Robot Guidance with Head Movements

Versiyon : v2.0
Tarih : 06/09/2020
Hazırlayan : Elif GENÇ

DÖKÜMAN REVİZYON SAYFASI

VERSIYON	TARİH	SAYFA	AÇIKLAMA
v1.0	31/08/2020	10	Ubuntu Bionic Beaver ve Ros Melodic Kurulumu, Ros Beginner Tutorials
v2.0	06/09/2020	14	Publisher/Subscriber ve Service/Client yapılarının Turtlesim üzerinde örnekleri

İÇİNDEKİLER

1. TANITIM VE KAPSAM

Bu çalışma kapsamında bizden istenilen Publisher - Subscriber ve Service - Client yapısını anlamak ve bu yapıları kullanarak TurtleSim üzerinde bir uygulama gerçekleştirmektir. "Düğüm" tabiri ROS jargonunda, birbiriyle haberleşen, programlanabilir bağlantılar olarak bilinir. [1] Publisher program içerisinde mesajı yayınlayan düğüm iken, Subscriber ise Publisher' ın gönderdiği mesajları alan ve okuyan düğümdür. Publisher ve Subscriber düğümlerinin kullandığı mesaj dosyaları ROS alanları içerisinde kullanılan mesajları içeren basit yazı dosyalarıdır. Bu mesajlar ve düğümler farklı kodlama dilleriyle yazılabilir ve kullanılabilir. [2]

Servis, Client' tan gelen mesajlara göre bir çıktı üreterek bu çıktıyı Client düğümüne geri gönderen bir düğümdür. Bu servis düğümünü bir kere çalıştırarak Client' tan gelen birden çok isteğe cevap veren bir yapısı vardır. Client ise Servis' te yapılması istenilen işlemler için mesajları gönderen düğümdür.

2. KULLANILAN MATERYALLER

Çalışmalarımız boyunca Publisher-Subscriber ve Service-Client yapıları için kullanacağımız dosya türleri; msg ve srv dosyalarıdır.

Service ve Client yapıları, ileti ve dönüt (request and response) olmak üzere iki farklı yapı kullanır. İleti ve dönüt yapıları srv dosyaları içerisinde iki farklı kısım olarak bulunur. srv dosyaları, servisleri tanımlayan dosyalardır. Bu dosyalar srv kütüphanelerinde tutulur. [3]

Çalışmalar kapsamında kullanacağımız düğümler, msg dosyalarını kullanır. msg dosyaları ROS alanları içerisinde kullanılan mesajları içeren basit yazı dosyalarıdır. msg dosyaları, paketler içinde yer alan msg kütüphanelerinde tutulur. msg' ler, mesajların döndüğü alanların tipi ve bu alanların ismini satır satır text dosyalarında tutar. Bu alan tipleri aşağıdaki listedeki gibidir. [3]

- int8, int16, int32, int64 (plus uint*)
- float32, float64
- string
- time, duration
- other msg files
- variable-length array[] and fixed-length array[C]

srv dosyaları da tıpkı msg' ler gibi bir içeriğe sahiptir ve bu içerikler ileti ve dönüt şeklinde iki kısımdır. Bu iki kısım dosya içerisinde "---" şekliyle ayrılmıştır. Aşağıda bir srv dosya örneği gösterilmektedir.

```
int 64 A

int 64 B

---

int 64 Sum
```

Yukarıdaki örnek dosya içeriğinde A ve B ileti (request), Sum ise dönüt' tür (response). [3]

3. ÖNERİLEN YÖNTEM / PROBLEM ÇÖZÜM AŞAMALARI

Publisher-Subscriber ve Sevice-Client yapılarını kullanırken srv ve msg dosyaları kullanılmaktadır. Bu dosyaları kullanabilmek için de "beginner_tutorials" isimli paket içerisindeki package.xml ve CMakeLists.txt dosyalarda gerekli ayarlamaların yapılması gerekmektedir. Msg dosyaları içerisinde değişiklik yapıldığında ilk olarak package.xml dosyası içerisine aşağıdaki kodlar eklenir. [3]

Srv ve msg dosyalarını c++, python gibi farklı dillerde kullanabilmek için aşağıdaki kodları package.xml dosyasına eklemek yeterli olacaktır.

