



Amirkabir University of Technology
(Tehran Polytechnic)



Department of
Computer Engineering

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش پروژه نهایی اصول علم رباتیک

دانشجویان :

محمدحسین بدیعی – 9531701

استاد : دکتر مهدی جوانمردی

نیم سال دوم 1399-1400

یک فیلم در کنار این فایل قرار داده‌ایم که دور زدنِ ربات را طبق مسیرِ بهینه‌ای که در صورتِ سوال ذکر شده است نشان می‌دهد. لطفاً به فایلِ فیلم برای مشاهده‌ی صحت عملکردِ ربات مراجعه بفرمایید. (لازم به ذکر است که در دورِ دوم بدلیلِ تغییراتی که در مواجهه با مانع مشاهده می‌کند در یک مسیر انحرافی می‌افتد ولی با توجه به الگوریتمِ ابداعی‌ای که پیاده نمودیم، خود را از آن منطقه نجات خواهد داد – همچنین مدتِ فیلم بدلیلِ حافظه‌ی کم گوشی 9 دقیقه است که صحتِ عملکردِ ربات در دور اول را به طور کامل و مشابه با مسیری که صورت سوال انتظار داشته است طی می‌کند.)

این ویدیو را نیز با کیفیتِ بالاتر در آپارات بارگذاری کرده‌ایم که می‌توانید به لینکِ زیر مراجعه بفرمایید:

<https://aparat.com/v/95W4b>

فورموله‌سازی مسأله

الگوریتم ابداعی که پیاده نمودیم، برگرفته از ایده‌ی موجود در الگوریتم VFH بود. در واقع ما سه محیط را در اطرافِ ربات که چپ و راست و روبرو می‌باشد را در نظر گرفتیم. محیطِ اطرافِ ربات شاملِ اسکنِ لیزر از 15- درجه تا 15+ درجه به عنوانِ front یا مقابل، و اسکنِ لیزر از 30 تا 60 درجه به عنوانِ right یا محیطِ راستِ ربات و در آخر نیز اسکنِ 300 تا 330 درجه به عنوانِ محیطِ سمتِ چپِ ربات در نظر گرفتیم. توجه داریم که لیزرِ ترتل‌بات 3 می‌تواند یک اسکنِ 360 درجه از محیط بگیرد ولی با توجه به پردازش‌هایی که نیاز بود و نیز دقتی که باید در نظر می‌گرفتیم، این سه محیط را به عنوانِ محیط‌های اسکن شده توسطِ لیزرِ ربات ذخیره کردیم.

حال برای بررسیِ وجودِ مانع یا عددِ وجودِ مانع در این سه محیط، مجموعاً با $2^3 = 8$ حالت مواجه هستیم. حال نحوه‌ی پیاده‌سازیِ safe بودن و effective بودنِ ربات را شرح می‌دهیم: برای safe بودن ما هشت حالت را در

نظر گرفتیم و ربات تنها در صورتی قادر به حرکت مستقیم است که در مقابل خود در محیط اسکن شده از 15- درجه تا 15+ درجه و با 0.7 threshold هیچگونه مانعی را مشاهده نکند. برای effective بودن نیز از الگوریتم VFH یا همان vector field histogram ایده گرفتیم و یک الگوریتم ابداعی را برای گشت گذار ربات به گونه‌ای که مدام بهترین مسیر را انتخاب کند طراحی کردیم.

نحوه‌ی کار این الگوریتم متأثر از مقدار دریافتی توسط اسکن لیزر است. در واقع ما با ایجاد یک رابطه‌ی خطی بین اختلاف مقدار اسکن شده از راست و چپ، زاویه را در مواقعی که مانعی در مقابل نباشد تنظیم نمودیم. این رابطه به صورت زیر است:

در مواقعی که مقادیر اسکن شده از سمت چپ و راست کمتر از threshold نباشد:

$$\text{angular_z} = (\text{regions}['\text{right}'] - \text{regions}['\text{left}'])/2.5$$

در مواقعی که مقادیر اسکن شده از سمت چپ و راست کمتر از threshold باشد:

$$\text{angular_z} = (\text{regions}['\text{right}'] - \text{regions}['\text{left}'])/2$$

روابط فوق کاملاً تجربی بدست آمده ولی نکته‌ای که حائز اهمیت است این است که اگر در سمتی موانع کمتری دیده شود، ربات به آن سمت میل می‌کند و این ایده‌ی اصلی مقداردهی به زاویه به صورت فوق بود.

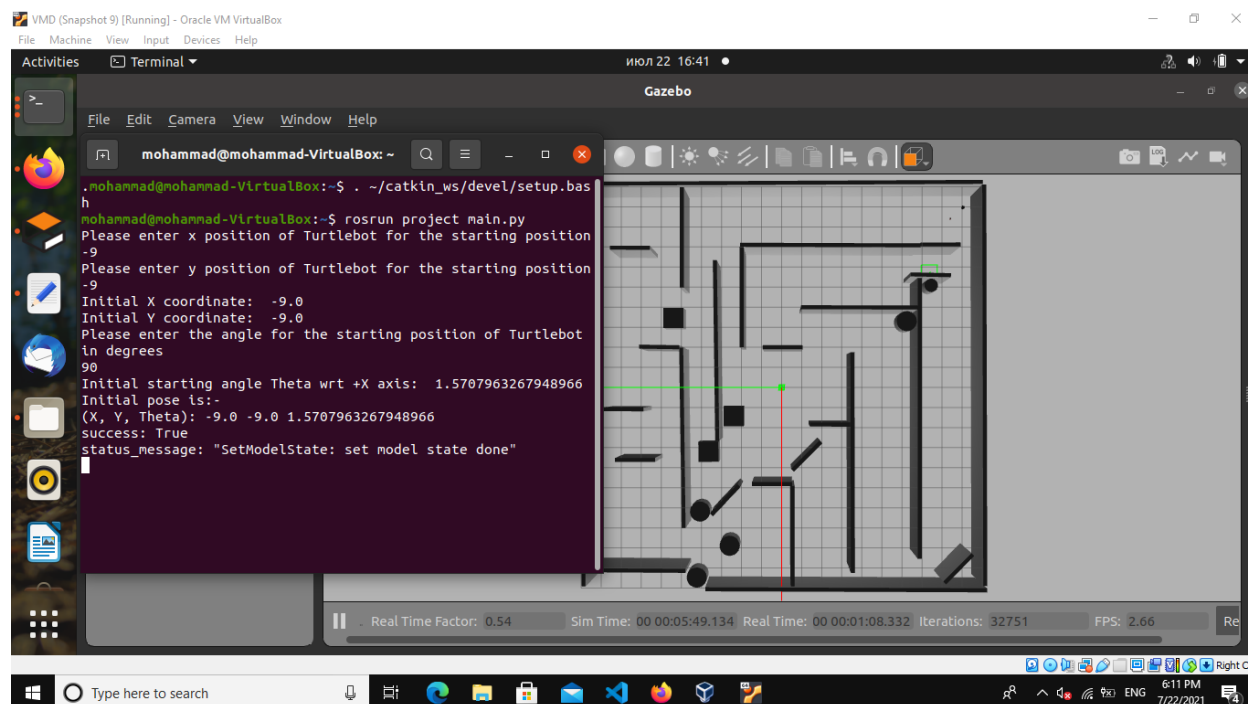
کارهای دیگری نیز در این مورد صورت گرفته است که روند ربات را smooth تر سازد و نیز به حرکت ربات سرعت ببخشد. از جمله‌ی این کارها حرکتی آهسته در مواقعی است که تنها در یک از طرفین چپ و راست مانعی وجود داشته باشد و کار دیگری که برای سرعت بخشیدن به ربات انجام شده، قرار دادن یک threshold جداگانه برای اختلاف مقدار اسکن شده‌ی موانع چپ و راست به جهت تنظیم سرعت زاویه چرخش فرمان است. در واقع با این threshold در صورتیکه اختلاف بدست آمده کمتر باشد، اخلاقی در حرکت ایجاد نمی‌کنیم و ربات با حداکثر سرعت ادامه می‌دهد ولی وقتی از حد آستانه عبود کرد، یعنی یک طرف موانع دارند افزایش می‌یابند (هر چند که

