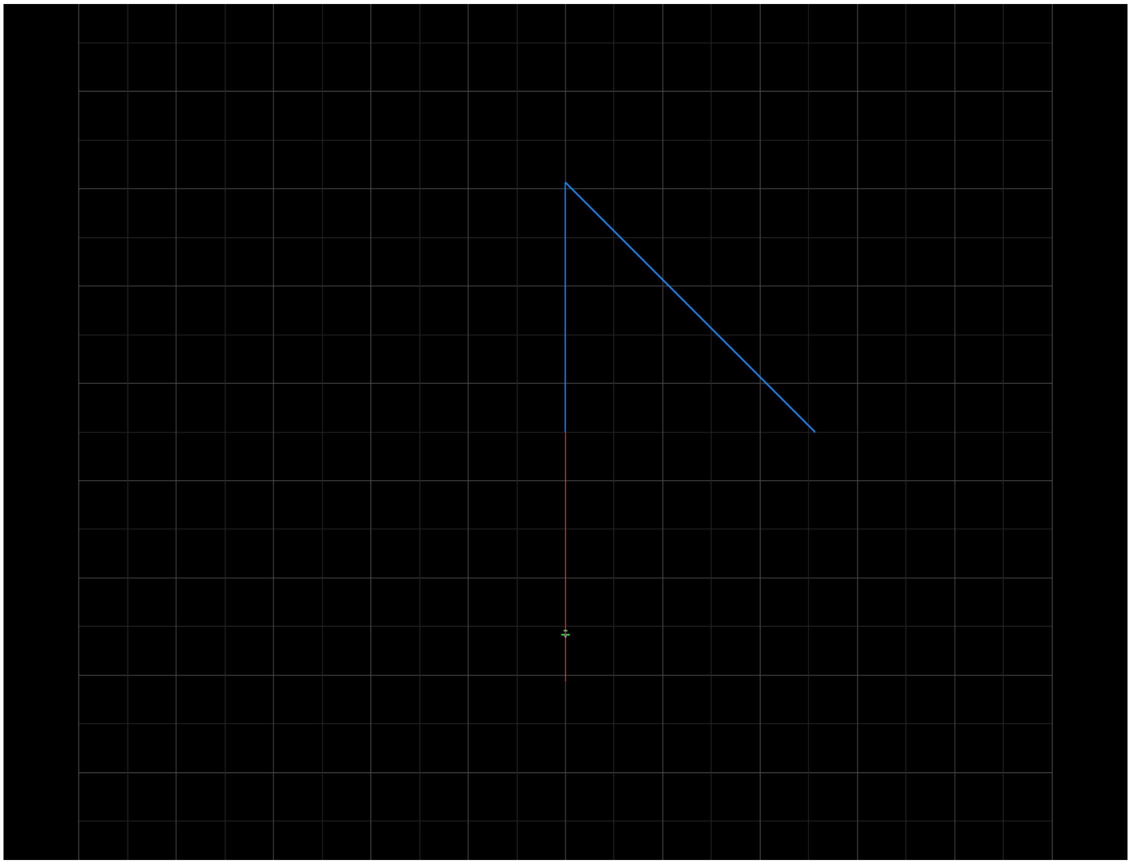
[4]:
```
# Waypoint definition
waypoints = MsgWaypoints()
waypoints.type = 'fillet'
Va = PLAN.Va0
waypoints.add(np.array([[0, 0, -100]]).T, Va, np.radians(0), np.inf, 0, 0)
waypoints.add(np.array([[1000, 0, -100]]).T, Va, np.radians(45), np.inf, 0, 0)
waypoints.add(np.array([[0, 0, -100]]).T, Va, np.radians(45), np.inf, 0, 0)

# Run the simulation for the fillet path manager
run_sim_and_display(waypoints=waypoints)
```

```
# Run the simulation for the straight line path manager
waypoints.type = 'straight_line'
run_sim_and_display(waypoints=waypoints)
```






pn
pn(m)
1000
0
pn
pn_e