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

Ardından srv ve msg dosyalarını kullanabilmek için CMakeLists.txt dosyasına aşağıdaki kodlar eklenerek gerekli ayarlamalar yapılmıştır.

```
find_package(catkin REQUIRED COMPONENTS
        roscpp
        rospy
        std_msgs
        message_generation
)

catkin_package{
...
CATKIN_DEPENDS message_runtime ...
...)

add_message_files(
        FILES
        Num.msg
)

generate_messages(
        DEPENDENCIES
        std_msgs
)

Add_service_files(
        FILES
        Request.srv
)
```

Ardından "beginner_tutorials" isimli paketin içerisinde srv klasörü içerisindeki "Request.srv" isimli dosyasına kullanacağımız değişkenler eklenmiştir. "Request.srv" isimli dosya içeriği aşağıdaki gibidir.

```
int64 a
int64 b
int64 c
---
int64 respon
```

4. GERÇEKLENEN GÖREVİN AŞAMALARI

- Srv ve msg dosyalarının oluşturulması için öncelikle mesajı taşıyan paket klasörü (beginner_tutorials) içerisine gidilmesi gerekmektedir. Daha sonra bu paket klasörü içerisinde msg isimli bir klasör oluşturulur. Bu klasör içerisine Num isimli msg dosyası oluşturulur ve bu dosya içerisine tipi int64 olan bir değişken tanımlanır.

  $ roscd beginner_tutorials

  $ mkdir msg

  $ echo "int64 num" > msg/Num.msg

  Eğer aradığımız msg'nin paketi unutulursa paket adı yazılmadan da msg bulunabilir. Bunun için aşağıdaki kodlar kullanılır.

  $ rosmsg show Num

  ```
  ||
  [beginner_tutorials/Num]:
                  ||
                  int64 num
  ```

- srv dosyası, oluşturulmak istenen pakete(beginner_tutorials) içerisine gidilerek "srv" isimli klasör oluşturulur. Ardından ileti ve dönüt için "*AddTwoInts*" adında srv dosyaları oluşturulur.

  $ roscd beginner_tutorials

  $ mkdir srv

  $ roscp beginner_tutorials AddTwoInts.srv srv/AddTwoInts.srv

4.1. PUBLISHER-SUBSCRIBER

- Öncelikle beginner_tutorials isimli paketine gidilir. Daha sonra bu paket içrisine 'scripts' isimli bir klasör oluşturulur. Bu klasör yazılacak Python kodlarını içerecektir.

  $ roscd beginner_tutorials
  $ mkdir scripts

  $ cd scripts

- Bu komut ile txt uzantılı bir dosya açılır. Bu açılan dosya içerisine Publisher düğümünün yapacağı işlerin python dilindeki kodları yazılır ve ardından uzantısı .py olacak şekilde scripts klasörü içerisine kaydedilir.

