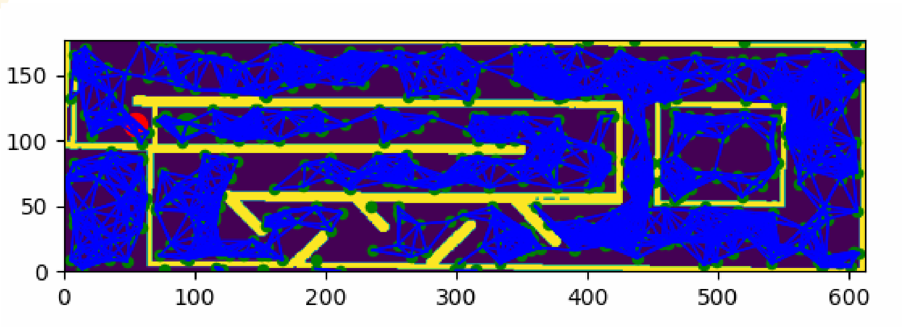
Mobile Robots - Wet HW2

PART 1 - PRM:

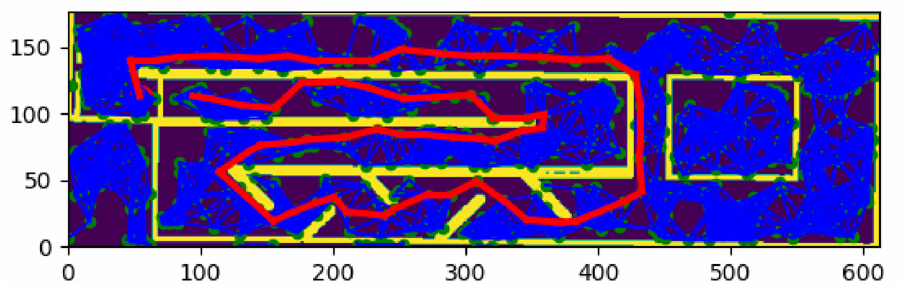
1. let's examine the RRT algorithm's performance as we increase iterations:

   `max_itr = 700`:

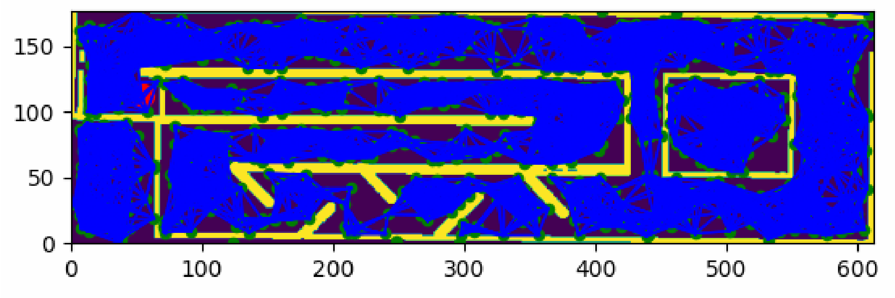
   `path cost: inf, time: 1.1930241584777832`

   we can see how the graph is a lot sparser than the next iterations and in-fact cuts off at several places, more iterations are needed to come up with a complete trajectory, the time is shorter since less iterations require less computations.
   trying another run:

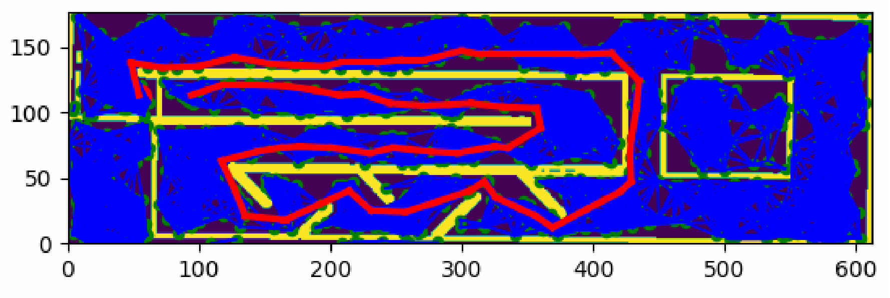
   `path cost: 1412.4489918614565, time: 1.3472466468811035`

