به نام خدا

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

اصول علم ربات

تمرین سری دوم

نام و نام خانوادگی	حمیدرضا همتی
شماره دانشجویی	۹۶۳۱۰۷۹
تاریخ ارسال گزارش	۲۰ اردیبهشت

فهرست گزارش سوالات

سوال ۱ – بازوی رباتی

سوال ۲ – orientation

$$R_{ab} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$R_{ba} = R_{ab}^{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$P_A = R_{ab} P_B \Longrightarrow P_B = (R_{ab})^{-1} P_B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\Longrightarrow P_B = \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix} \Longrightarrow (1, -3, 2)$$

سوال ۳ – kinematic

forward kinematic :

$$x = d_3 \cos(\theta_2)\cos(\theta_1)$$

$$y = d_3 \cos(\theta_2)\sin(\theta_1)$$

$$Z = a - d_3 \sin(\theta_2)$$

$$\begin{vmatrix} x \\ y \\ z \end{vmatrix} = \begin{vmatrix} d_3 \cos(\theta_2)\cos(\theta_1) \\ d_3 \cos(\theta_2)\sin(\theta_1) \\ a - d_3 \sin(\theta_2 \end{vmatrix}$$

inverse kinematic :

$$\tan(\theta_1) = (y/x) \longrightarrow \theta_1 = \arctan(y/x)$$

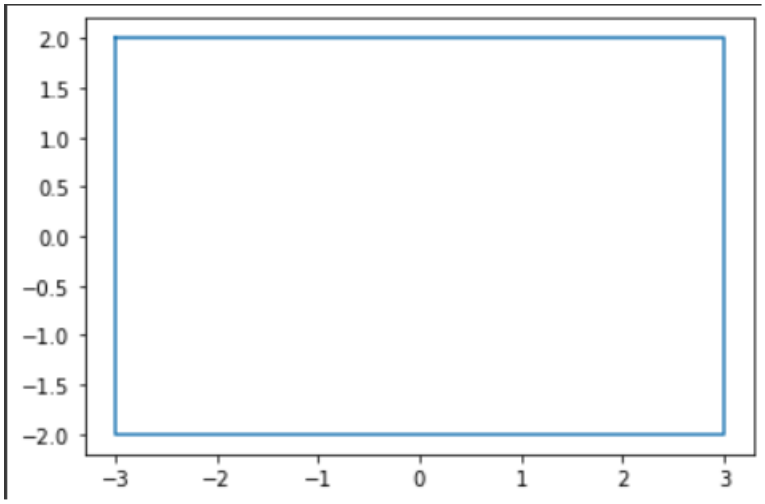
$$Z - a = d_3 \sin(\theta_2) \quad \& \quad \frac{y}{\sin(\theta_1)} = d_3 \cos(\theta_2)$$

$$\Longrightarrow \tan\theta_2 = \frac{-d_3 \sin\theta_2}{d_3 \cos\theta_2} = \frac{Z-a}{\frac{y}{\sin\theta}} \Rightarrow \theta_2 = \arctan$$

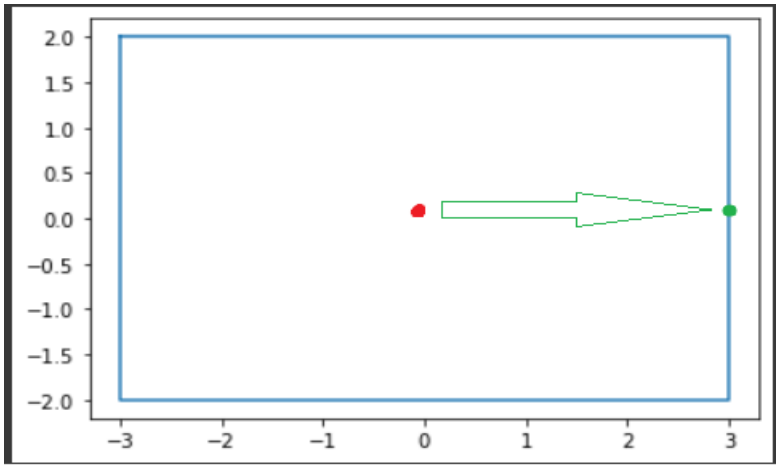
$$\longrightarrow \theta_2 = \arctan\left(\frac{Z-a}{\frac{y}{\sin(\theta_2}}\right)$$

