A* Motion Planning

In [34]:
```
# The autoreload extension will automatically load in new code as you
edit files,
# so you don't need to restart the kernel every time
%load_ext autoreload
%autoreload 2
import numpy as np
import matplotlib.pyplot as plt
from P1_astar import DetOccupancyGrid2D, AStar
from utils import generate_planning_problem
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Simple Environment

Workspace

(Try changing this and see what happens)

In [35]:
```
width = 10
height = 10
obstacles = [((6,7),(8,8)),((2,2),(4,3)),((2,5),(4,7)),((6,3),(8,5))]
occupancy = DetOccupancyGrid2D(width, height, obstacles)
```

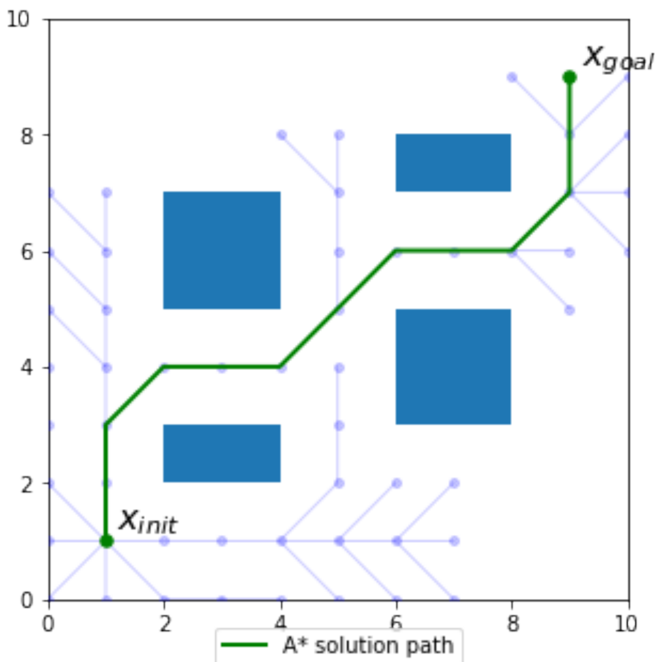
Starting and final positions

(Try changing these and see what happens)

In [36]:
```
x_init = (1, 1)
x_goal = (9, 9)
```

Run A* planning

In [37]:
```
astar = AStar((0, 0), (width, height), x_init, x_goal, occupancy)
if not astar.solve():
    print("No path found")
else:
    plt.rcParams['figure.figsize'] = [5, 5]
    astar.plot_path()
    astar.plot_tree()
```



Random Cluttered Environment

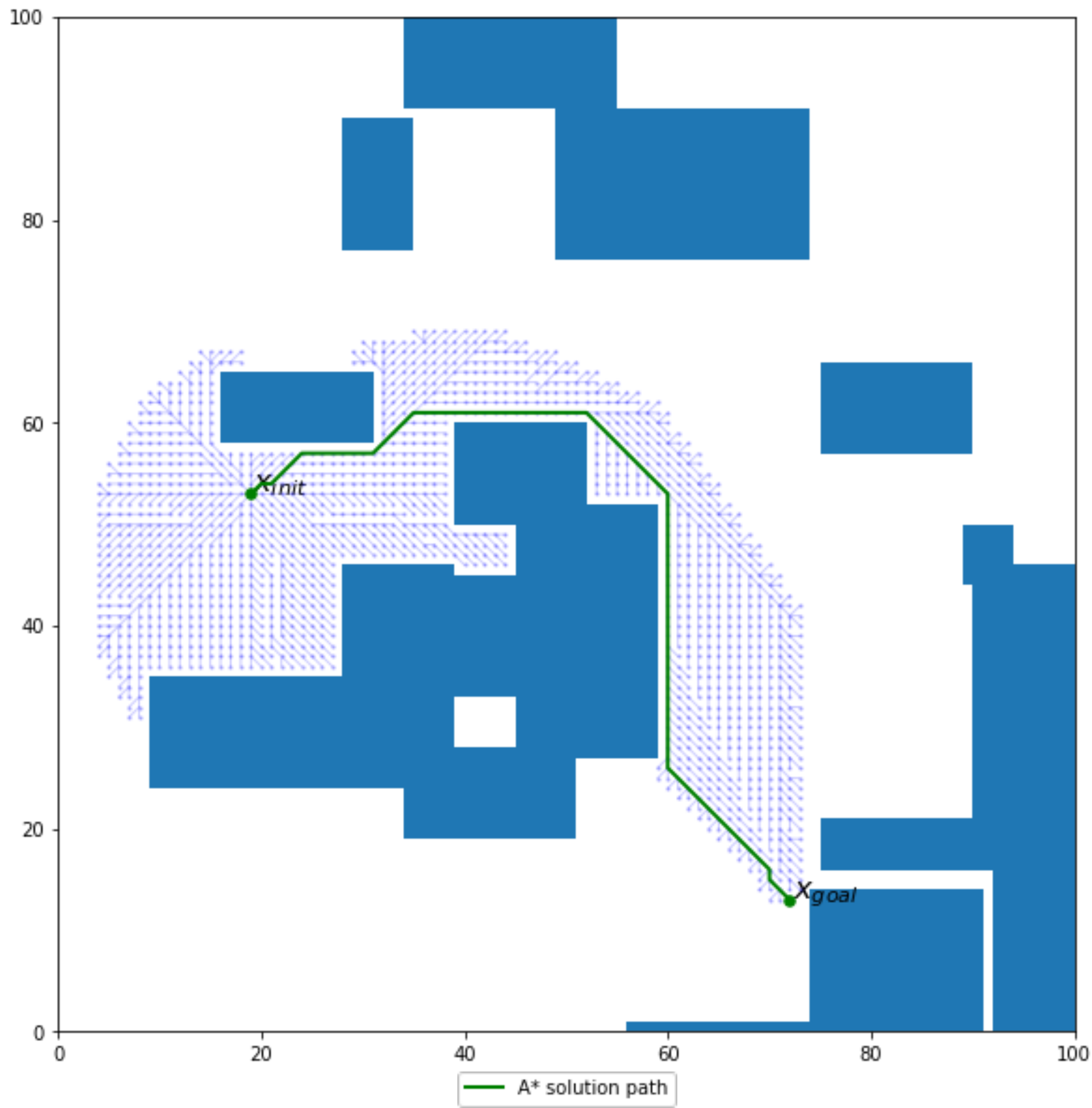
Generate workspace, start and goal positions

(Try changing these and see what happens)

In [42]:
```
width = 100
height = 100
num_obs = 25
min_size = 5
max_size = 30

occupancy, x_init, x_goal = generate_planning_problem(width, height, n
um_obs, min_size, max_size)
```

Run A* planning

In [43]:
```
astar = AStar((0, 0), (width, height), x_init, x_goal, occupancy)
if not astar.solve():
    print("No path found")
else:
    plt.rcParams['figure.figsize'] = [10, 10]
    astar.plot_path()
    astar.plot_tree(point_size=2)
```



RRT Sampling-Based Motion Planning

In [107]:
```
# The autoreload extension will automatically load in new code as you
edit files,
# so you don't need to restart the kernel every time
%load_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt
from P2_rrt import *

plt.rcParams['figure.figsize'] = [10, 10] # Change default figure size
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

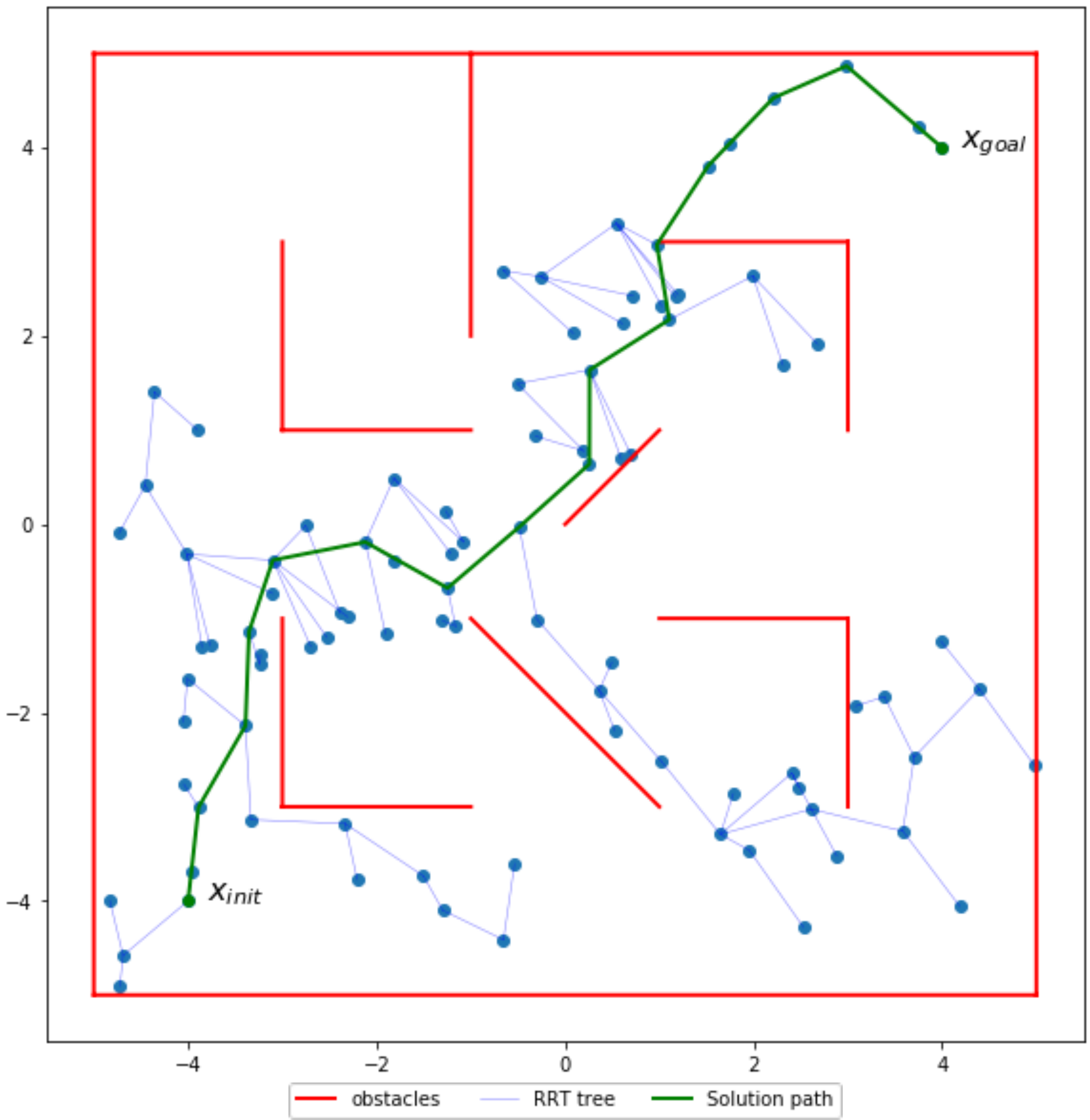
Set up workspace

In [108]:
```
MAZE = np.array([
    (( 5, 5), (-5, 5)),
    ((-5, 5), (-5,-5)),
    ((-5,-5), ( 5,-5)),
    (( 5,-5), ( 5, 5)),
    ((-3,-3), (-3,-1)),
    ((-3,-3), (-1,-3)),
    (( 3, 3), ( 3, 1)),
    (( 3, 3), ( 1, 3)),
    (( 1,-1), ( 3,-1)),
    (( 3,-1), ( 3,-3)),
    ((-1, 1), (-3, 1)),
    ((-3, 1), (-3, 3)),
    ((-1,-1), ( 1,-3)),
    ((-1, 5), (-1, 2)),
    (( 0, 0), ( 1, 1))
])

# try changing these!
x_init = [-4,-4] # reset to [-4,-4] when saving results for submission
x_goal = [4,4] # reset to [4,4] when saving results for submission
```