هنوز از حدِ آستانه‌ی مربوط به خود عبور نکرده باشند) و لذا یک پیش‌بینی نیز در اینجا انجام می‌دهیم که به حرکتِ ربات سرعت می‌بخشد. در ادامه تمامی این مراحل را بر روی کد توضیح می‌دهیم.

توضیح کد

برای اجرای کد پس از `make` نمودنِ پکیج و راه‌اندازیِ گزبو، کافی است که از `project`، فایلِ `main.py` را ران بفرمایید.

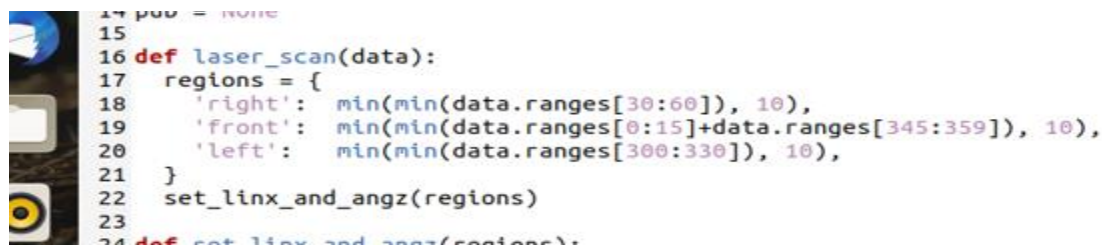
لازم به ذکر است که کدِ ما، نقطه‌ی شروع را از کاربر به عنوانِ ورودی دریافت می‌کند. این قابلیت را قرار دادیم که از هر کجای `map` که مایل باشید، ربات را به حرکت در آورید. لذا توجه داریم که نقطه‌ی شروعی که مسأله در نظر گرفته است، $(x, y, \theta) = (-9, -9, 90)$ می باشد. پس لذا در هنگامِ اجرای کد مطابقِ زیر، این نقطه را به عنوانِ نقطه‌ی شروع به سیستم بدهید. (دقت فرمایید که نقشه از ویوی بالا است)



حال به سراغ کد می‌رویم:

در این پکیج یک نود با نام `safe_efficient_obstacle_avoidance` را تعریف کردیم که دو تابع اصلی (جز `main`) دارد: تابع اول که در واقع `callback` مربوط به اسکن لیزر است و تابع دوم که به واسطه‌ی تابع اول فراخوانی می‌شود، مربوط به تعیین سرعت خطی و زاویه‌ای اعمالی به فرمان است.

همانطور که گفتیم، سه ناحیه را به صورت زیر از مقادیر اسکن شده‌ی لیزر استخراج می‌نماییم.



```
14 pub = None
15
16 def laser_scan(data):
17     regions = {
18         'right': min(min(data.ranges[30:60]), 10),
19         'front': min(min(data.ranges[0:15]+data.ranges[345:359]), 10),
20         'left': min(min(data.ranges[300:330]), 10),
21     }
22     set_linx_and_angz(regions)
23
24 def set_linx_and_angz(regions):
```

و سپس تابع تنظیم سرعت خطی و زاویه‌ای را مطابق فوق صدا می‌کنیم.

سپس مقادیر دیفالت سرعت‌ها و نیز حد آستانه‌ی مقدار خطر برای مانع را تعریف نمودیم.

```
--
24 def set_linx_and_angz(regions):
25     threshold_dist = 0.7
26     linear_speed = 0.5
27     angular_speed = 0.3
28
29     msg = Twist()
30     linear_x = 0
31     angular_z = 0
```

سپس در این تابع تنظیم سرعت خطی و زاویه‌ای به تنظیم سرعت‌ها مطابق با تقسیم بندی‌ای که در نتیجه‌ی

سه ناحیه بوجود می‌آید که در واقع همان هشت حالت است، کرده‌ایم:

ناحیه اول:

```
state = ''
if regions['front'] > threshold_dist and regions['left'] > threshold_dist and regions['right'] > threshold_dist:
    state = 'No Obstacle'
    linear_x = linear_speed
    angular_z = 0
if regions['left']+0.15 < regions['right']:
    angular_z = (regions['right'] - regions['left'])/2.5 # Increase this angular speed for avoiding obstacle faster
elif regions['right']+0.15 < regions['left']:
    angular_z = (regions['right'] - regions['left'])/2.5
```

ناحیه دوم:

```
42 elif regions['front'] < threshold_dist and regions['left'] < threshold_dist and regions['right'] < threshold_dist:
43     state = 'Obstacle in Front and Left and Right'
44     linear_x = -0.05#-linear_speed
45     angular_z = (regions['right'] - regions['left'])/2
```

ناحیه سوم:

```
46 elif regions['front'] < threshold_dist and regions['left'] > threshold_dist and regions['right'] > threshold_dist:
47     state = 'Obstacle in Front'
48     linear_x = 0
49     angular_z = angular_speed
```