pe
pe(m)
200
100
0
pe
pe_e
h
h(m)
100.4
100.2
100
99.8
99.6
99.4
h
h_e
h_c
wn
wind(m/s)
0.1
0
-0.1
wn
wn_e
we
we_e
Va
Va(m/s)
25.04
25.02
25.0
24.98
24.96
Va
Va_e
alpha
alpha(deg)
4
3
alpha
alpha_e
beta
beta(deg)
2
1
0
beta
beta_e
Vg
Vg(m/s)
25.1
25.0
Vg
Vg_e
phi
phi(deg)
30
20
10
0
phi
phi_e
phi_c
theta
theta(deg)
8
6
4
2
theta
theta_e
theta_c
psi
psi(deg)
200
100
0
-100
-200
psi
psi_e
chi
chi(deg)
200
100
0
-100
-200
chi
chi_e
chi_c
p
p(deg/s)
100
0
p
p_e
q
q(deg/s)
8
6
4
2
0
q
q_e
r
r(deg)
10
0
r
r_e
bx
bias(deg/s)
0.4
0.2
0
-0.2
-0.4
bx
bx_e
by
by_e
bz
delta_e
delta_e(deg)
-7
-8
-9
-10
-11
delta_e
delta_a
delta_a(deg)
30
20
10
0
delta_a
delta_r
delta_r(deg)
2
1
0
delta_r
delta_t
delta_t(deg)
0.7
0.68
0.66
0.64
delta_t
20
40
60
80
100
120
Time (s)



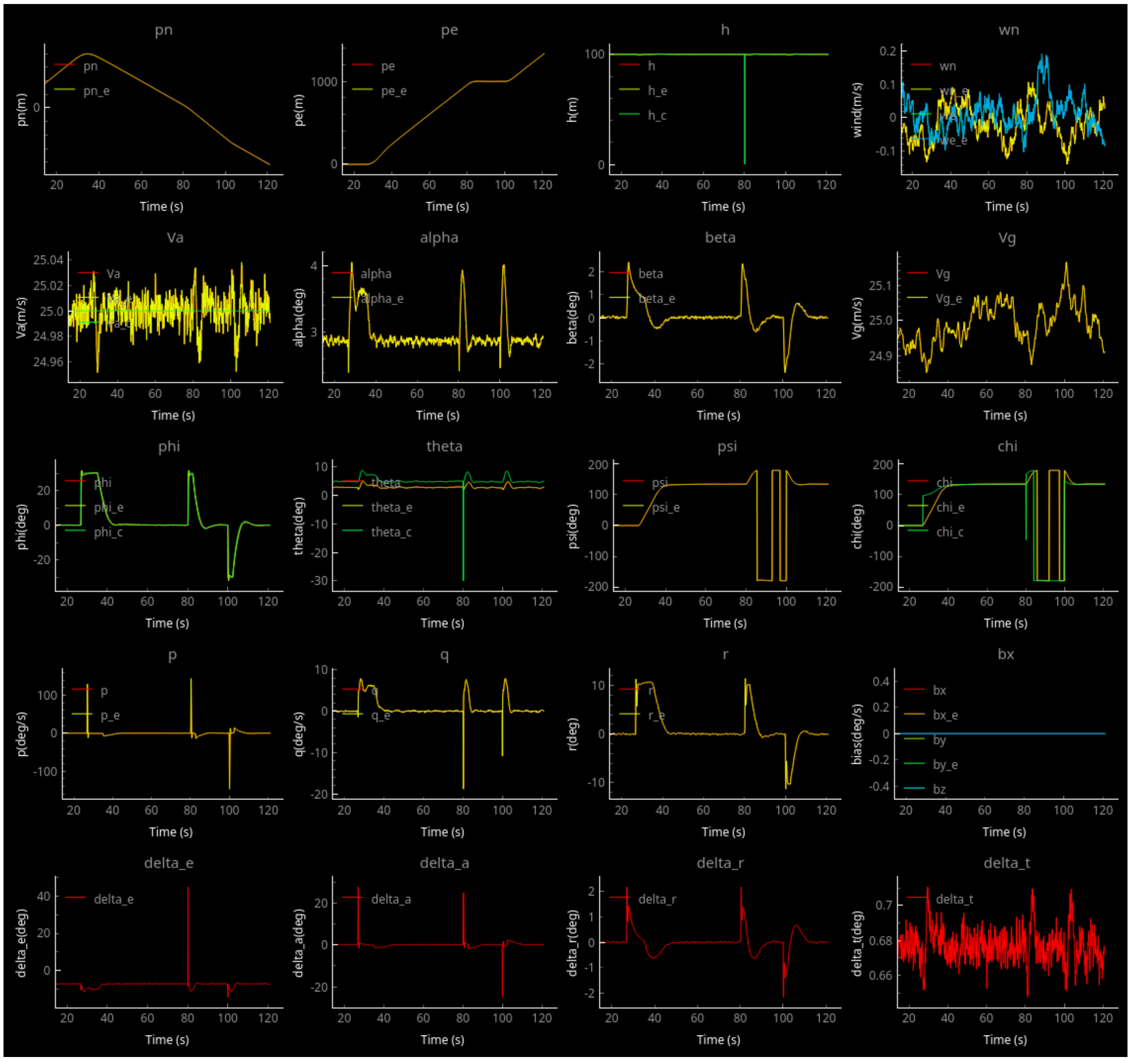
## 0.4 Problem 3 - Compare paths

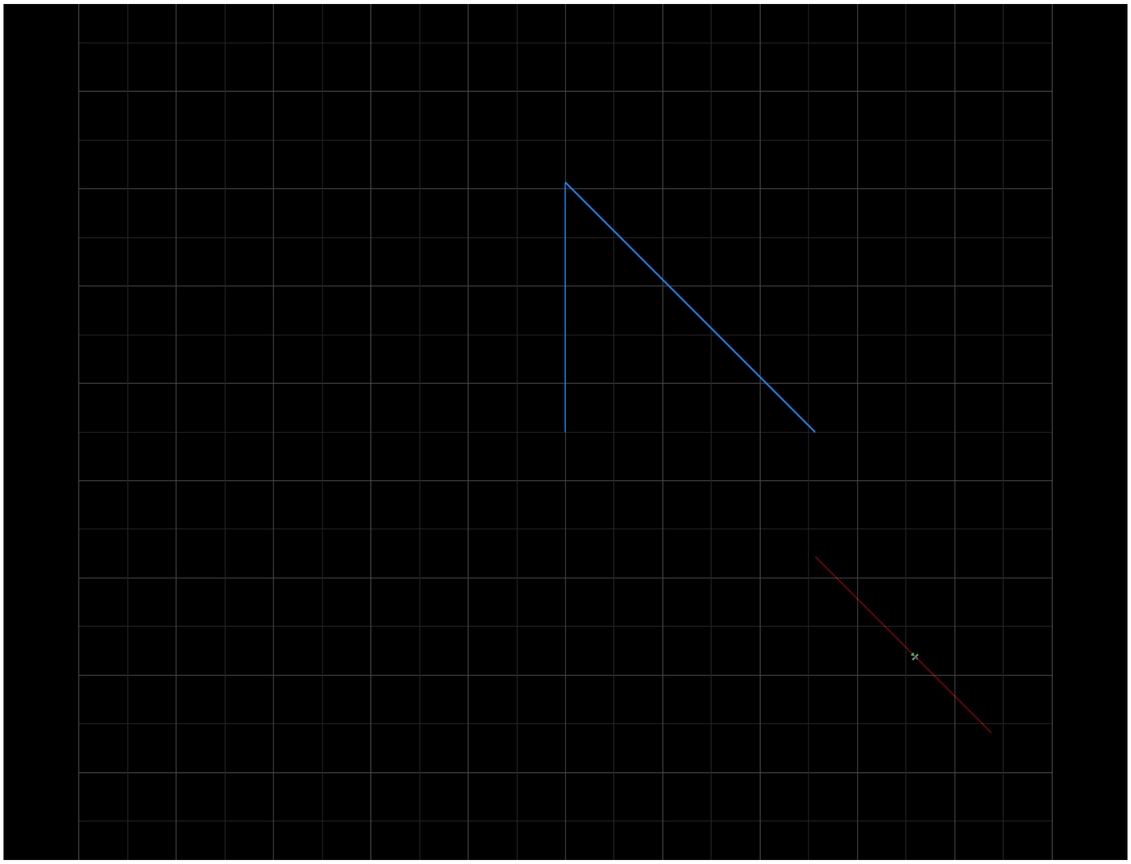
Create five waypoints that are not colinear with the first and last waypoints being $w = \begin{bmatrix} 0 \\ 0 \\ -100 \end{bmatrix}$. Ensure that there is sufficient spacing between waypoints such that the fillet circles do not overlap. Run it for both the `straight-line` and `fillet` approaches. Ensure that the simulation time is sufficient to run passed the final waypoint.

