

PROJET FITENTH



...

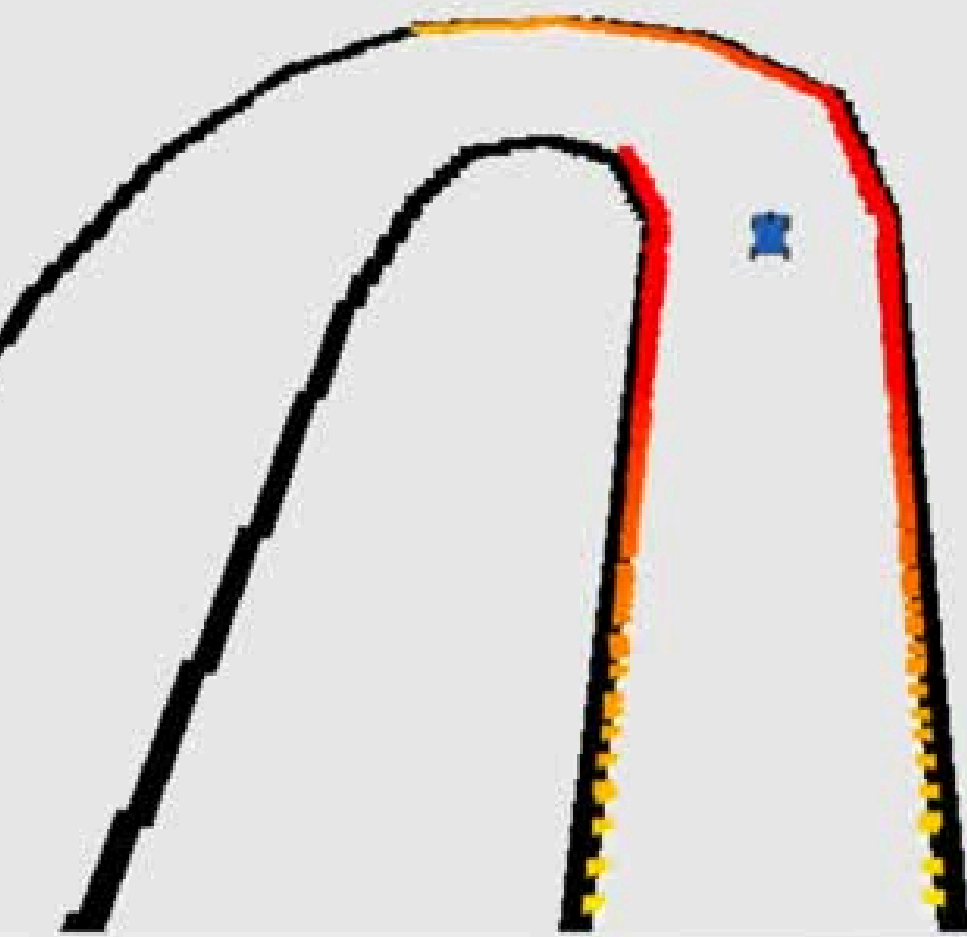
► ANTONIN LEROY



ÉCOLE
POLYTECHNIQUE



Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal



on: Off ROS Time: 191.15 ROS Elapsed: 156.76 Wall Time: 191.1

me/antonin/catkin_ws/src/f1tenth_simulator/launch/simulator.launch http://

catkin_ws/src/f1tenth_simulator/launch/simulator. antonin@antonin: ~/catkin_ws/s

e/scripts\$ python3 safety_nod ^Cantonin@antonin:~/catkin_ws

e/scripts\$ python3 safety_nod ^Cantonin@antonin:~/catkin_ws

e/scripts\$ python3 safety_nod ^Cantonin@antonin:~/catkin_ws

986.456845924]: Emergency brake turne

987.404098782]: Emergency brake turne

e/scripts\$ python3 safety_nod

EMERGENCY BRAKE

...

OBJECTIF DU NOEUD

L'objectif de ce projet est de développer un système de freinage d'urgence autonome pour la F1TENTH. Le système doit être capable d'intercepter les informations du lidar pour connaître les risques qui l'entourent et être capable, si il y en a de stopper le véhicule, indépendamment de l'action du conducteur ou de l'algorithme de pilotage. Le déclenchement repose sur le calcul du Time To Collision (TTC).