`max_itr = 1000`:



`path cost: inf, time:2.753676652908325`

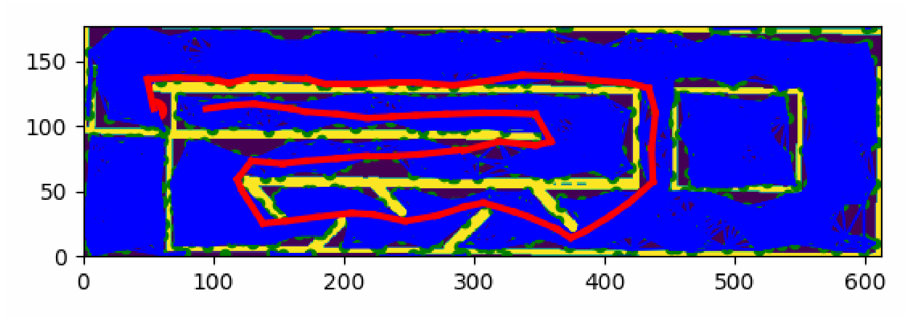
running another iteration:



```
path cost: 1410.3569118128148, time: 2.062098264694214
```

we can see how the graph is significantly denser in this iteration than previous, the addition of more iterations leads to a greater probability of success but but the time it takes to run the algorithm is longer since we do it for more iterations.

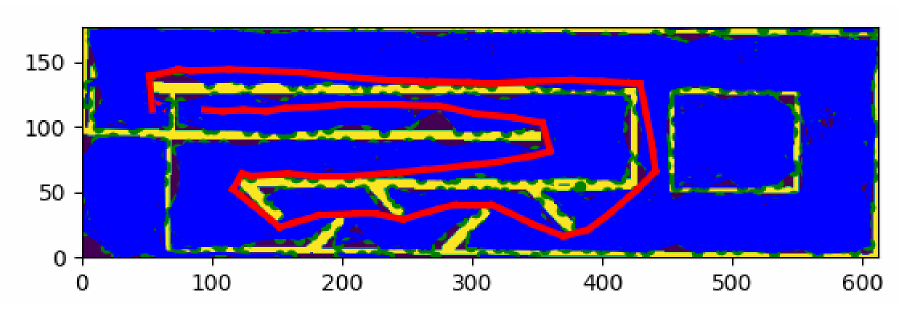
`max_itr=1500` :



```
path cost: 1372.7953501803677, time: 5.495332956314087
```

we can see how the path is more optimized since we have more iterations - more vertexes and the ability to connect them in a more optimized way - the trajectory hugs the wall more, and is straighter at places. The path cost is suitably smaller. The graph is a lot more dense, the computation time is suitably longer since we need to check more vertex & edge connections, as well as run an A^* algorithm in a denser graph.

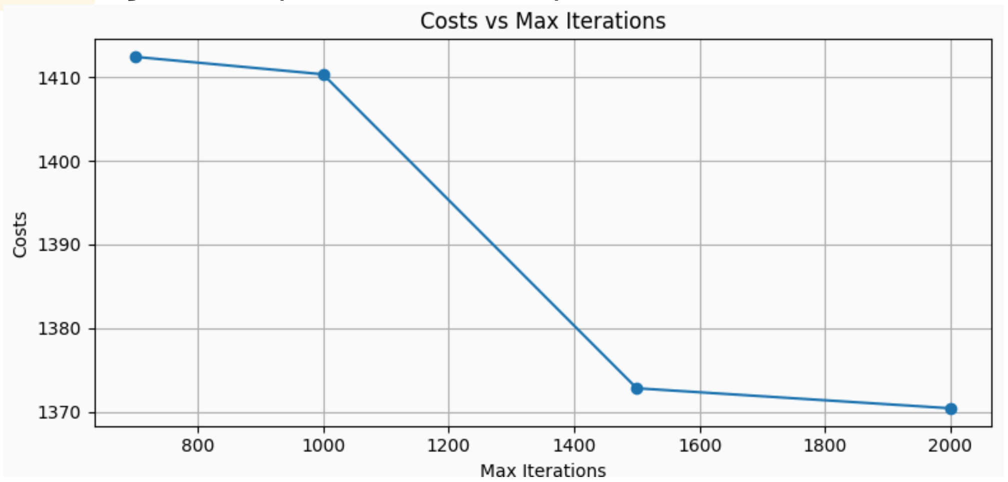
`max_itr = 2000` :

