Navigation Robotique

PROJET DU COURS ROB201

EDOUARD CLOCHERET

Objectif : Cartographier un lieu pour s'y déplacer efficacement

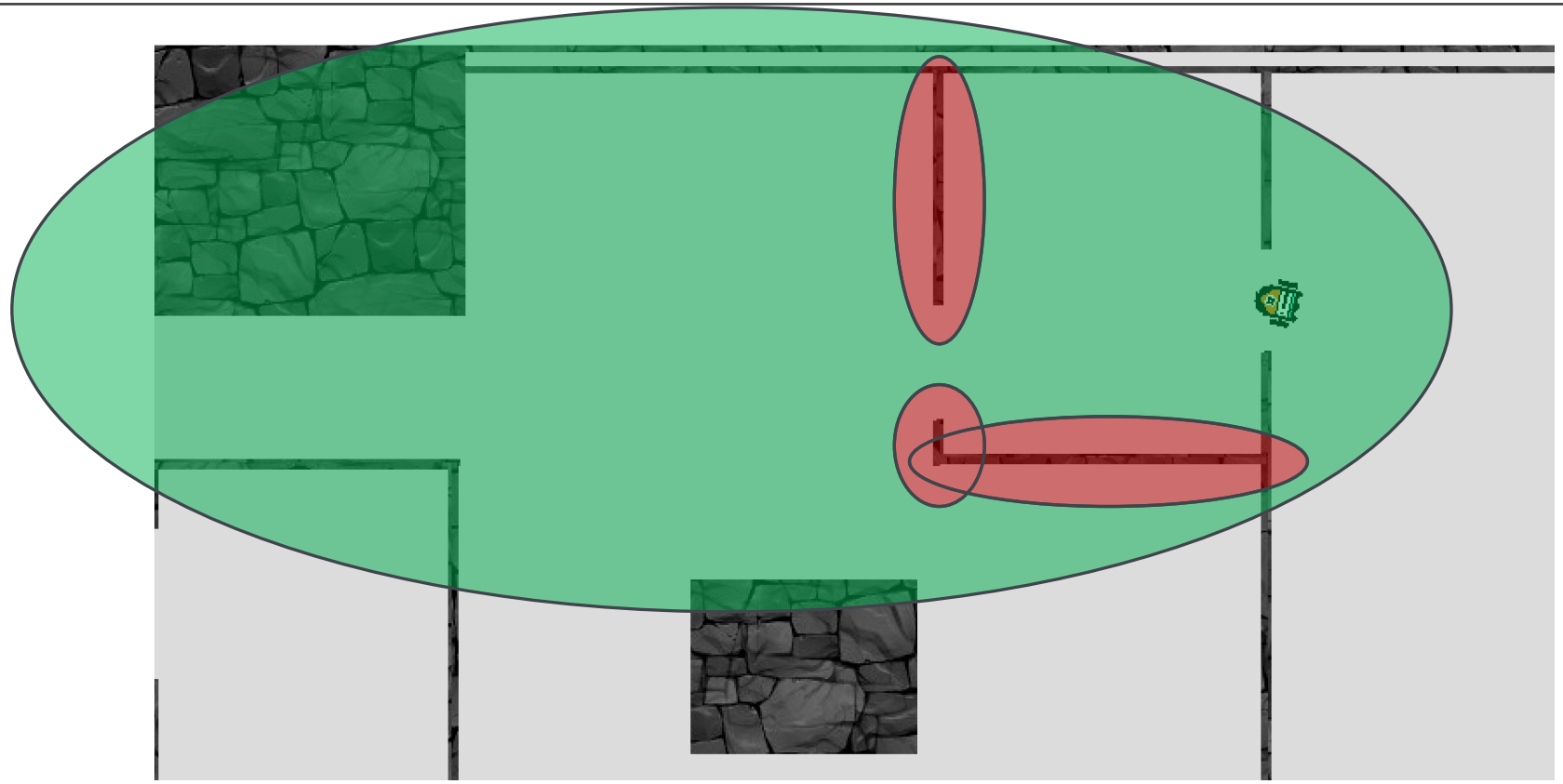
I - POTENTIAL FIELD CONTROL

II – CARTOGRAPHIE LIDAR

III – LOCALISATION

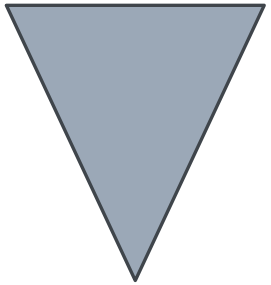
IV – PLUS COURT CHEMIN : A*

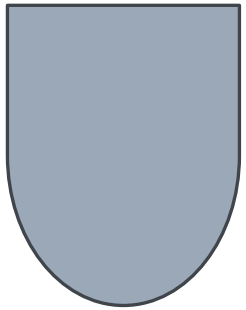
Potential Field Control



Code

```
if distance_to_goal < d_quad :
    #Potentiel attractif quadratique vers le goal (norme qui diminiue)
    gradient_goal = K_goal/d_quad * (goal_pose[0:2] - current_pose[0:2])
else :
    #Potentiel attractif linéaire vers le goal (norme constante)