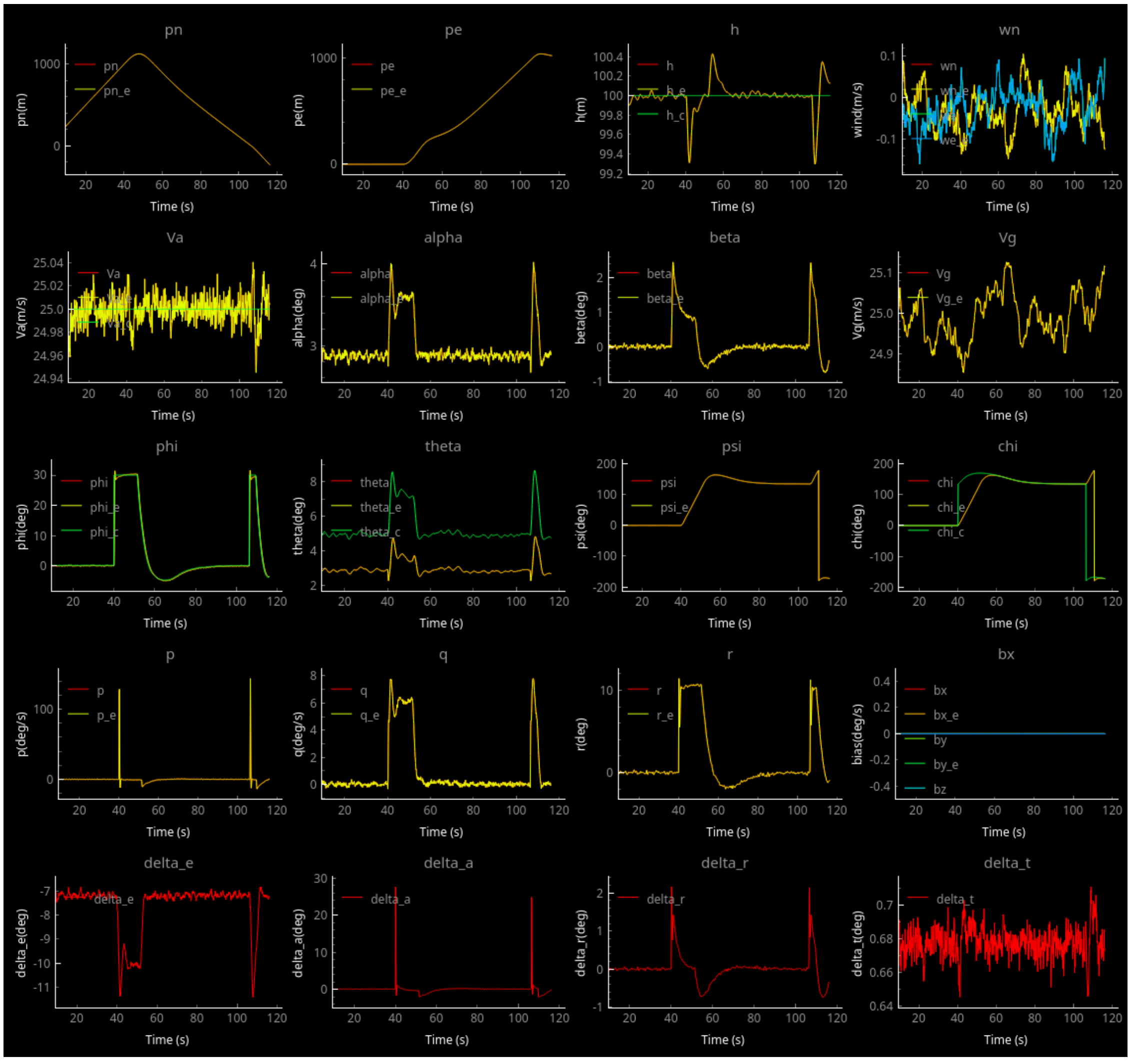
### 0.4.1 Question: what is the difference between the performance of the straight_line path and the fillet path?

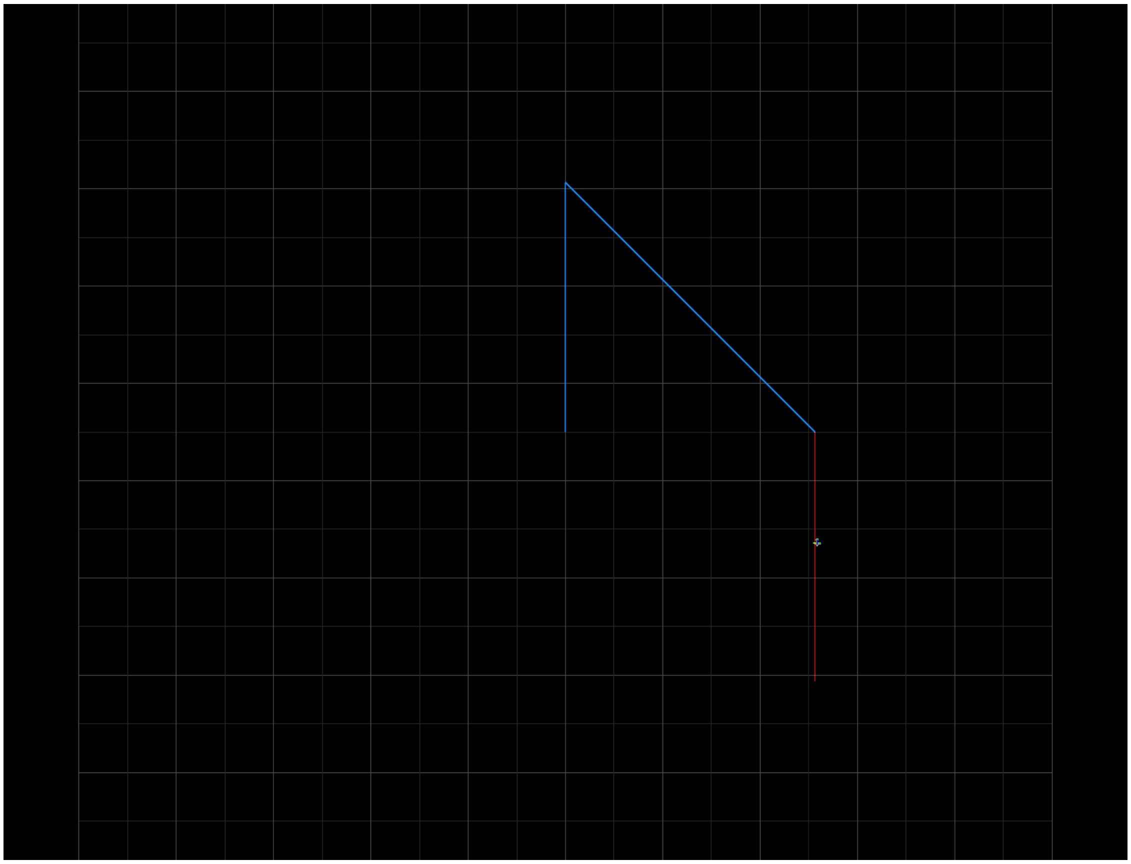
- The filet path is much more efficient than the straight line, however it does not reach the waypoints.

[5]:
```
# Waypoint definition
waypoints = MsgWaypoints()
waypoints.type = 'fillet'
Va = PLAN.Va0
waypoints.add(np.array([[0, 0, -100]]).T, Va, np.radians(0), np.inf, 0, 0)
waypoints.add(np.array([[1000, 0, -100]]).T, Va, np.radians(45), np.inf, 0, 0)
waypoints.add(np.array([[0, 1000, -100]]).T, Va, np.radians(45), np.inf, 0, 0)
waypoints.add(np.array([[-500, 1000, -100]]).T, Va, np.radians(0), np.inf, 0, 0)
waypoints.add(np.array([[-1000, 1500, -100]]).T, Va, np.radians(45), np.inf, 0,␣
  ↪0)

# Initialize the simulation parameters
sim_params_default = MsgSimParams(end_time=125., video_name="cha11.avi") # Sim␣
  ↪ending in 10 seconds

# Run the simulation for the fillet path manager
run_sim_and_display(waypoints=waypoints, sim_params=sim_params_default)

# Run the simulation for the straight line path manager
waypoints.type = 'straight_line'
