دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

گزارش پروژه نهایی درس اصول علم ربات

دانشجویان :
امیر حسین عسکری ۹۷۳۱۰۴۳
محمد مهدی مرادی ۹۷۳۱۰۶۰

استاد :
دکتر جوانمردی

تیر ۱۴۰۰

بخش اول: در بخش اول راجب سیر مراحل توضیح داده می‌شود.

برای obstacle detection از سنسور laser scan استفاده می‌کنیم. (در بخش دوم که توضیح کد هست این قسمت‌ها کامل آورده می‌شود)

بعد از subscribe کردن scan و دریافت msg های LaserScan به obstacle avoidance می‌پردازیم.

برای این قسمت دو راه معروف وجود دارد. (VFF,VFH) البته برای قسمت real-time تا حدودی پیاده سازی شدند و به صورت کامل در قسمت امتیازی بعد از ایجاد map بهتر عمل کردند.

در کد ارسال شده هر دو راه پیاده سازی شده‌اند. که هر دو به صورت real-time داده‌ی به دست آمده از لیزر اسکن را تحلیل کرده و و بر اساس آن مسیر و زاویه حرکت را تصمیم میگیریم.

چالش‌های بخش اول:

- در turtlebot2 زاویه سنسور ۱۲۰ درجه می‌باشد و در turtlebot3 زاویه سنسور ۳۶۰ درجه در نتیجه ما ابتدا که با turtlebot2 ران کرده بودیم خطای زیادی داشت و با وجود اینکه به مانعی برخورد نمی‌کرد bumper triggers = 0 بود ولی در loop می‌افتاد.
- در turtlebot3 سنسور bumper وجود نداشت به همین دلیل تعدادهای گزارش شده از سنسور bumper مربوط به turtlebot2 بوده است.
- در پیاده سازی VFH بهترین مقدار threshold بسیار سخت بدست آمده بود.

بخش دوم: توضیحی بر کد و مراحل اجرا در گزبو است.

ابتدا world داده شده را در مسیر launch فایل gazebo قرار داده و دستور launch مربوطه را برای بالا آمدن آن زدیم.

```
amir@funAut:~/Desktop/catkin_ws$ export TURTLEBOT3_MODEL=burger
amir@funAut:~/Desktop/catkin_ws$ roslaunch turtlebot3_gazebo funky-maze.launch
```

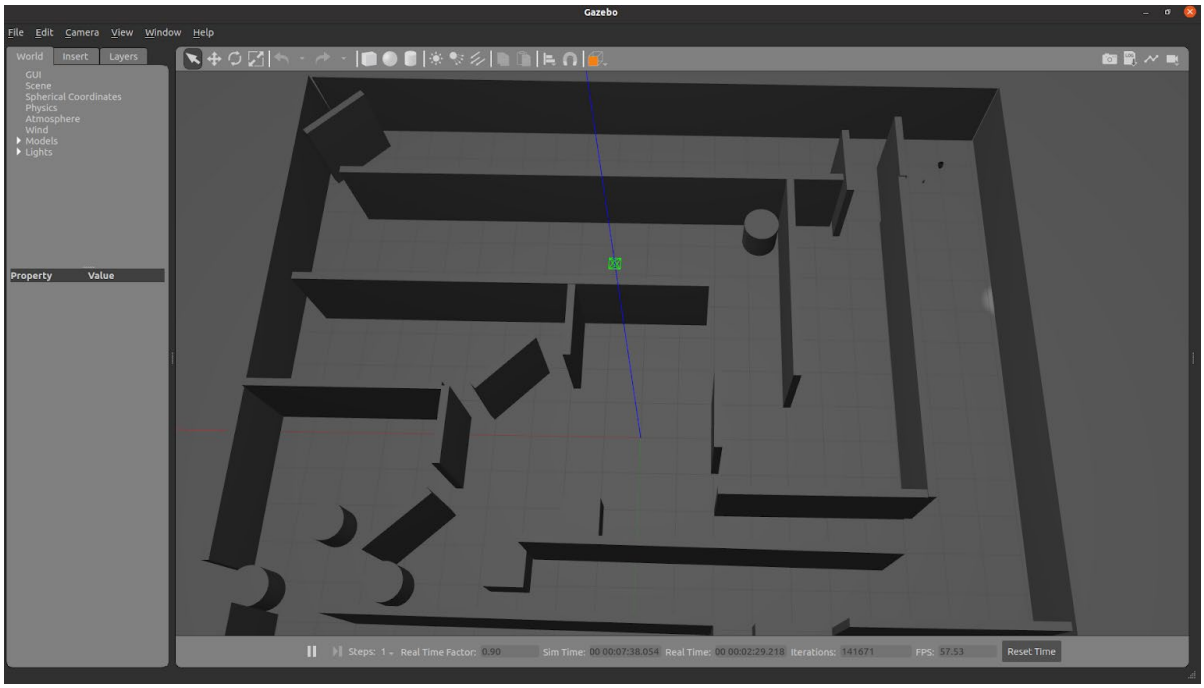
که محیط گزبو کاملا بالا آمد.