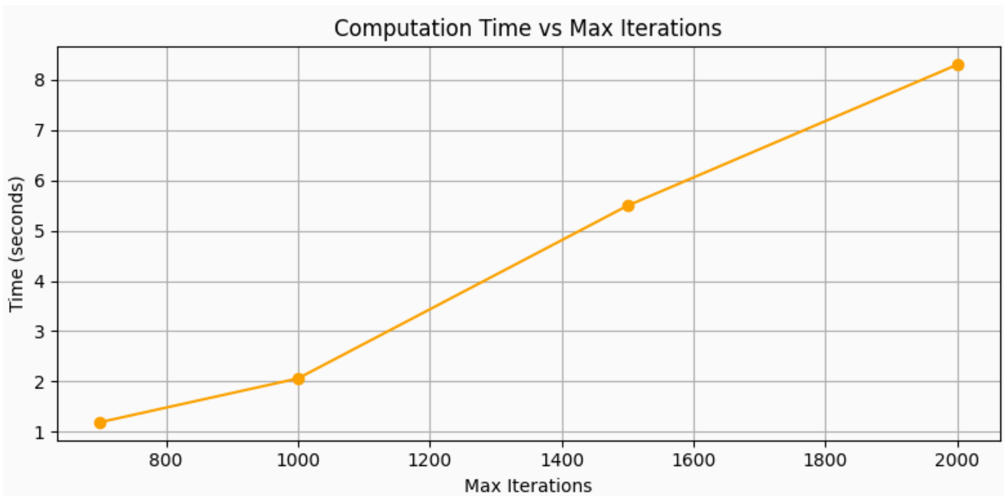


```
path cost: 1370.4019330203898, time: 8.296157121658325
```

we can see how the trends we outlined continue - the higher the iterations, the more time it takes to run the algorithm due to the density of the graph and computations required for each edge (& potential edge), but the solutions do get further optimized, we can see how at this point though we've hit diminishing returns - for a significant bump to computation time our improvements are negligible.

graphing `max_itr` against computation time and path cost:





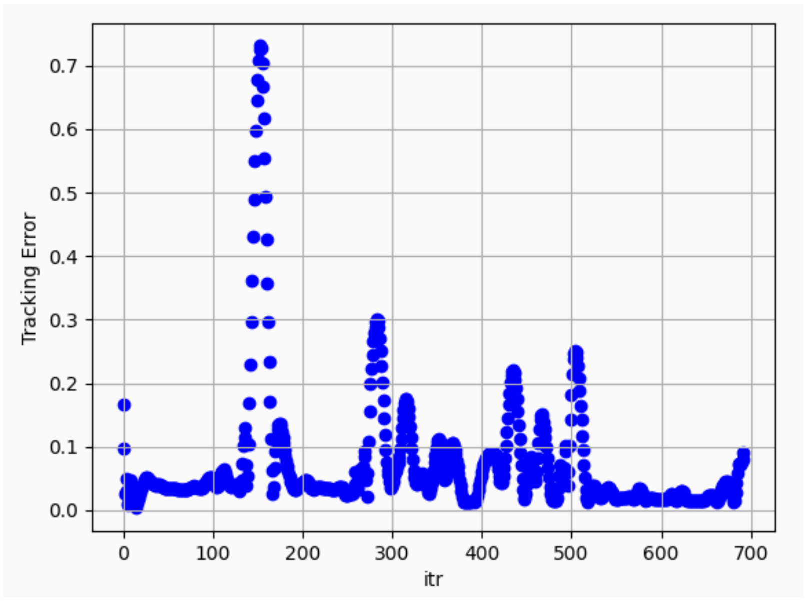
we can see how the cost does appear to get a reduction in the benefit whereas the computation time still soars with the additions of iterations.

