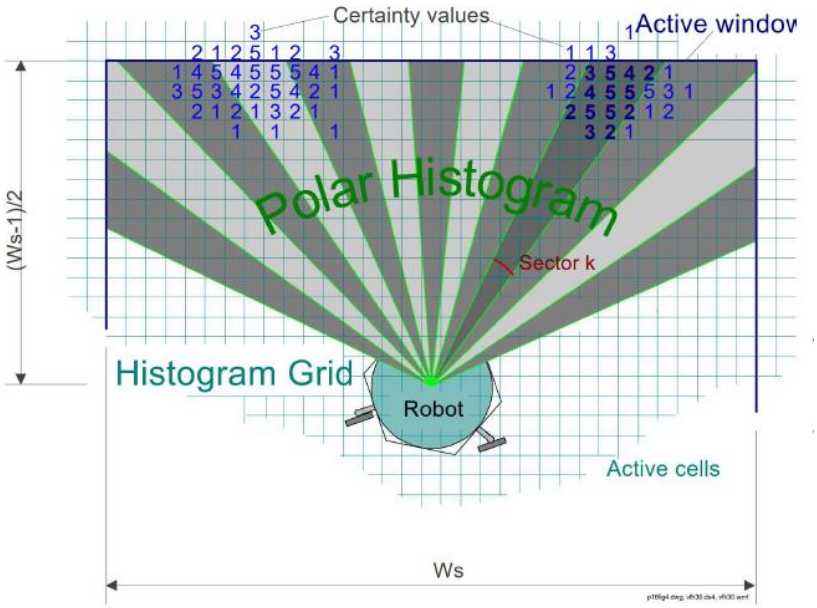
اصول علم ربات – پروژه نهایی

متین جعفرآقایی – ۹۶۳۱۹۰۳

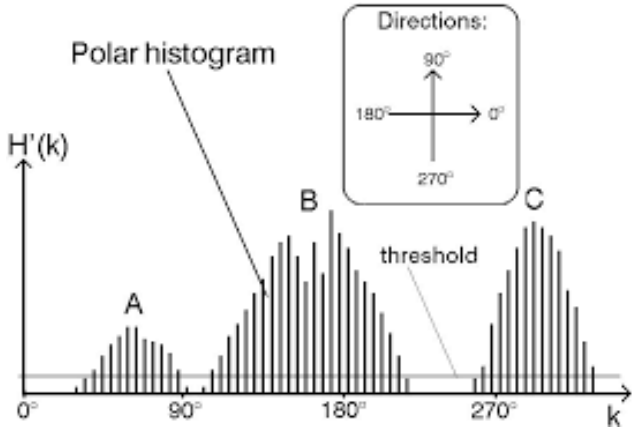
علی نظری - ۹۶۳۱۰۷۵

شرح پیاده‌سازی

در این پروژه تنها توانستیم از سنسور laser استفاده کنیم و متاسفانه سنسور bumber به دلیل اختلاف نسخه دستور پروژه قابل استفاده نبود. با استفاده سنسور laser و با روش VFH یک نقشه از محیط بدست می‌آوریم. به این صورت که با هر ورودی سنسور، مقدار خانه متناظر در نقشه یکی زیاد می‌شود:



سپس برای اینکه ربات مسیر درست را تشخیص دهد باید یک هیستوگرام قطبی از زوایای مختلف تشکیل دهیم:



در این مرحله sectorهایی که از threshold مقدار کمتری دارند یک دره تشکیل می‌دهند. در مرحله آخر زاویه میانی دره‌ها را بدست آورده و آن را به یه cost function می‌دهیم:

- ***Apply cost function G to each opening***

  $$G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_direction}$$

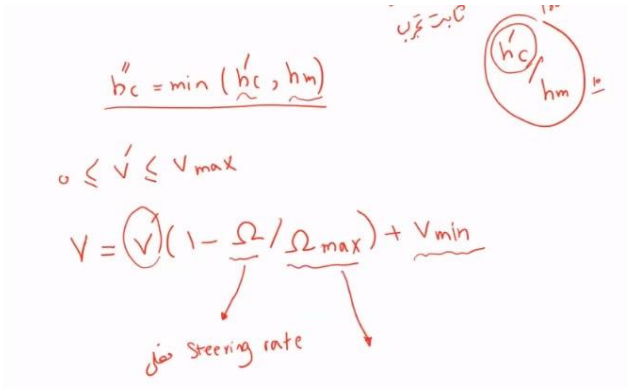
  where:
  - *target_direction = alignment of robot path with goal*
  - *wheel_orientation = difference between new direction and current wheel orientation*
  - *previous_direction = difference between previously selected direction and new direction*

زاویه‌ای که کمتری cost را داشته باشد به عنوان زاویه نهایی انتخاب می‌شود.