Geometric Planning

In [109]:
```
grrt = GeometricRRT([-5,-5], [5,5], x_init, x_goal, MAZE)
grrt.solve(1.0, 2000)
```

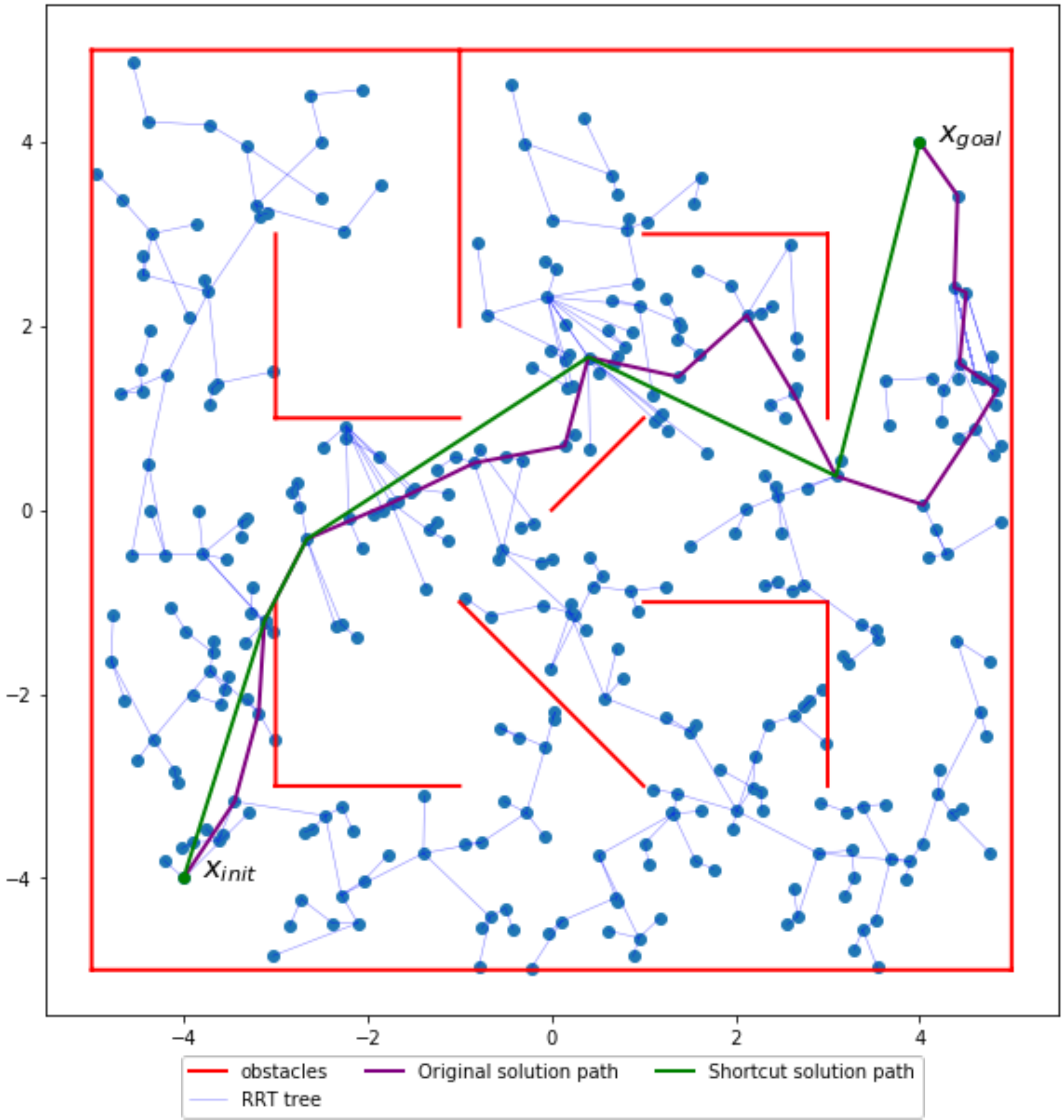
Out[109]: True



Adding shortcutting

In [110]:
```
grrt.solve(1.0, 2000, shortcut=True)
```

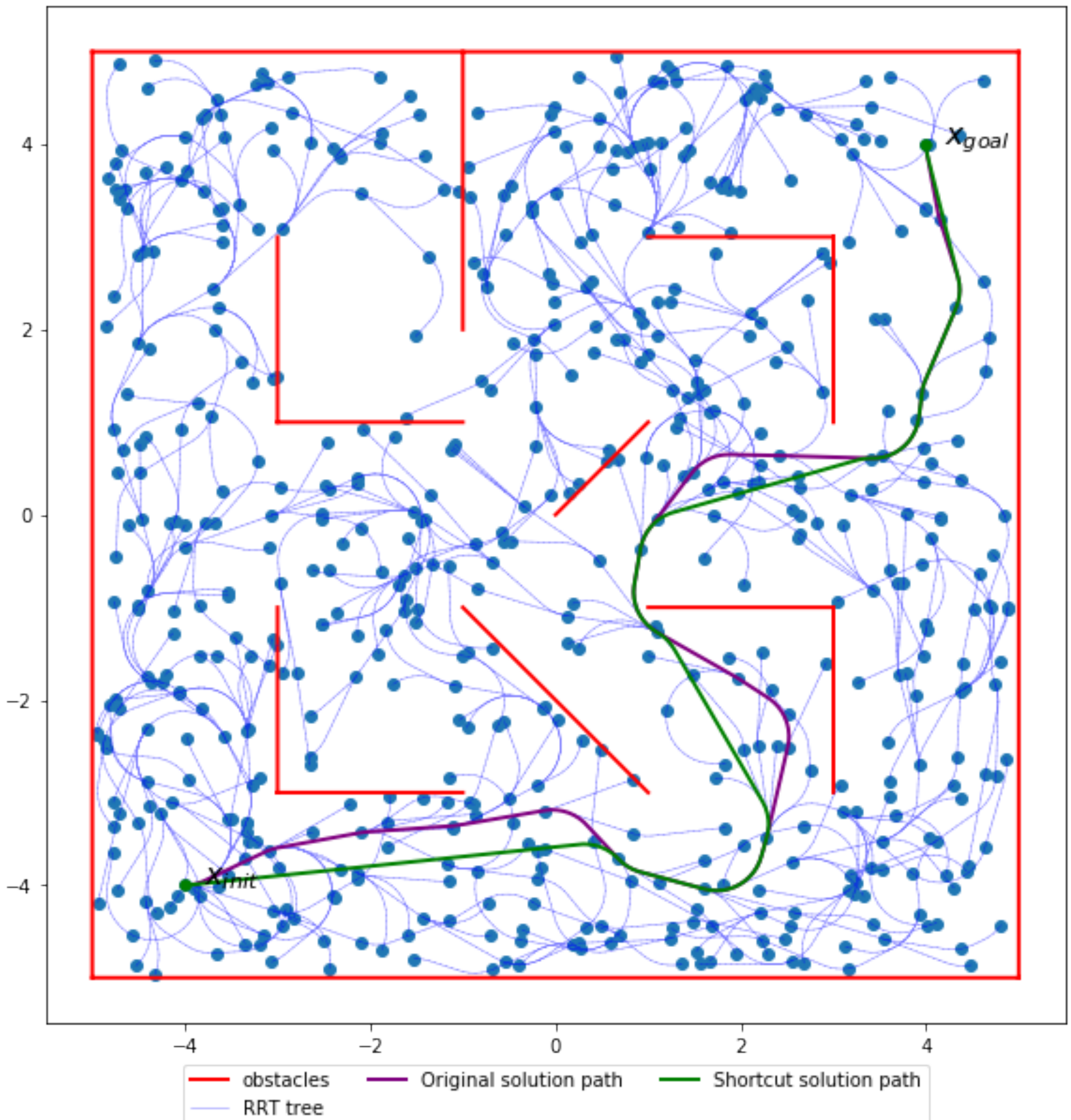
Out[110]: True



Dubins Car Planning

In [111]:
```
x_init = [-4,-4,0]
x_goal = [4,4,np.pi/2]

drrt = DubinsRRT([-5,-5,0], [5,5,2*np.pi], x_init, x_goal, MAZE, .5)
drrt.solve(1.0, 1000, shortcut=True)
```