$ gedit

```
#!/usr/bin/env python

import rospy #rospy kütüphanesi eklendi

from geometry_msgs.msg import Twist #Turtlesimden linear ve angular hız
#verilerini almak için kullanıldı.

from turtlesim.msg import Pose #Turtlesimden konum bilgilerini almak için kullanılır.

import sys #Kullanıcıdan parametre almak için kullanılır.

x = 0.0

def poseCallback(pose_message):

        global x

        rospy.loginfo("Robotun x koordinati = %f\n", pose_message.x)

        x = pose_message.x #Robotun x koordinatı x değişkenine aktarılır.

def circular_turtle(lineer,angular,distance):

   global x

   rospy.init_node(' circular_turtle ', anonymous=False)

   pub = rospy.Publisher('/turtle1/cmd_turtle_velocity', Twist, queue_size=10)

   rospy.Subscriber('/turtle1/pose',Pose, poseCallback)

   rate = rospy.Rate(10)

   turtle_velocity= Twist()

   while not rospy.is_shutdown():

        turtle_velocity.linear.x = lineer # Kullanıcının girdiği ilk parametre lineer
                                          # hızın x değişkenine aktarılır.

        turtle_velocity.linear.y = 0

        turtle_velocity.linear.z = 0


        turtle_velocity.angular.x = 0

        turtle_velocity.angular.y = 0
```

```
        turtle_velocity.angular.z = angular #Kullanıcının girdiği ikinci parametre
                                            #açısal hızın z değişkenine aktarılır.

        if(x >= distance) #Robotun x koordinatı, kullanıcının girdiği üçüncü#
                          #parametresinden büyük olunca robot durdurulur.

            rospy.loginfo("Robot Hedefe Ulasti")

            rospy.logwarn("Robot Durdu.")

            break

    pub.publish(turtle_velocity)

    rate.sleep()
if __name__ == '__main__':

  try:

    circular_turtle (float(sys.argv[1]), float(sys.argv[2]), float(sys.argv[3]))

#Kullanıcıdan alınan parametreler ile circular_turtle isimli fonksiyon çağrılır.

  except rospy.ROSInterruptException:

    pass
```

- Yazılan python kodlarını derlemek etmek için öncelikle "beginner_tutorials" isimli pakete gidilmesi gerekmektedir. Daha sonra yazılan python dosyasını derlemek için ikinci kod kullanıldı. Daha sonra "catkin_ws" isimli klasöre gidildi. Bu klasör içerisinde "$ catkin_make" komutu ile catkinin kendi standart araçlarıyla doldurulması sağlanır.

  $ roscd beginner_tutorials/

  $ chmod +x scripts/distan.py

  $ cd

  $ cd catkin_ws

  $ catkin_make

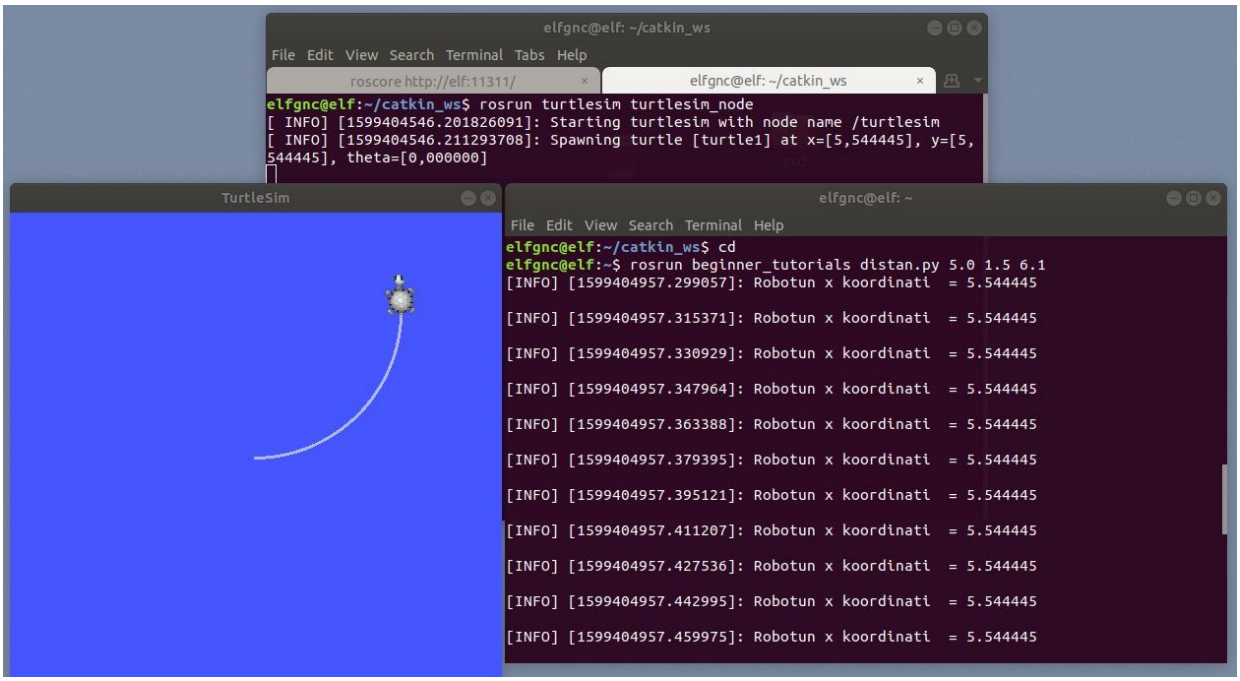
- Başka bir terminal açılarak "$ roscore" komutu çalıştırılır. Daha sonra turtlesim paketi içerisindeki düğümler çalıştırılır.