    gradient_goal = K_goal/distance_to_goal * (goal_pose[0:2] - current_pose[0:2])
```



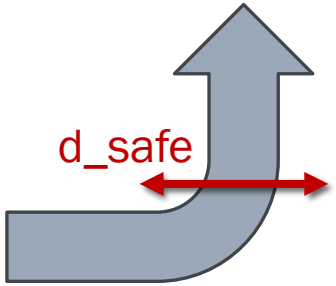


Potentiel attractif :
Linéaire loin du but

Quadratique près du but

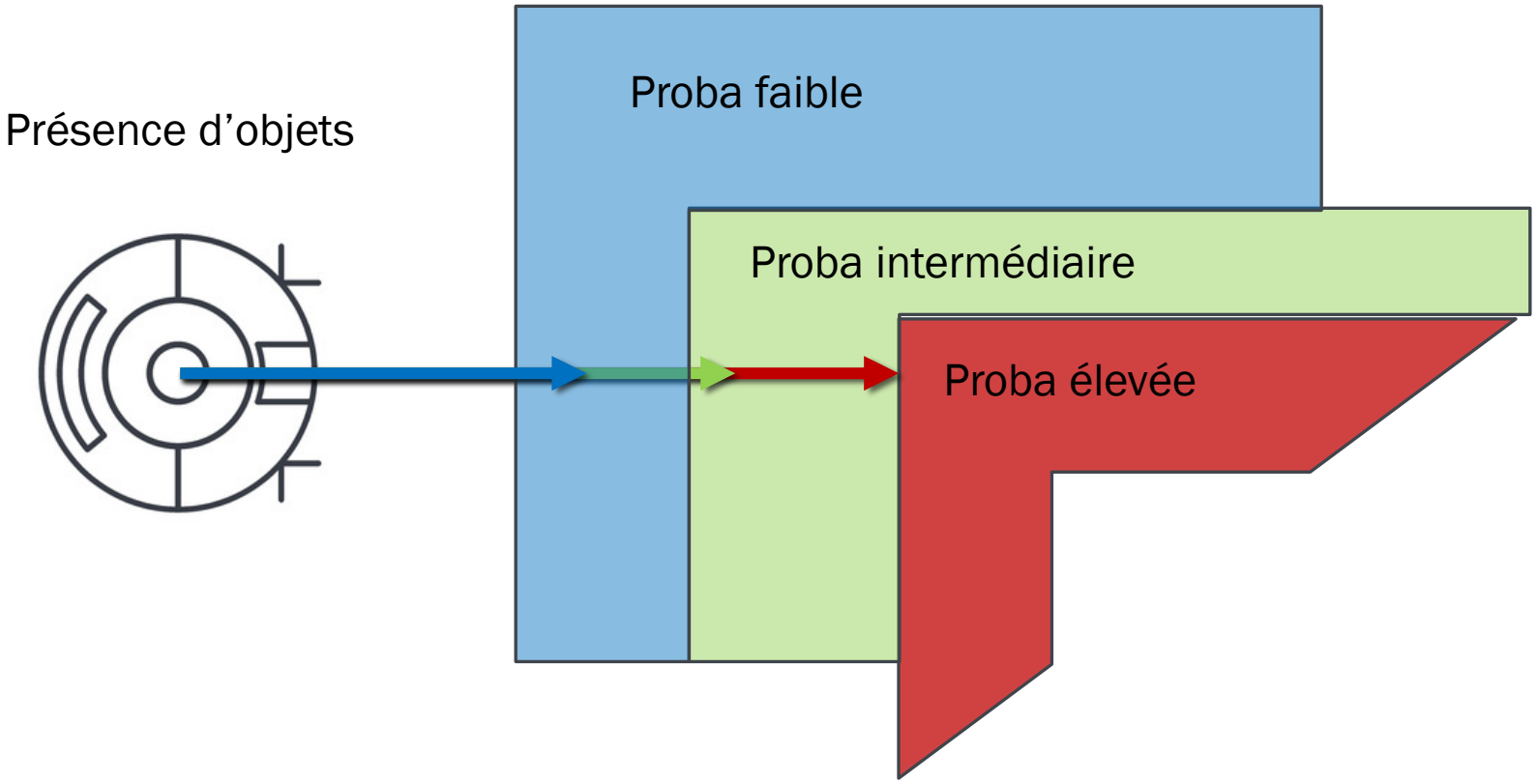
```
d_safe = 350 #Un obstacle n'a pas d'influence à plus de d_safe
if distance_to_obstacle > d_safe :
    gradient_obstacle = [0,0]
else :
    gradient_obstacle = K_obs/ (distance_to_obstacle)**3 *(1/distance_to_obstacle -

#Combinaison de la partie du gradient provennant du goal et de l'obstacle
gradient = gradient_goal - gradient_obstacle
```