یکی از چالش‌های همین قسمت بالا آوردن محیط بود چرا که به درستی ذکر نشده بود که کجا اضافه شود و فایل لانچ باید براش ساخته می‌شد که بعد از کلی سرچ موفق شدیم.

برای اینکه ربات در همان نقطه‌ای که در دستورکار پروژه بود قرار بگیرد نیاز بود در فایل لانچ x_pos , y_pos به صورت زیر قرار بگیرند.

x_pos= -9

y_pos= -8



توضیح کدها:

دو نود داریم یکی برای obstacle avoid و دیگری برای حرکت ربات.

از آنجایی که کد حرکت ربات مانند کدهای قدیمی move_robot در تمرین یک و دو میباشد توضیح داده نمی‌شود و به کد obstacle_avoidance.py می‌پردازیم.

```
14  class ObstacleAvoiance:
15    def __init__(self):
16      rospy.init_node('reading_laser', anonymous=True)
17      self.vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
18      self.scan_sub= rospy.Subscriber("/scan", LaserScan , self.callback_scan)
19      self.odom_sub = rospy.Subscriber('/odom', Odometry, self.callback_odometry)
```

در این قسمت از laserScan و odom استفاده کردیم.

```
46      def callback_scan(self, msg):
47          self.ranges = list(msg.ranges)
48          self.angle_min = msg.angle_min
49          self.angle_max = msg.angle_max
50          self.angle_increment = msg.angle_increment
51          self.time_increment = msg.time_increment
52          self.scan_time = msg.scan_time
53          self.range_min = msg.range_min
54          self.range_max = msg.range_max
```

در این قسمت msg که از laserScan گرفتیم را به متغیرهای کلاس نسبت دادیم.

```python
def smoothing(self, ranges):
  window_size = 5
  size = len(ranges) - 1

  for i in range(0, len(ranges)):
      if i == 0 or i == 1:
          ranges[i] = (2*ranges[i] + 6*ranges[i + 1] + 2*ranges[i + 2]) / 10
      elif i == 2:
          ranges[i] = (2*ranges[i - 1] + 6*ranges[i] + 1*ranges[i + 1] + 1*ranges[i + 2]) / 10
      elif i == size - 2:
          ranges[i] = (1*ranges[i - 2] + 1*ranges[i - 1] + 6*ranges[i] + 2*ranges[i + 1]) / 10
      elif i == size - 1 or i == size:
          ranges[i] = (1*ranges[i - 2] + 3*ranges[i - 1] + 6*ranges[i]) / 10

      else:
          ranges[i] = (1*ranges[i-2] + 3*ranges[i-1] + 6*ranges[i] + 3*ranges[i+1] + 1*ranges[i+2]) / 14

  return ranges
```

قسمت smoothing VFH که بر اساس فرمول ارائه شده در کلاس است. (اندازه پنجره ۵ در نظر گرفته شده است.

قسمت اصلی کد:

در قسمت زیر VFH پیاده سازی شده است که ۳۶۰ سکتور در نظر گرفتیم به توجه به واقعی بودن محیط عملکرد بهتری داشته.

Threshold = 3

که اگر دره پهن پیدا کنیم (gap) به همان سمت می‌رویم.

```python
def main_VFH(self):
    rate = rospy.Rate(10) # 10hz
    ang_K_p = 1.8
    threshold = 3

    while not rospy.is_shutdown():
      twist = Twist()

      if not self.ranges or not self.x or not self.y:
        continue

      # ranges = self.smoothing(copy.copy(self.ranges))
      slice_indices, filtered_data = self.make_slice_indices(self.pair_making(self.ranges), threshold)
      gaps = self.slicing_gaps(slice_indices, filtered_data)


      final_list = []
      for gap in gaps:
        weights = [np.interp(gap[i][1], [threshold, self.range_max], [1, 10]) for i in range(len(gap))]
        avg_angle = sum([gap[i][0] * weights[i] for i in range(len(gap))]) / sum(weights)

        final_list.append({
            "gap_list": gap,
            "avg_angle": avg_angle,
            "diff": avg_angle if avg_angle <= 180 else -(360 - avg_angle),
        })


      width_limit = 15
      final_list.sort(key=lambda item: abs(item['diff']))

      widest_list = []
      for l in final_list:
        if abs(l['diff']) < 100 and len(l['gap_list']) >= width_limit:
          widest_list.append(copy.deepcopy(l))

      widest_list.sort(key=lambda item: len(item['gap_list']), reverse=True)
```