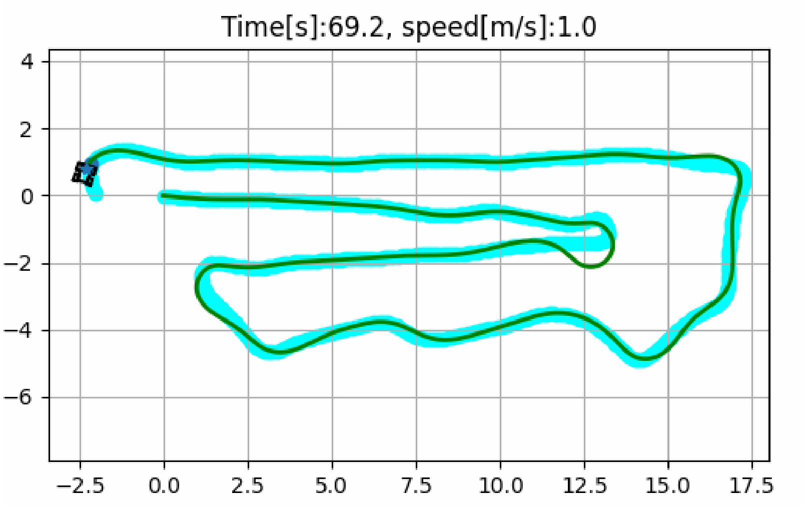
To convert the path to meters as requested we appended the following lines of code to `main()` : (for formatting)

```
            # Ensure yaw = 0 for each point
        path_index = [list(p) + [0] if len(p) == 2 else list(p) for p in path]

        # Convert to meters
        path_meter = converter.pathindex2pathmeter(path_index)

        # Remove duplicate points to avoid spline errors
        path_meter_clean = remove_consecutive_duplicates(path_meter)

        # Save clean path
        np.save("path_in_meters.npy", np.array(path_meter_clean))
```

where `remove_consecutive_duplicates` is defined here:

```
def remove_consecutive_duplicates(path):
    cleaned = [path[0]]
    for p in path[1:]:
        if not np.allclose(p, cleaned[-1]):
            cleaned.append(p)
    return cleaned
```





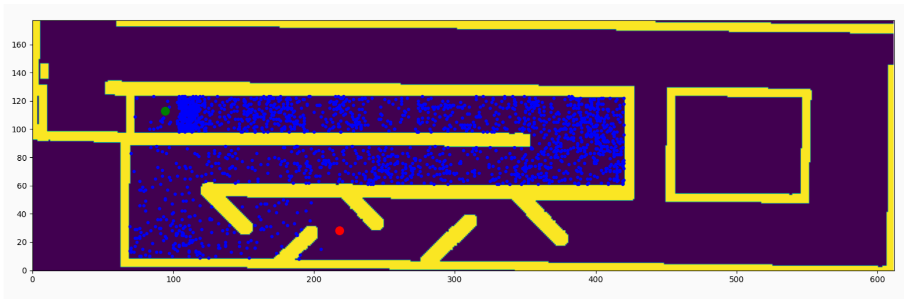
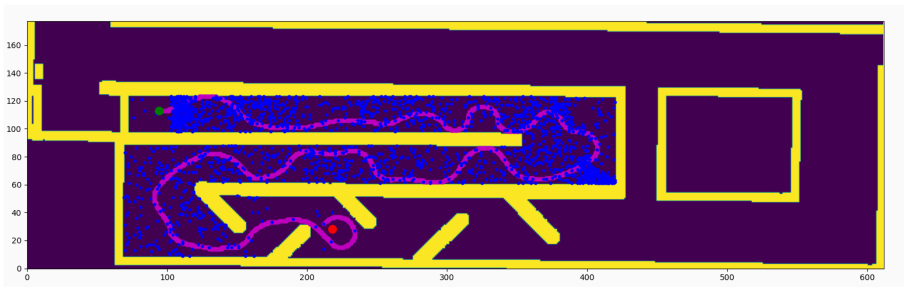
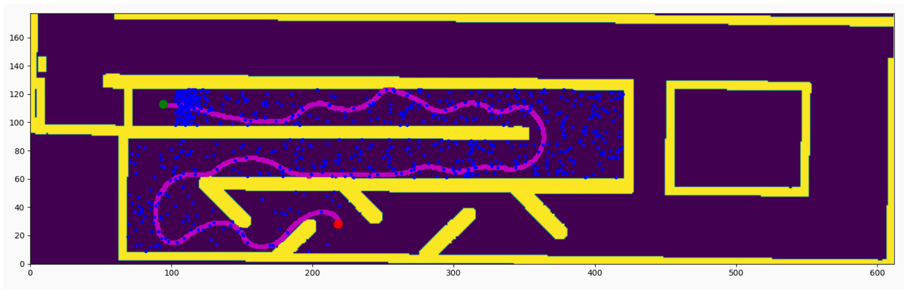
cool! so our robot now follows the path we charted out for it inside our tracking simulator!

