

ROS D    m Yazma Temelleri

November 25, 2023

 Anonymous

ros da c++ ile d    m ler yazmak i in ne gibi bilgilere sahip olmam z gerekir

 ChatGPT

ROS (Robot Operating System), robot yazılımları geli tirmek i in kullanılan a ık kaynaklı bir platformdur. C++ ile ROS d    m (node) yazmak i in a a ıdaki temel bilgilere sahip olmanız  nemlidir:

1. C++ Programlama Bilgisi:

ROS, C++ dilini kullanarak yazılmıştır. Bu nedenle, C++ dilinde temel bilgi sahibi olmanız gereklidir. C++ dilinde veri t rleri, d    ler, ko ul ifadeleri, fonksiyonlar ve nesne y nelimli programlama (OOP) gibi temel konuları bilmek  nemlidir.

2. ROS Temel Kavramlar:

ROS'un temel kavramlarını anlamak  nemlidir. Bu kavramlar arasında d    m (node), konu (topic), hizmet (service), parametre (parameter), yayın (publish), abone (subscribe) gibi terimler bulunmaktadır.

3. ROS D    m Olu turma:

ROS d    mlerini olu turmak i in `ros::init` fonksiyonunu kullanmalısınız. Bu, ROS sistemini ba latır ve d    m n z  tanımlar.

cpp

 Copy code

```
#include "ros/ros.h"

int main(int argc, char **argv) {
    // ROS sistemini ba lat
    ros::init(argc, argv, "my_node");

    // ROS d    m  olu tur
    ros::NodeHandle nh;
```

```
// D ğ m iřlemleri burada yapılır

return 0;
}
```

4. ROS Yayın ve Abonelik:

ROS, iletiřimi temsil etmek i in "yayın" ve "abonelik" konseptlerini kullanır. Yayın yapmak i in `ros::Publisher` kullanılırken, bir konuyu dinlemek i in `ros::Subscriber` kullanılır.

```
cpp Copy code

#include "ros/ros.h"
#include "std_msgs/String.h"

int main(int argc, char **argv) {
    ros::init(argc, argv, "my_node");
    ros::NodeHandle nh;

    // Yayın oluřtur
    ros::Publisher pub = nh.advertise<std_msgs::String>("my_topic");

    // Abone ol
    ros::Subscriber sub = nh.subscribe("my_topic", 10, callbackFu);

    // D ğ m iřlemleri burada yapılır

    ros::spin(); // Etkileřimli d ğ mlerde kullanılır

    return 0;
}
```

5. Callback Fonksiyonları:

ROS, olay tabanlı bir mimariye sahiptir. D ğ mler genellikle asenkron olarak  alıřır ve mesajları almak i in callback fonksiyonları kullanılır.

```
cpp Copy code
```

```
void callbackFunction(const std_msgs::String::ConstPtr& msg) {  
    // Mesaj işleme burada yapılır  
}
```

6. ROS Parametreleri:

ROS parametreleri, düğümler arasında yapılandırma bilgilerini paylaşmak için kullanılır.

cpp

Copy code

```
ros::NodeHandle nh;  
int my_param;  
nh.getParam("my_param", my_param);
```

Bu temel bilgilerle başlayarak, ROS'un daha karmaşık konseptlerini ve kullanım durumlarını öğrenmeye devam edebilirsiniz. ROS'un resmi belgeleri ve örnekleri de size bu konuda yardımcı olacaktır.

Anonymous

peki ben orta seviyede c++ biliyorum. ros için de ayrıca yeni şeyler öğrenecek miyim ?
eğer evet ise neler öğrenmem gerekiyor

ChatGPT

Eğer orta seviyede C++ biliyorsanız, ROS kullanarak yeni beceriler kazanacaksınız. ROS, robot yazılımlarını geliştirmek için tasarlanmış bir framework olduğu için genellikle aşağıdaki konularda yeni bilgiler edinmeniz gerekebilir:

1. ROS Mesajları ve Servisleri:

ROS içinde iletişim, mesajlar ve servisler üzerinden gerçekleşir. ROS mesajları, düğümler arasında veri iletimini sağlar. Ayrıca, ROS servisleri, düğümler arasında işlem çağrıları yapmak için kullanılır. Bu konseptleri anlamak ve kendi özel mesajlarınızı ve servislerinizi nasıl tanımlayacağınızı bilmek önemlidir.