  $ roscore

  $ rosrun turtlesim turtlesim_node

- Başka bir terminal açılarak "distan" isimli python dosyası üç parametre verilerek çalıştırılır. Şekil 1'de bu yapının çıktısı verilmektedir.

$ rosrun beginner_tutorials distan.py 5.0 1.5 6.1



Şekil 1: Örnek Çıktı

4.2. SERVICE-CLIENT

- Service için yazılan kodlar

```
#!/usr/bin/env python

import rospy #rospy kütüphanesi eklendi

from beginner_tutorials.srv import Request, Response #beginner_tutorials isimli pakette
                    # srv klasöründe bulunan Request ve Response isimli srv dosyalarını
                    # tanımlar

from geometry_msgs.msg import Twist   #Turtlesimden linear ve angular hız
                                      #verilerini almak için kullanıldı.

from turtlesim.msg import Pose #Turtlesimden konum bilgilerini almak için kullanılır.

robot_x = 0

def pose_callback(pose):

        global robot_x

        rospy.loginfo("Robotun x koordinati = %f\n",pose.x)

        robot_x = pose.x #Robotun x koordinatı x değişkenine aktarılır.

def move_turtle(req):
```

```
global robot_x

rospy.init_node('move_turtle', anonymous=False)

pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

rospy.Subscriber('/turtle1/pose',Pose, pose_callback)

rate = rospy.Rate(10)

vel = Twist()

while not rospy.is_shutdown():

    vel.linear.x = req.a # Kullanıcının girdiği ilk parametre lineer hızın x değişkenine
                                                                    # aktarılır.

    vel.linear.y = 0

    vel.linear.z = 0

    vel.angular.x = 0

    vel.angular.y = 0

    vel.angular.z = req.b #Kullanıcının girdiği ikinci parametre açısal hızın z
                                #değişkenine aktarılır.

    if(robot_x >= req.c): #Robotun x koordinatı, kullanıcının client üzerinden girdiği
                                #üçüncü parametresinden büyük olunca robot durdurulur.

        rospy.loginfo("Robot Hedefe Ulasti")

        rospy.logwarn("Robot Durdu")

        vel.linear.x = 0

        vel.linear.y = 0

        vel.linear.z = 0

        vel.angular.x = 0

        vel.angular.y = 0

        vel.angular.z = 0

        pub.publish(vel)

        rate.sleep()

        return Response(1)
```

```
                    break
        pub.publish(vel)
        rate.sleep()
def new_server():
    rospy.init_node('move_turtle')
    s = rospy.Service('new_server', Request, move_turtle) #Service'in ismi, tipi ve
                                                          #çalıştırdığı fonksiyon tanımlanır.
    rospy.spin()
if __name__ == '__main__':
    try:
       new_server()
    except rospy.ROSInterruptException:
        pass
```

