Turtlesim ve ROS Kullanarak Turtle Takibi

Busenaz Kerimgil

November 14, 2024

1 Giriş

Raporda Ros Noetic kullanılarak TurtleSim simulasyon ortamında birden fazla turtle oluşturuldu ve takip edilme algoritmaları açıklandı.Bu doğrultuda iki ana görev tanımlandı:

- turtleların oluşturulması ve rastgele konumlandırılması,
- Belirli bir turtlenın lider olarak seçilmesi ve diğer turtleların bu lideri takip etmesi.

2 Algoritma

2.1 turtleların Oluşturulması

İlk aşama, kullanıcı tarafından belirtilen sayıda turtlenın oluşturulmasıdır. Bu işlem, 'turtles_create' servisi ile gerçekleştirildi.

1. Başlangıçta, 'turtle1' turtle'ı mevcut kabul edildi ve listeye eklendi.
2. Kullanıcıdan alınan sayıya göre, 'turtle2', 'turtle3', ... gibi yeni turtlelar oluşturuldu.
3. Her turtle için rastgele bir x, y koordinatı ve başlangıç yönü (θ) belirlendi.
4. Bu turtlelar, ROS servisi kullanılarak çalışma alanına yerleştirildi.
5. turtleların isimleri bir listeye eklendi ve bu liste, '/turtle_names' kanalında yayınlandı.

Bu adımların sonunda, her turtle için rastgele bir başlangıç konumu ve yönü belirlenmiş oldu.

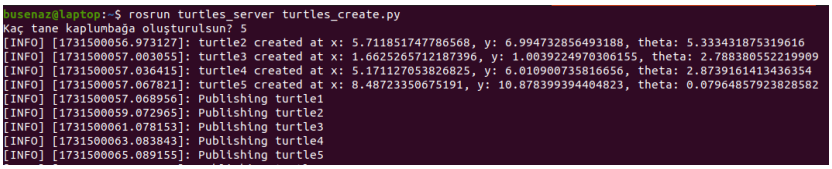

Figure 1: Turtle yaratma komutu

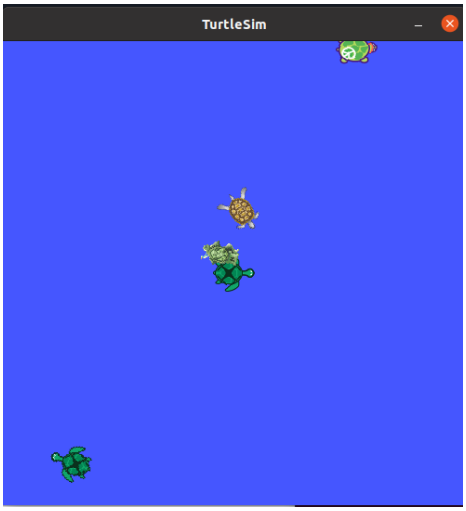

Figure 2: Simulasyon ortamı

2.2 Turtleların Takibi

Turtleların takip işlemi, ‘turtles_follow‘ servisi ile yapıldı. Bu aşamada, bir turtle lider olarak seçildi ve diğer turtlelar bu lideri takip etti.

1. Kullanıcı, lider turtleyı seçdi.
2. Her turtle, liderin pozisyonunu takip etti. Liderin pozisyonu, ‘/leader_name/pose‘ kanalında yayınlandı.
3. Takipçi turtlelar, liderle arasındaki mesafeyi ölçdü. Eğer mesafe belirli bir eşikten (bu kodda 0.5 metre) küçükse, takipçi turtle kısa süreliğine durdu.
4. Aksi takdirde, takipçi turtle, liderin konumuna doğru hareket etti. Bu hareket, PID kontrol algoritması kullanılarak yapıldı.
5. PID parametreleri, doğrusal hız (K_P) ve açısal hız (K_θ) için ayarlandı. Bu parametreler, takip mesafesini korumak ve liderin yönüne doğru hareket etmek için kullanıldı.

6. Lider turtlenın hareketi, klavye girdileri ile kontrol edildi. Klavye tuşları (w, s, a, d) ile hız ve açı ayarlandı.

7. Liderin hareketiyle birlikte, diğer turtlelar lideri takip etmek için aynı hızda ve açıyla hareket ettiler.

Figure 3: Takip eden turtle

2.3 PID Kontrolü

Takipçi turtleların lideri takip ederken doğru mesafeyi korumaları için PID kontrolü kullanıldı.

- **Proportional (P)**: Hata ile doğru orantılıdır. Yani, hata ne kadar büyükse düzeltme o kadar büyük olur.
- **Integral (I)**: Zamanla biriken hatayı düzeltir. Küçük ve sürekli hataları giderir.
- **Derivative (D)**: Hata değişim hızına bakar. Hata hızındaki ani değişimleri azaltmak için kullanılır.