Potentiel répulsif en 1/x
près des obstacles

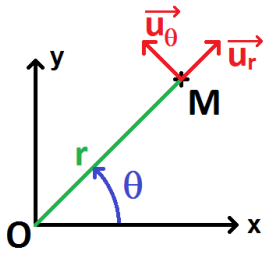
Cartographie Lidar



Code

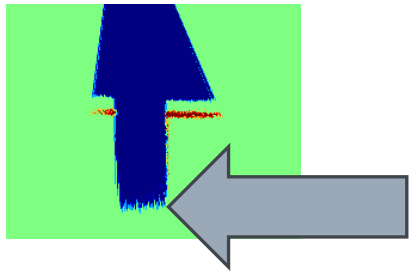
```
radiuses = lidar.get_sensor_values()
angles = lidar.get_ray_angles()

x_obs = pose[0] + radiuses * np.cos(angles + pose[2])
y_obs = pose[1] + radiuses * np.sin(angles + pose[2])
```



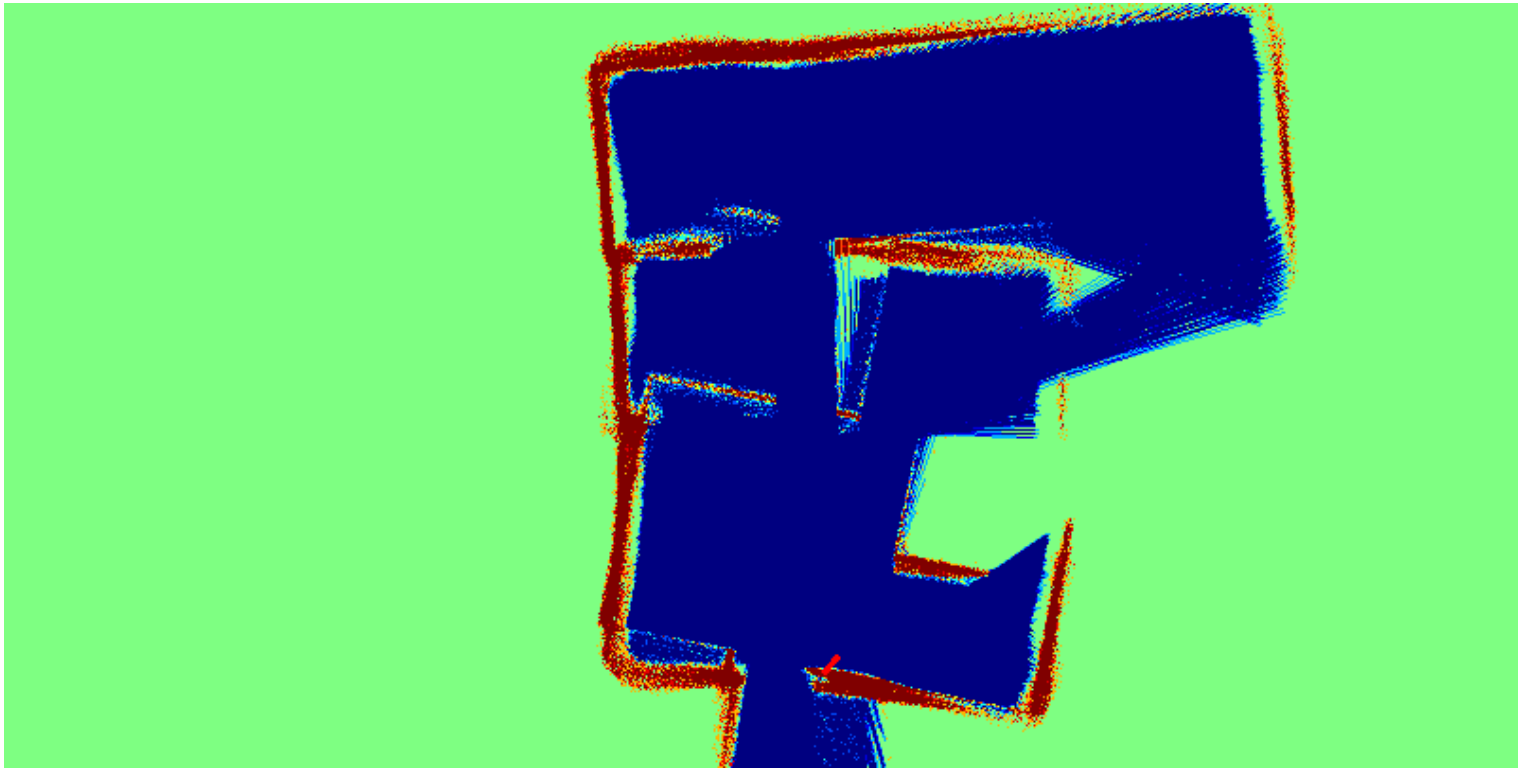
Changement de coordonnées

```
#Si la portée maximale du lidar est atteinte,
#on n'ajoute pas de mur
masque = radiuses<550
#ajout d'une proba élevée à la position de l'obstacle
self.grid.add_map_points(x_obs[masque], y_obs[masque], 2 )
```