BOUCLE DES DISTANCES

```
for i in range(len(distances)):
    dist = distances[i]
```

- Boucler sur toutes les distances du LIDAR

```
alpha = angle_min + (i * incr)
vitesse_proj = self.speed * math.cos(alpha)
ttc = dist / vitesse_proj
```

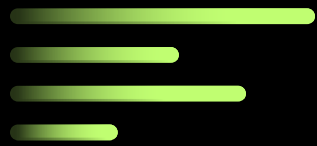
CALCUL DU TTC

- On projette pour connaître les vitesses en fonction de l'angle
- On calcule Time To Collision

```
if ttc < TTC_THRESHOLD:
    brake_msg = AckermannDriveStamped()
    brake_msg.drive.speed = 0.0
    bool_msg = Bool()
    bool_msg.data = True
```

TEST ET ENVOIE

- Test du seuil
- Création du message

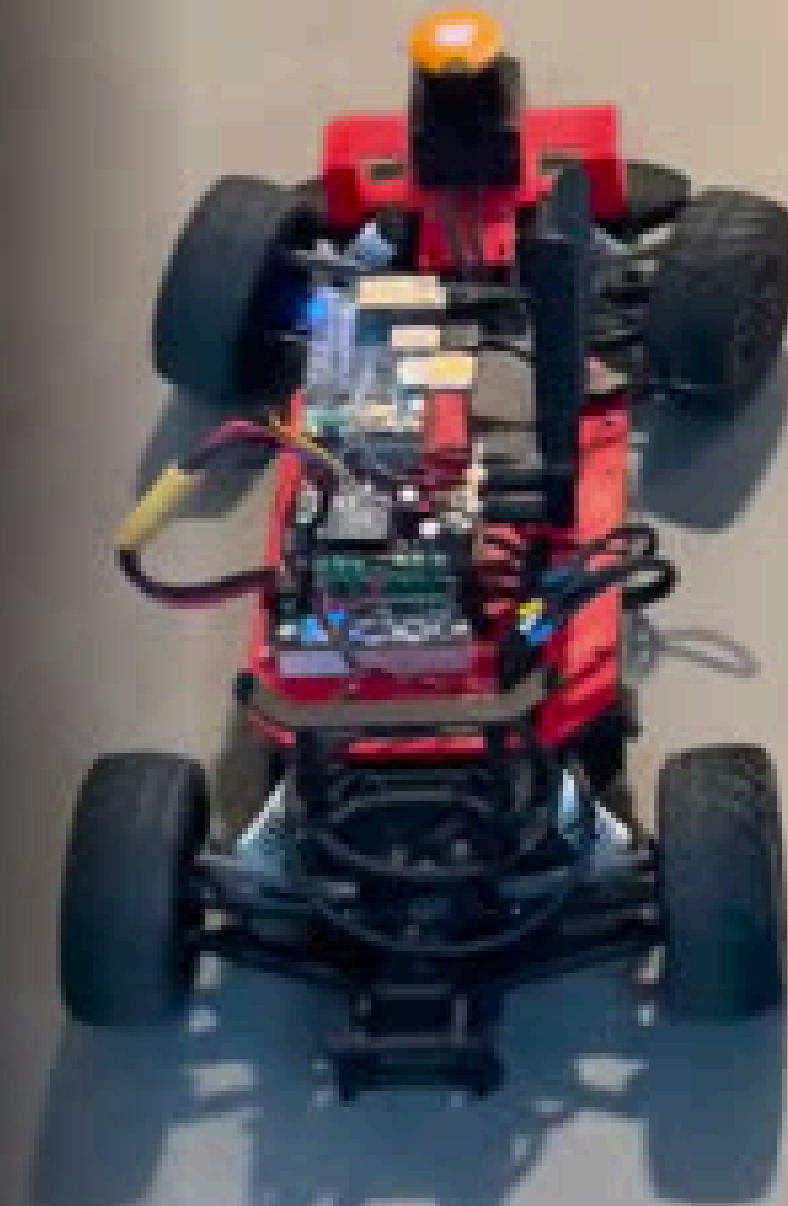


...

OBJECTIF DU NOEUD

L'objectif était de permettre à la voiture de naviguer de manière autonome en maintenant une distance fixe par rapport au mur de droite, tout en adaptant sa vitesse par rapport à la situation.