گام اول

در این بخش باید ربات را بر روی یک مسیر مستطیلی شکل همانند زیر با استفاده از دو عملگر حرکت به سمت مقصد و چرخش ۹۰ درجه به سمت چپ کنترل کنیم. از آنجایی که در حرکت و چرخش ربات خطا وجود دارد با استفاده از این دو عملگر نمیتوان کنترل خوبی بر روی حرکت ربات داشت. چون که ربات مسیر حرکت خود را کنترل نمیکند و صرف نظر از خطایی که الان دارد صرفا دستوراتی را اجرا میکند.



نقطه شروع ربات در مرکز این مستطیل و نقطه ۰ و ۰ است. پس ربات ابتدا باید از خارج از محیط مستطیل خود را به محیط مستطیل برساند سپس حرکت خود را برروی مستطیل شروع کند. در شکل زیر مسیر اولیه ایی که ربات برای رسیدن به path انجام میدهد را مشاهده میکنید. ابدا ربات از نقطه ۰ و ۰ به نقطه x=3 و y=0 میرود و ۹۰ درجه به سمت چپ چرخش میکند.



در کد این بخش با استفاده از دو state تعریف شده on_path و of_path مدیریت میشود. در ابتدای شروع حرکت ربات state ربات of_path است و ربات به سمت نقطه x=3 و y=0 حرکت میکند و بعد از rotate ربات state به حالت on_pathتغییر میکند. در تابع controller در شکل زیر این قسمت را میتوانید مشاهده کنید.

```
153  def controller(self):
154      counter = 0
155      rospy.sleep(1)
156      while not rospy.is_shutdown():
157          if self.state == self.OFF_PATH:
158              ## beginning of the code
159              dist = self.calculate_distance()
160              self.move(dist)
161              # self.rotate(self.goal[0], self.goal[1])
162              self.rotate_90degree(90)
163              self.state = self.ON_PATH
164          print("goal: ", self.goal[0], self.goal[1])
165          dist = self.calculate_distance()
166          self.move(dist)
167          self.goal = next(self.path)
168          # self.rotate(self.goal[0], self.goal[1])
169          self.rotate_90degree(90)
170
171          if counter == 5:
172              break
173          counter += 1
174
175      plt.title("AVG error: {:.2f}".format(sum(self.errors)/len(self.errors)))
176      plt.plot(list(range(1, len(self.errors) + 1)), self.errors)
177      plt.show()
```

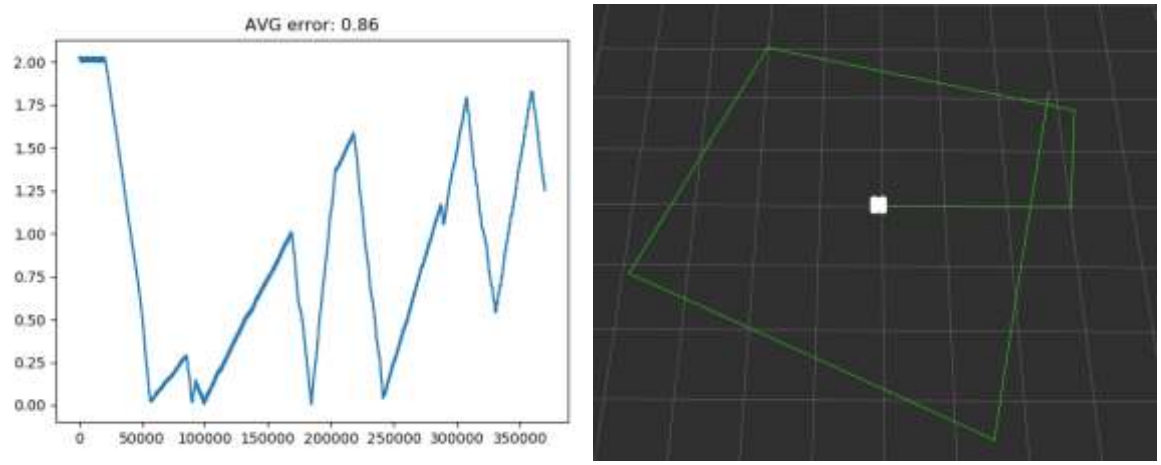
کد بخش حرکت ربات به سمت نقطه بعدی در شکل زیر آمده است.

```
120  def move(self, distance):
121      travelled_dist = 0
122      start_x = self.position_x
123      start_y = self.position_y
124
125      twist = Twist()
126
127      while distance > travelled_dist:
128          twist.linear.x = self.linear_speed
129          self.cmd_publisher.publish(twist)
130          travelled_dist = abs(self.position_y - start_y) + abs(self.position_x - start_x)
131
132          self.x_histogram.append(self.position_x)
133          self.y_histogram.append(self.position_y)
134          self.errors.append(min(min(abs(X1 - self.position_x) + abs(Y1 - self.position_y)), min(abs(X2 - self.position_x) + abs(Y2 - self.position_y))
135                             min(abs(X3 - self.position_x) + abs(Y3 - self.position_y)), min(abs(X4 - self.position_x) + abs(Y4 - self.position_y))))
136
137          print("distance: ", distance, " travelled: ", travelled_dist)
138
139      self.cmd_publisher.publish(Twist())
140      rospy.sleep(1)
```