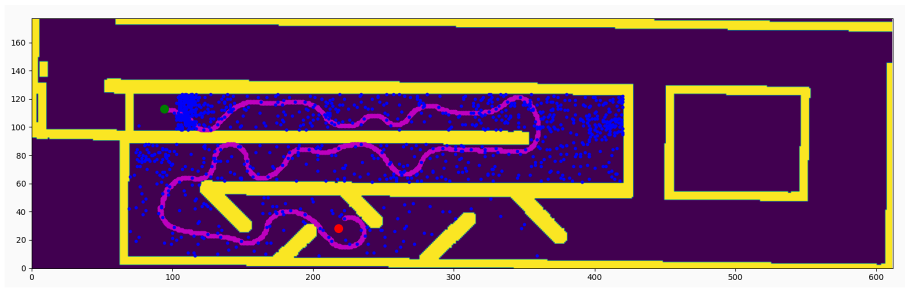
PART 2 - KinoRRT:

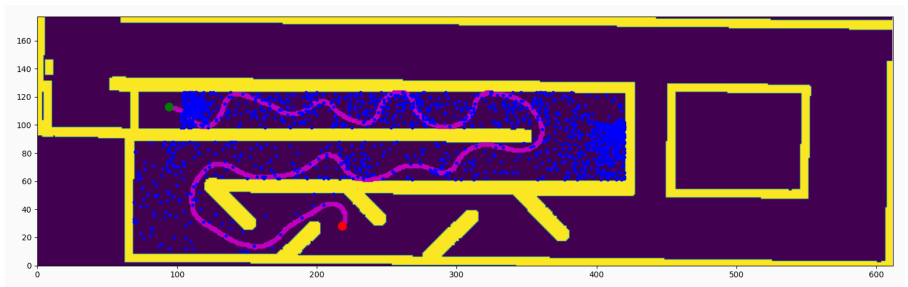
Run	iteration the path was found	cost	explanation
1	Not Found	None	KinoRRT did not get close enough to target (by our threshold) to finish the run, points mostly concentrated near start area, and first bend. sparser near target.
2	8414	1094.5	KinoRRT arrived at the desired threshold at a relatively high cost due to a wobbly and unoptimized trajectory - even did something of a loop near the end! (where the sparse datapoints allowed for it)
3	3562	701.11	KinoRRT achieved a far better result, being more direct and getting good sampled controls has led to an early convergence and a really small cost.
4	5501	896.0691128680553	later convergence relative to the previous trajectory, the trajectory is also curvier leading to a longer path.

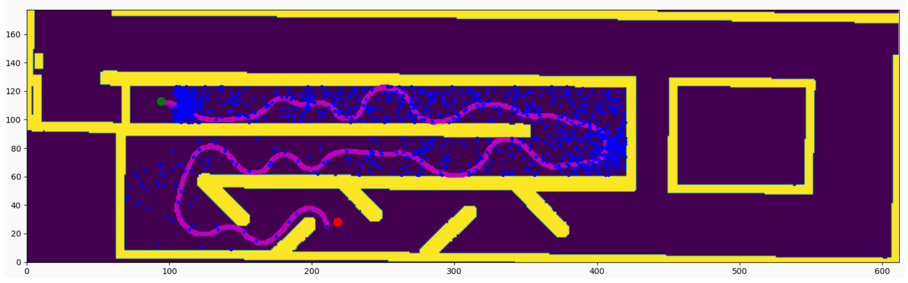
Run	iteration the path was found	cost	explanation
5	6419	743.5225240657132	
6	5123	914.9157417117681	
7	5447	735.8793976367659	here we can see the importance of rolling a random point and extending a branch from the closest - we overshot the target and the tree continued developing, however, a lucky sample helped us take a nearby point and finish the course.
8	2436	783.1859738037281	we can see how the graph here contains much fewer samples and iterations, we managed to hit the target (within threshold) really fast. Didn't leave a lot of room to spread and consequently our trajectory is one of the better ones.
9	7510	648.3911333811134	a new champion! we can see how there's not much of a correlation between iteration number and a good cost - the reason for that is that KinoRRT it is *not* asymptotically normal - we're not guaranteed a better solution!
10	Not Found	None	