نمونه ای از عملکرد:

تا زمانی که مانعی وجود نداشته باشه theta مستقیم میره در صورت مشاهده‌ی مانع به بزرگترین دره پهن چپ یا راست تمایل پیاده می‌کند و بر اساس vhf وسط دره را می‌گیرد و مسیر خود را به آن سمت ادامه می‌دهد.

```
WIDEST
    avg: 309.1667442785901,  diff: -50.8332557214099 , len: 75
selected diff: -50.8332557214099

theta
THETA
    avg: 309.1667442785901,  diff: -50.8332557214099 , len: 75
    avg: 208.7743188486796,  diff: -151.2256811513204 , len: 72
WIDEST
    avg: 309.1667442785901,  diff: -50.8332557214099 , len: 75
selected diff: -50.8332557214099

theta
THETA
    avg: 2.8859036252961108,  diff: 2.8859036252961108 , len: 8
    avg: 321.5831557755222,  diff: -38.41684422447781 , len: 75
    avg: 225.66112412177986,  diff: -134.33887587822014 , len: 75
WIDEST
    avg: 321.5831557755222,  diff: -38.41684422447781 , len: 75
selected diff: 2.8859036252961108

theta
THETA
    avg: 2.8859036252961108,  diff: 2.8859036252961108 , len: 8
    avg: 321.5831557755222,  diff: -38.41684422447781 , len: 75
    avg: 225.66112412177986,  diff: -134.33887587822014 , len: 75
WIDEST
    avg: 321.5831557755222,  diff: -38.41684422447781 , len: 75
selected diff: 2.8859036252961108

theta
THETA
    avg: 351.8135663862044,  diff: -8.1864336137956 , len: 16
    avg: 9.07946867722768,  diff: 9.07946867722768 , len: 20
    avg: 313.5,  diff: -46.5 , len: 50
    avg: 233.00813548915266,  diff: -126.99186451084734 , len: 81
WIDEST
    avg: 313.5,  diff: -46.5 , len: 50
    avg: 9.07946867722768,  diff: 9.07946867722768 , len: 20
    avg: 351.8135663862044,  diff: -8.1864336137956 , len: 16
selected diff: -8.1864336137956
```

بخش سوم: گزارشی بر عملکرد پیاده سازی

سوال‌های قسمت اول =>

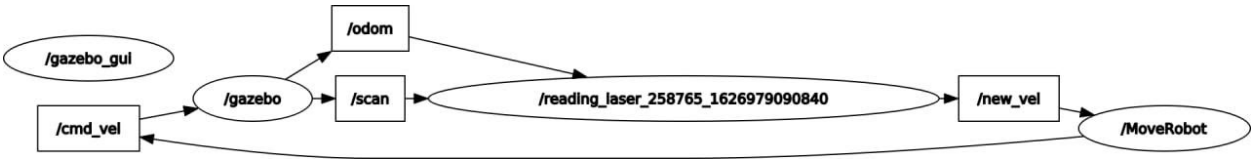
- در زمان ۱۵ دقیقه ربات burger توانست پنج بار از کنار coke رد شود. (با VHF)
- در زمان ۱۵ دقیقه ربات burger توانست یک بار از کنار coke رد شود. (با VFF)
- در زمان ۱۵ دقیقه برای turtlebot2 سنسور bumper به تعداد ۰ بار برخورد گزارش داده است. (turtlebot3 سنسور bumper را ندارد)
- سرعت اولیه ۰,۸ در نظر گرفته شده بود که برای میانگین گیری هر بار سرعت را در یک لیست می ریختیم و در آخر میانگین گرفتیم. که میانگین سرعت تقریبا ۰,۷۳ بوده است. (دلیلش هم مشخص است مقدار زمانی را صرف تغییر زاویه می‌کند برای جلوگیری از برخورد و در آن زمان‌ها سرعت برابر صفر می‌شود.)
- برای قسمت اختیاری نرم بودن حرکت اینگونه برداشت کردیم که هر چقدر مجموع تغییرات زاویه بیشتر باشد smoothing کمتری را داشتیم. (که در کد این قسمت پیاده سازی شده بود و بهترین threshold را بر این اساس ۱,۵ بدست آوردیم)