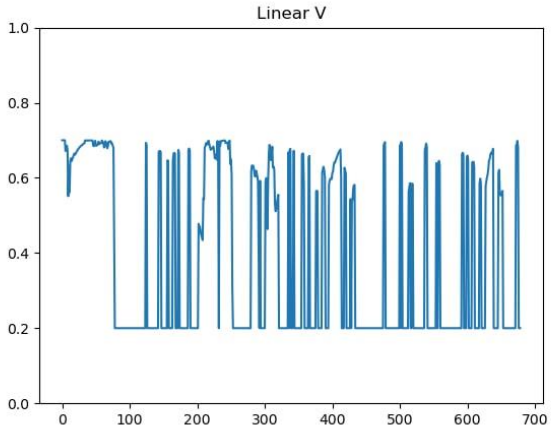
به این ترتیب ربات با استفاده از یک obstacle map محل کنونی خود و موانع را تشخیص داده و بر اساس آن تصمیم‌گیری می‌کند.

تحویلات پروژه

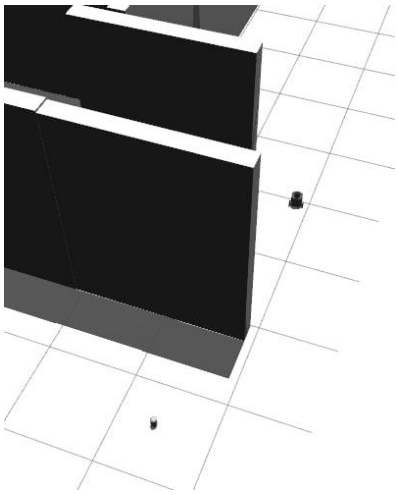
با استفاده از فرمول زیر سرعت ربات محاسبه شد:



نمودار سرعت را در اینجا مشاهده می‌کنید:



لحظه‌ای که مجددا به محل نخست خود می‌رسد:



بحث

– count how many times the robot is passing near the can of coke (i.e., passing by the origin);

۵ بار از کنار قوطی نوشابه رد شد.

– count how many times the bumpers get triggered;

متاسفانه در ROS Neotic امکان استفاده از سنسور bumber وجود نداشت. همچنین این موضوع در تالار مطرح شد اما پاسخی نگرفتیم.

– compute an estimate of your average speed;

میانگین سرعت ربات حدود ۰.۴ بود.

– Optional: can you compute a measure of smoothness of your motion?

به دلیل استفاده از obstacle map و الگوریتم VFH، ربات می‌توانست به صورت یکنواخت زاویه خود را تنظیم کند. همچنین با استفاده از فرمول سرعت ربات که در کلاس تدریس شده بود، توانستیم سرعت ربات را نیز یکنواخت تغییر دهیم.

– Was the performance satisfactory? Justify and discuss the answer.

با توجه به اینکه سنسور بامپر در دسترس نبود، عملکرد نسبتا مطلوبی داشتیم و می‌توانستیم با استفاده از سنسور laser نواحی خالی را تشخیص داده و به سمت آن‌ها حرکت کنیم. با این حال در مواردی به دلیل خطای سنسورها و odometry ربات به دیوار برخورد می‌کرد و بدون وجود سنسور bumper امکان اصلاح خطا وجود نداشت.

– What are the issues (if any) of your controller?

همانطور که گفته شد به دلیل عدم وجود سنسور بامپر در بعضی مواقع ربات با دیوار بخورد می‌کرد.

– Identify precise ways to proceed in order to improve the navigation performance of the reactive controller.

اگر از سنسورهای متنوعی برای هدایت ربات استفاده شود، آنگاه احتمال خطای ربات کاهش می‌یابد.

بخش امتیازی

همه‌ی ویژگی‌های مورد نظر بخش امتیازی پیاده‌سازی شده است.