ناحیه چهارم:

```
50 elif regions['front'] > threshold_dist and regions['left'] > threshold_dist and regions['right'] < threshold_dist:
51     state = 'Obstacle in Right'
52     linear_x = 0.05
53     angular_z = -angular_speed
```

ناحیه پنجم:

```
54 elif regions['front'] > threshold_dist and regions['left'] < threshold_dist and regions['right'] > threshold_dist:
55     state = 'Obstacle in Left'
56     linear_x = 0.05
57     angular_z = angular_speed
```

ناحیه ششم:

```
58 elif regions['front'] < threshold_dist and regions['left'] > threshold_dist and regions['right'] < threshold_dist:
59     state = 'Obstacle in Front and Right'
60     linear_x = 0
61     angular_z = -angular_speed
```

ناحیه هفتم:

```
62 elif regions['front'] < threshold_dist and regions['left'] < threshold_dist and regions['right'] > threshold_dist:
63     state = 'Obstacle in Front and Left'
64     linear_x = 0
65     angular_z = angular_speed
```

ناحیه هشتم:

```
66 elif regions['front'] > threshold_dist and regions['left'] < threshold_dist and regions['right'] < threshold_dist:
67     state = 'Obstacle in Left and Right'
68     linear_x = linear_speed
69     angular_z = 0
```

اگر مشاهده بفرمایید در ناحیه‌ی اول و دوم، الگوریتم ابداعی خود را اعمال کردیم و بقیه‌ی نواحی نیز متناسب با

حالاتی که ایجاد می کنند به صورت متفاوت پیاده شده اند. این نحوه ی پیاده سازی باعث شد که تنها یک دید ثابت بر اساس نواحی نداشته باشیم بلکه برخی نواحی بتوانند خود تصمیم گیرنده ای ادپتیو و متناسب با شرایط برای تنظیم سرعت های ربات باشند.

برای وقتی که مانعی در هیچ سمتی نباشد: (با در نظر گرفتن حد آستانه برای اختلاف)

```
state = 'No Obstacle'
linear_x = linear_speed
angular_z = 0
if regions['left']+0.15 < regions['right']:
    angular_z = (regions['right'] - regions['left'])/2.5 # Increase this angular speed for avoiding obstacle faster
elif regions['right']+0.15 < regions['left']:
    angular_z = (regions['right'] - regions['left'])/2.5
```

برای وقتی که در موانع از سه جهت محاصره شده است:

```
state = 'Obstacle in Front and Left and Right'
linear_x = -0.05#-linear_speed
angular_z = (regions['right'] - regions['left'])/2
```

لذا کد اصلی شامل این دو تابع است. حال بقیه ی کد را که روتین است فقط با تیر قرار می دهیم.

تعریف تابع main و callback function مربوط به لیزر و نیز پابلیشر سرعت ها:

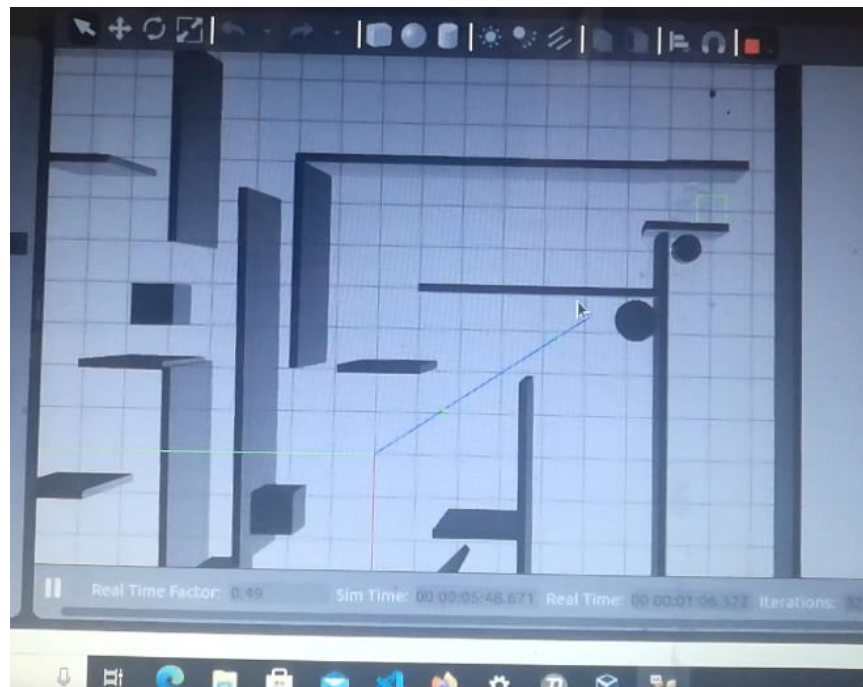
```
78
79 def main():
80     global pub
81
82     rospy.init_node('safe_efficient_obstacle_avoidance')
83     pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
84     sub = rospy.Subscriber('/scan', LaserScan, laser_scan)
85     rospy.spin()
86
87
```

در آخر نیز کد گرفتن نقطه ی شروع از کاربر است که چون جز اصلی پروژه نیست و یک قابلیت اضافه است، می توانید به محتوای main.py برای نحوه ی پیاده سازی آن مراجعه بفرمایید.