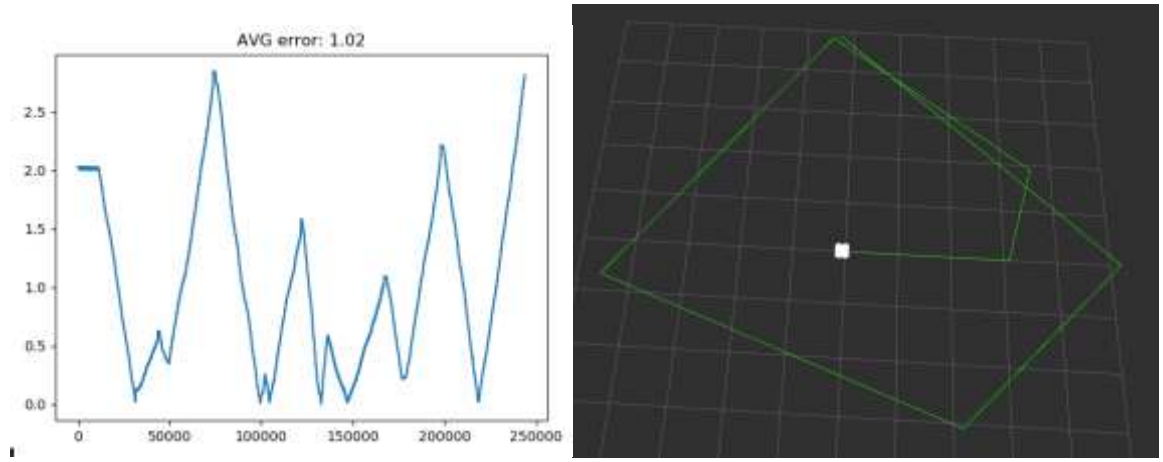
کد چرخش ۹۰ درجه به سمت چپ در شکل زیر آمده است.

```
80       def rotate_90degree(self, degree):
81           remaining = math.radians(degree)
82           prev_angle = abs(self.yaw)
83
84           twist = Twist()
85           twist.angular.z = self.angular_speed
86           self.cmd_publisher.publish(twist)
87
88           while remaining >= self.epsilon:
89               current_angle = abs(self.yaw)
90               delta = abs(prev_angle - current_angle)
91               remaining -= delta
92               print("yaw: ", math.degrees(current_angle), " remaning: ", math.degrees(remaining))
93               prev_angle = current_angle
94
95           twist.angular.z = 0
96           self.cmd_publisher.publish(twist)
97           self.cmd_publisher.publish(Twist())
98           rospy.sleep(1)
```