2. ROS Paketleri ve Yapılandırma:

ROS projeleri genellikle paketler halinde düzenlenir. Her paket, bir veya birden fazla düğüm içerebilir. ROS paketleri oluşturmak, yapılandırmak ve yönetmek, geliştirme sürecinde önemli bir beceridir.

3. ROS'ün Diğer Modülleri:

ROS, çeşitli modüller ve kütüphaneler içerir. Örneğin, tf kütüphanesi, dönüşümleri ve robot konumunu takip etmek için kullanılır. ROS Navigation Stack, robotların harita

oluřturma ve yol planlama gibi karmařık grevleri gerekleřtirmelerine yardımcı olur.

4. ROS Hata Ayıklama ve İzleme:

ROS projelerinde hata ayıklama becerileri kazanmak nemlidir. roscpp iindeki hata ayıklayıcı araları ve ROS ile ilgili hata ayıklama tekniklerini ğrenmek, geliřtirme srecinizi kolaylařtırabilir.

5. ROS İletişim Desenleri:

ROS iletişimde kullanılan yayın/abonelik deseni, servis ağrılar ve diğeri iletişim desenlerini anlamak, karmařık sistemlerin nasıl entegre edileceğini anlamana yardımcı olabilir.

6. ROS Simülasyon Araları:

ROS, robot simülasyonu iin eřitli aralar sunar. Örneğini, Gazebo simülasyon ortamı ve ROS üzerinde simülasyon iin özel paketler. Bu araları kullanarak robot yazılımlarınızı gerek dnya kořullarında test edebilirsiniz.

7. ROS Topluluğı ve Kaynakları:

ROS topluluğunu takip etmek ve ROS ile ilgili kaynakları incelemek nemlidir. ROS topluluğı, forumlar, eğitim materyalleri ve örnek projeler ile doludur. Diğeri ROS kullanıcılarıyla etkileřimde bulunarak daha fazla ğrenme fırsatı elde edebilirsiniz. Bu konularda alışarak ve projeler geliřtirerek, ROS ile daha derinlemesine bir anlayış kazanabilir ve karmařık robot yazılımlarını geliřtirebilirsiniz.



Anonymous

ros daki dğüm ve paketleri açıkla



ChatGPT

ROS'ta dğümler (nodes) ve paketler (packages) temel yapı taşlarıdır. İşte bu kavramların açıklamaları:

1. ROS Dğümleri (Nodes):

- ROS dğümleri, bir ROS sistemini oluřturan temel alışma birimleridir.
- Her bir dğüm, genellikle belirli bir görevi gerekleřtiren bir işlemi temsil eder. Örneğini, sensör verilerini okuma, hareket kontrolü, görüntü işleme gibi görevler bir dğümde gerekleřtirilebilir.
- Dğümler, ROS üzerinde yayın yapabilir (publish) veya bir konuyu dinleyebilir (subscribe) ve servislere erişebilir.

2. ROS Paketleri (Packages):

- ROS paketleri, bir veya daha fazla dğümü ieren ve bağımsız olarak alışabilen birimlerdir.

- Her paket, belirli bir amaca hizmet eden dosyaları ve kodları içerir. Bu dosyalar arasında genellikle düğüm tanımlamaları, yapılandırma dosyaları, launch dosyaları, derleme talimatları, bağımlılık tanımlamaları ve benzeri bulunabilir.
- Paketler, projeyi modüler ve yönetilebilir kılar. Örneğin, bir sensör sürücüsünü içeren bir paket veya bir simülasyon ortamını yöneten bir paket olabilir.
- ROS paketleri, catkin adı verilen bir derleme sistemini kullanarak derlenir ve yürütülür.