- Client yapısı için yazılan kodlar

```
#!/usr/bin/env python
from __future__ import print_function
import sys  #Kullanıcıdan parametre almak için kullanılır.
import rospy # rospy kütüphanesi eklendi
from beginner_tutorials.srv import *
def client_request(x, y, z):
    rospy.wait_for_service('new_server')
    try:
        server_request = rospy.ServiceProxy('new_server', Request)
        resp1 = server_request(x, y, z) #Servise gönderilen isteği temsil eder.
        return resp1.respon #Servisten dönen cevaptır.
```

```
    except rospy.ServiceException as e: #Eğer bu try içinde bir hata oluşursa
                                        #dönülecek exception
        print("Servis hatalidir: %s"%e)
def usage():
    return "%s [x y z]"%sys.argv[0]
if __name__ == "__main__":
    if len(sys.argv) == 4:
        x = int(sys.argv[1]) #Kullanıcının girdiği birinci parametre
        y = int(sys.argv[2]) #Kullanıcının girdiği ikinci parametre
        z = int(sys.argv[3]) #Kullanıcının girdiği üçüncü parametre
    else:
        print(usage())
        sys.exit(1)
    print("Requesting %s,%s,%s"%(x, y, z))
    print("%s"%(client_request(x, y, z)))
```

- Bu dosyalar oluşturulduktan sonra beginner_tutorials isimli paketine gidilir. Daha sonra bu dosyalar derlenir.

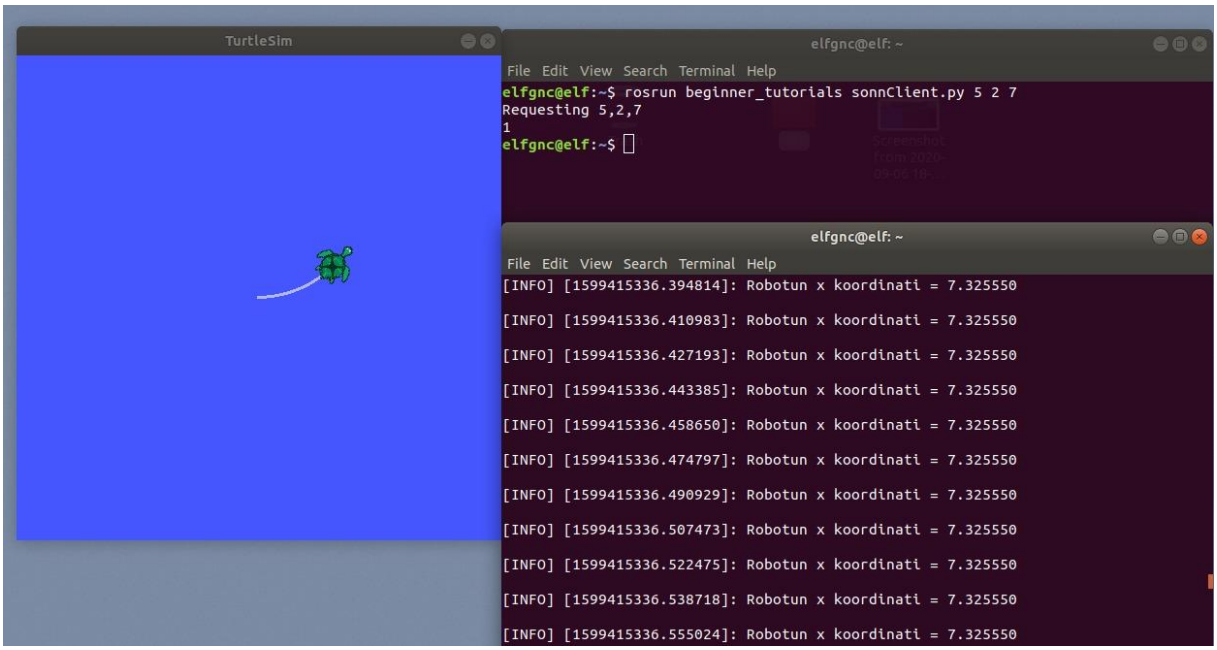
  $ roscd beginner_tutorials
  $ chmod +x scripts/sonn.py
  $ chmod +x scripts/sonnClient.py
  $ cd
  $ cd catkin_ws
  $ catkin_make
  $ roscore
  $ rosrun turtlesim turtlesim_node

- Daha sonra başka bir terminale geçilir ardından Service ve Client düğümleri ayrı terminallerde aşağıdaki çalıştırılır.