WALL FOLLOW



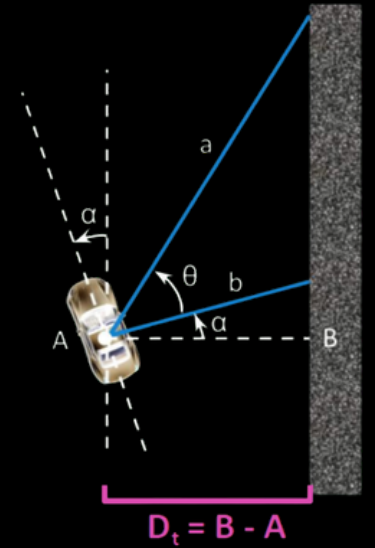
```

L = 1
theta = 42*(math.pi/180)
b=self.getRange(data,-90*(math.pi/180))
a=self.getRange(data,(-90*(math.pi/180))+theta)
alpha = math.atan((a*math.cos(theta)-b)/(a*math.s
distance=b * math.cos(alpha)
distancepredite= distance + L*math.sin(alpha)
error = DESIRED_DISTANCE_RIGHT - distancepredite

```

CALCUL DE L'ERREUR

- En utilisant 2 angles différents on calcule la distance avec le mur
- On la prédit, et on calcule l'erreur par rapport à ce qu'on souhaite



```

integral += error
derive = error-prev_error
angle = error * kp + ki * integral + kd * derive
prev_error=error

```

CONTROL PID

- On vient utiliser un contrôleur PID pour calculer l'angle optimal à donner à nos roues

```

if 0<abs(angle) <10*((math.pi)/180) :
    drive_msg.drive.speed = 1.5
elif 10*((math.pi)/180) <= abs(angle) <= 20*((math.pi)/180) :
    drive_msg.drive.speed = 1
else :
    drive_msg.drive.speed = 0.5
self.drive_pub.publish(drive_msg)

```

CHOIX DE LA VITESSE

- Choix intelligent de la vitesse en fonction de l'angle
- Plus l'angle est élevée plus la vitesse est faible

ARCHITECTURE



FOLLOW THE GAP

...

OBJECTIF DU NOEUD

L'objectif était de développer un comportement réactif robuste permettant au véhicule d'éviter les obstacles et de naviguer à vitesse modérée sans collision, en se basant sur les données LIDAR brutes.

```

car_width = 0.50
threshold = 0.30

diffs = np.diff(proc_ranges)
disparities = np.where(np.abs(diffs) > threshold)[0]

```

DISPARITIES

- Trouver les points où il y a un gap

```

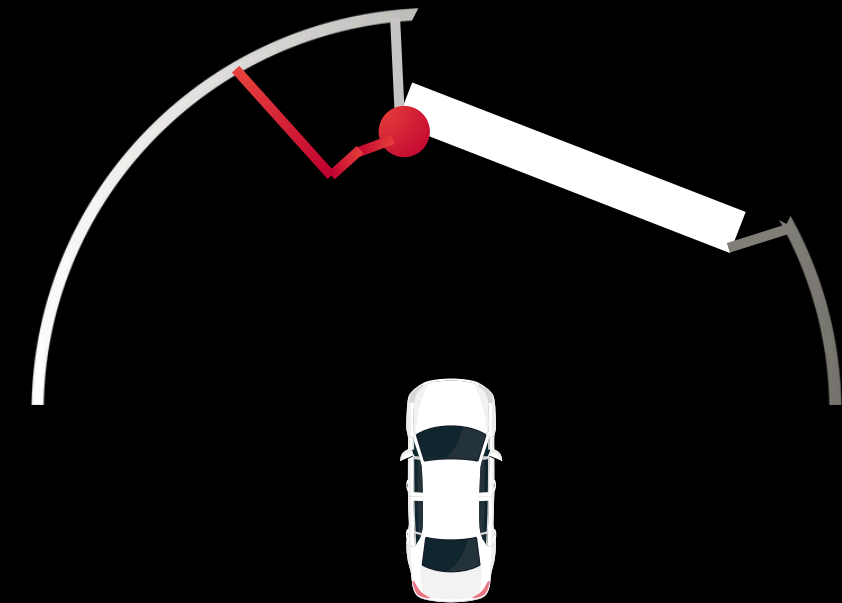
for i in disparities:
    # i est l'index du saut
    val_current = proc_ranges[i]
    val_next = proc_ranges[i+1]

    if val_current < val_next:
        closer_val = val_current
        # Le mur est à gauche du saut, on étend DROITE
        extend_direction = 1
        start_idx = i + 1
    else:
        closer_val = val_next
        # Le mur est à droite du saut, on étend GAUCHE
        extend_direction = -1
        start_idx = i

```

SENS DU GAP

- Trouver le sens de l'obstacle pour savoir dans quel sens le traiter



```

# Sécurité
if len(starts) == 0:
    return 0, len(free_space_ranges) - 1

# Trouver la séquence la plus longue
lengths = ends - starts
longest_idx = np.argmax(lengths)