ROS Konsepti:

3. ROS, dağıtılmış sistemlerin tasarımına odaklanır. Bu nedenle, bir ROS uygulaması
 - genellikle birçok düğümden (nodes) oluşur.Düğüm, bir ağ üzerinde iletişim kurarak bilgi alışverişinde bulunur. Bu iletişim
 - genellikle "konular" (topics) ve "hizmetler" (services) üzerinden gerçekleşir.Konular, düğümlerin asenkron bir şekilde veri paylaşmasına izin verir. Yani bir düğüm bir
 - konuya mesaj gönderirken, diğer düğümler o konuyu dinleyebilir.Hizmetler, bir düğümün diğer bir düğüme özel bir görevi gerçekleştirmesi için senkron
 - bir şekilde talepte bulunmasına izin verir.

ROS Yapılandırması:

4. ROS paketleri, CMakeLists.txt ve package.xml adlı iki temel yapılandırma dosyası
 - içerir.CMakeLists.txt, paketin nasıl derleneceği ve bağımlılıklarını nasıl yöneteceği ile ilgili
 - bilgiler içerir.package.xml, paketin ROS'taki diğer paketlerle olan bağımlılıklarını ve paketle ilgili
 - diğer meta bilgileri içerir.

ROS'taki düğümler ve paketler, modüler bir yapı oluşturarak karmaşık robot yazılımlarını daha kolay yönetilebilir ve genişletilebilir kılar.



Anonymous

düğümleri daha detaylı açıkla




ChatGPT

ROS düğümleri, bir ROS sistemini oluşturan temel çalışma birimleridir. Düğümler genellikle belirli bir görevi gerçekleştiren bir işlemi temsil eder. İşte ROS düğümleri hakkında daha fazla detay:

1. Düğüm Oluşturma ve Başlatma:

- ROS düğümleri, `ros::init()` fonksiyonuyla başlatılır. Bu fonksiyon, ROS sistemini başlatır ve düğümünüzü tanımlar.

cpp

 Copy code

```
#include "ros/ros.h"

int main(int argc, char **argv) {
    // ROS sistemini başlat
    ros::init(argc, argv, "my_node");

    // ROS düğümü oluştur
    ros::NodeHandle nh;


    // Düğüm işlemleri burada yapılır

    return 0;
}
```

2. Konu (Topic) Yayını ve Abonelik:

- Düğümler, birbirleriyle iletişim kurmak için konu (topic) tabanlı bir iletişim modelini kullanır. Yayın (publish) yapan bir düğüm, belirli bir konuya mesaj gönderir. Abone (subscribe) olan düğümler, o konuyu dinler ve gelen mesajları işler.

cpp

 Copy code

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void callbackFunction(const std_msgs::String::ConstPtr& msg) {
    // Mesaj işleme burada yapılır
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "my_node");
    ros::NodeHandle nh;

    // Yayın oluştur
    ros::Publisher pub = nh.advertise<std_msgs::String>("my_top
```

```

// Abone ol
ros::Subscriber sub = nh.subscribe("my_topic", 10, callback

// D ğ m iřlemleri burada yapılır

ros::spin(); // Etkileřimli d ğ mlerde kullanılır


return 0;
}

```

3. Servisler (Services):

- D ğ mler, birbirlerine belirli g revleri yerine getirmeleri i in servis  ağrılarını yapabilirler. Servis  ağrısı yapan d ğ m, belirli bir hizmete talepte bulunur ve karřılığında bir cevap alır.

cpp

 Copy code

```

#include "ros/ros.h"
#include "std_srvs/Trigger.h"

bool myServiceCallback(std_srvs::Trigger::Request& req, std_srvs::Trigger::Response& res) {
    // Servis talebi karřılandığında yapılacak iřlemler
    res.success = true;
    res.message = "Servis  ağrısı bařarıyla tamamlandı";
    return true;
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "my_node");
    ros::NodeHandle nh;

    // Servis oluřtur
    ros::ServiceServer service = nh.advertiseService("my_service", myServiceCallback);

    // D ğ m iřlemleri burada yapılır

    ros::spin(); // Etkileřimli d ğ mlerde kullanılır
}

```

```
return 0;
}
```

4. Parametreler:

- Düğümler, çalışma zamanında belirli parametreleri alabilir ve bu parametreleri kullanabilirler.

cpp

Copy code