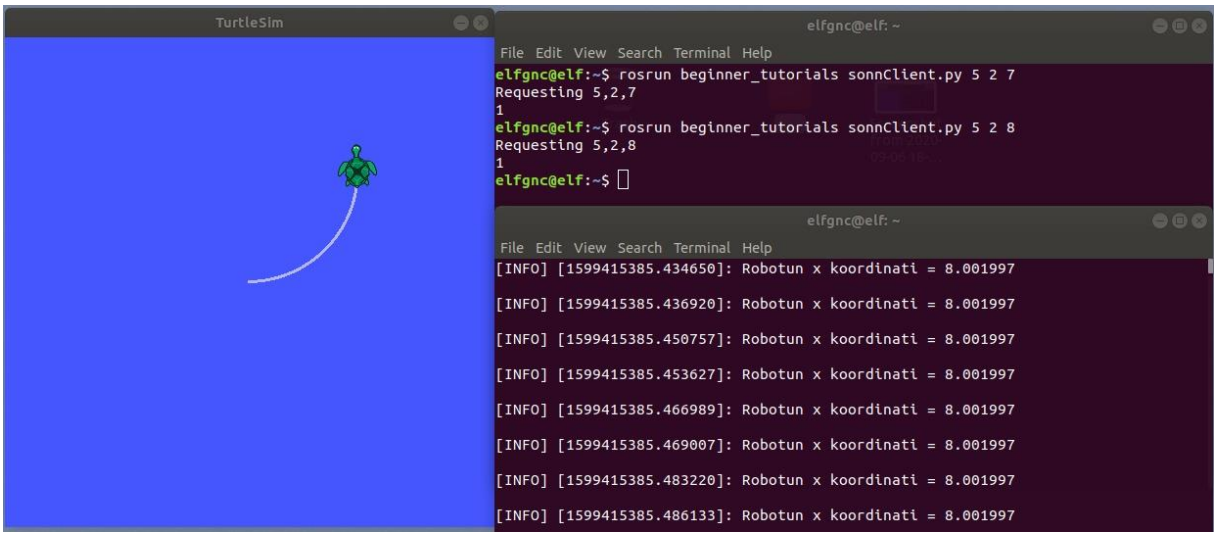
  $ rosrun beginner_tutorials sonn.py

  $ rosrun beginner_tutorials sonn_client.py 5 2 7

- Örnek çıktı şekil 2 ve şekil 3' teki gibidir.



Şekil 2: Service-Client Örnek Çıktı



Şekil 3: Service-Client Örnek Çıktı

5. SONUÇLAR

Yapılan çalışmalar kapsamında Publisher-Subscriber ve Service-Client yapılarını, bu yapıların gerektirdiği ayarlamaların nasıl yapıldığını ve bu yapıları kullanarak Turtlesim üzerinde çalışmalar yapılmıştır. Msg ve srv yapısını kullanabilmek için package.xml ve CMakeLists.txt dosyalarında gerekli ayarlamaların ve bu yapılar sayesinde terminaller arası iletişimin nasıl gerçekleştirildiği öğrenildi. Bu yapıları kullanarak bir terminalden girilen veriler ile TurtleSim düğümü üzerinde robotun konumunun ve hareketlerinin değiştirilmesi sağlandı.

6. KAYNAKÇA

[1] http://wiki.ros.org/tr/ROS/Tutorials/Python%20Kullan%C4%B1larak%20Yay%C4%B1nc%C4%B1%20%28Publisher%29%20ve%20%C4%B0zleyici%20%28Subscriber%29%20D%C3%BC%C4%9F%C3%BCmleri%20Yazma [Erişim tarihi: Ağustos, 2020]

[2] http://wiki.ros.org/tr/ROS/Tutorials/ROS%27ta%20%22msg%22%20ve%20%22srv%22%20Dosyalar%C4%B1n%C4%B1%20Olu%C5%9Fturma [Erişim tarihi: Ağustos, 2020]

[3] http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv [Erişim tarihi: Ağustos, 2020]

[4] http://yapbenzet.kocaeli.edu.tr/ros-dagitimlari/ [Erişim tarihi: Ağustos, 2020]