Out[111]: True



In [292]:
```
# The autoreload extension will automatically load in new code as you
edit files,
# so you don't need to restart the kernel every time
%load_ext autoreload
%autoreload 2

import numpy as np
from P1_astar import AStar
from P2_rrt import *
from P3_traj_planning import compute_smoothed_traj, modify_traj_with_l
imits, SwitchingController
import matplotlib.pyplot as plt
from HW1.P1_differential_flatness import *
from HW1.P2_pose_stabilization import *
from HW1.P3_trajectory_tracking import *
from utils import generate_planning_problem
from HW1.utils import simulate_car_dyn

plt.rcParams['figure.figsize'] = [14, 14] # Change default figure size
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Generate workspace, start and goal positions

In [293]:
```
width = 100
height = 100
num_obs = 25
min_size = 5
max_size = 30

occupancy, x_init, x_goal = generate_planning_problem(width, height, n
um_obs, min_size, max_size)
```

In [294]:
```
x_goal
```

Out[294]:
```
(18, 73)
```

Solve A* planning problem

In [295]:
```
astar = AStar((0, 0), (width, height), x_init, x_goal, occupancy)
if not astar.solve():
    print("No path found")
```

Smooth Trajectory Generation

Trajectory parameters

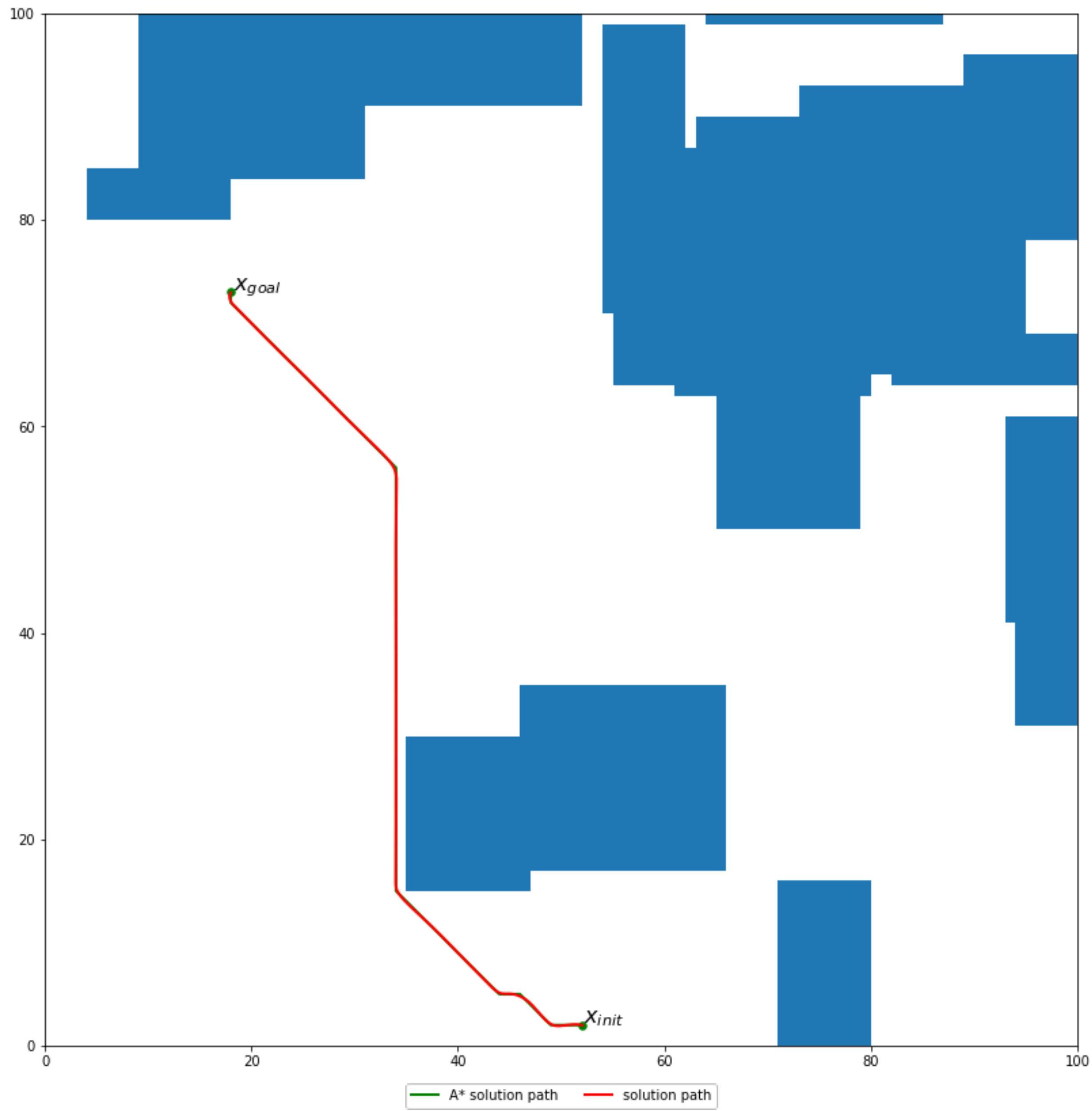
(Try changing these and see what happens)

In [296]:
```
V_des = 0.3  # Nominal velocity
alpha = 0.1   # Smoothness parameter
dt = 0.05
```

Generate smoothed trajectory

In [297]:
```
traj_smoothed, t_smoothed = compute_smoothed_traj(astar.path, V_des, a
lpha, dt)

fig = plt.figure()
astar.plot_path(fig.number)
def plot_traj_smoothed(traj_smoothed):
    print(traj_smoothed[:,0])
    print(traj_smoothed[:,1])
    plt.plot(traj_smoothed[:,0], traj_smoothed[:,1], color="red", line
width=2, label="solution path", zorder=10)
plot_traj_smoothed(traj_smoothed)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=T
rue, ncol=3)
plt.show()
```

```
[52.04219378 52.0272388  52.01231248 ... 17.97970831 17.98598673
 17.9924706 ]
[ 1.99117545  1.99464452  1.99802059 ... 72.91849702 72.92941025
 72.94032666]
```



Control-Feasible Trajectory Generation and Tracking

Robot control limits

In [298]:
```
V_max = 0.5 # max speed
om_max = 1 # max rotational speed
```

Tracking control gains

Tune these as needed to improve tracking performance.

In [299]:
```
kpx = 2
kpy = 2
kdx = 2
kdy = 2
# if control gains are equal to 2. The trajectory is pretty close to t
he smoothed traj.
```

Generate control-feasible trajectory

In [300]:
```
t_new, V_smooth_scaled, om_smooth_scaled, traj_smooth_scaled = modify_
traj_with_limits(traj_smoothed, t_smoothed, V_max, om_max, dt)
```

Create trajectory controller and load trajectory

In [301]:
```
traj_controller = TrajectoryTracker(kpx=kpx, kpy=kpy, kdx=kdx, kdy=kdy
, V_max=V_max, om_max=om_max)
traj_controller.load_traj(t_new, traj_smooth_scaled)
```

Set simulation input noise

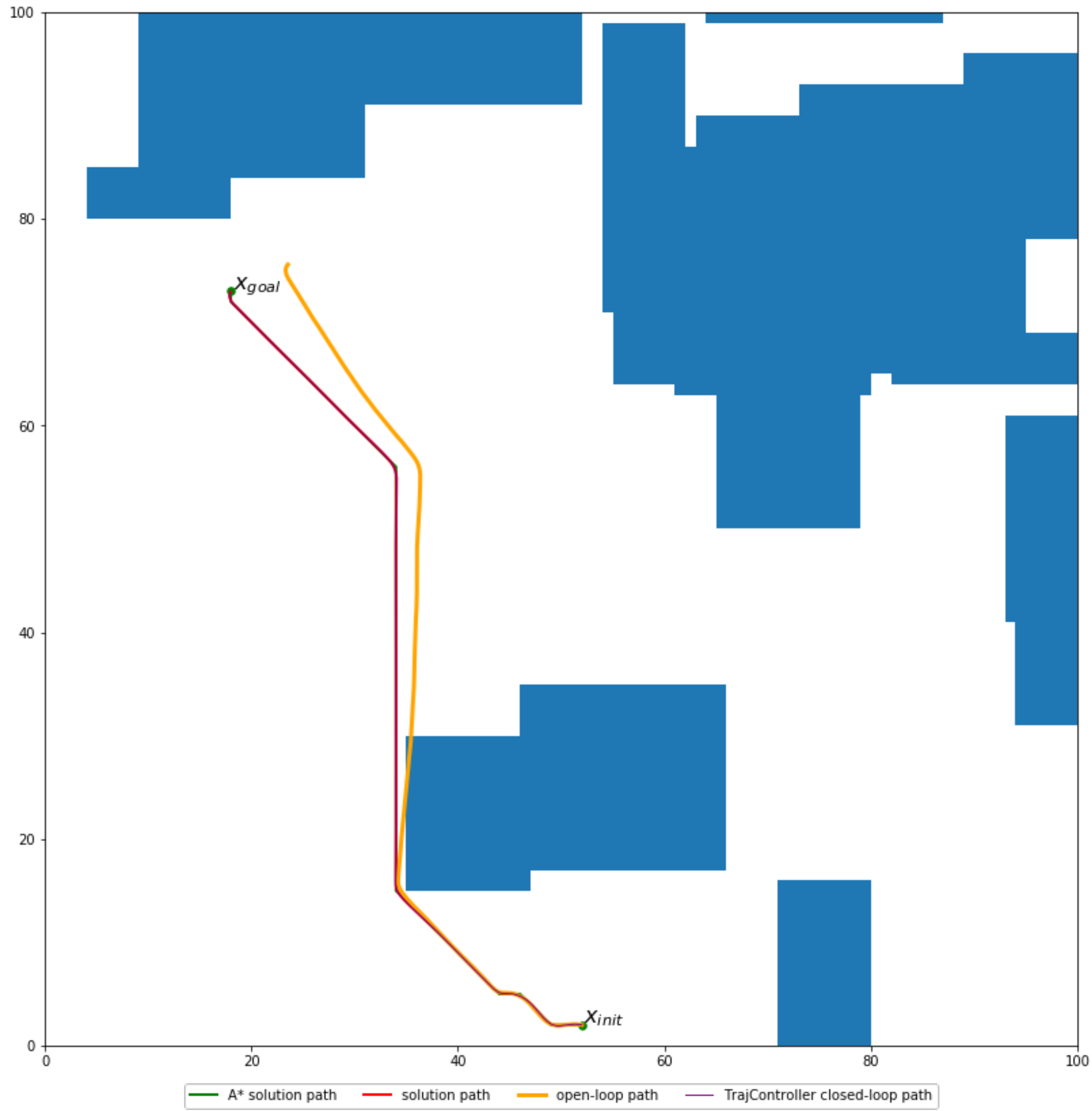
(Try changing this and see what happens)

In [302]:
```
noise_scale = 0.05
```

Simulate closed-loop tracking of smoothed trajectory, compare to open-loop

In [303]:
```
tf_actual = t_new[-1]
times_cl = np.arange(0, tf_actual, dt)
s_0 = State(x=x_init[0], y=x_init[1], V=V_max, th=traj_smooth_scaled[0
,2])
s_f = State(x=x_goal[0], y=x_goal[1], V=V_max, th=traj_smooth_scaled[-
1,2])

actions_ol = np.stack([V_smooth_scaled, om_smooth_scaled], axis=-1)
states_ol, ctrl_ol = simulate_car_dyn(s_0.x, s_0.y, s_0.th, times_cl,
actions=actions_ol, noise_scale=noise_scale)
states_cl, ctrl_cl = simulate_car_dyn(s_0.x, s_0.y, s_0.th, times_cl,
controller=traj_controller, noise_scale=noise_scale)

fig = plt.figure()
astar.plot_path(fig.number)
plot_traj_smoothed(traj_smoothed)
def plot_traj_ol(states_ol):
    plt.plot(states_ol[:,0],states_ol[:,1], color="orange", linewidth=
3, label="open-loop path", zorder=10)
def plot_traj_cl(states_cl):
    print(states_cl[:,0])
    print(states_cl[:,1])
    plt.plot(states_cl[:,0], states_cl[:,1], color="purple", linewidth
=1, label="TrajController closed-loop path", zorder=10)
plot_traj_ol(states_ol)
plot_traj_cl(states_cl)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=T
rue, ncol=4)
plt.show()
```