```

MEILLEUR GAP

- --

ARCHITECTURE

```
num_indices_to_cover = int(angle_width / angle_increment)
```

```
for k in range(num_indices_to_cover):  
    idx_to_change = start_idx + (k * extend_direction)  
  
    if 0 <= idx_to_change < len(proc_ranges):  
        proc_ranges[idx_to_change] = min(proc_ranges[idx_to_change], closer_val)
```

SUPPRIMER LES VALEURS DANGEREUSES

- Utiliser la projection de la voiture pour savoir les directions “dangereuse”
- Choisir le min pour ne pouvoir que “rapprocher”

```
gap = ranges[start_i:end_i]
```

```
# 2. On trouve la profondeur maximale dans ce gap  
max_dist = np.max(gap)
```

```
# Cela crée un plateau de valeurs sûres  
safe_threshold = max_dist * 0.90
```

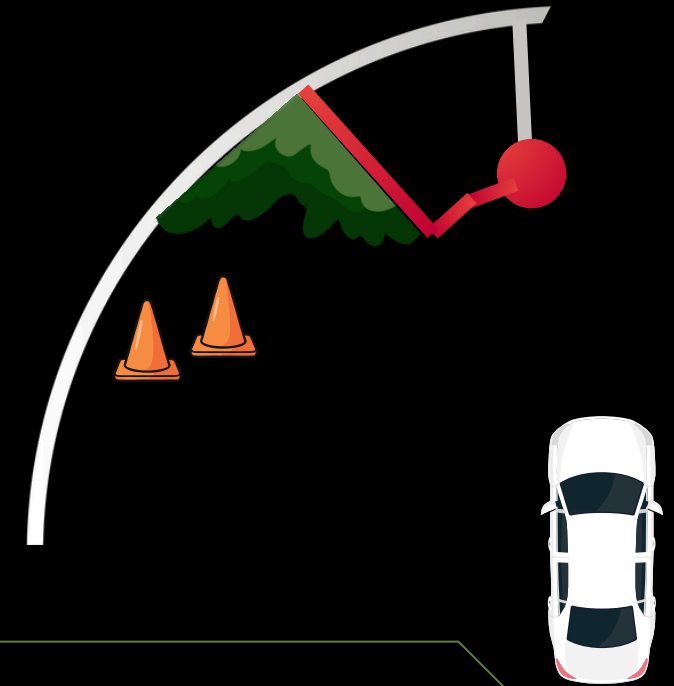
```
best_indices = np.where(gap >= safe_threshold)[0]
```

```
# 4. On vise le MILIEU
```

```
if len(best_indices) > 0:  
    avg_idx_relative = int(np.mean(best_indices))  
else:  
    avg_idx_relative = int(len(gap) / 2)  
return start_i + avg_idx_relative
```

TROUVER OU ALLER DANS LE GAP

- PLATEAU DE VALEUR SUR
- LE MILIEU DE CES VALEURS



```
target_steering_angle = data.angle_min + (real_best_idx * angle_inc)
```

```
alpha = 0.3
```

```
smoothed_angle = (alpha * target_steering_angle) + ((1.0 - alpha) * self.prev_steering_angle)
```

```
self.prev_steering_angle = smoothed_angle
```

ANGLE ROUES

- On vient prendre en compte l'ancien angle pour “lisser” a sortie, evitent des zigzags lors d'incertitude