```
ros::NodeHandle nh;
int my_param;
nh.getParam("my_param", my_param);
```

ROS Hata Ayıklama:

- 5.
- ROS düğümleri üzerinde hata ayıklama yapmak için `roscpp` kütüphanesinin sağladığı araçları kullanabilirsiniz. `ROS_ERROR`, `ROS_WARN`, `ROS_INFO` ve `ROS_DEBUG` gibi makrolar, farklı hata düzeylerinde mesajlar yazmanıza olanak tanır.

Bu örnekler, temel ROS düğüm konseptlerini açıklamaktadır. Gerçek bir ROS projesinde, bu temel konseptlere ek olarak, konfigürasyon dosyaları, launch dosyaları, zamanlayıcılar ve ROS tarafından sağlanan diğer hizmetler gibi çeşitli araçları da kullanmanız gerekebilir.



Anonymous

tam anlayamadım.mesela bir ros pakedinin içerisindeki src klasöründeki her cpp dosyası bir node mudur



ChatGPT

Evet, doğru. Bir ROS paketinin içindeki `src` klasöründeki her C++ dosyası, genellikle bir ROS düğümünü temsil eder. Her dosya genellikle bir düğümü ifade eder ve belirli bir görevi gerçekleştirmek üzere tasarlanmıştır.

Her ROS düğümü genellikle bir veya birkaç konu (topic) yayını ve aboneliği içerir veya belirli servislere erişim sağlar. Bu dosyalar, genellikle düğümün işlevselliğini belirleyen C++ kodunu içerir. Aynı zamanda, `CMakeLists.txt` ve `package.xml` dosyaları, ROS paketinin nasıl derleneceğini ve bağımlılıklarını nasıl yöneteceğini tanımlar.

İşte tipik bir ROS paketinin iç yapısının örnek bir gösterimi:

lua

Copy code


```
my_package/  
|-- src/  
|   |-- node1.cpp  
|   |-- node2.cpp  
|-- CMakeLists.txt  
|-- package.xml
```

- node1.cpp ve node2.cpp dosyaları, her biri bir ROS düğümünü temsil eder.
- CMakeLists.txt, ROS paketinin derlenmesi için gerekli talimatları içerir.
- package.xml, paketin ROS'taki diğer paketlerle olan bağımlılıklarını ve paketle ilgili diğer meta bilgileri içerir.

Her bir .cpp dosyası, main fonksiyonunu içerir ve genellikle `ros::init()`, düğüm oluşturma, konu yayını ve abonelikleri, servis oluşturma veya servis taleplerini karşılama gibi temel ROS fonksiyonlarını içerir.

Örneğin, node1.cpp şu şekilde olabilir:

```
cpp Copy code  
  
#include "ros/ros.h"  
#include "std_msgs/String.h"  
  
int main(int argc, char **argv) {  
    ros::init(argc, argv, "node1");  
    ros::NodeHandle nh;  
  
    ros::Publisher pub = nh.advertise<std_msgs::String>("my_topic");  
  
    // Düğüm işlemleri burada yapılır  
  
    ros::spin();  
  
    return 0;  
}
```

Ve node2.cpp şu şekilde olabilir:

```
cpp Copy code  
  
#include "ros/ros.h"
```

```
#include "std_msgs/String.h"

void callbackFunction(const std_msgs::String::ConstPtr& msg) {
    // Mesaj işleme burada yapılır
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "node2");
    ros::NodeHandle nh;

    ros::Subscriber sub = nh.subscribe("my_topic", 10, callbackFu

    // Düğüm işlemleri burada yapılır

    ros::spin();

    return 0;
}
```

Bu dosyalar, iki ayrı ROS düğümünü temsil eder ve belirli bir konuda veri paylaşımı yapabilirler.



Anonymous

peki bana detaylı olacak şekilde düğümleri anlat



ChatGPT

Tabii ki, ROS düğümleri (nodes) ROS ekosisteminde önemli bir rol oynarlar. Her bir düğüm, bağımsız olarak çalışabilen ve diğer düğümlerle iletişim kurabilen birimleri temsil eder. İşte ROS düğümleri hakkında daha ayrıntılı bir açıklama:

1. Düğüm Oluşturma ve Başlatma:

- ROS düğümleri, `ros::init()` fonksiyonuyla başlatılır. Bu fonksiyon, ROS sistemini başlatır ve düğümünüzü tanımlar. Ayrıca, düğümün adını ve benzersiz kimliğini belirler.
- Örneğin:

cpp

Copy code