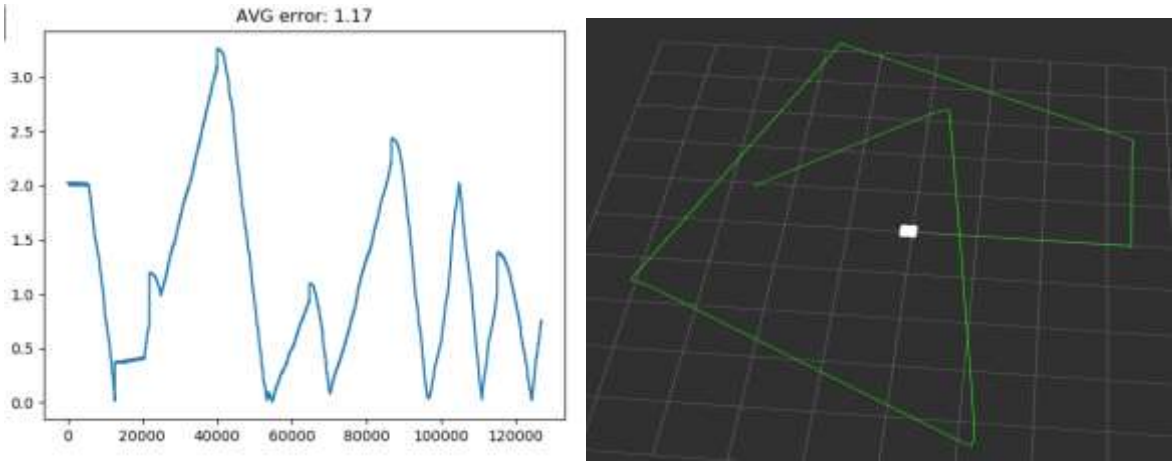
در شکل ها زیر نماهای تولید شده توسط RViz به ازای هر سرعت خطی همراه با خطای انحراف از مسیر مشخص شده است.



سرعت خطی برابر ۰٫۲



سرعت خطی برابر ۰٫۴



سرعت خطی برابر ۰٫۸

سوال: آیا افزایش سرعت خطی باعث افزایش انحراف ربات شده است؟

بله. با افزایش سرعت خطی ربات، مقدار انحراف ربات در گوشه های مسیر بیشتر میشود. دلیل این امر این است که هنگامی که ربات به نقطه ایی میرسد که باید توقف کند به دلیل اینرسی که در سرعت های بالا دارد باعث میشود ربات مقداری سر بخورد و از مسیر اصلی خارج شود. همچنین باعث میشود که ربات در گوشه ها مقداری کج هم بشود.

گام دوم

توجه: برای گام دوم و سوم کنترلر زاویه برای همه سناریو ها P است

در کل برای کنترل کردن mobile robot هایی که بر روی زمین حرکت میکنند (منظور این است که مانند کوادرو کوپتر ها قرار نیست پرواز کنند و در یک ارتفاع hover بکنند) و هدف کنترلشان رسیدن از یک نقطه به نقطه دیگر است کنترلر P کافی است و نیازی به مقادیر I و D نیست. در همین گزارش هم شکل مسیر طی شده توسط ربات با استفاده از کنترلر P آمده است که بسیار خوب است.

البته مطلب گفته شده در پاراگراف قبل موقعی صادق است که زمین اصطکاک زیادی نداشته باشد. مثلا فرض کنید که یک ربات بر روی یک موکت ضخیم قرار است از یک نقطه به هدف برود. در این صورت هنگامی که به نزدیکی هدف میرسد با اینکه مقدار error هنوز صفر نشده است ولی چون مقدار آن بسیار اندک هست و همچنین چون که اصطکاک زمین زیاد است موتور های ربات قدرت کافی برای ایجاد گشتاور ندارند و حرکت ربات ندارند. در این حالت یک steady state error به وجود آمده است. این سناریو مشابه سناریو کوادروکوپتری هست که میخواهد در یک ارتفاع ثابت قرار بگیرد. در این حالت تنها استفاده از مقدار P برای کنترل کافی نیست و باید از مقدار I هم استفاده کرد.