```
[52.04219378 52.0272388  52.01231248 ... 17.97970831 17.98598673
 17.9924706 ]
[ 1.99117545  1.99464452  1.99802059 ... 72.91849702 72.92941025
 72.94032666]
[52.         51.98381207 51.97199767 ... 17.97677353 17.98444102
 17.98875597]
[ 2.          2.00378781  2.00652015 ... 72.91503008 72.92862465
 72.93603367]
```

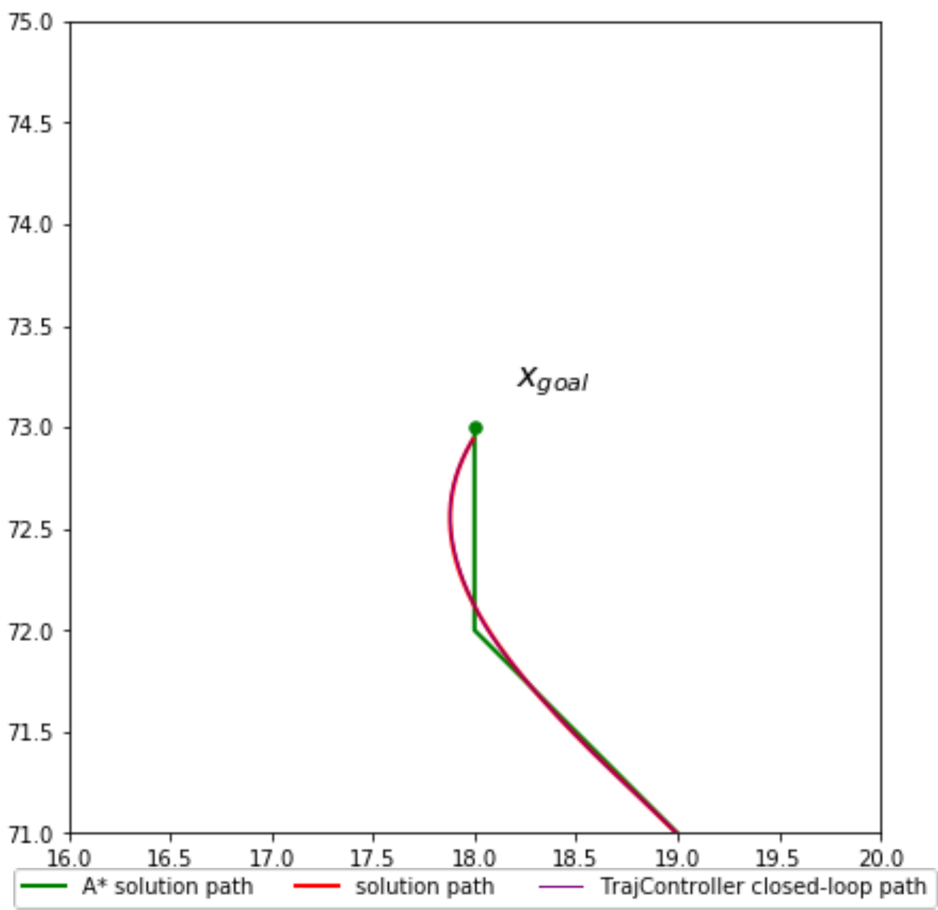


Switching from Trajectory Tracking to Pose Stabilization Control

Zoom in on final pose error

In [304]:
```
l_window = 4.

fig = plt.figure(figsize=[7,7])
astar.plot_path(fig.number, show_init_label = False)
plot_traj_smoothed(traj_smoothed)
plot_traj_cl(states_cl)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=T
rue, ncol=3)
plt.axis([x_goal[0]-l_window/2, x_goal[0]+l_window/2, x_goal[1]-l_wind
ow/2, x_goal[1]+l_window/2])
plt.show()
```

```
[52.04219378 52.0272388  52.01231248 ... 17.97970831 17.98598673
 17.9924706 ]
[ 1.99117545  1.99464452  1.99802059 ... 72.91849702 72.92941025
 72.94032666]
[52.         51.98381207 51.97199767 ... 17.97677353 17.98444102
 17.98875597]
[ 2.          2.00378781  2.00652015 ... 72.91503008 72.92862465
 72.93603367]
```



Pose stabilization control gains

Tune these as needed to improve final pose stabilization.

In [341]:
```
k1 = 1.
k2 = 1.
k3 = 1.
```

Create pose controller and load goal pose

Note we use the last value of the smoothed trajectory as the goal heading θ

In [342]:
```
pose_controller = PoseController(k1, k2, k3, V_max, om_max)
pose_controller.load_goal(x_goal[0], x_goal[1], traj_smooth_scaled[-1,
2])
```

Time before trajectory-tracking completion to switch to pose stabilization

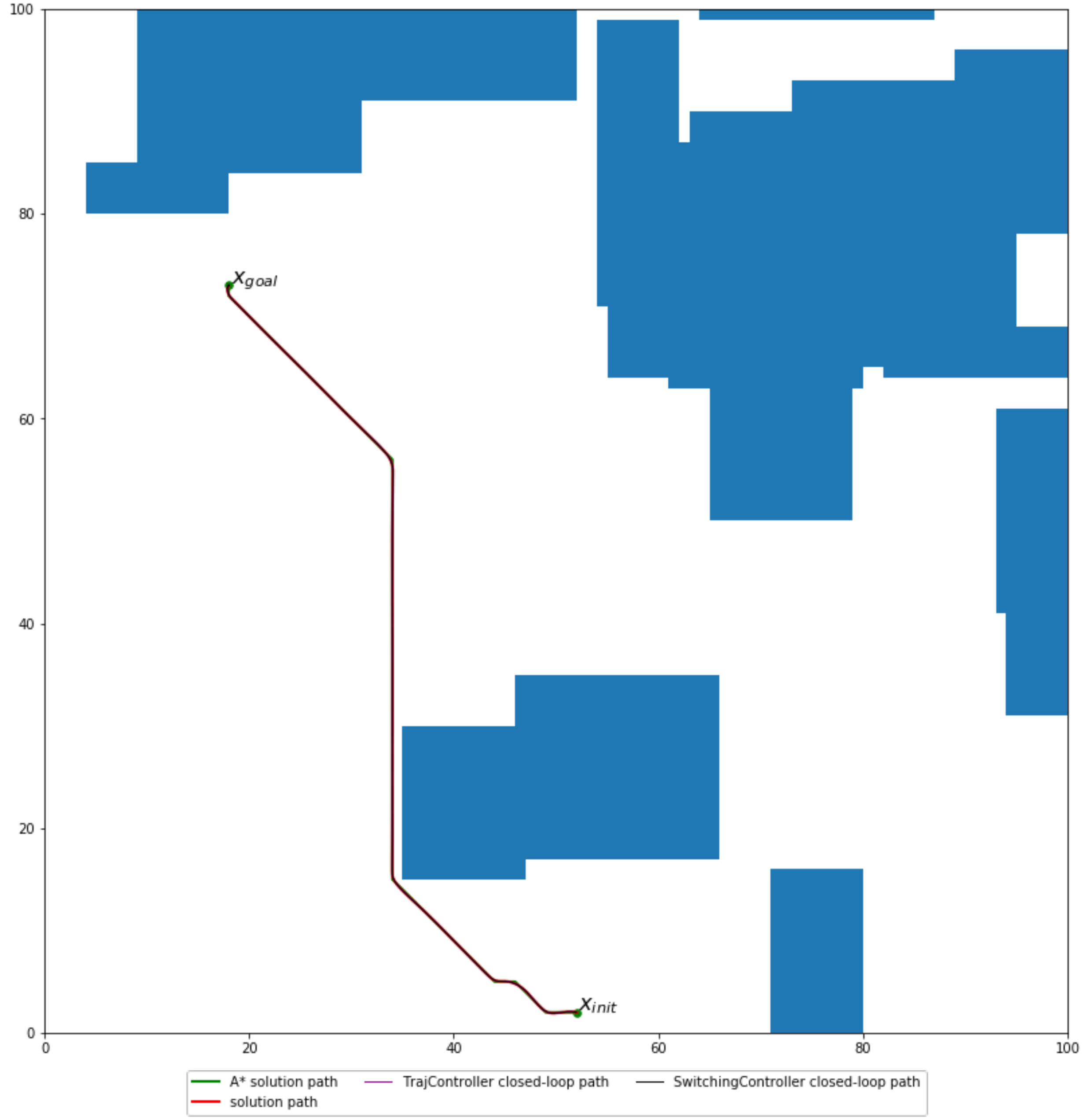
Try changing this!

In [343]:
```
t_before_switch = 5.0
```

Create switching controller and compare performance

In [344]:
```
switching_controller = SwitchingController(traj_controller, pose_contr
oller, t_before_switch)

t_extend = 60.0 # Extra time to simulate after the end of the nominal
trajectory
times_cl_extended = np.arange(0, tf_actual+t_extend, dt)
states_cl_sw, ctrl_cl_sw = simulate_car_dyn(s_0.x, s_0.y, s_0.th, time
s_cl_extended, controller=switching_controller, noise_scale=noise_scal
e)