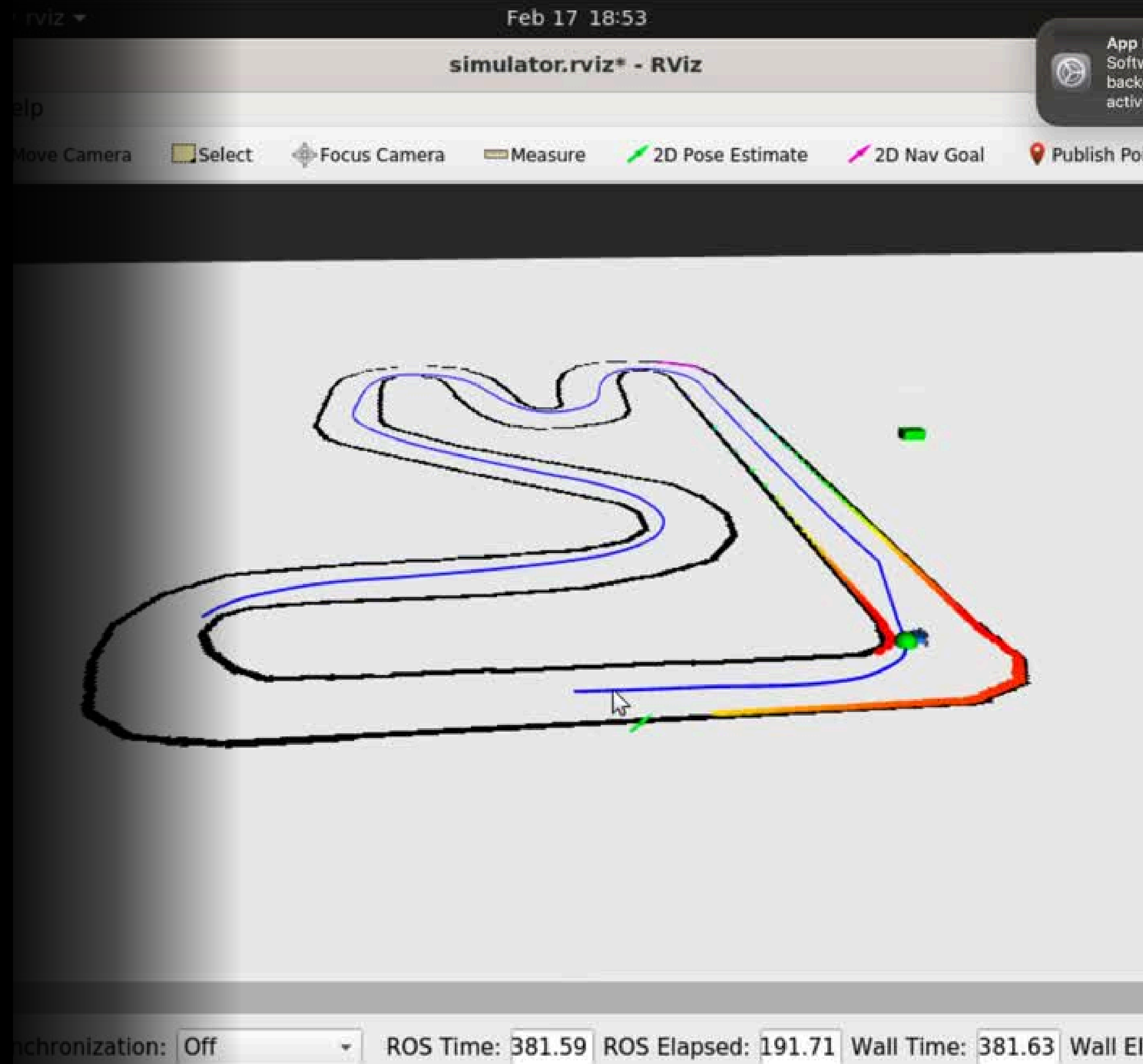
ARCHITECTURE

...

OBJECTIF DU NOEUD

L'objectif était de développer un contrôleur de type Pure Pursuit capable de suivre une trajectoire issu d'un csv. Si la théorie du Pure Pursuit est simple ($\gamma = 2y/L^2$), son implémentation pratique dans un environnement ROS dynamique (avec bruit de capteurs, latence et réinitialisations manuelles) à présenté de nombreuses difficultés.

PURE PURSUIT



```
# TRANSFORMATION LOCALE
indices = [(closest_index + i) % len(self.waypoints) for i in range(100)]
local_chunk = self.waypoints[indices]

# X_local = Devant, Y_local = Gauche
dx = local_chunk[:, 0] - x_car
dy = local_chunk[:, 1] - y_car

x_local = dx * math.cos(-yaw) - dy * math.sin(-yaw)
y_local = dx * math.sin(-yaw) + dy * math.cos(-yaw)
```

CHANGEMENT DE REPERE

- Choisir les indices proches de la voiture
- les mettre dans le repere de celle ci

```
lx < 0:
    continue

# Si le point est devant et qu'il dépasse la distance L
if dist >= self.L:
    target_x = lx
    target_y = ly
    target_found = True
```

CHOIX DU POINT

- Prendre le point devant la voiture qui est juste apres L

```
gamma = (2 * target_y) / (self.L**2)
steer = math.atan(gamma * self.wheelbase)
steer = np.clip(steer, -self.max_steer, self.max_steer)
```

ANGLE DES ROUES

- On utilise la formule vu dans les slides pour sortir un angle en fonction de la cible et du L choisie

≡ MERCI ≡