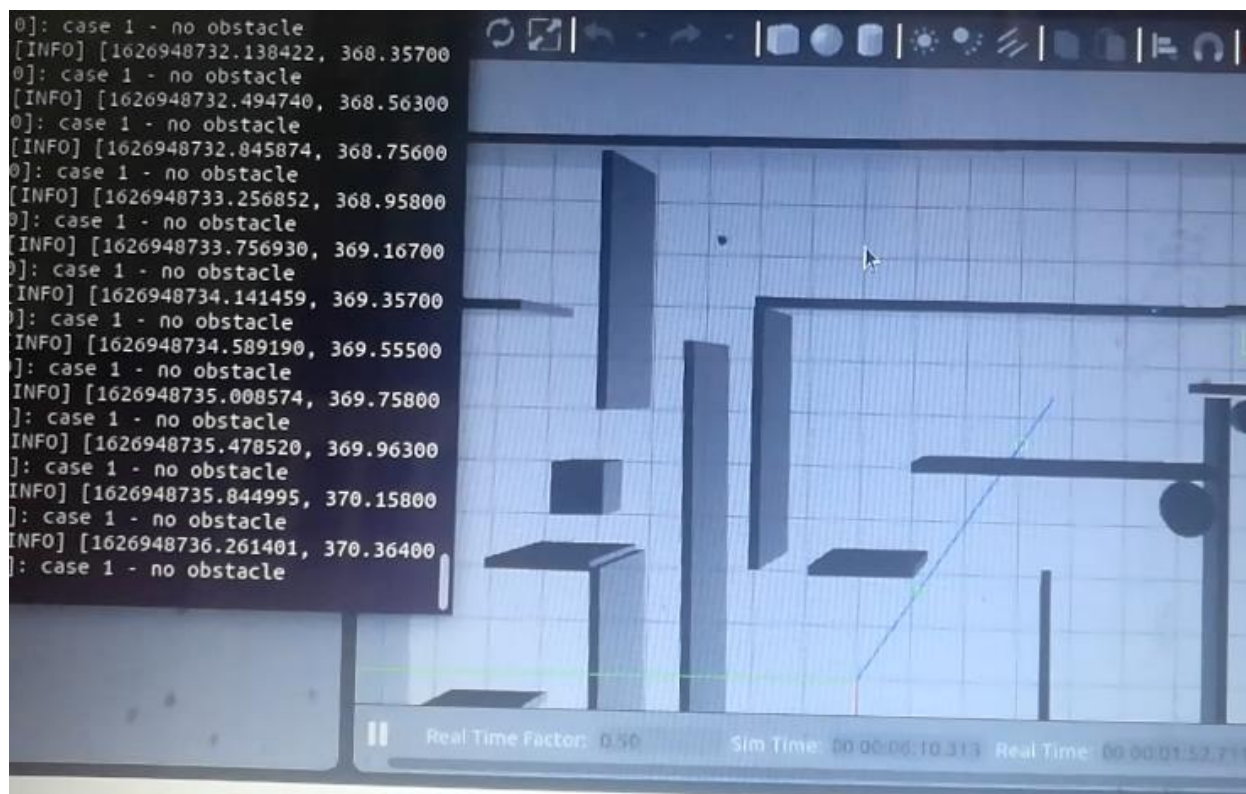
خروجی گرفته شده

با پیاده‌سازی‌ای که در فوق انجام دادیم، ربات در اکثر اوقات قادر است مسیر اصلی را تشخیص داده و یک دور کامل را بپیماید و مجدداً این کار را تکرار کند. در ویدیوی قرار گرفته صحت این موضوع را حتماً مشاهده بفرمایید. گاهی ربات ممکن است بدلیل ورود به مواجهه با ناحیه‌ای خاص با یک زاویه‌ی مشخص، ناحیه‌ی بهینه را یک ناحیه جدا از مسیر تشخیص دهد که این موضوع کاملاً درست است و نشان از ادپتیو بودن ربات در تشخیص نواحی خالی دارد اما منشا آن همین زاویه‌ی ورود است که باعث می‌شود در هر دور بهترین نواحی را برای ادامه‌ی حرکت تشخیص دهد. در دور اول چون شرایط کاملاً نرمال و ربات با یک زاویه‌ی ثابت مسیر مستقیم را می‌پیماید، لذا یک دور کامل را مطابق شکل که بهینه‌ترین حالت است می‌زند ولی چون پس از اولین دور، زوایای ورودی ربات در ناحیه‌ی اولیه متفاوت می‌شود، کمی تغییر عملکرد مشاهده می‌کنیم که نشان از صحت کد می‌باشد. نمونه‌ای را برای شما در اینجا قرار داده ایم. (این نمونه از فیلمی که گرفتیم قرار داده شده که نحوه‌ی عملکرد ربات را مشاهده بفرمایید).

شروع به حرکت ربات پس از تنظیم مکان اولیه (90 , -9 , -9)



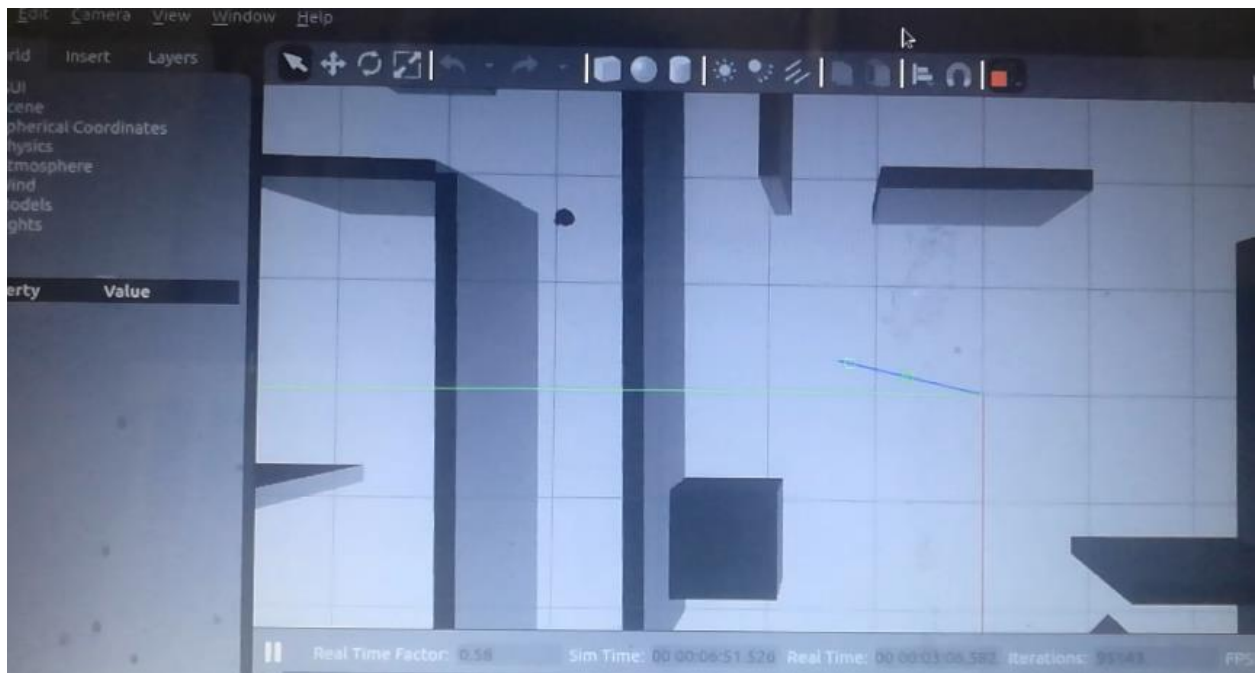
تنظیم مسیر پیش از رسیدن به مانع به جهتِ smooth کردنِ حرکت



