Car:

Planner

Tracker

Drone:

Planner

Tracker

Mapping for car model:

Upsample to approximately 1 by 1 m grid if necessary. Else leave it as it is. Add 2m padding around every obstacle (C-space)

Hybrid A\* for car model:

Allow 3 motion primitives going forward (straight forward, turning left, turning right). Move a distance equal to sqrt(2). Allow several orientations per grid?

Hybrid A\* for drone model:

Allow 8 motion primitives in every direction.

Mapping for drone model:

Upsample to approximately 1 by 1 m grid if necessary. Else leave it as it is. Add 2m padding around every obstacle (C-space)

Tracker for car model:

Obstacle recovery in terms of returning to a waypoint (4 m lookbehind) 1 second. Slow down before turns and increase lookahead when speed increases.

Tracker for drone:

Should not hit any obstacles! Therefore, we must go slow and slow down before turns.

Todo:

Vary lookahead based on speed. (CarAI.cs)

Limit maximum speed based on angle ahead. (CarAI.cs)

Allow several orientations in same cell. (Planner.cs)

Fix mapper so it upsamples big cell maps (low resolution) and keeps small cell maps (high resolution). (Mapper.cs)

Fix mapper so it adds 2m padding regardless of resolution. (Mapper.cs)

Make A\* version for the drone (Planner.cs)

Test Douglas-Peucker algorithm for removing abundant nodes (Yuxin, Tuesday, New function)

If it works, apply to CarAI on Wednesday.

Create a map that contains the distance to the closest obstacle for every cell in the obstacle map (Mapper.cs, Yuxin, Tuesday)

If it works, apply to Planner today or Wednesday.

Make CarAI.cs readable and keep tuning things (CarAI.cs, Herman, Tuesday)

If it works, apply to DroneAI on Wednesday

Apply tracking to the drone (DroneAI.cs, Herman, Wednesday)

Try different terrains and fine tune the models individually (CarAI.cs and DroneAI.cs) (Wednesday and Thursday)

For every cell in the distance map

If the same grid index in the obstacle map is 1

Return 0

Else

For every 1 in the obstacle map:

Calculate distance to cell in distance map

Return minimum distance

For every node n in dp\_path

If not last

If Euclidean(n, n + 1) < distance minimum

New\_path.Add(n)

Continue;

Else

New\_path.Add(n)

L = Euclidean(n, n + 1) / distance minimum

For I = 0:floor(L)

New\_path.Add(n + I \* Euclidean(n, n + 1) / floor(L))