```
#include "ros/ros.h"
```

```

int main(int argc, char **argv) {
    // ROS sistemini başlat
    ros::init(argc, argv, "my_node");

    // ROS düğümü oluştur
    ros::NodeHandle nh;

    // Düğüm işlemleri burada yapılır

    return 0;
}


```

2. Konu (Topic) Yayını ve Abonelik:

- ROS düğümleri, konu (topic) tabanlı bir iletişim modelini kullanarak birbirleriyle etkileşimde bulunurlar. Konular, düğümlerin asenkron olarak veri paylaşmasına izin verir. Yayın (publish) yapan düğümler, belirli bir konuya mesaj gönderir. Abone (subscribe) olan düğümler, o konuyu dinler ve gelen mesajları işler.

Örneğin:

cpp

 Copy code

```

#include "ros/ros.h"
#include "std_msgs/String.h"

void callbackFunction(const std_msgs::String::ConstPtr& msg) {
    // Mesaj işleme burada yapılır
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "my_node");
    ros::NodeHandle nh;

    // Yayın oluştur
    ros::Publisher pub = nh.advertise<std_msgs::String>("my_top

    // Abone ol

```

```

ros::Subscriber sub = nh.subscribe("my_topic", 10, callback

// D ğ m iřlemleri burada yapılır

ros::spin(); // Etkileřimli d ğ mlerde kullanılır

return 0;
}

```

Servisler (Services):

3. D ğ mler, belirli g revleri yerine getirmeleri i in servis  ağrılarını yapabilirler. Servis  ağrısı yapan d ğ m, belirli bir hizmete talepte bulunur ve bir cevap alır.

 rneğ n:

```

cpp
Copy code

#include "ros/ros.h"
#include "std_srvs/Trigger.h"

bool myServiceCallback(std_srvs::Trigger::Request& req, std_srvs::Trigger::Response& res) {
    // Servis talebi karřılandığında yapılacak iřlemler
    res.success = true;
    res.message = "Servis  ağrısı bařarıyla tamamlandı";
    return true;
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "my_node");
    ros::NodeHandle nh;

    // Servis oluřtur
    ros::ServiceServer service = nh.advertiseService("my_service", myServiceCallback);

    // D ğ m iřlemleri burada yapılır

    ros::spin(); // Etkileřimli d ğ mlerde kullanılır

```

```
return 0;
}
```

Parametreler:

4. D    mler,   alı ma zamanında belirli parametreleri alabilir ve bu parametreleri kullanabilirler. Bu parametreler, d    m  n davranı ını yapılandırmak i in kullanılır.  rne in:

```
cpp Copy code
ros::NodeHandle nh;
int my_param;
nh.getParam("my_param", my_param);
```

ROS Hata Ayıklama:

5. D    mler  zerinde hata ayıklama yapmak i in roscpp k t phanesinin sa ladığı ara ları kullanabilirsiniz. ROS_ERROR, ROS_WARN, ROS_INFO ve ROS_DEBUG gibi makrolar, farklı hata d zeylerinde mesajlar yazmanıza olanak tanır.

Bu  rnekler, temel ROS d    m konseptlerini i ermektedir. Ger ek bir projede, bu temel konseptlere ek olarak, zamanlayıcılar, ROS Action'ları, dinamik yeniden yapılandırma,  oklu konu yayını ve servis  a rırları gibi daha geli mi  konseptleri de   renmek  nemlidir.

Anonymous

rosun genel yapısını bir analogi ile anlat

ChatGPT

ROS'un genel yapısını anlamak i in bir ofis veya i yeri analogisi kullanabiliriz.

-   yeri (Workspace):**   yeri, ROS projesinin genel yapısını temsil eder. Bu, projenin ana klas r  veya dizini olabilir. Bu alanda projeye ait bir ok  eyi bulabilirsiniz.
- Birim (Package):** Birim, projeyi olu turan ba ımsız mod llerdir. Her bir birim, projenin bir par asını temsil eder ve ba ımsız olarak geli tirilebilir, test edilebilir ve kullanılabilir.  rne in, bir ofiste her birim bir departman olabilir.
- Ki i (Node):** Ki i, birim i indeki bireyleri temsil eder. Bu bireyler, belirli bir g revi yerine getirmek  zere bir araya gelirler.  rne in, bir departmanda bir ki i bir belge  zerinde

çalışabilir.

4. **Konu (Topic):** Konu, kişiler arasında bilgi paylaşımını sağlayan ortamları temsil eder. Bu, bir kişinin diğer birime bir bilgiyi bildirmesi veya birime bilgi alması anlamına gelir. Örneğin, bir konu toplantı odası beyaz tahtası olabilir.
5. **Yayın (Publish) ve Abone (Subscribe):** Yayın ve abone, kişilerin birbirleriyle iletişim kurmasını sağlayan yöntemleri temsil eder. Yayın yapan kişi, belirli bir konuda bilgiyi paylaşır ve bu bilgiyi alan kişi, bu bilgiyi dinler. Örneğin, bir kişi bir belgeyi yayınlarken, diğer kişi bu belgeyi dinler.
6. **Toplantı Odası (Servis):** Toplantı odası, belirli bir görevin gerçekleştirilmesi için kullanılan özel bir alanı temsil eder. Kişiler bu odada toplanabilir, bir konuyu tartışabilir ve bir hizmet talebinde bulunabilir. Örneğin, bir servis çağrısı bir toplantı odasına benzetilebilir.
7. **Beyanname (Launch File):** Beyanname, projenin belirli bir durumunu tanımlayan ve belirli düğümleri başlatan bir dosyayı temsil eder. Örneğin, sabah birimdeki kişilerin nasıl başlatılacağını ve hangi konular arasında iletişim kuracaklarını tanımlayan bir sabah toplantısı olabilir.

Bu analogi, ROS'un genel yapısını anlamak için bir başlangıç noktası olarak

kullanılabilir. ROS'un daha karmaşık konseptlerini keşfetmeye başladıkça, bu analoginin sınırlarını aşacak ve daha spesifik özellikleri anlayacaksınız.



Anonymous

topic i tam anlamadım



ChatGPT

ROS'ta bir "konu" (topic), düğümler arasında veri paylaşımını sağlayan bir iletişim mekanizmasıdır. Temel olarak, bir konu, bir düğüm tarafından yayınlanan verileri (mesajları) almak isteyen bir veya daha fazla düğüm tarafından dinlenebilir.

İşte konuların temel özellikleri:

1. **Yayın (Publish) ve Abone (Subscribe):**
 - Bir düğüm, belirli bir konu üzerine mesajlar yayınlatabilir (publish), ve diğer düğümler bu konuyu dinleyebilir (subscribe).
 - Yayın yapan düğüm, belirli bir konuya mesajları düzenli aralıklarla gönderir. Abone olan düğümler ise bu konuyu dinleyerek gelen mesajları işler.
2. **Mesaj (Message) Yapısı:**
 - Konular, belirli bir yapıya sahip olan mesajlar aracılığıyla veri alışverişi yaparlar. Bu mesajlar, önceden belirlenmiş bir veri tipine sahiptir ve bu veri tipi, belirli bir konuda paylaşılacak olan bilginin formatını belirler.

- Örneğin, bir "String" türünde konu, metin verilerini içeren mesajları paylaşabilir.

3. Asenkron İletişim:

- Konular, düğümler arasında asenkron (zaman bağlamında eşzamansız) bir iletişim sağlar. Yayın yapan düğüm, mesajları yayınladığı anlamına gelir ve bu mesajları dinleyen diğer düğümler, kendi hızlarında ve zamanlamalarında bu mesajlara erişebilirler.

Genel Kullanım Alanları:


4. Konular, genellikle sensör verileri, robot durumu, görüntü verileri, komutlar gibi sürekli akış halindeki verilerin paylaşımında kullanılır.
Örneğin, bir sensör düğümü, çevresindeki ortamın verilerini bir konu üzerinden yayınlabilir ve bu verileri işleyen bir başka düğüm, bu konuyu dinleyerek çevresel bilgileri kullanabilir.

Kanal Adlandırma:

5. Her konu, bir ad (channel name) ile tanımlanır. Bu ad, konuyu benzersiz bir şekilde tanımlar. Konu adları genellikle birer string şeklinde ifade edilir.
Örneğin, bir "imu_data" adlı konu, bir inertial ölçüm birimi (IMU) tarafından üretilen sensör verilerini içerebilir.

İşte basit bir örnek:

cpp

 Copy code