سوال‌های قسمت دوم =>

- با توجه به نتیجه‌های بدست آمده عملکرد خوب بود ولی درصورت داشتن planning و استفاده از deliberative control نتیجه بهتر می‌بود.
- Reactive controlبرای فضاهای باز خوب عمل می‌کند چرا که هدف فقط به جایی نخوردن است و همیشه هدف مهم ما این نیست.
- روش‌های زیادی وجود دارد مانند:
  - ترکیب deliberative و reactive
  - استفاده از PI feedback
  - Path planning
  - Map-based plan

و تمام کارهایی که نیاز بود در بخش امتیازی انجام دهیم.

رابطه نودها و تاپیک‌ها



بخش امتیازی:

برای قسمت امتیازی در کد فقط قسمت یک map در نظر گرفتیم و با استفاده از احتمال پر و خالی بودن و فرمول‌های بخش obstacle detection را ایجاد کردیم.

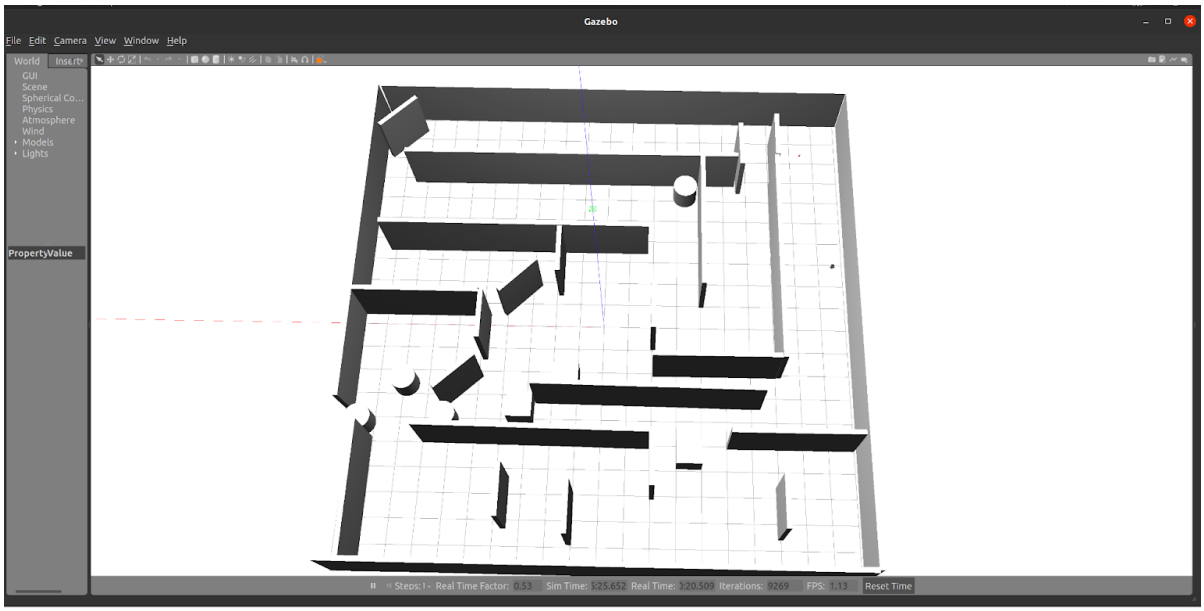
که در ادامه از vhf و اضافه کردن smoothing استفاده کردیم.