fig = plt.figure()
astar.plot_path(fig.number)
plot_traj_smoothed(traj_smoothed)
plot_traj_cl(states_cl)
def plot_traj_cl_sw(states_cl_sw):
    print(states_cl_sw[:,0])
    print(states_cl_sw[:,1])
    plt.plot(states_cl_sw[:,0], states_cl_sw[:,1], color="black", line
width=1, label="SwitchingController closed-loop path", zorder=10)
plot_traj_cl_sw(states_cl_sw)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=T
rue, ncol=3)
plt.show()
```

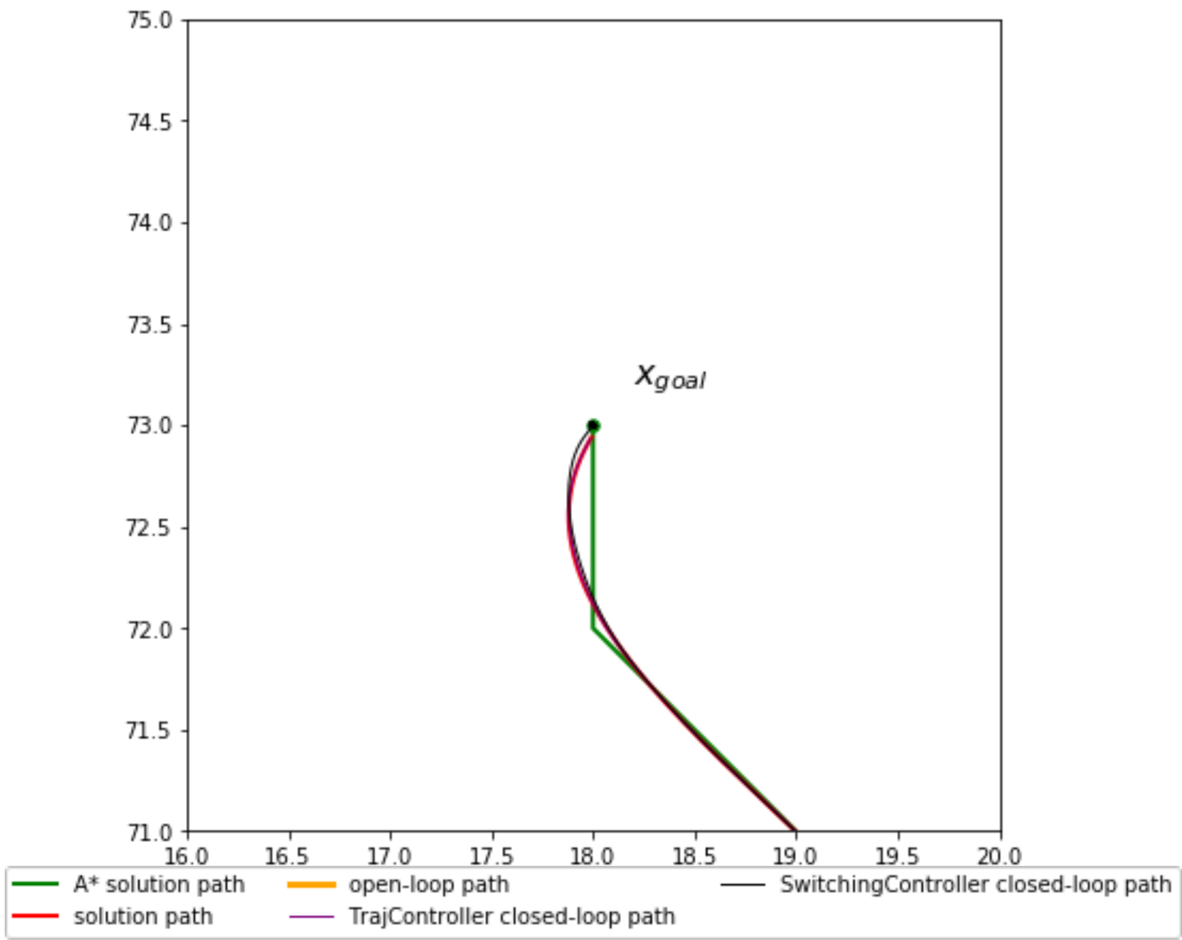
```
[52.04219378 52.0272388  52.01231248 ... 17.97970831 17.98598673
 17.9924706 ]
[ 1.99117545  1.99464452  1.99802059 ... 72.91849702 72.92941025
 72.94032666]
[52.         51.98381207 51.97199767 ... 17.97677353 17.98444102
 17.98875597]
[ 2.          2.00378781  2.00652015 ... 72.91503008 72.92862465
 72.93603367]
[52.         51.97995835 51.95434654 ... 17.98789058 17.98907926
 17.99001039]
[ 2.          2.00460474  2.01030952 ... 72.9906493  72.99136491
 72.99191652]
```



Zoom in on final pose

In [345]:
```
l_window = 4.

fig = plt.figure(figsize=[7,7])
astar.plot_path(fig.number, show_init_label = False)
plot_traj_smoothed(traj_smoothed)
plot_traj_ol(states_ol)
plot_traj_cl(states_cl)
plot_traj_cl_sw(states_cl_sw)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=True, ncol=3)
plt.axis([x_goal[0]-l_window/2, x_goal[0]+l_window/2, x_goal[1]-l_window/2, x_goal[1]+l_window/2])
plt.show()
```