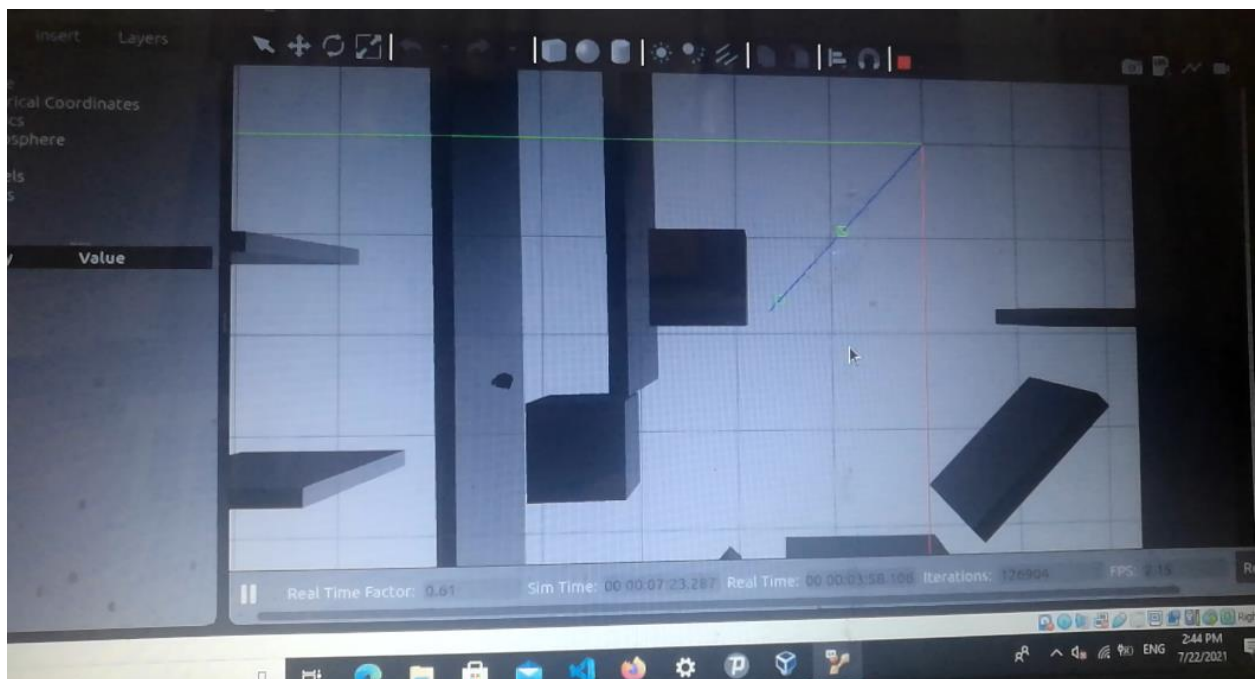
همانطور که در شکل فوق مشاهده می‌کنید هیچ مانعی که از حد آستانه کمتر شده باشد، مشاهده نشده است ولی ربات سرعت زاویه‌ای خود را پیش از آنکه بسیار نزدیک شود تغییر می‌دهد که نتیجه‌ی الگوریتمی است که بکار بردیم. سپس ناحیه مناسب برای ورود را پیدا کرده و وارد می‌شود: (ربات در سایه دیوار قرار دارد)



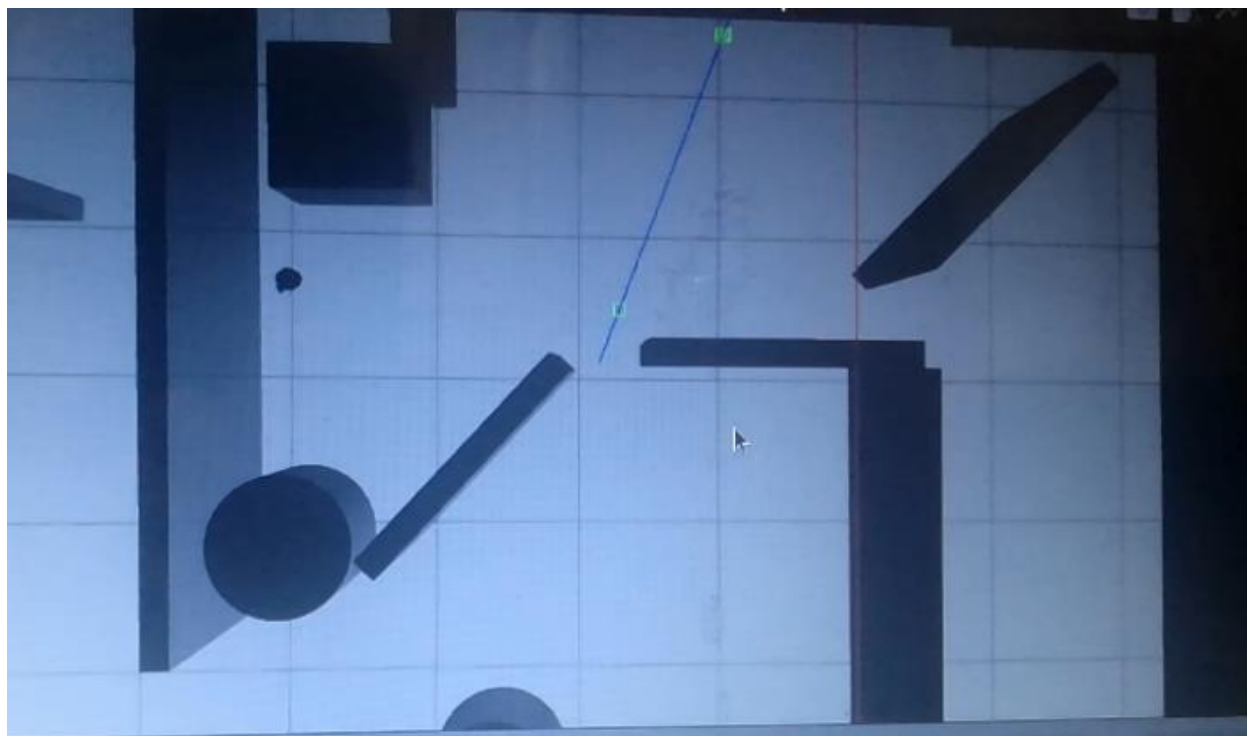
سپس :