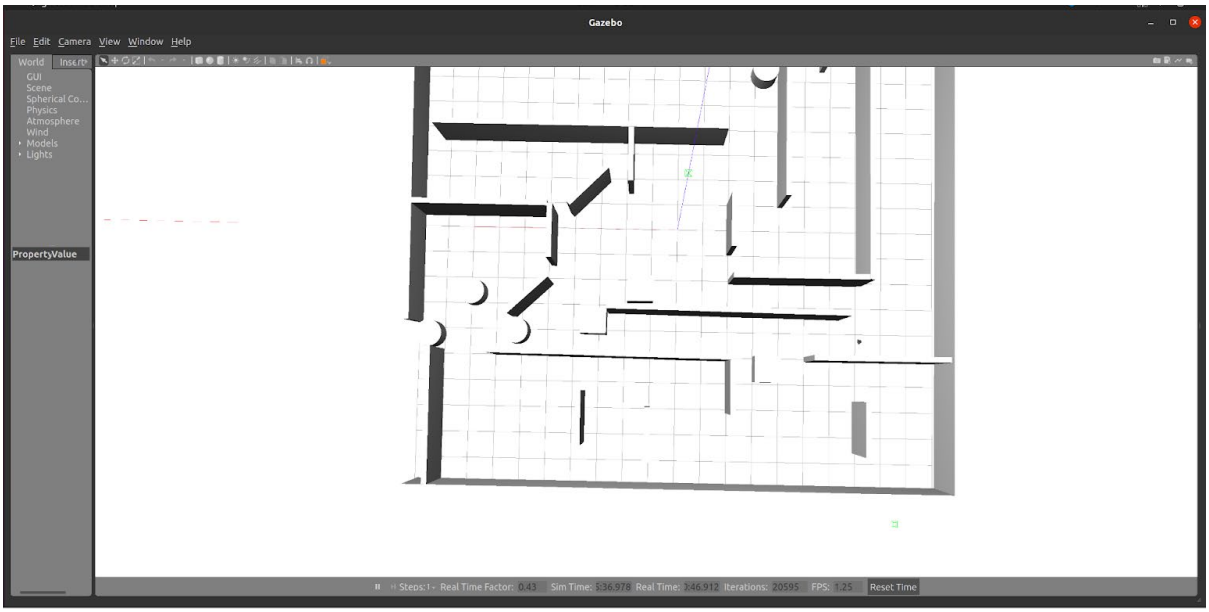
که در قسمت کد به صورت زیر اضافه شده است.

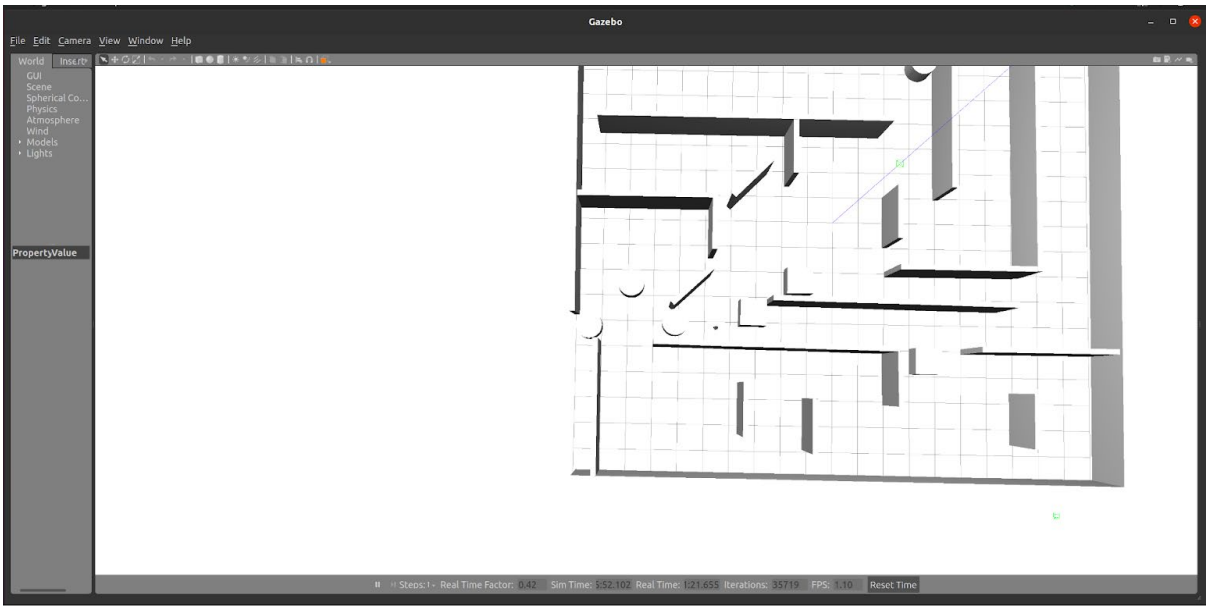
```python
def pair_making(self, ranges):
    pairs = []
    for i in range(len(ranges)):
        if ranges[i] == math.inf:
            ranges[i] = self.range_max
        pairs.append((i, round(ranges[i], 3)))
    return pairs
```

```python
def make_slice_indices(self, pairs, threshold, mode='gap'):
    if mode == 'gap':
        filtered_data = [x for x in pairs if x[1] >= threshold]
        slice_indices = []
        for index, pair in enumerate(filtered_data):
            try:
                angle = pair[0]
                if filtered_data[index + 1][0] != (angle + 1):
                    slice_indices.append(index + 1)
            except:
                pass

        return slice_indices, filtered_data
    else:
        filtered_data = [x for x in pairs if x[1] <= threshold]
        slice_indices = []
        for index, pair in enumerate(filtered_data):
            try:
                angle = pair[0]
                if filtered_data[index + 1][0] != (angle + 1):
                    slice_indices.append(index + 1)
            except:
                pass

        return slice_indices, filtered_data
```