```
[52.04219378 52.0272388  52.01231248 ... 17.97970831 17.98598673
 17.9924706 ]
[ 1.99117545  1.99464452  1.99802059 ... 72.91849702 72.92941025
 72.94032666]
[52.         51.98381207 51.97199767 ... 17.97677353 17.98444102
 17.98875597]
[ 2.          2.00378781  2.00652015 ... 72.91503008 72.92862465
 72.93603367]
[52.         51.97995835 51.95434654 ... 17.98789058 17.98907926
 17.99001039]
[ 2.          2.00460474  2.01030952 ... 72.9906493  72.99136491
 72.99191652]
```

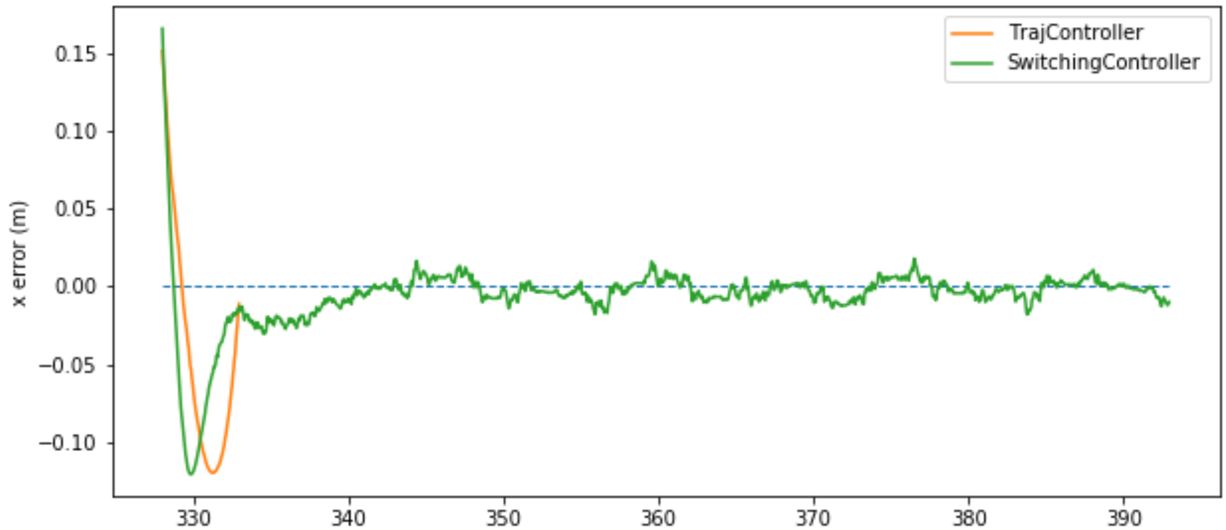


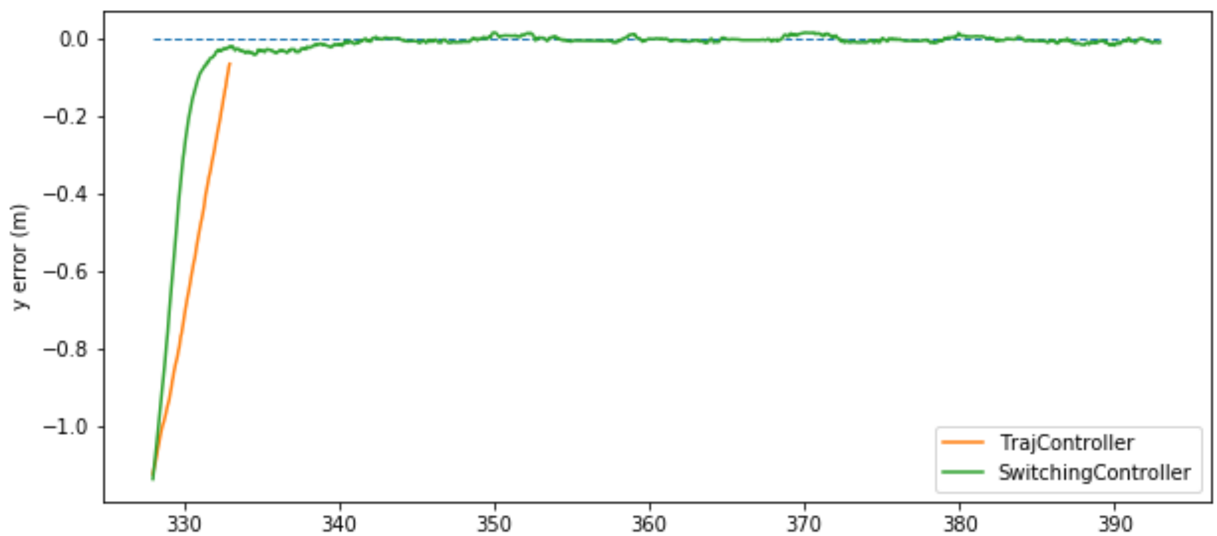
Plot final sequence of states

To see just how well we're able to arrive at the target point (and to assist in choosing values for the pose stabilization controller gains k_1, k_2, k_3), we plot the error in x and y for both the tracking controller and the switching controller at the end of the trajectory.

In [346]:
```
T = len(times_cl) - int(t_before_switch/dt)
fig = plt.figure(figsize=[10,10])
plt.subplot(2,1,1)
plt.plot([times_cl_extended[T], times_cl_extended[-1]], [0,0], linesty
le='--', linewidth=1)
plt.plot(times_cl[T:], states_cl[T:,0] - x_goal[0], label='TrajControl
ler')
plt.plot(times_cl_extended[T:], states_cl_sw[T:,0] - x_goal[0], label=
'SwitchingController')
plt.legend()
plt.ylabel("x error (m)")
plt.subplot(2,1,2)
plt.plot([times_cl_extended[T], times_cl_extended[-1]], [0,0], linesty
le='--', linewidth=1)
plt.plot(times_cl[T:], states_cl[T:,1] - x_goal[1], label='TrajControl
ler')
plt.plot(times_cl_extended[T:], states_cl_sw[T:,1] - x_goal[1], label=
'SwitchingController')
plt.legend()
plt.ylabel("y error (m)")
```

Out[346]:
```
Text(0, 0.5, 'y error (m)')
```



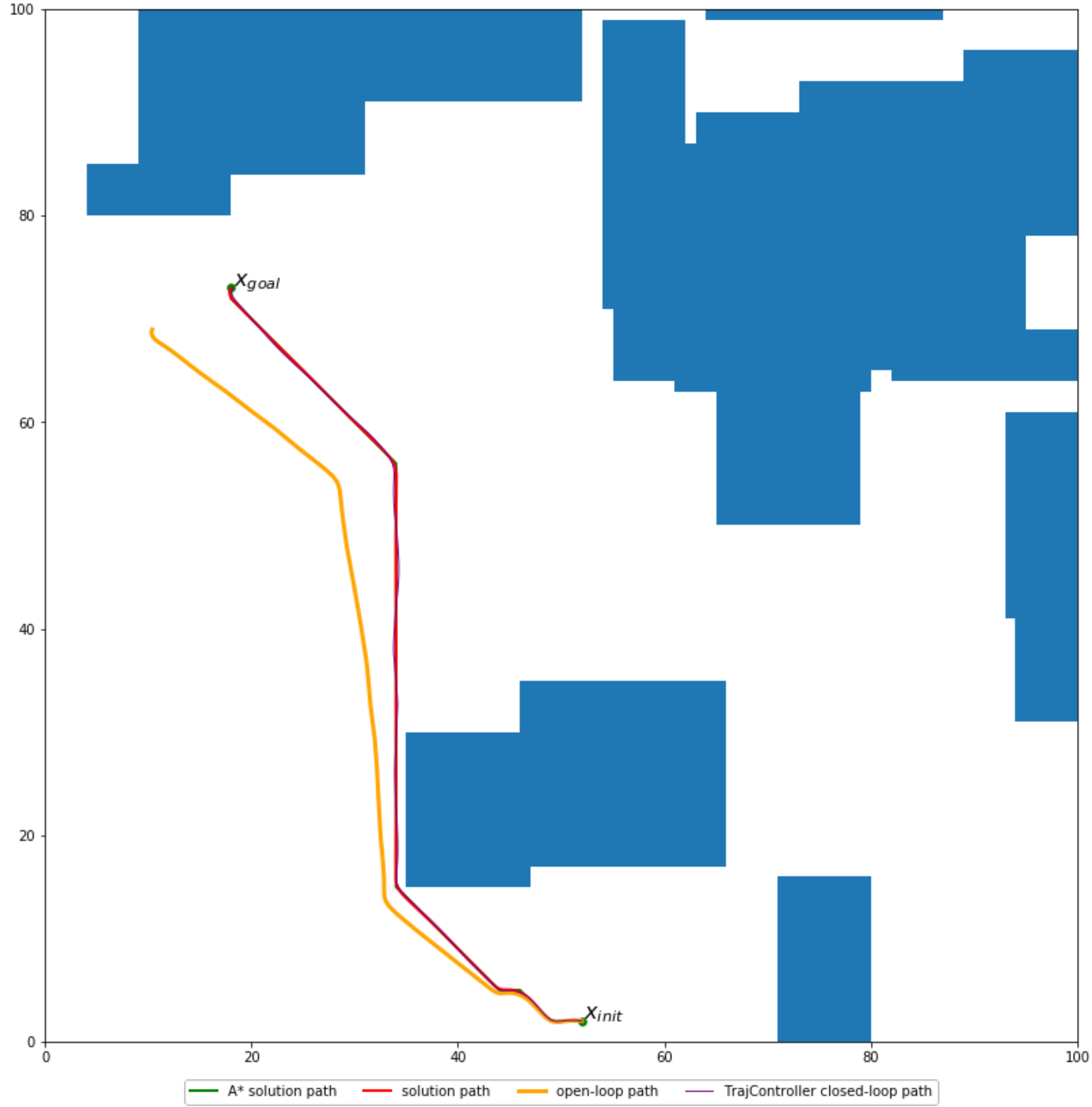


Try Lower Control Gains for Trajectory Tracking

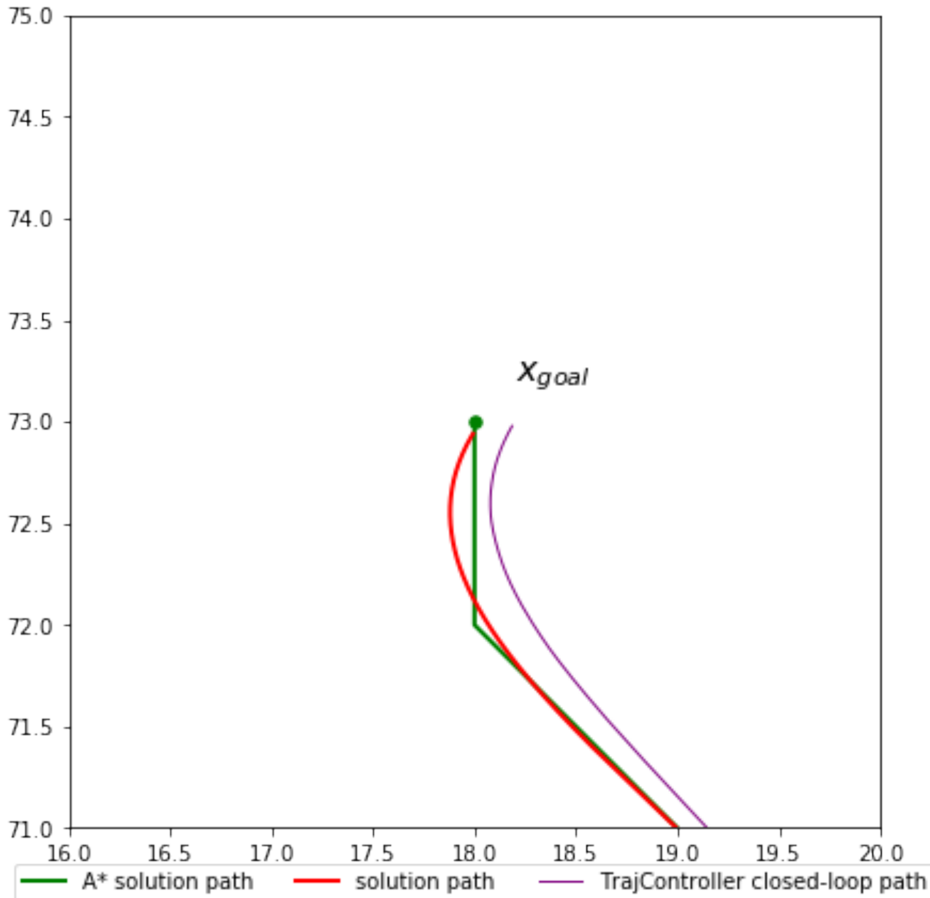
In [347]:
```
kpx = 0.01
kpy = 0.01
kdx = 0.01
kdy = 0.01

t_new, V_smooth_scaled, om_smooth_scaled, traj_smooth_scaled = modify_
traj_with_limits(traj_smoothed, t_smoothed, V_max, om_max, dt)
traj_controller = TrajectoryTracker(kpx=kpx, kpy=kpy, kdx=kdx, kdy=kdy
, V_max=V_max, om_max=om_max)
traj_controller.load_traj(t_new, traj_smooth_scaled)
tf_actual = t_new[-1]
times_cl = np.arange(0, tf_actual, dt)
```

```
s_0 = State(x=x_init[0], y=x_init[1], V=V_max, th=traj_smooth_scaled[0
,2])
s_f = State(x=x_goal[0], y=x_goal[1], V=V_max, th=traj_smooth_scaled[-
1,2])

actions_ol = np.stack([V_smooth_scaled, om_smooth_scaled], axis=-1)
states_ol, ctrl_ol = simulate_car_dyn(s_0.x, s_0.y, s_0.th, times_cl,
actions=actions_ol, noise_scale=noise_scale)
states_cl, ctrl_cl = simulate_car_dyn(s_0.x, s_0.y, s_0.th, times_cl,
controller=traj_controller, noise_scale=noise_scale)

fig = plt.figure()
astar.plot_path(fig.number)
plot_traj_smoothed(traj_smoothed)
def plot_traj_ol(states_ol):
    plt.plot(states_ol[:,0],states_ol[:,1], color="orange", linewidth=
3, label="open-loop path", zorder=10)
def plot_traj_cl(states_cl):
    print(states_cl[:,0])
    print(states_cl[:,1])
    plt.plot(states_cl[:,0], states_cl[:,1], color="purple", linewidth
=1, label="TrajController closed-loop path", zorder=10)
plot_traj_ol(states_ol)
plot_traj_cl(states_cl)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=T
rue, ncol=4)
plt.show()

l_window = 4.

fig = plt.figure(figsize=[7,7])
astar.plot_path(fig.number, show_init_label = False)
plot_traj_smoothed(traj_smoothed)
plot_traj_cl(states_cl)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=T
rue, ncol=3)
plt.axis([x_goal[0]-l_window/2, x_goal[0]+l_window/2, x_goal[1]-l_wind
ow/2, x_goal[1]+l_window/2])
plt.show()
```

```
[52.04219378 52.0272388  52.01231248 ... 17.97970831 17.98598673
 17.9924706 ]
[ 1.99117545  1.99464452  1.99802059 ... 72.91849702 72.92941025
 72.94032666]
[52.         51.98702603 51.96837998 ... 18.17402464 18.1796322
 18.1855498 ]
[ 2.          2.00299368  2.00719343 ... 72.95815779 72.96829372
 72.97865904]
```