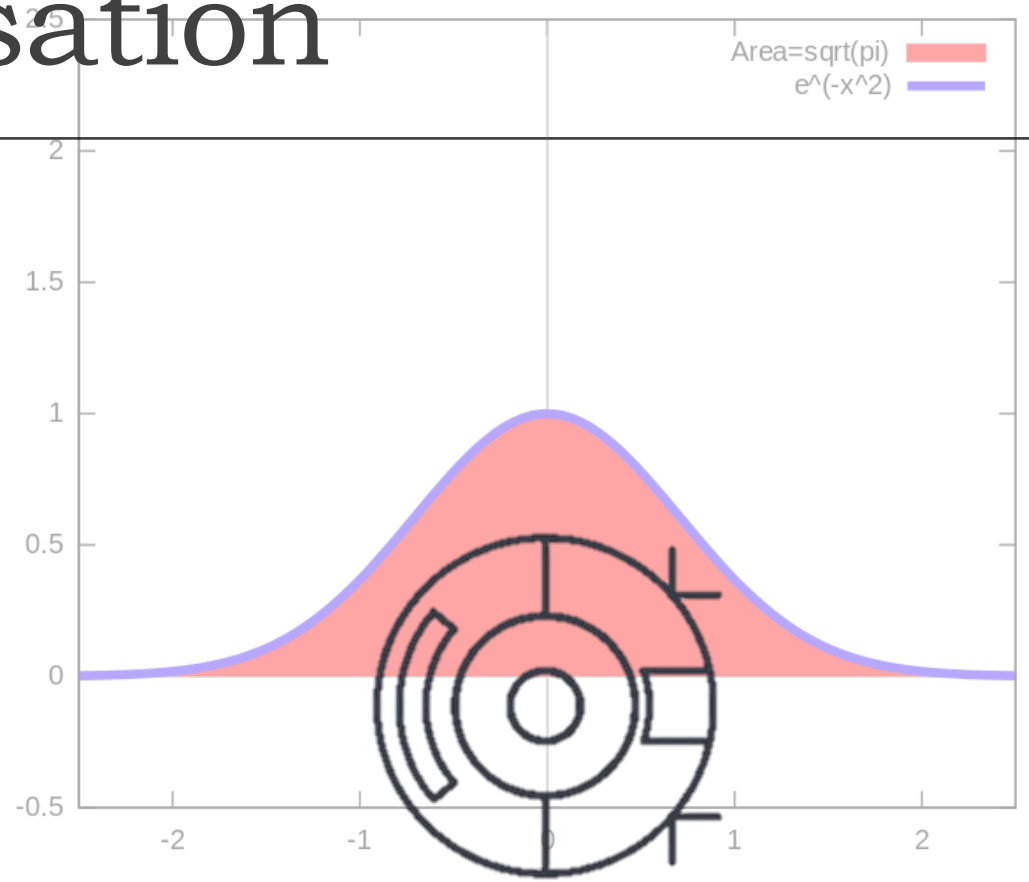


Pas d'ajout de mur si portée maximale atteinte

Problème : dérive liée à l'odométrie



Localisation

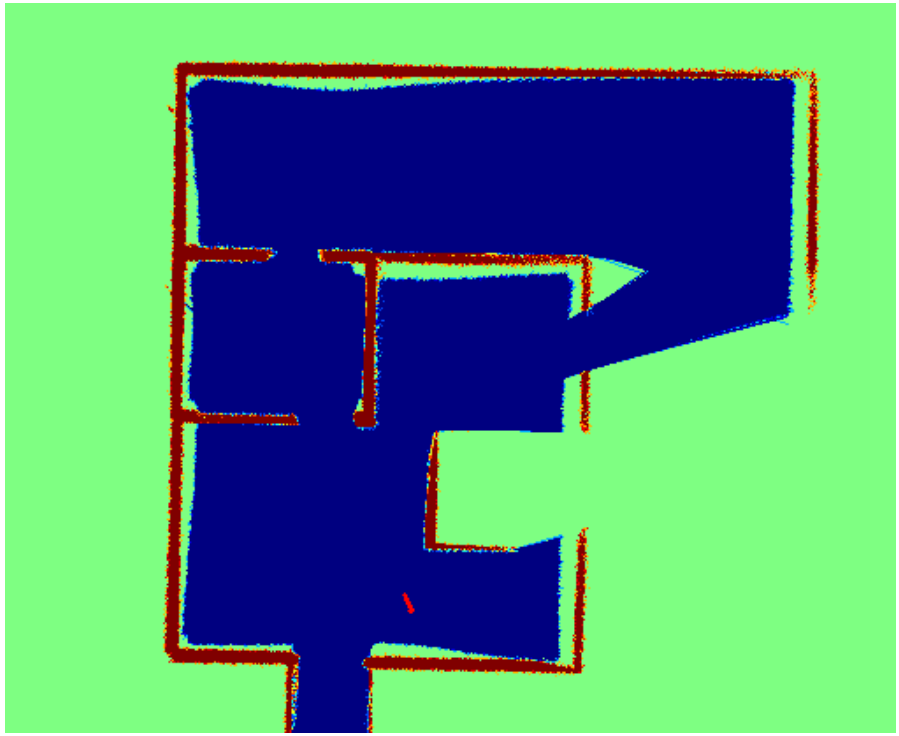


Code

```
def localise(self, lidar, raw_odom_pose):
    N = 200  # nombre de tirages de bruits
    for i in range (N):
        variance_pos = 0.1 #en x et y
        variance_theta = 0.01
        # Generate a random offset in 3 dimensions
        offset = np.random.multivariate_normal(mean=np.zeros(3), cov=np.diag([varia
        #print("offest",offset)
        new_ref_pose = best_ref_pose + offset
```