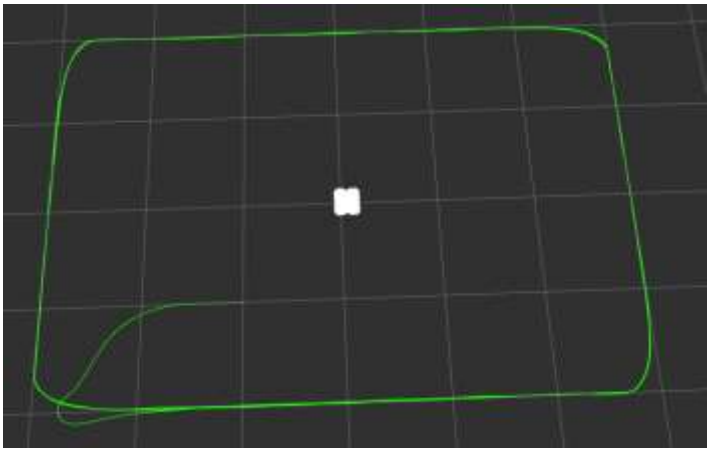
مقدار D هم برای smooth کردن تغییرات حرکت ربات ها استفاده میشود. از آنجایی با عملیات مشتق گیری انگار که داریم آینده را پیشبینی میکنیم میتوانیم از تغییرات ناگهانی در حرکت ربات جلوگیری کنیم. مثلا برای کنترل یک ربات wall following هنگامی که ربات به گوشه یک دیوار میرسد و باید تغییر زیادی در قرارگیری خود انجام دهد اگر مقدار D به درستی tune شده باشد میتوان از تغییرات ناگهانی ربات جلو گیری کرد و ربات در این صورت در مسیر smooth تری تغییر جهت میدهد.

تاثیرات ضرایب P، I و D بر روی کنترل ربات:

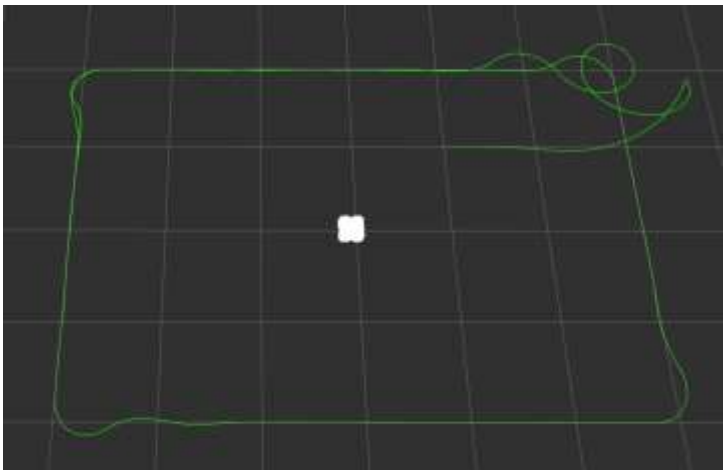
ضریب P: هرچقدر مقدار KP کوچکتر باشد سرعت ربات در کم کردن مقدار error کمتر میشود. با توجه به معادله زیر هم هرچقدر ضریب KP بزرگتر باشد سرعت بیشتری داریم. اما مقدار زیاد KP یک تاثیر بد خواهد داشت که باعث میشود عملکرد ربات جالب نباشد. اگر KP خیلی زیاد باشد باعث overshoot کردن ربات میشود چون وقتی فاصله با هدف کم میشود به دلیل مقدار زیاد ضریب باز هم بزرگ میشود و نمیتواند به موقع ربات را متوقف یا سرعت آن را به خوبی کاهش دهد.

$$P = kp * (goal - current)$$

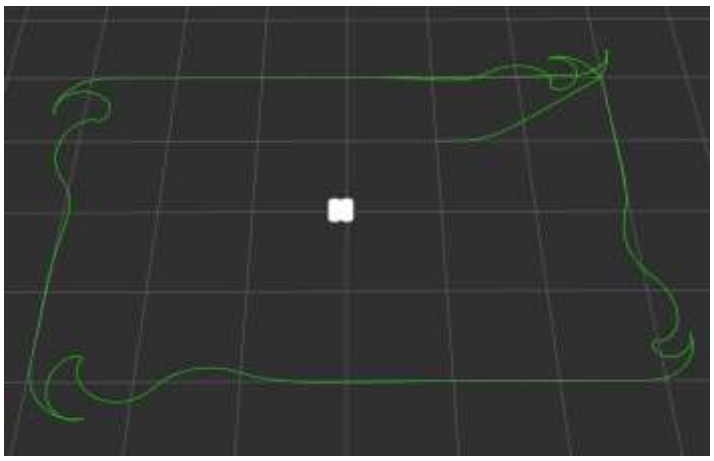
برای مدیریت کردن این مشکل باید بتوان مقدار KP را با آزمون و خطا tune کرد. در شکل های زیر دو نمونه از کنترلر P با مقدار KP زیاد و کم نشان داده شده است. همانطور که مشاهده میشود مقدار overshoot با افزایش مقدار kp زیاد میشود.