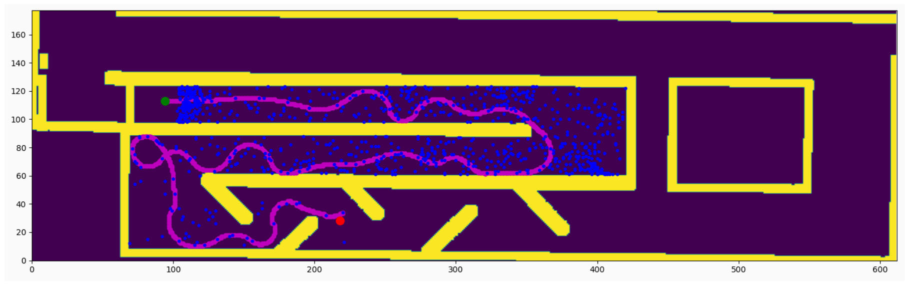
Graphs (presented in run order)

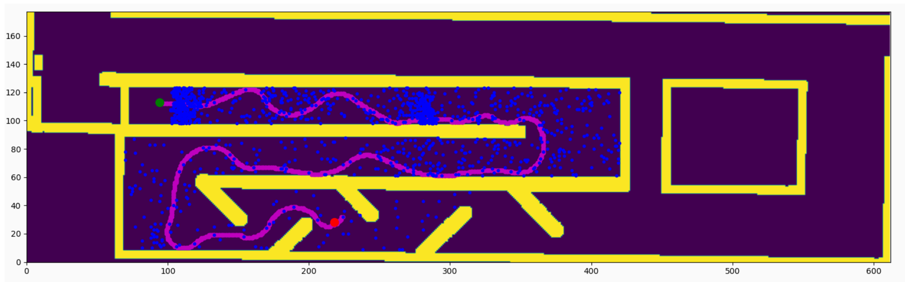




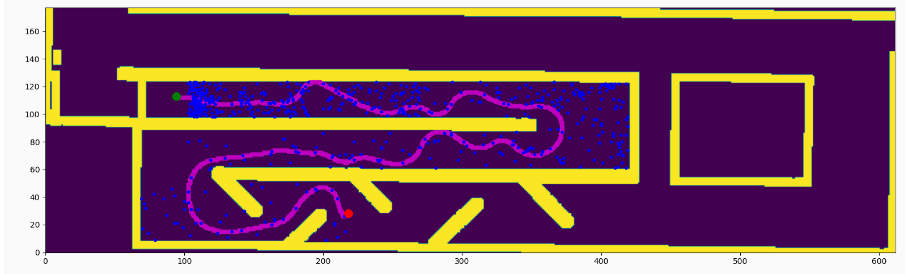

0 20 40 60 80 100 120 140 160
0 100 200 300 400 500 600

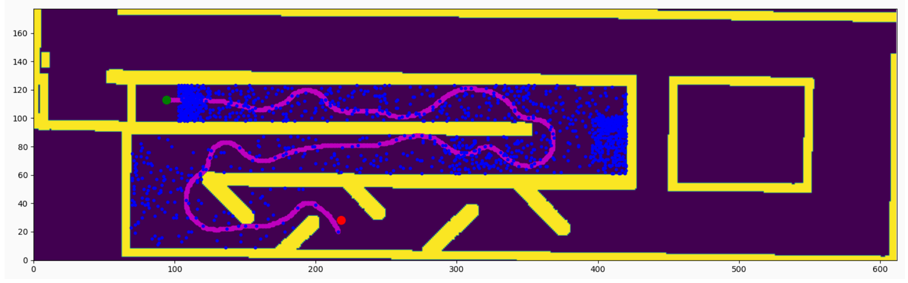

0 20 40 60 80 100 120 140 160
0 100 200 300 400 500 600


0 20 40 60 80 100 120 140 160
0 100 200 300 400 500 600

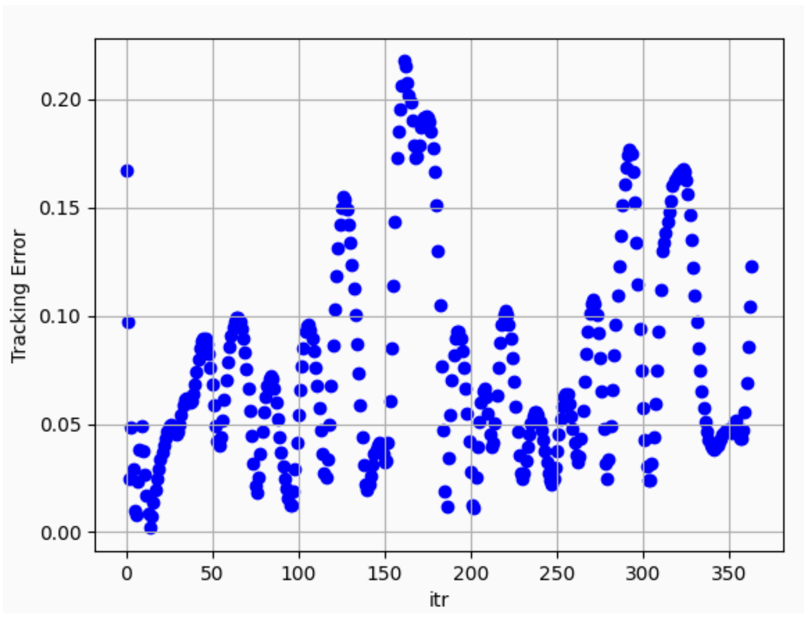

0 20 40 60 80 100 120 140 160