```
#include "ros/ros.h"
#include "std_msgs/String.h"

int main(int argc, char **argv) {
    // ROS sistemini başlat
    ros::init(argc, argv, "publisher_node");

    // ROS düğümü oluştur
    ros::NodeHandle nh;

    // "my_topic" adlı bir konu oluştur (String türünde mesajları)
    ros::Publisher pub = nh.advertise<std_msgs::String>("my_topic");

    // Düğüm işlemleri burada yapılır

    // "Hello, ROS!" mesajını oluştur
    std_msgs::String msg;
```


```
msg.data = "Hello, ROS!";

// Mesajı "my_topic" konusuna yayınla
pub.publish(msg);

// ROS'u çalıştır
ros::spin();

return 0;
}
```

Bu örnekte, `my_topic` adlı bir konu oluşturulmuş ve "Hello, ROS!" mesajı bu konuya yayınlanmıştır. Diğer düğümler, bu konuyu dinleyerek gelen mesajları alabilirler.

 **Anonymous**
düğüm

 **ChatGPT**

ROS'ta bir "düğüm" (node), bağımsız olarak çalışabilen bir yazılım birimidir ve genellikle belirli bir görevi yerine getirmek üzere tasarlanmıştır. Düğümler, bir ROS sisteminin temel yapı taşlarıdır ve genellikle bir veya birkaç düğüm bir araya gelerek bir robot veya başka bir otomasyon sistemini kontrol eder.

İşte ROS düğümleri hakkında temel bilgiler:

1. **Bağımsızlık:**

- Her ROS düğümü, genellikle belirli bir görevi gerçekleştiren bağımsız bir işlemi temsil eder. Örneğin, bir düğüm sensör verilerini okuyabilir,

 **Anonymous**

düğüm ile topic arasındaki bağlantıyı anlamadım

 **ChatGPT**

Düğümler (nodes) ve konular (topics) arasındaki bağlantıyı daha iyi anlamak için şu adımları düşünebiliriz:

1. **Düğüm Oluşturma:**

- ROS'ta bir düğüm, belirli bir görevi yerine getirmek üzere tasarlanmış bağımsız bir yazılım birimidir.
- Örneğin, bir sensör okuma düğümü, bir hizmet sunan düğüm veya bir hareket kontrol düğümü olabilir.

2. Veri Paylaşımı:

- Düğümler arasındaki iletişim, genellikle konu (topic) tabanlı bir yapı kullanılarak sağlanır. Bir düğüm, belirli bir konuya mesajlar gönderir (yayar), diğer düğümler ise bu konuyu dinleyerek (abone olur) gelen mesajları alır.

Konu (Topic):

3. Bir konu, belirli bir konsept etrafında veri alışverişini sağlayan bir iletişim kanalını temsil eder.
Örneğin, bir sensörün okuma değerlerini içeren bir "sensor_data" konusu olabilir.

Yayın (Publish) ve Abone (Subscribe):

4. Düğümler, belirli bir konuda mesajları yayınlar (publish) veya bu konuda mesajları dinler (subscribe).
Örneğin, bir sensör düğümü, "sensor_data" konusuna sensör verilerini yayınlayabilir ve bir kontrol düğümü bu konuyu dinleyerek bu verileri alabilir.

Örnek Senaryo:

5. Bir sensör düğümü, çevresel ışık düzeyini ölçen bir sensörden veri alır.
 - Bu sensör düğümü, "light_data" adlı bir konu oluşturur ve bu konuya çevresel ışık düzeyini içeren mesajları yayınlar.Başka bir düğüm, örneğin bir kontrol düğümü, "light_data" konusunu dinleyerek gelen ışık verilerini kullanabilir.

Bu bağlamda, düğümler belirli görevleri yerine getirir ve bu görevleri gerçekleştirmek için diğer düğümlerle veri paylaşımı yaparlar. Bu veri paylaşımı, belirli konular aracılığıyla gerçekleşir. Her konu, belirli bir veri türüne (mesaj tipine) ve bir isme sahiptir, ve düğümler bu konular üzerinden birbirleriyle iletişim kurarlar.



Anonymous

topic ler package klasor hiyerşisinde nerede yer alırlar



ChatGPT

ROS paketleri genellikle bir belirli bir amaca hizmet eden dosya ve klasörlerin bir koleksiyonudur. Konular (topics) ise bu paketlerin içinde genellikle src (source) veya msg (message) klasörlerinde yer alırlar.

İşte tipik bir ROS paketi hiyerarşisinin örnek bir gösterimi:

lua

Copy code