Kp = 0.5

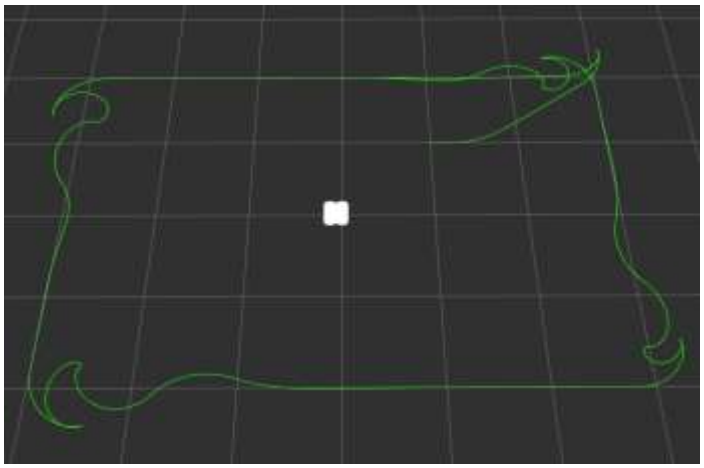


Kp = 1

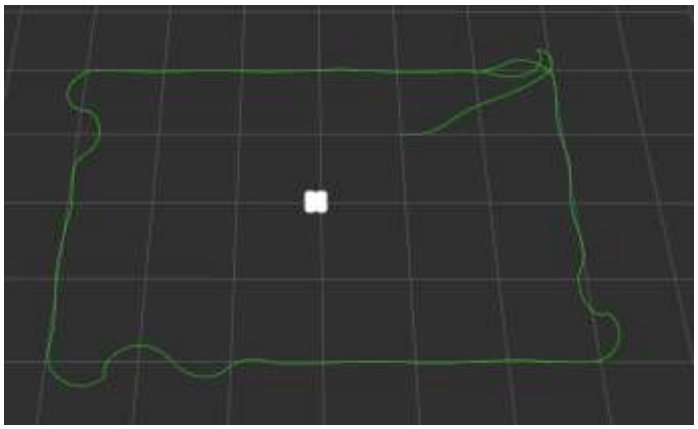


Kp = 2

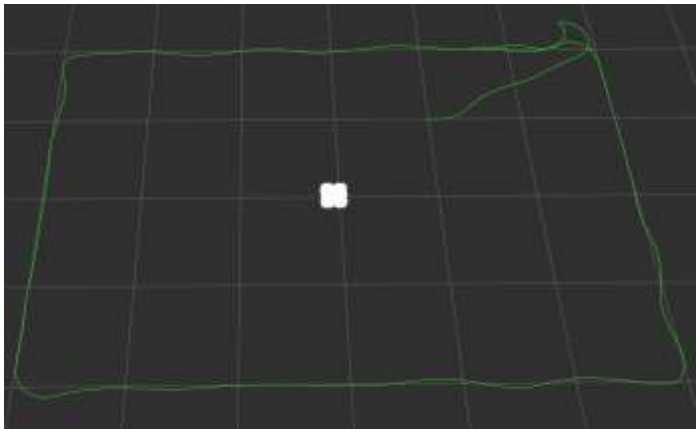
ضریب D: این ضریب برای کم کردن تغییرات ناگهانی و smooth کردن حرکات ربات اضافه میشود. هرچقدر مقدار KD بیشتر باشد سرعت تغییرات ربات کمتر میشود. برای تست کردن این ویژگی در شکل های زیر ابتدا حالتی که kp برابر ۲ و kd برابر ۰ است آوره شده. سپس میبینیم که اگر از مقدار مناسبی برای kd استفاده کنیم میتوان overshoot را کنترل کرد.