0 100 200 300 400 500 600

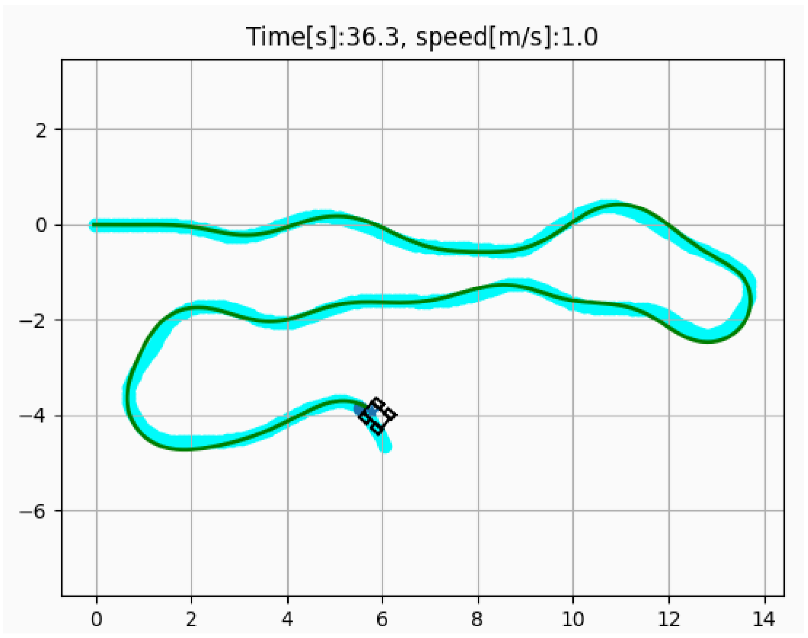

0 20 40 60 80 100 120 140 160
0 100 200 300 400 500 600


0 20 40 60 80 100 120 140 160
0 100 200 300 400 500 600


0 20 40 60 80 100 120 140 160
0 100 200 300 400 500 600

- Tracking error:





we can see how the car can track the trajectory at a much better consistency due to KinoRRT taking into account the robot dynamical model whereas the PRM didn't.