```
[52.04219378 52.0272388  52.01231248 ... 17.97970831 17.98598673
 17.9924706 ]
[ 1.99117545  1.99464452  1.99802059 ... 72.91849702 72.92941025
 72.94032666]
[52.         51.98702603 51.96837998 ... 18.17402464 18.1796322
 18.1855498 ]
[ 2.          2.00299368  2.00719343 ... 72.95815779 72.96829372
 72.97865904]
```



75.0
74.5
74.0
73.5
73.0
72.5
72.0
71.5
71.0
16.0
16.5
17.0
17.5
18.0
18.5
19.0
19.5
20.0
x_{goal}
A* solution path
solution path
TrajController closed-loop path

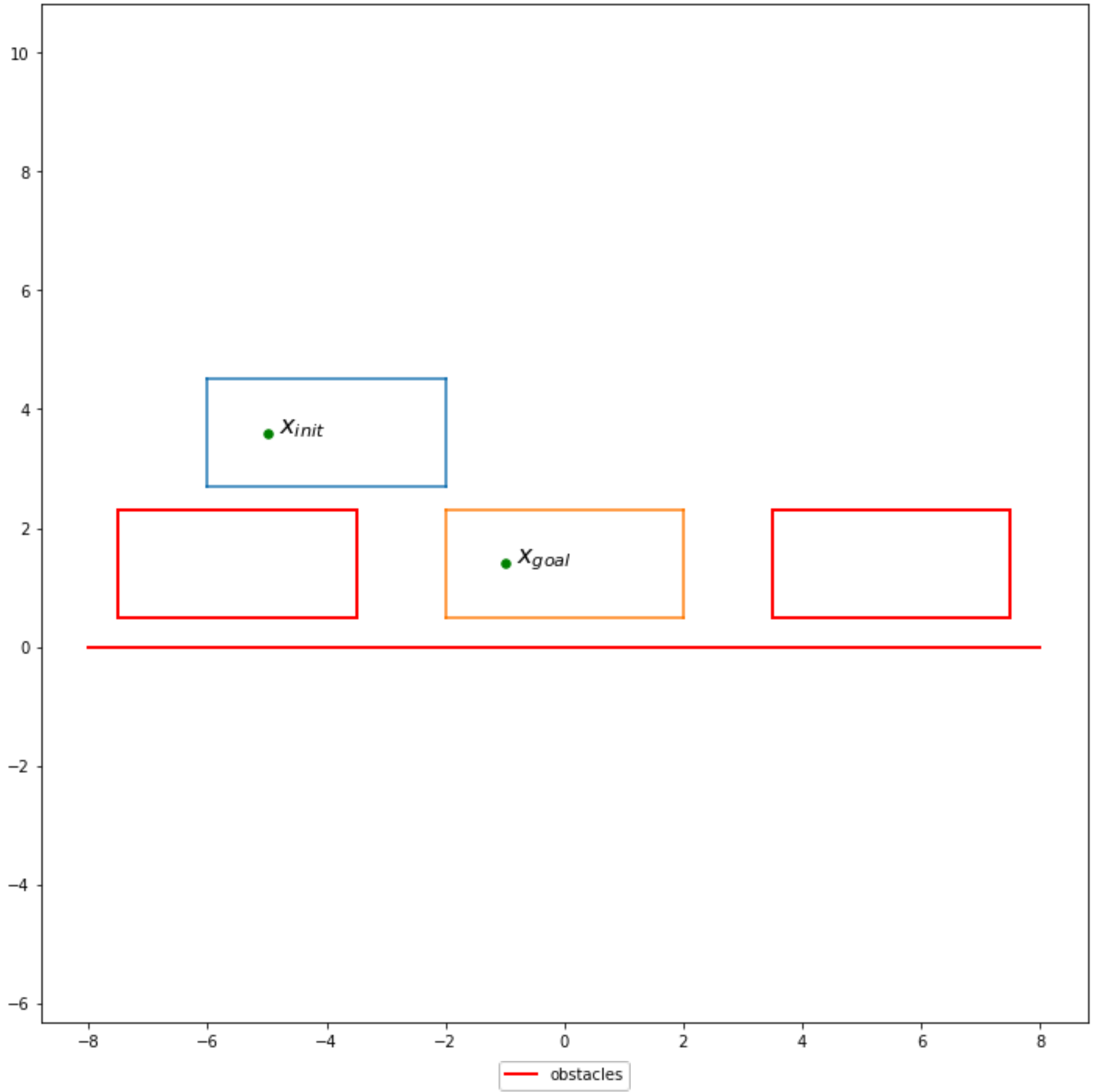
In [1]:
```
!pip3 install reeds-shepp
```

```
Collecting reeds-shepp
  Downloading reeds_shepp-1.0.7.tar.gz (45 kB)
     |████████████████████████████████| 45 kB 1.8 MB/s
Building wheels for collected packages: reeds-shepp
  Building wheel for reeds-shepp (setup.py) ... done
  Created wheel for reeds-shepp: filename=reeds_shepp-1.0.7-cp37-cp37m-linux_x86_64.whl size=181940 sha256=933ff9136e01aa19258bcc6246827e1da43ca5421bd62a1a8e7aa521ab62b77b
  Stored in directory: /root/.cache/pip/wheels/db/8f/b0/cc244db2ac9927783f636ecb40a683cb2a39578c234dde3a86
Successfully built reeds-shepp
Installing collected packages: reeds-shepp
Successfully installed reeds-shepp-1.0.7
```

In [11]:
```
# The autoreload extension will automatically load in new code as you edit files,
# so you don't need to restart the kernel every time
%load_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt
from P4_parallel_parking import ParkingRRT

plt.rcParams['figure.figsize'] = [12, 12]  # Change default figure size
```

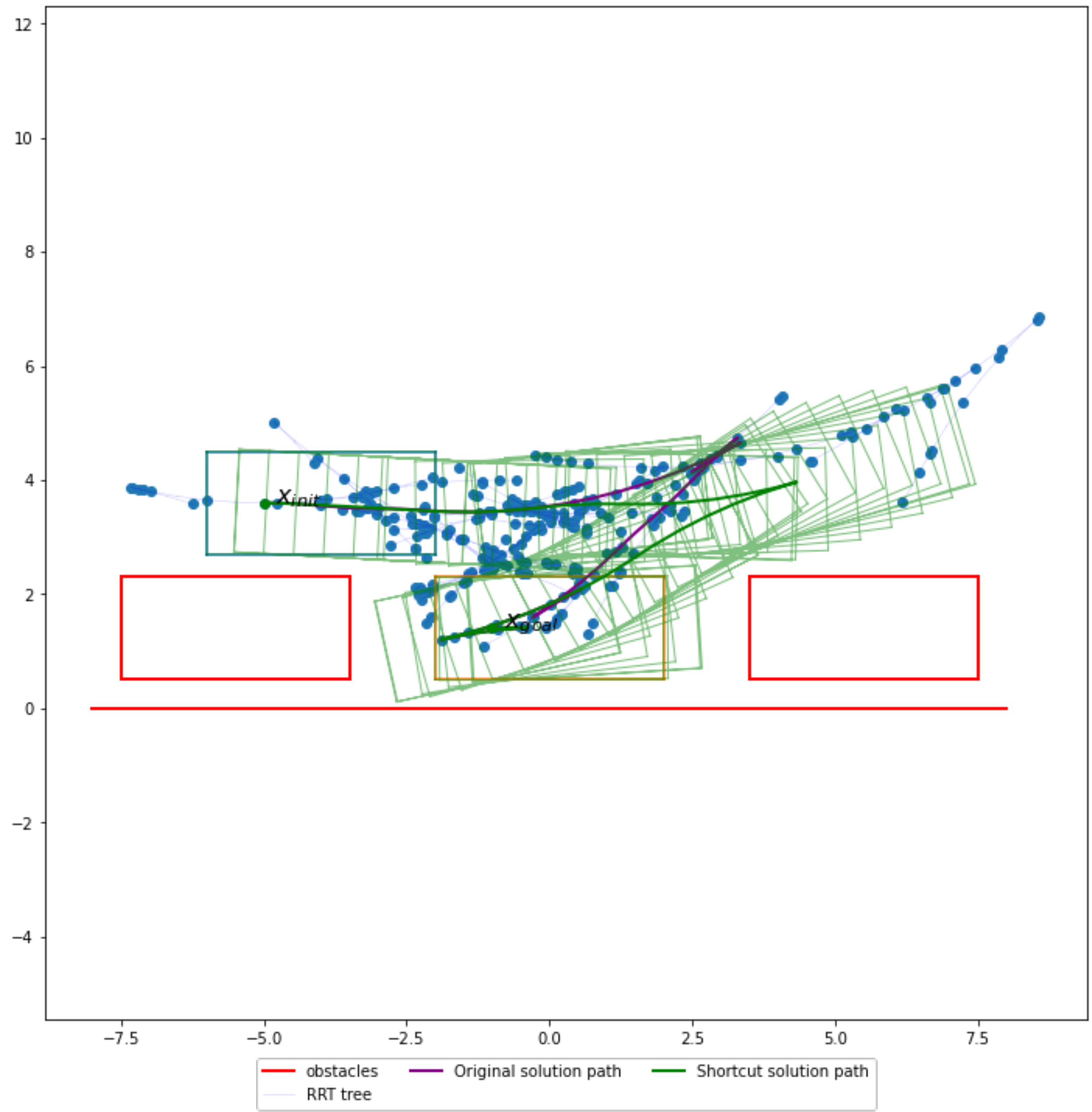
```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [12]:
```
x_init = [-5, 3.6, 0]
x_goal = [-1, 1.4, 0]
CAR = np.array([[[-2, -0.9], [2, -0.9]], [[2, -0.9], [2, 0.9]], [[2, 0.9], [-2, 0.9]], [[-2, 0.9], [-2, -0.9]]])
PARKING_SPOT = np.concatenate([CAR + np.array([5.5, 1.4]), CAR + np.array([-5.5, 1.4]),
                               np.array([[[-8, 0], [8, 0]]])], 0)

pp_rrt = ParkingRRT([-5, 0, -np.pi / 3], [5, 4, np.pi / 3], x_init, x_goal, PARKING_SPOT)
pp_rrt.plot_problem()
```