- Centaines de tirages aléatoires
- Tirage d'un offset
- Evaluation de l'offset

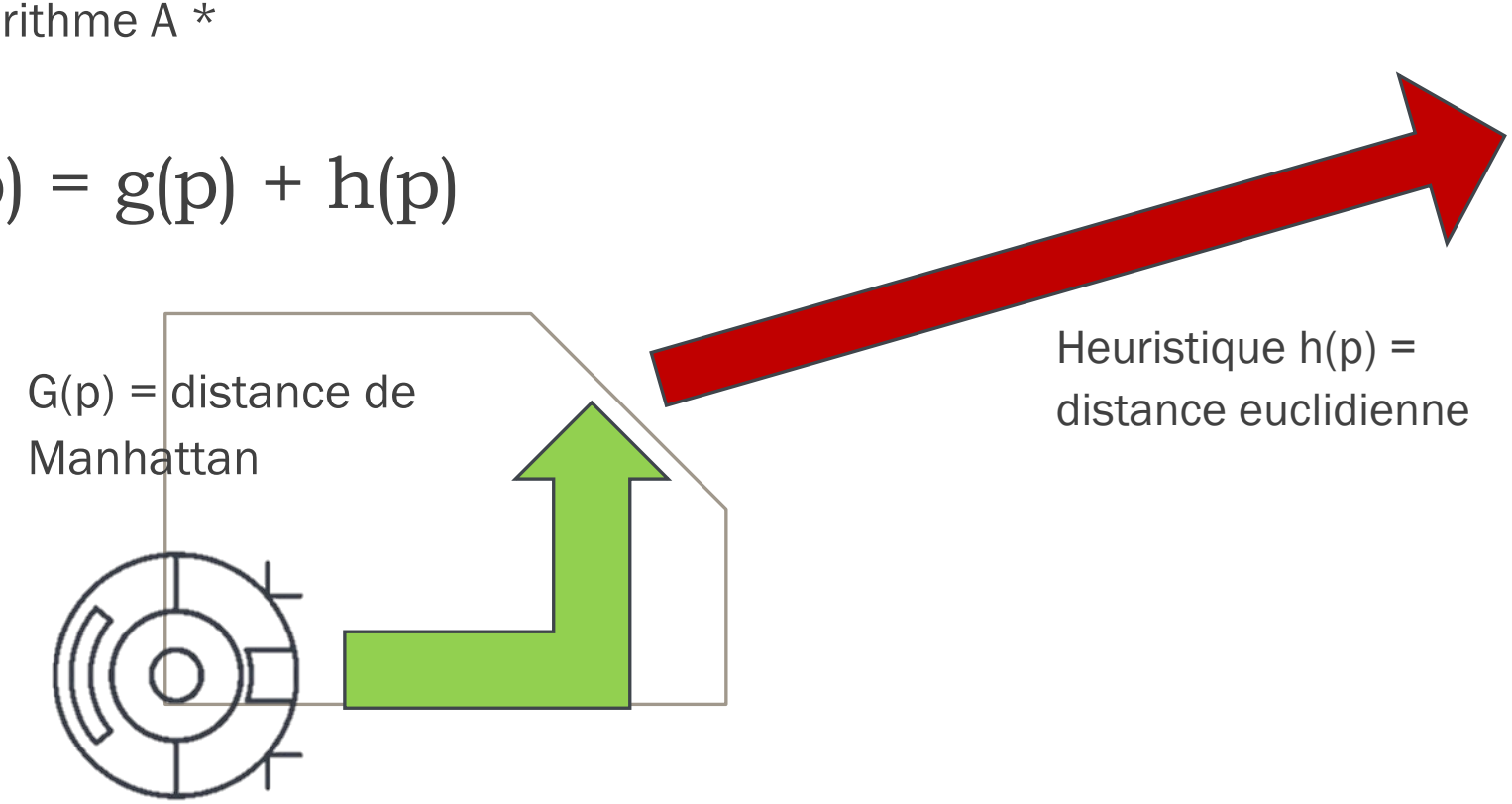
Résultat de localisation



Planification de trajectoire

Algorithme A *

$$F(p) = g(p) + h(p)$$



Mon résultat :