run_sim_and_display(waypoints=waypoints, sim_params=sim_params_default)
```






pn
pn
pn_e
pn(m)
Time (s)
pe
pe
pe_e
pe(m)
Time (s)
h
h
h_e
h_c
h(m)
Time (s)
wn
wn
wn_e
we
wind(m/s)
Time (s)
Va
Va
Va(m/s)
Time (s)
alpha
alpha
alpha_e
alpha(deg)
Time (s)
beta
beta
beta_e
beta(deg)
Time (s)
Vg
Vg
Vg_e
Vg(m/s)
Time (s)
phi
phi
phi_e
phi_c
phi(deg)
Time (s)
theta
theta
theta_e
theta_c
theta(deg)
Time (s)
psi
psi
psi_e
psi(deg)
Time (s)
chi
chi
chi_e
chi_c
chi(deg)
Time (s)
p
p
p_e
p(deg/s)
Time (s)
q
q
q_e
q(deg/s)
Time (s)
r
r
r_e
r(deg)
Time (s)
bx
bx
bx_e
by
by_e
bz
bias(deg/s)
Time (s)
delta_e
delta_e
delta_e(deg)
Time (s)
delta_a
delta_a
delta_a(deg)
Time (s)
delta_r
delta_r
delta_r(deg)
Time (s)
delta_t
delta_t
delta_t(deg)
Time (s)



## 0.5 Simple code checking

The following code does not need to change. It should just be used as a sanity check so that you know the code is implemented properly. The output should not have any lines reading `Failed test!`

[6]:
```
from mav_sim.unit_tests.ch11_line_and_fillet_path_manager_test import␣
  ↪run_all_tests
run_all_tests()
```

```
Starting inHalfSpace test
Failed test!