Kp =2, kI = 0, KD = 0

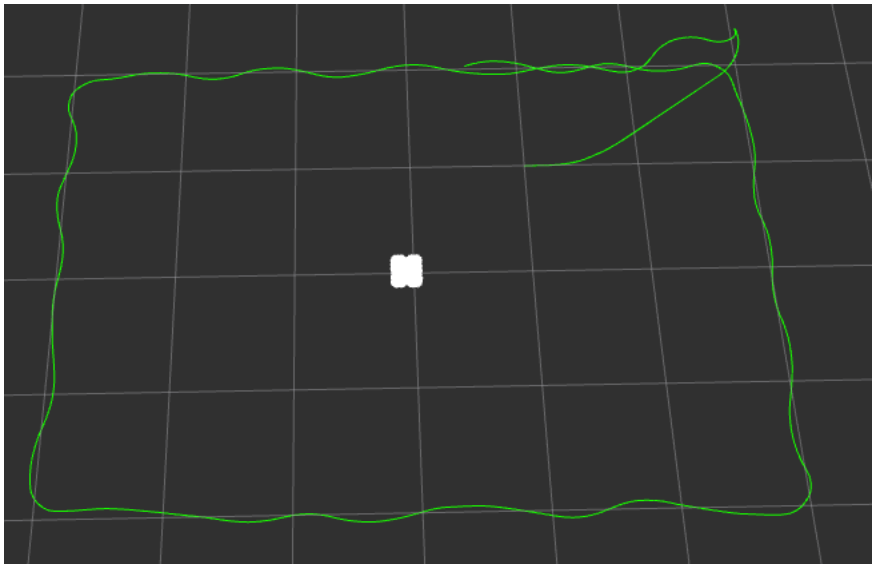


Kp =2, Ki= 0, KD = 10



Kp = 2, Ki = 0, KD = 20

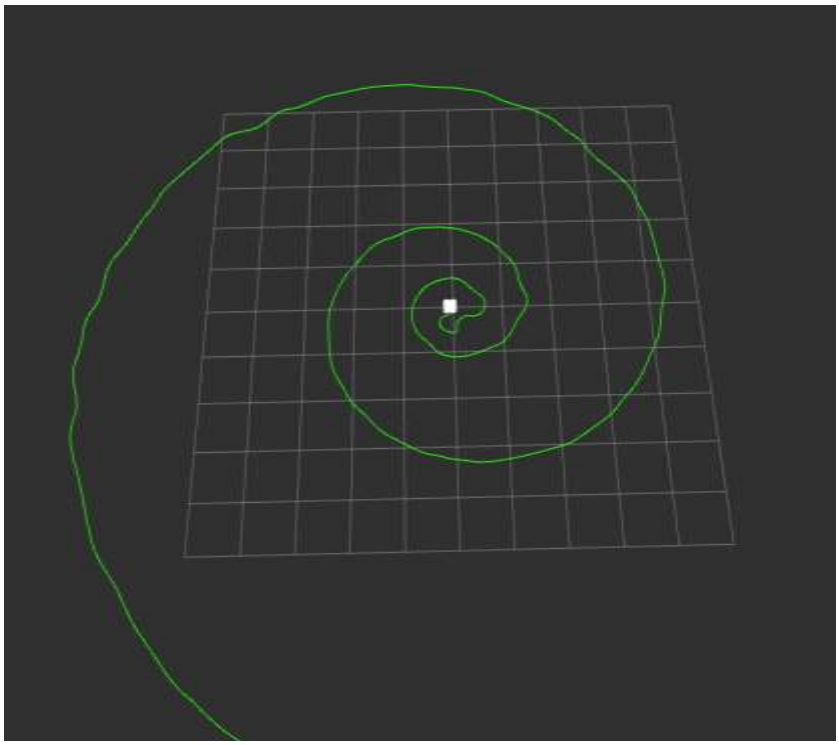
ضریب I: از انتگرال برای حذف steady state error استفاده میشود و باعث میشود که ربات کاملا بر روی مسیر خود حرکت بکند. البته اگر مقدار آن درست انتخاب نشود باعث ossilation میشود. معمولا مقدار ضریب I همیشه خیلی کم انتخاب میشود که مانند شکل زیر به این مشکل بر نخوریم.