ورودی گرفتن از سنسور:

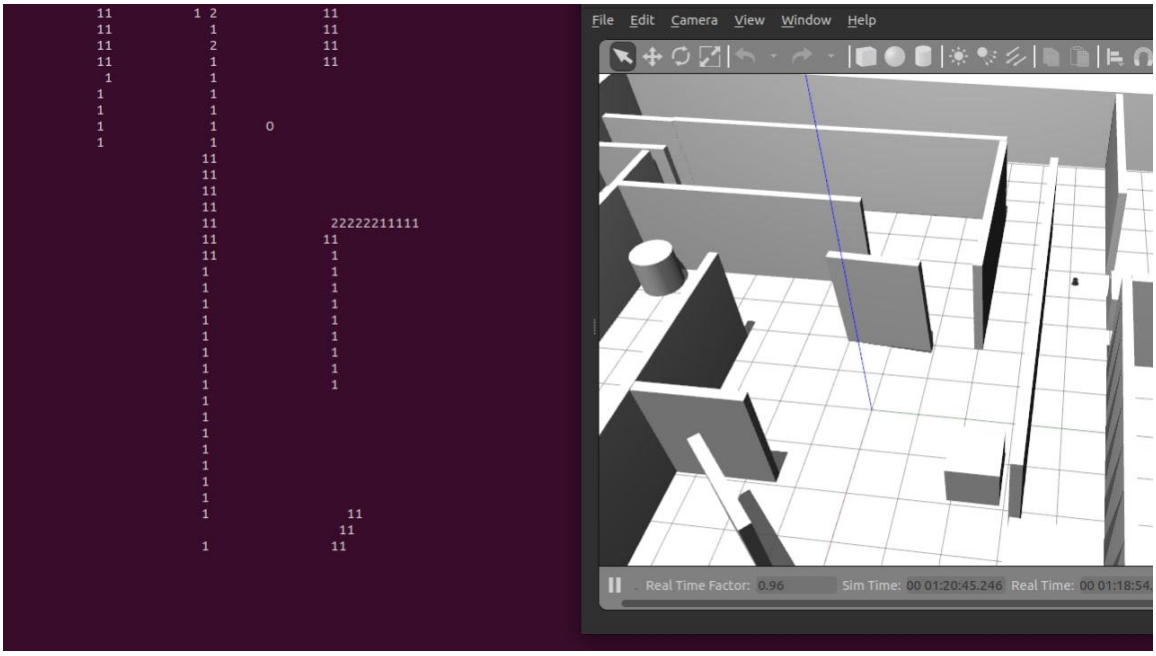
```
def callback_laser(self, msg):
  if not self.current:
    return

  robot_x, robot_y, robot_yaw = self.current

  range_len = len(msg.ranges)
  for i in range(range_len):
    if msg.ranges[i] == math.inf:
      continue
    angle = robot_yaw + msg.angle_min + i * msg.angle_max / range_len
    x, y = self.scale_to_map(robot_x + np.cos(angle) * msg.ranges[i],
                             robot_y + np.sin(angle) * msg.ranges[i])
    self.obstacle_map[x, y] += msg.ranges[i]
```

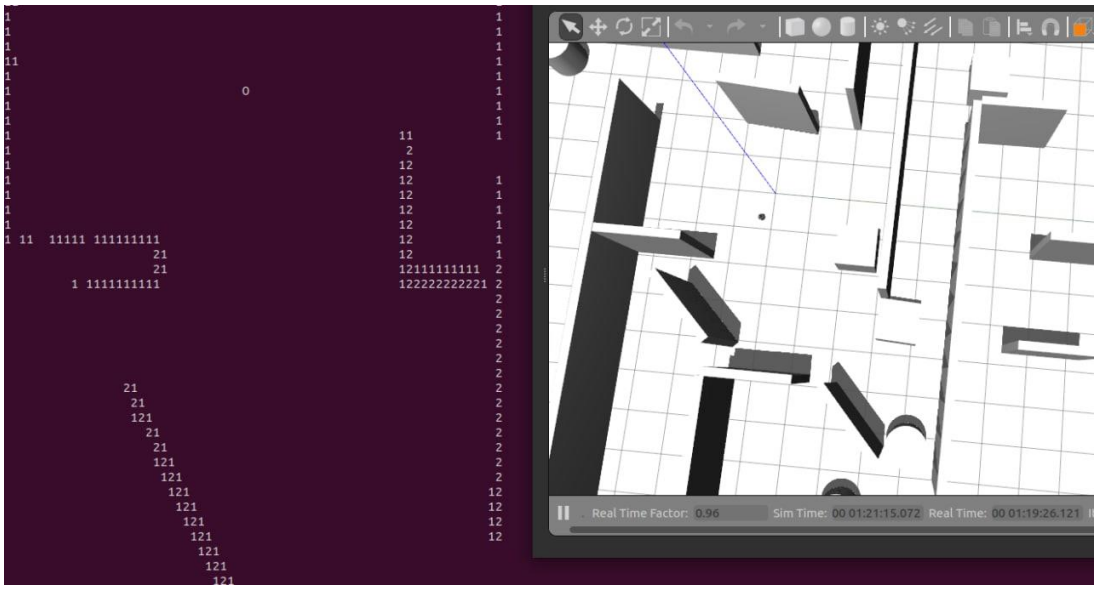
localization:

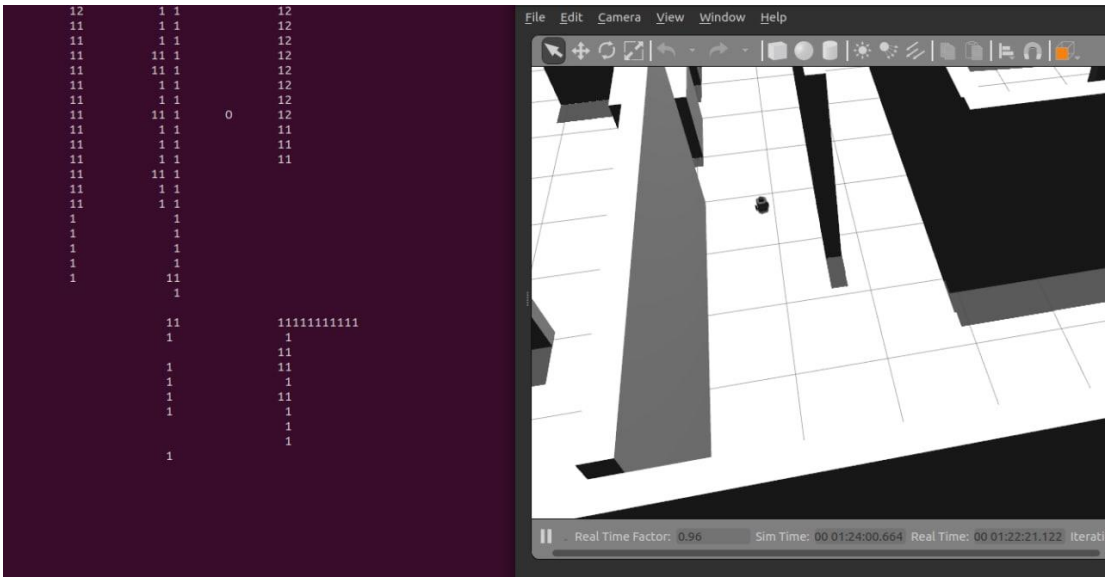
```
def callback_odometry(self, msg):
  quaternion = (msg.pose.pose.orientation.x, msg.pose.pose.orientation.y,
                msg.pose.pose.orientation.z, msg.pose.pose.orientation.w)
  (roll, pitch, yaw) = tf.transformations.euler_from_quaternion(quaternion)
  self.current = (msg.pose.pose.position.x, msg.pose.pose.position.y, yaw)
```

تولید نقشه:

```
range_len = len(msg.ranges)
for i in range(range_len):
  if msg.ranges[i] == math.inf:
    continue
  angle = robot_yaw + msg.angle_min + i * msg.angle_max / range_len
  x, y = self.scale_to_map(robot_x + np.cos(angle) * msg.ranges[i],
                           robot_y + np.sin(angle) * msg.ranges[i])
  self.obstacle_map[x, y] += msg.ranges[i]
```


File Edit Camera View Window Help
Real Time Factor: 0.96
Sim Time: 00 01:20:45.246
Real Time: 00 01:18:54


Real Time Factor: 0.96
Sim Time: 00 01:21:15.072
Real Time: 00 01:19:26.121



:Planning

```
heading_coef = 1
previous_coef = 1
path_coef = 1
valley_threshold = 65000
min_angle = 2
cell_size = 0.1
map_size = round((20 + 5) / cell_size)
number_of_sectors = 72
alpha = (2 * np.pi) / number_of_sectors
ws = 7 / cell_size
a = ((ws - 1) / 2) * math.sqrt(2)
b = 1
d_star = 0.2
path = [[-6, 4], [5, 4], [-3, -3], [9, -7], [-8, -9]]
```

:Action

```
def callback_target(self, msg):
    target_yaw = msg.data[0]
    h_prime = msg.data[1]
    h_zegond = min(h_prime, h_m)
    v_prime = v_max * (1 - h_zegond / h_m)
    yaw = self.current[2]
    yaw_diff = target_yaw - yaw
    while not -np.pi <= yaw_diff <= np.pi:
        if yaw_diff > np.pi:
            yaw_diff -= 2 * np.pi
        if yaw_diff < -np.pi:
            yaw_diff += 2 * np.pi
    self.vel.angular.z = k_yaw * yaw_diff
    self.vel.linear.x = v_prime * (1 - abs(yaw_diff) / np.pi) + v_min
    self.linear_v_history.append(self.vel.linear.x)
```