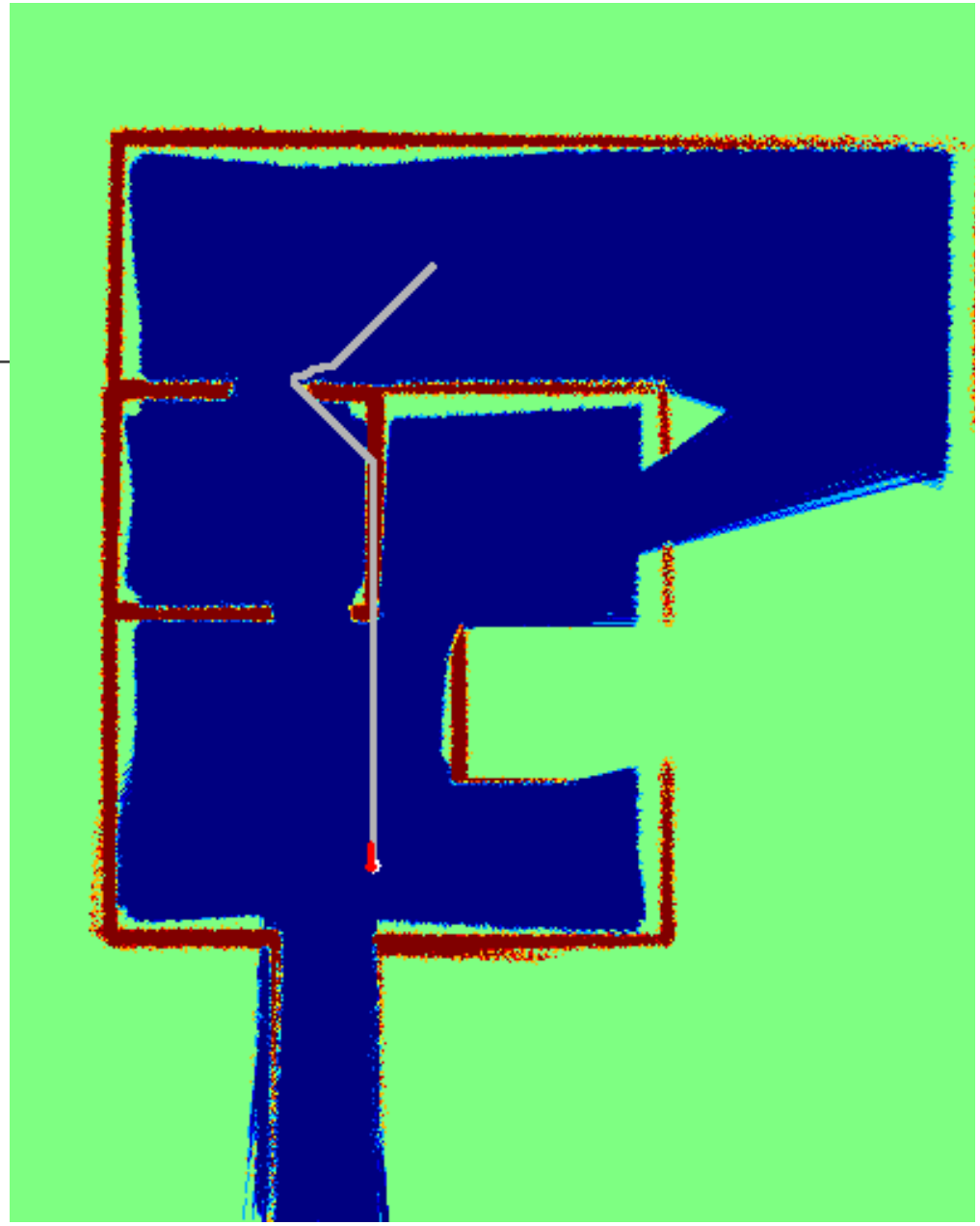
A* star efficace

Problèmes de pénétration
dans les murs trop fins

```
bad_environment =False
for a in range(-1,2,1):
    for b in range(-1,2,1):
        if self.grid.occupancy_map[observed_cell[0]+a][observed_cell[1]+b] >= 1.0:
            bad_environment =True

if not bad_environment:
    result.append(observed_cell)
```

Idée de solution : ne pas ajouter un voisin s'il est entouré de rouge



Conclusion

Découverte de la navigation robotique, cartographie, localisation

Implémentation d’algos de recherche opérationnelle (A*)

Travail sur le long terme sur un gros projet en python