Calculated output:
True
Expected output:
False
Failed on test id: 377
End of test

Starting construct_line test
Failed test!
```

```
Calculated output:

type= line
plot_updated= False
airspeed= 10.0
line_origin= [[-100.]
 [-100.]
 [-100.]]
line_direction= [[-0.57735027]
 [-0.57735027]
 [-0.57735027]]
orbit_center= [[0.]
 [0.]
 [0.]]
orbit_radius= 50
orbit_direction= CW

normal= [[-0.64055498]
 [-0.64055498]
 [-0.42353116]]
point= [[-108.]
 [-108.]
 [-108.]]
Expected output:

type= line
plot_updated= False
airspeed= 11.0
line_origin= [[-100.]
 [-100.]
 [-100.]]
line_direction= [[-0.57735027]
 [-0.57735027]
 [-0.57735027]]
orbit_center= [[0.]
 [0.]
 [0.]]
orbit_radius= 50
orbit_direction= CW

normal= [[-0.64055498]
 [-0.64055498]
 [-0.42353116]]
point= [[-108.]
 [-108.]
 [-108.]]
Failed on test id: 0 list index: 0
End of test
```

```
Starting construct_fillet_line test
Failed test!
Calculated output:

type= line
plot_updated= False
airspeed= 25
line_origin= [[-100.]
 [-100.]
 [-100.]]
line_direction= [[-0.57735027]
 [-0.57735027]
 [-0.57735027]]
orbit_center= [[0.]
 [0.]
 [0.]]
orbit_radius= 50
orbit_direction= CW

normal= [[-0.57735027]
 [-0.57735027]
 [-0.57735027]]
point= [[-106.96048968]
 [-106.96048968]
 [-106.96048968]]
Expected output:

type= line
plot_updated= False
airspeed= 11.0
line_origin= [[-100.]
 [-100.]
 [-100.]]
line_direction= [[-0.57735027]
 [-0.57735027]
 [-0.57735027]]
orbit_center= [[0.]
 [0.]
 [0.]]
orbit_radius= 50
orbit_direction= CW

normal= [[-0.57735027]
 [-0.57735027]
 [-0.57735027]]
point= [[-106.96048968]
 [-106.96048968]
 [-106.96048968]]
Failed on test id: 0 list index: 0
End of test

Starting construct_fillet_circle test
Failed test!
Calculated output:

type= line
plot_updated= False
airspeed= 25
line_origin= [[0.]
 [0.]
 [0.]]
line_direction= [[1.]
 [0.]
 [0.]]
orbit_center= [[0.]
 [0.]
 [0.]]
orbit_radius= 50
orbit_direction= CW

normal= [[-0.68348613]
 [-0.68348613]
 [-0.2563073 ]]
point= [[-106.29212334]
 [-106.29212334]
 [-106.29212334]]
Expected output:

type= orbit
plot_updated= False
airspeed= 11.0
line_origin= [[0.]
 [0.]
 [0.]]
line_direction= [[1.]
 [0.]
 [0.]]
orbit_center= [[-111.04297259]
 [-111.04297259]
 [ -98.79552387]]
orbit_radius= 10
orbit_direction= CCW

normal= [[-0.68348613]
 [-0.68348613]
 [-0.2563073 ]]
point= [[-109.23060631]
 [-109.23060631]
 [-108.46147736]]
Failed on test id: 0 list index: 0
End of test
```