```
my_package/  
|-- src/  
|   |-- node1.cpp  
|   |-- node2.cpp  
|-- msg/  
|   |-- MyMessage.msg  
|-- CMakeLists.txt  
|-- package.xml
```

Bu örnekte:

- `src` klasörü, ROS düğümlerinin (nodes) C++ kaynak kodlarını içerir.
- `msg` klasörü, ROS mesaj türleri tanımlayan `.msg` dosyalarını içerir. Bu dosyalar, genellikle bir konu (topic) için kullanılacak olan mesaj formatını tanımlar.
- `CMakeLists.txt`, ROS paketinin derleme talimatlarını içerir.
- `package.xml`, paketin ROS bağımlılıklarını ve diğer meta bilgilerini içerir.

Eğer bir ROS paketinde konuları kullanmak istiyorsanız, genellikle bu konuları tanımlayan `.msg` dosyalarını `msg` klasöründe yerleştirmeniz gerekir. Mesaj tanımlama dosyalarını tanımladıktan sonra, bu mesajları kullanacak olan düğümleri (`src` klasöründe yer alan C++ dosyalarınızı) yazabilirsiniz. Örneğin, bir `.msg` dosyası şu şekilde olabilir:

csharp

Copy code

```
# MyMessage.msg  
string name  
int32 age
```

Bu dosya, "MyMessage" adında bir mesaj tipi tanımlar ve bu mesaj tipi içinde bir isim (string) ve bir yaş (integer) alanı bulunur. Bu mesaj tipini kullanarak bir konu tanımlayabilir ve düğümler arasında bu konu üzerinde veri paylaşımı yapabilirsiniz.



Anonymous


srv klasoru neden vardır

`.srv` klasörü, ROS paketlerinde servis (service) tanımlamak için kullanılır. Servisler, bir ROS düğümünden diğerine özel görevler gerçekleştirmesi için talepte bulunma ve bu taleplere cevap verme mekanizmasını sağlarlar. Servisler, genellikle bir düğümün başka bir düğüme belirli bir işlemi gerçekleştirmesi için bir talepte bulunmasını ve bu işlemin sonuçlarını almasını sağlar.

`.srv` klasörü içinde tanımlanan `.srv` (service) dosyaları, servislerin giriş ve çıkış parametrelerini tanımlar. Bu dosyalar, belirli bir servisin talep ve cevap mesaj yapısını belirler.

Örnek bir `.srv` dosyası şu şekilde olabilir:

yaml

 Copy code

```
# AddTwoInts.srv
int64 a
int64 b
---
int64 sum
```

Bu örnekte, `AddTwoInts` adında bir servis tanımlanmıştır. Bu servis, iki tane 64-bit tamsayı parametre alır (a ve b), ve bir sonuç olarak bir tane 64-bit tamsayı (sum) döndürür. `.srv` dosyalarını tanımladıktan sonra, bu servisi çağıran ve bu servise cevap veren C++ kodları `src` klasörü içinde yazılır. Ayrıca, `CMakeLists.txt` dosyasında servisleri tanımlamak için gerekli eklemeler yapılır. Bu adımlar, servisleri kullanmak üzere ROS paketinizi konfigüre etmenizi sağlar.

Servisler, özellikle robot kontrolü, algılama veya diğer özel görevlerin gerçekleştirilmesi gibi senaryolarda kullanılır. Bir düğüm, diğer bir düğüme bir servis talebinde bulunarak belirli bir işlemi gerçekleştirmesini sağlar ve bu işlemin sonuçlarını alır. Bu, düğümler arasında etkileşimli ve belirli görevlere odaklanmış bir iletişim sağlar.