و نتایج بسیار بهتر بود.
نمایی از حرکت ربات:


Gazebo
File Edit Camera View Window Help
World Insert
GUI
Scene
Spherical Co...
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.53 Sim Time: 5:25.652 Real Time: 9:20.505 Iterations: 9269 FPS: 1.13 Reset Time

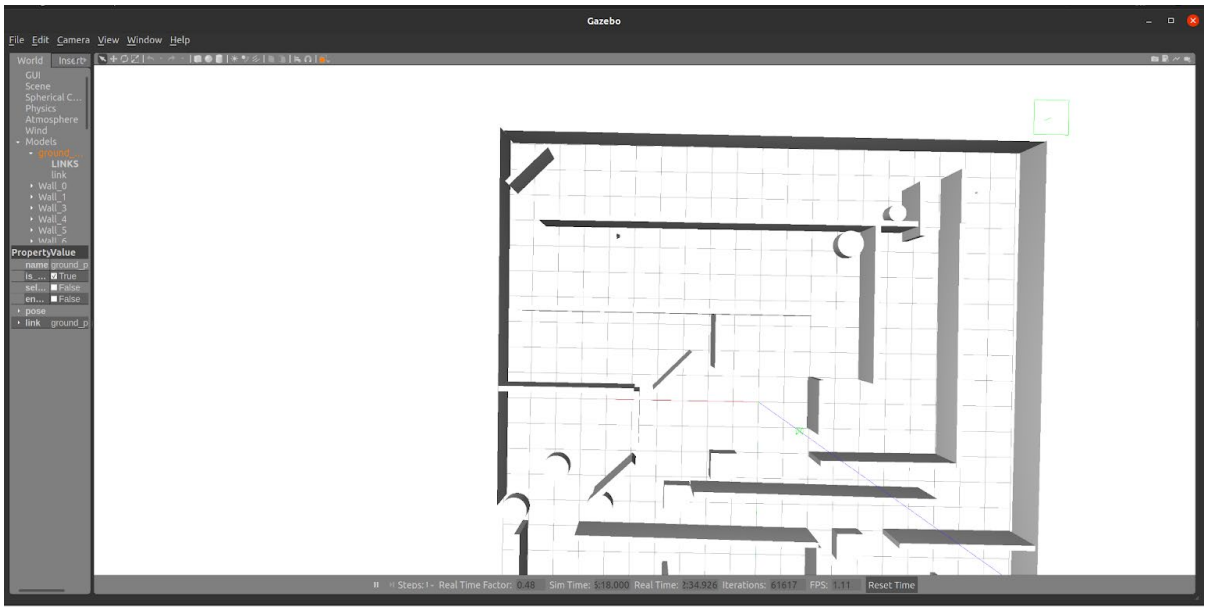

Gazebo
File Edit Camera View Window Help
World Insert
GUI
Scene
Spherical Co...
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.43 Sim Time: 5:36.978 Real Time: 7:46.912 Iterations: 20595 FPS: 1.25 Reset Time


Gazebo
File Edit Camera View Window Help
World Insert
GUI
Scene
Spherical Co...
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.42 Sim Time: 52.102 Real Time: 121.655 Iterations: 35719 FPS: 1.10 Reset Time


Gazebo
File Edit Camera View Window Help
World Insert
GUI
Scene
Spherical Co...
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.28 Sim Time: 55.687 Real Time: 132.971 Iterations: 39304 FPS: 0.82 Reset Time


Gazebo
File Edit Camera View Window Help
World
GUI
Scene
Physics
Atmosphere
Wind
Models
Lights
Property
Value
Reset Time


Gazebo
File Edit Camera View Window Help
World
GUI
Scene
Physics
Atmosphere
Wind
Models
Lights
Property
Value
Reset Time



با تشکر فراوان - امیدواریم کامل بوده باشد.