PID denetleyici şu şekilde çalışır:

1. Mesafe hatası, lider ile takipçi arasındaki doğrusal mesafe ile hesaplandı.
2. Açı hatası, liderin yönü ile takipçinin mevcut yönü arasındaki fark ile hesaplandı.
3. Bu hatalar, PID parametreleri ile çarpılarak doğrusal hız (v) ve açısal hız (ω) belirlendi.

Burada, `Kp` ve `K_angular` PID parametreleridir ve bu parametreler, takipçi turtleların hareketini düzenlemek için kullandı.

4. Hızlar, belirtilen sınırlarla kısıtlandı.

   Uygulama sırasında, kullanıcılar lider turtleyı yön tuşlarıyla hareket ettirebilir. Lider turtle hareket ettikçe, diğer turtlelar lideri takip eder. Sistemin düzgün çalışması için takip mesafesi (`FOLLOW_DISTANCE`) ve minimum mesafe (`MIN_DISTANCE`) parametreleri ayarlandı. Ayrıca, takipçi turtleların birbirleriyle çakışmasını engellemek için mesafeler kontrol edildi.

5. `move_turtle` fonksiyonunda, lider ve takipçi turtlelar arasındaki mesafe ve açıya dayalı olarak lineer ve angular hızlar hesaplandı.

Kodda Kullanılan PID Parametreleri

Bu kodda belirlenen hız ve PID parametreleri aşağıdaki gibidir:

Hız Tanımları

Aşağıda, turtleların hareketini kontrol etmek için kullanılan hız parametreleri verilmiştir:

- LINEAR_SPEED = 2.0
- ANGULAR_SPEED = 2.0
- FOLLOW_DISTANCE = 1
- MIN_DISTANCE = 0.2

PID Kontrol Parametreleri

- KP_LINEAR = 1.0
- KD_LINEAR = 0.05
- KI_LINEAR = 0.1
- KP_ANGULAR = 4.0
- KD_ANGULAR = 0.5
- KI_ANGULAR = 0.6

Kodda Kullanılan PID Kontrolörü

Aşağıdaki Python classı, PID algoritmasını temel alarak takipçi turtleların hareketini kontrol eder. Bu classın amacı, hata değerini (mesafe ve açı hatası) hesaplayarak uygun hız değerlerini çıkarmaktı.

```
class PIDController:
    def __init__(self, kp, ki, kd):
        self.kp = kp  # Proportional Gain
        self.ki = ki  # Integral Gain
        self.kd = kd  # Derivative Gain
        self.prev_error = 0.0  # Önceki hata
        self.integral = 0.0

    def update(self, error, dt):
        # Temel PID hesaplaması
        self.integral += error * dt
        derivative = (error - self.prev_error) / dt if dt > 0 else 0.0
        output = self.kp * error + self.ki * self.integral + self.kd * derivative
        self.prev_error = error  # Önceki hatayı güncelle
        return output
```

Buradaki PID algoritması, `error` ve zaman farkı (`dt`) parametreleriyle hesaplamalar yaparak çıkış sinyali üretti.

Takipçi turtleların Hız Hesaplaması

Turtleların lideri takip etmesi için her bir takipçi turtle, liderin konumuna göre lineer ve açısal hızlarını PID kontrolü ile hesapladı. Aşağıda bu hesaplamaların nasıl yapıldığı gösterilmektedir:

- Lider ile takipçi arasındaki mesafe şu şekilde hesaplanır:

  $$dx = \text{leader\_pose.x} - \text{follower\_pose.x}, \quad dy = \text{leader\_pose.y} - \text{follower\_pose.y}$$

  $$\text{distance} = \sqrt{dx^2 + dy^2}$$

- Mesafe hatası:

  $$\text{distance\_error} = \text{distance} - \text{FOLLOW\_DISTANCE}$$

- Lineer hız, PID kontrolörü ile hesaplanır:

  $$\text{linear\_speed} = \text{linear\_pid.update(distance\_error, dt)}$$

- Liderin hedef konumuna olan açı hesaplanır:

  $$\text{angle\_to\_target} = \text{atan2}(dy, dx)$$

- Açısal hız, PID kontrolörü ile hesaplanır:

  $$\text{angular\_error} = \text{angle\_to\_target} - \text{follower\_pose.theta}$$

  $$\text{angular\_speed} = \text{angular\_pid.update(angular\_error, dt)}$$

Hız Limitleri ve Mesafe Kontrolü

Takipçi turtleların birbirine çok yakın olmalarını engellemek için mesafe kontrolü uygulandı. Eğer takipçi turtle diğerine çok yakınsa, hızları sınırlandırıldı:

$$\text{if distance\_to\_other} < \text{MIN\_DISTANCE:}$$

$$\text{linear\_speed} = \max(\text{linear\_speed} - 0.2, 0)$$

Bu kontrol, robotların birbirine çarpmasını engellemeye yardımcı oldu.