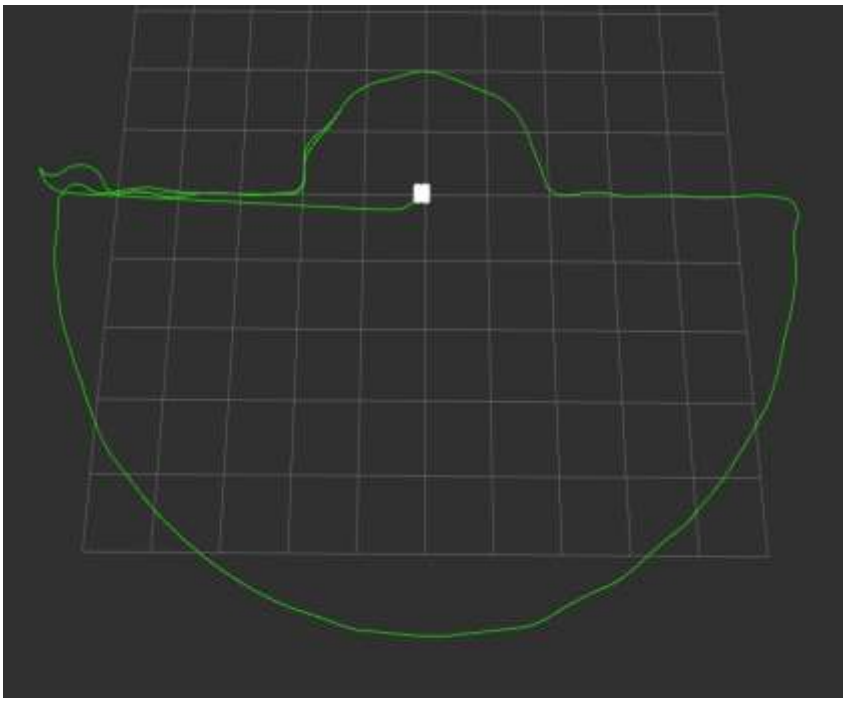
Kp=2, Ki = 0.1, Kd = 20

گام سوم

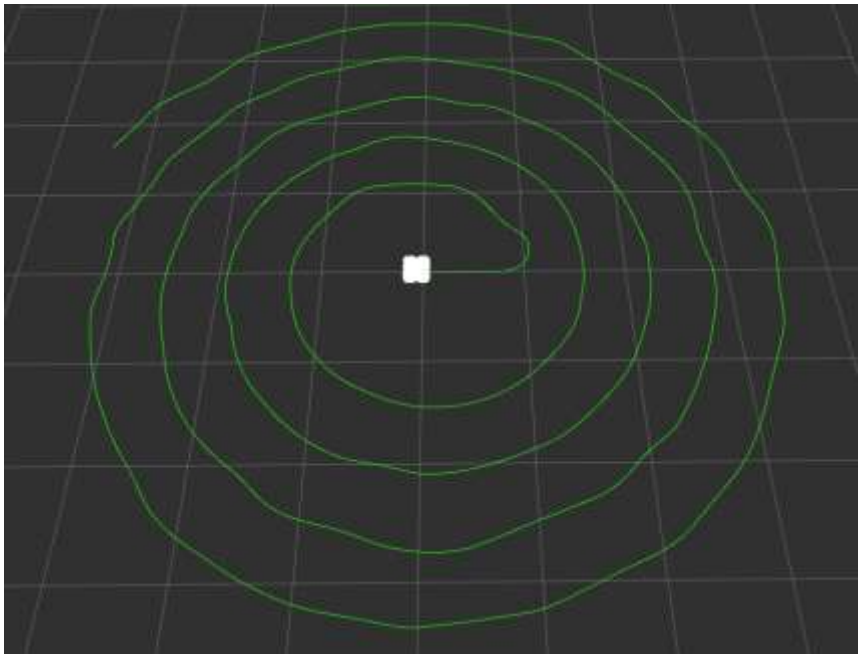
۱. مارپیچ لگاریتمی



۲. ترکیب دو نیم دایره



۳. مارپیچ ارشمیدسی



۴. هشت ضلعی منتظم