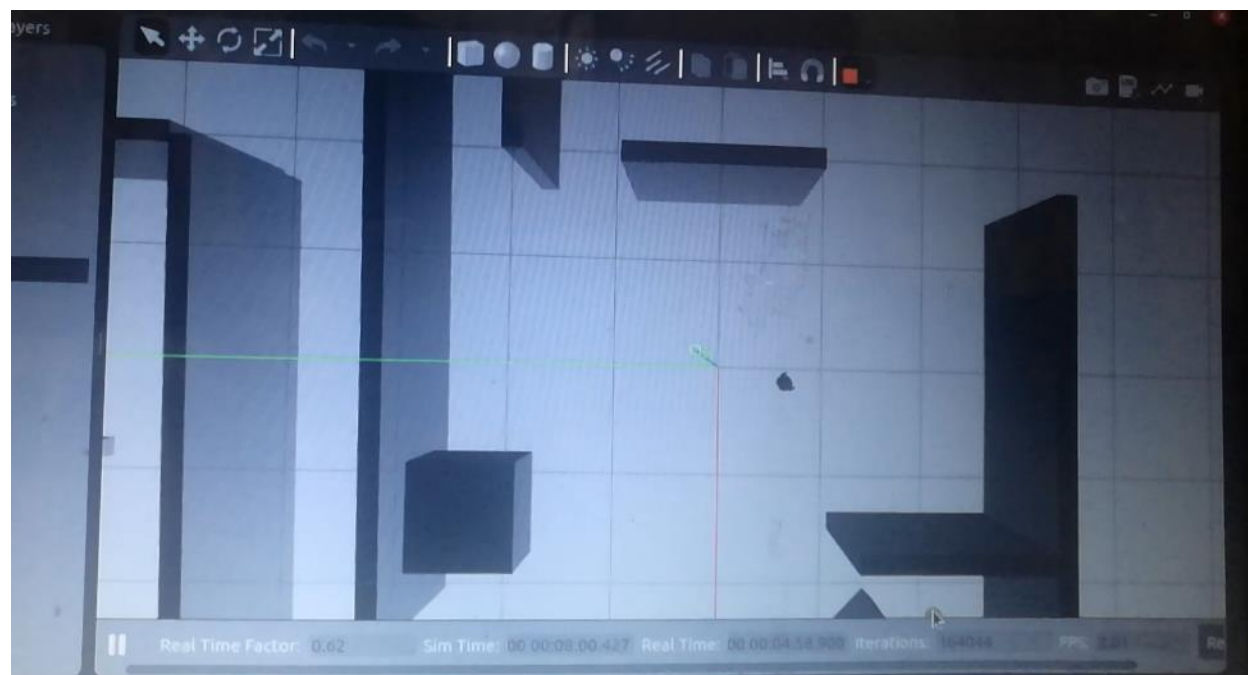
سپس:



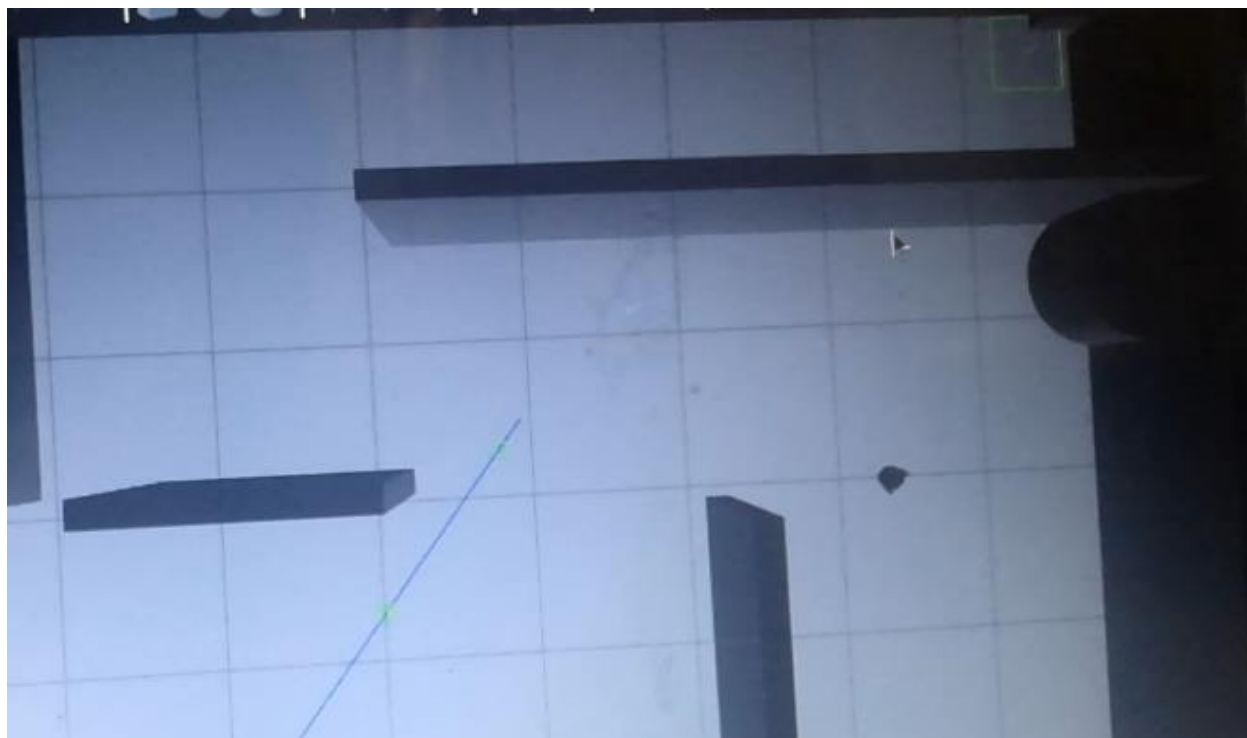
عبور از این ناحیه:



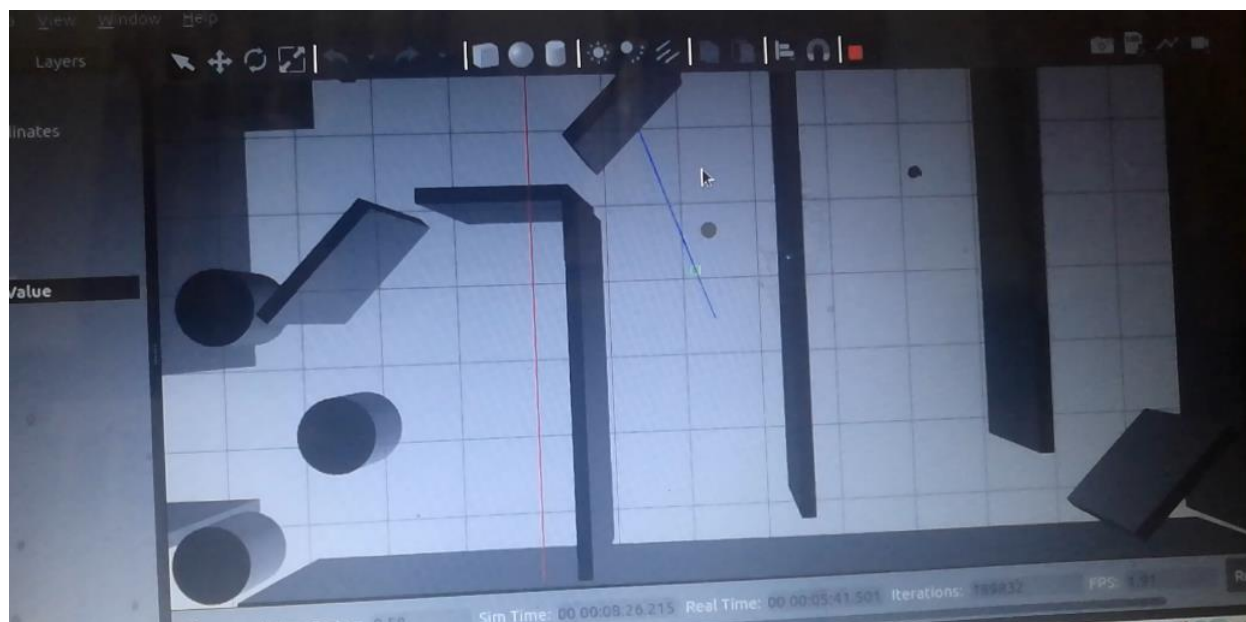
مجددا تنظیم زاویه در ناحیه‌ی جدید:



ورود به ناحیه‌ی بعدی:



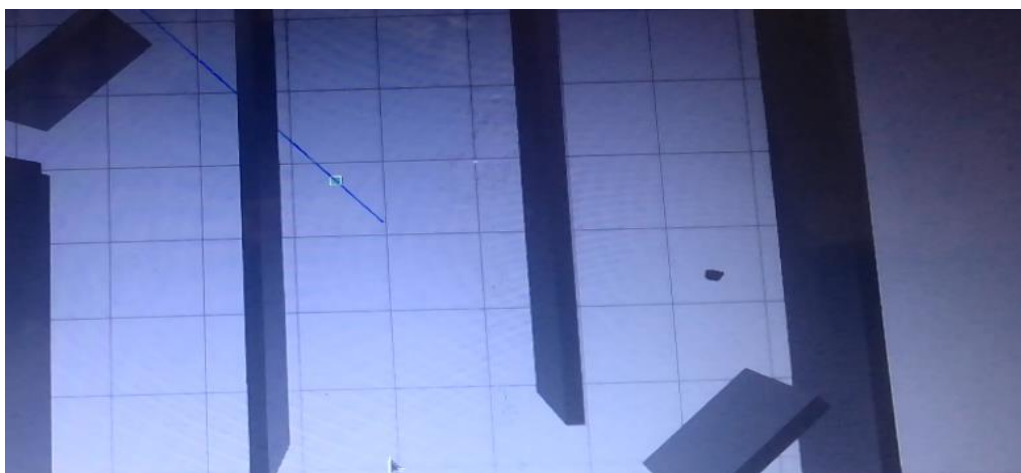
تنظیم سرعت‌ها:



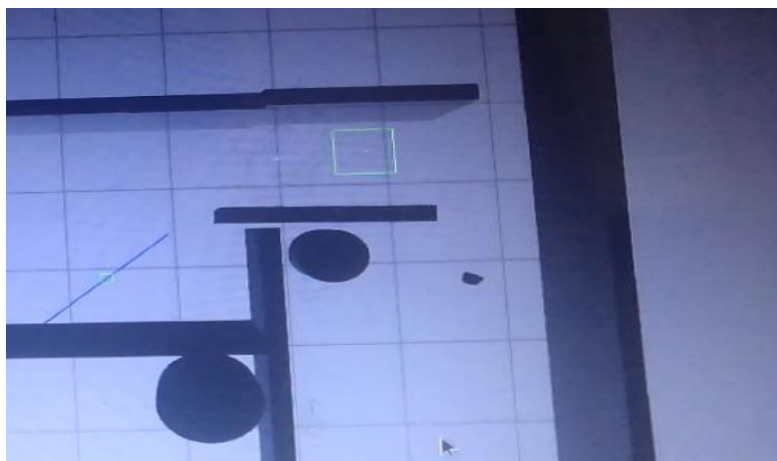
ورود به ناحیه بعدی:



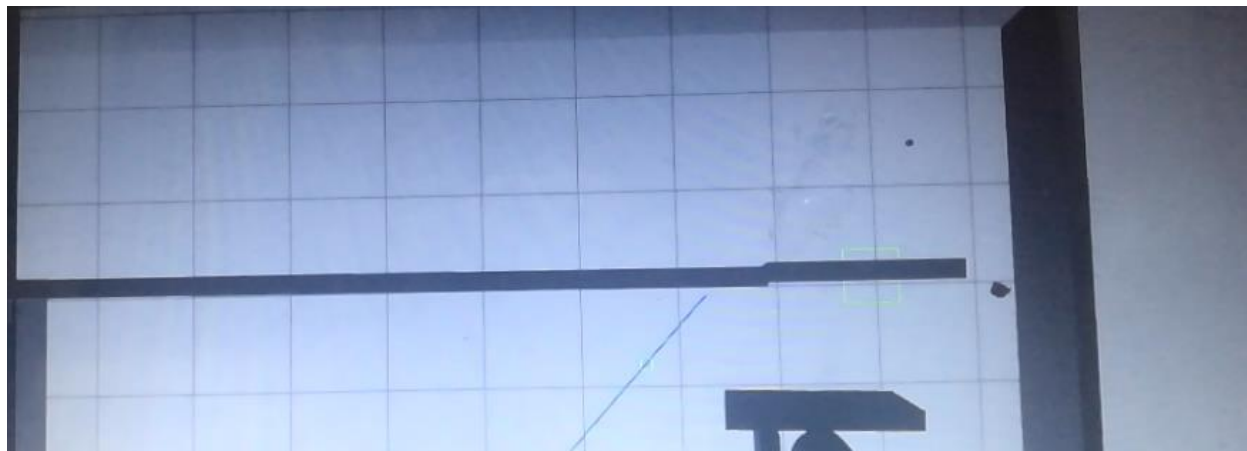
تنظیم سرعت‌ها و حرکت در این ناحیه:



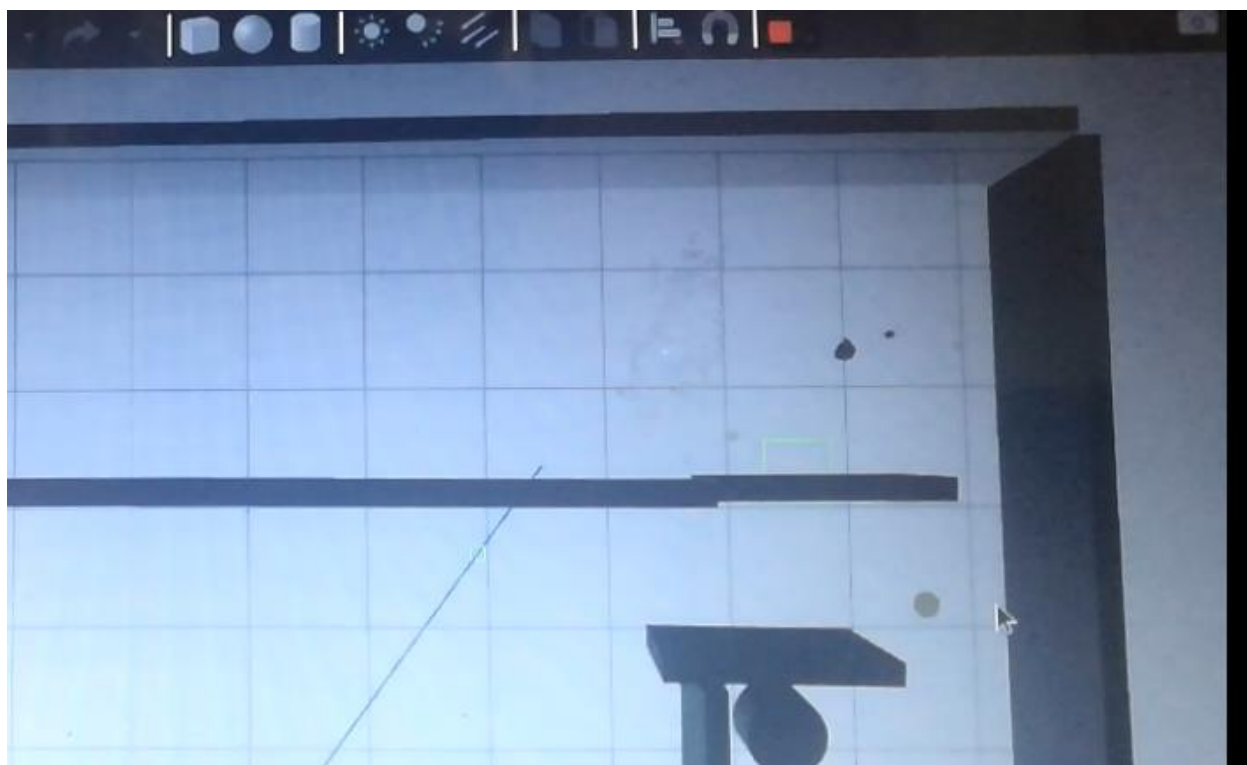
رسیدن به محیط بعدی (در حال رسیدن به نقطه‌ی شروع)



و در نهایت زدنِ یک دورِ کامل به صورتِ کورکورانه مطابقِ صورتِ سوال و رسیدن به ناحیه اولیه که شروع به حرکت کرده است:



رسیدن به نقطه‌ی شروع:



همانطور که مشاهده می‌کنید این دور رو به صورت کامل و فقط با سنسور لیزر پیمایش کرد. لطفا ویدیو را مشاهده بفرمایید.

پاسخ به سوالات:

در 15 دقیقه دو دور را ربات توانست طی کند. (البته در دور دومش یکم از مسیر بهینه منحرف شد که بدلیل زاویه‌ی خاص ورودش در مواجهه با ناحیه‌ی سمت چپ بود ولی دوباره به مسیر برگشت (در ویدیو نه دقیقه اول قرار داده شده که بدلیل محدودیت حافظه به این میزان ریکورد اکتفا کردیم که در هفت دقیقه اول آن پیمایش مسیر مطابق صورت سوال مشهود است، که می‌توانید به آن ویدیو یا ویدیوی قرار داده شده در آپارات با کیفیت بالاتر مراجعه بفرمایید)

در مورد بامپر لازم به ذکر است که ما تصمیم داشتیم با حداقل سنسور یعنی یک سنسور به مسیر خواسته شده در صورت سوال برسیم که رسیدیم ولی در کنار آن توجه داشتید که این سنسور برای turtlebot3 موجود نیست و لذا ما حتی نمی‌توانستیم از این سنسور استفاده کرده و یک خروجی ساده که اکتیو شدن آن در برخورد با مانع است بگیریم که این محدودیت نسخه‌ی راسی noetic و همچنین پکیج‌های turtlebot3 است. سرعت خطی و زاویه‌ای متوسط با ذخیره‌سازی مقادیر پابلیش شده در یک وکتور برای هر سرعت و میانگین‌گیری از آنها در مدت زمان ذکر شده، برای سرعت خطی برابر 0.4171 و برای سرعت زاویه نیز حدودا برابر 0.083 بدست آمد.

معیاری که برای smooth بودن در نظر داشتیم در واقع مسافت طی شده و نیز مجموع چرخش‌های ربات بود که دو متد را در ابتدای گزارش توضیح دادیم و سعی کردیم که حرکت ربات را smooth تر کنیم که توضیحات آن در ابتدای گزارش ارائه شد. و اما دلیل اینکه مسافت و مجموع چرخش گفته شد آن است که ربات هر چه حرکت‌های اضافه‌تری را انجام دهد، این دو پارامتر از مقادیر بالاتری برخوردار خواهند بود و لذا هر چه این دو کمتر شود می‌توان گفت که حرکت ربات smooth تر است.

نتیجه‌ای که از پیاده‌سازی گرفتیم برای ما کاملاً رضایت بخش بود بالاخص که الگوریتم حرکت را خودمان با ایده گرفتن از VFH طراحی کردیم. در نتایج هم مشاهده می‌کنید که ربات مسیر کم مانع را به خوبی می‌پیماید. هر چند به نظر می‌رسد که اگر ضرایب دقیق‌تر و تابع نوشته شده نیز بهینه‌تر این نگاشت را بین اختلاف مقادیر خروجی و سرعت زاویه‌ای برقرار کند، حتماً به نتایج بهتری خواهیم رسید ولی در همین حد و طراحی این الگوریتم، نتیجه‌ی خوبی را در پی داشته و ما رضایت کامل داشتیم.

در مورد چالش‌ها نیز همینکه می‌توان این الگوریتم را با تعریف یک نگاشت دقیق‌تر کامل نمود. به نظرم این می‌تواند زمینه‌ی طراحی یک الگوریتم جدید در مسیریابی کورکورانه باشد.

لینک آپارات

<https://aparat.com/v/95W4b>

با تشکر