2.4 Kullanılan ROS Komutları

turtles_create.py

- `rospy.init_node('turtles_create')`: Bu komutla `turtles_create` adlı bir ROS düğümü başlatılır.
- `rospy.Publisher('/turtle_names', String, queue_size=50)`: `/turtle_names` adlı bir kanal üzerinde turtle isimleri yayınlanır.
- `rospy.ServiceProxy('spawn', Spawn)`: Bu servis kullanılarak yeni turtlelar yaratılır.

turtles_follow.py

Bu dosya, lider turtleyı kontrol etmek için kullanılan klavye girdilerini aldı ve lider turtlenın hareketini, takip eden turtleların hareketini hesaplamak için PID kontrolü ile yönlendirdi. Ayrıca, her bir turtlenın pozisyonu dinlenir ve liderin pozisyonuna göre takipçi turtleların hareketi sağlandı.

ROS Komutları

- `rospy.Subscriber('/turtle_names', String, turtle_name_callback)`: turtle isimlerini dinler.
- `rospy.Subscriber(f'/leader_name/pose', Pose, leader_pose_callback)`: Lider turtlenın pozisyonunu dinler.
- `rospy.Subscriber(f'/turtle_name/pose', Pose, follower_pose_callback(turtle_name))`: Takipçi turtleların pozisyonlarını dinler.
- `rospy.Publisher(f'/follower_name/cmd_vel', Twist, queue_size=10)`: Takipçi turtleların hareket komutlarını yayınlar.

Hareket Kontrolü

- `get_key()`: Klavye tuşlarından gelen girişleri alır ve lider turtlenın hareketini yönlendirir.
- `move_turtle(follower_name)`: Takipçi turtleların lideri takip etmesi için mesafe ve açı hatalarını PID kontrolü ile hesaplar.

OK Mesajı İçin ROS Servisi

OK mesajı yayınlamak için, TurtleFollow adında bir srv dosyası yazıldı. Bu servis, bir turtlenın başarılı bir şekilde hareket ettiğini bildiren OK mesajlarını içerir.

srv dosyası input olarak string "turtle_name" almakta ve output olarak "OK" mesajını basmakta.

Servis dosyasının içeriği:

```
# TurtleFollow.srv
string turtle_name  # İstek kısmı: Takip edilecek turtlenın adı
---
string response       # Yanıt kısmı: "OK" mesajı
```

3 Sonuç

Bu algoritma, birden fazla turtlenın oluşturulması ve lider turtlenın etrafında takipçi turtleların hareket etmesini sağlamaktadır. turtlelar arasında belirli bir mesafe ve düzen korunur, bu sayede istenilen görev başarıyla tamamlanır. ROS Publisher ve Subscriber yapıları kullanılarak turtleların pozisyonları sürekli olarak güncellenir ve iletişim sağlanır. OK mesajı ile her işlemin başarılı olduğu bildirilir.

Proje boyunca, echo komutu ile ROS terminal çıktıları izlenebilir. Bu çıktılar, her bir turtle'ın konum ve hareket bilgilerini içerir ve PID kontrolünün doğru çalışıp çalışmadığını gösterir.

Aşağıda, ekranda görülen bazı çıktılar yer almaktadır:

```
rostopic echo /turtle_names
```

Bu komut, yayımlanan turtle isimlerini ekrana yazdırır.

Figure 4: yayımlanan turtle isimleri

`rostopic echo /{turtle_name}/cmd_vel`

Bu komut, herhangi istenilen bir kamplumbağanın hareket komutlarını yayınlar.

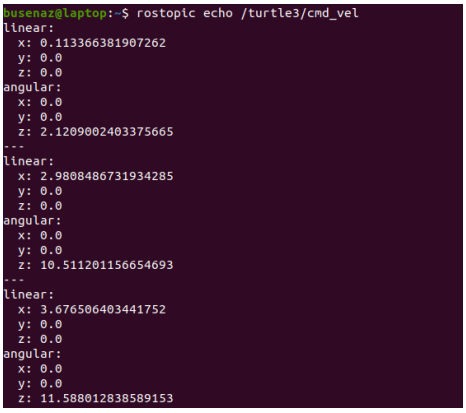

Figure 5: turtle3 için olan haraket komutları

En son tüm topicleri görebimek için şu komutu kullanabiliriz.

`rostopic list`

```
busenaz@laptop:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
/turtle2/cmd_vel
/turtle2/color_sensor
/turtle2/pose
/turtle3/cmd_vel
/turtle3/color_sensor
/turtle3/pose
/turtle4/cmd_vel
/turtle4/color_sensor
/turtle4/pose
/turtle5/cmd_vel
/turtle5/color_sensor
/turtle5/pose
/turtle_names
```

Figure 6: Tüm topicler