In [35]:
```
# RRT is a randomized algorithm; even though this planning problem is feasible, with a finite number of samples
# success is not guaranteed (though we see that with 1000 samples it seems to work more often than not). It's fun
# to see the different solutions RRT comes up with, but for debugging you may wish to use the fixed seed below.
#np.random.seed(1235)
pp_rrt.solve(1.0, 1000, shortcut=True)
```

Out[35]: True


12
10
8
6
4
2
0
−2
−4
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
x_{init}
x_{goal}
obstacles
Original solution path
Shortcut solution path
RRT tree

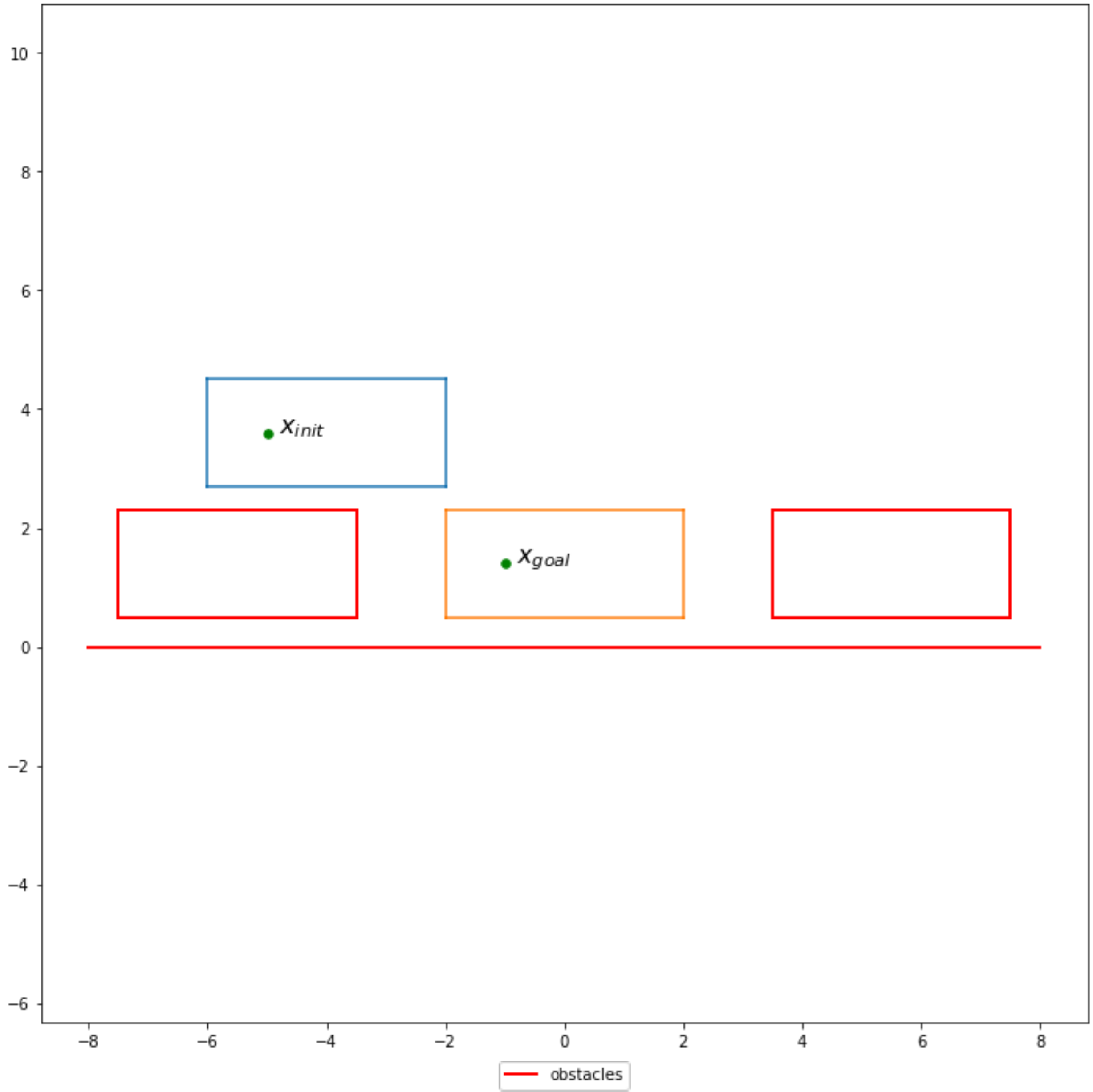
In [5]:
```
!pip3 install reeds-shepp
```

```
Collecting reeds-shepp
  Downloading reeds_shepp-1.0.7.tar.gz (45 kB)
     |████████████████████████████████| 45 kB 1.7 MB/s
Building wheels for collected packages: reeds-shepp
  Building wheel for reeds-shepp (setup.py) ... done
  Created wheel for reeds-shepp: filename=reeds_shepp-1.0.7-cp37-cp3
7m-linux_x86_64.whl size=181917 sha256=406e4ed5ae1c7b35b8070f15cbc6b
f9305d5cb9d1e564c335a8136a1ac4eb1d6
  Stored in directory: /root/.cache/pip/wheels/db/8f/b0/cc244db2ac99
27783f636ecb40a683cb2a39578c234dde3a86
Successfully built reeds-shepp
Installing collected packages: reeds-shepp
Successfully installed reeds-shepp-1.0.7
```

In [6]:
```
# The autoreload extension will automatically load in new code as you
edit files,
# so you don't need to restart the kernel every time
%load_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt
from P4_parallel_parking import ParkingRRT

plt.rcParams['figure.figsize'] = [12, 12]  # Change default figure siz
e
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

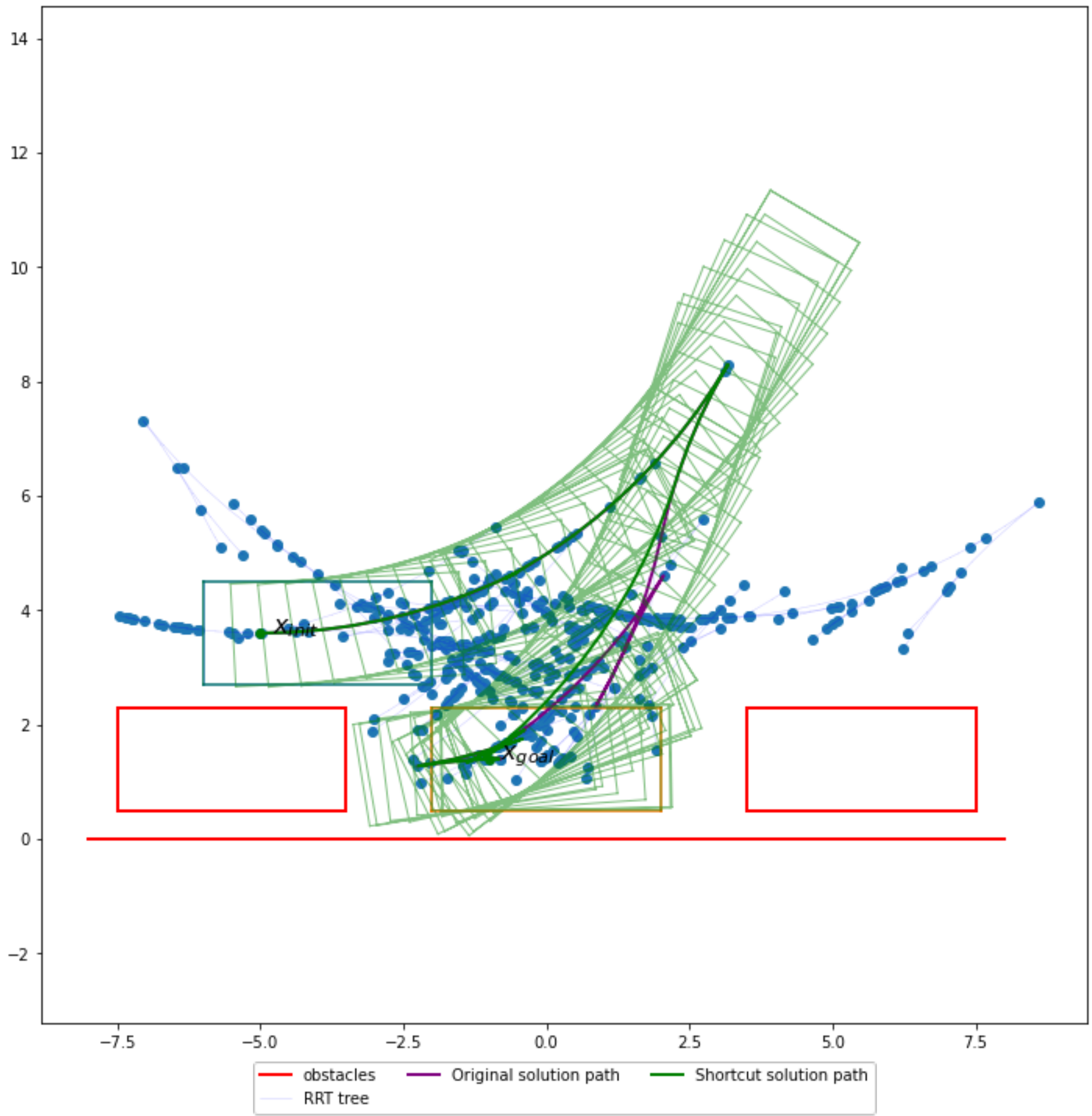
In [8]:
```
x_init = [-5, 3.6, 0]
x_goal = [-1, 1.4, 0]
CAR = np.array([[[-2, -0.9], [2, -0.9]], [[2, -0.9], [2, 0.9]], [[2, 0
.9], [-2, 0.9]], [[-2, 0.9], [-2, -0.9]]])
PARKING_SPOT = np.concatenate([CAR + np.array([5.5, 1.4]), CAR + np.ar
ray([-5.5, 1.4]),
                               np.array([[[-8, 0], [8, 0]]])], 0)

pp_rrt = ParkingRRT([-5, 0, -np.pi / 3], [5, 4, np.pi / 3], x_init, x_
goal, PARKING_SPOT)
pp_rrt.plot_problem()
```



In [23]:

```
# RRT is a randomized algorithm; even though this planning problem is
feasible, with a finite number of samples
# success is not guaranteed (though we see that with 1000 samples it s
eems to work more often than not). It's fun
# to see the different solutions RRT comes up with, but for debugging
you may wish to use the fixed seed below.
#np.random.seed(1235)
pp_rrt.solve(5, 2000, shortcut=True)
```

Out[23]: True



In [ ]: