ARM24 ws dökümantasyonu

Atahan Yenipınar, Aktuğ İşbilen, Şeyma Gürbüz

5 Ağustos 2025

Özet

Moviet workpace'in detaylı ve açıklayıcı dökümantasyonu.

1 Giriş

Ws içindeki tüm klasörlerin, dosyaların içerikleri ve işlevlerine değinilecektir. Geçmiş yıllardan beri kullanıldığı ve eklemeler yapıldığı için karmaşayı engellemek için gerekli gereksiz tüm dosyalar incelenecektir.

2 arm24_ws Kurulumu

1. Terminalde aşağıdaki komutu çalıştırarak veya GitHub üzerinden zip dosyasını indirerek workspace'i kurun: `git clone https://github.com/itu-rover/arm24_ws`

2. Workspace içerisindeki `build`, `devel` ve `log` klasörlerini silin.

3. Workspace'i derlemek için `catkin_build` veya `catkin_make` komutlarını kullanın. Bu işlem, ortamı kendi bilgisayarınıza uygun şekilde kurmak için gereklidir.

4. Build işlemi muhtemelen eksik paketler nedeniyle başarıyla tamamlanamayacaktır. Çıktı mesajlarından eksik paketleri tespit edip yükleyin.

5. Eksik paketleri yükledikten sonra bu sefer `build`, `devel` ve `log` klasörlerini silmeden workspace'i tekrar build edin.

3 Launch

Genellikle ROS üzerinde bir proje geliştirilirken, birden fazla node'un, parametre dosyasının ve yapılandırmanın aynı anda çalıştırılması gereken durumlar olur. Küçük projelerde veya tekrar tekrar aynı anda çağırılması gerekmeyen dosya tiplerinde bu durum çok büyük sıkıntı yaratmayabilir; ancak büyük projelerde, hele ki aynı dosyaların düzenli olarak çağırılması gerektiği durumlarda, bu dosyaların tek tek elle açılması can sıkıcı bir hal alır.

Launch dosyaları tam olarak bu sorunu çözen, XML formatında dosyalardır. Simülasyon ortamlarını, kendi yazdığınız kodları, ROS'un kendisinde bulunan *default* node'ları hatta farklı launch dosyalarını, launch dosyaları ile tek bir launch dosyasını çalıştırarak çalıştırabilirsiniz. En çok kullanılan etiketler `<arg>`, `<node>`, `<rosparam>`, `<param>`, `<include>` ve `<group>` etiketleridir.

`arg` Launch dosyalarında değişken atamak için kullanılır. Farklı bir launch dosyası çağırılırken o dosyaya bu değişkenleri göndermek için de kullanılabilir. Kimi zaman değişken adıyla değer adının aynı atandığı görülebilir. Bu durum değişken değerinin mevcut launch dosyasına çağırıldığı launch dosyasından atanmaması takdirinde default değerinin kendisine eşit olması içindir.

`node` ROS depolarında halihazırda bulunan node'ları veya bizzat workspace'in yazarları tarafından hazırlanan Python kodu veya C++ halinde bulunan node'ları çalıştırmak için kullanılan etikettir.

`rosparam` Node'ların çalıştırılması sırasında belli parametreleri atamak için kullanılır. Doğrudan doğruya ROS'ta halihazırda bulunan parametreleri değiştirmek için kullanılır. Genellikle bu parametreler fazla olabileceğinden ve kodun daha düzensiz durmasını sağlayabileceğinden, `yaml` dosyaları ile birlikte kullanılır.

`param` Belli bir parametreler listesini değil de tek bir parametreye değer atamak için kullanılır.

`include` Diğer launch dosyalarını çağırmak için kullanılır.

`group` Launch dosyalarını daha düzenli hale getirmek, aynı işlev için kullanılan parametrelerin, node'ların ve launch dosyalarının bir arada kullanılması için kullanılır. Dilenirse içinde `if` tarzı koşullar kullanılarak, `group` içerisindeki bütün olayların belli bir durumda çalıştırılması daha kolayca sağlanır.

4 URDF

Unified robot description file. URDF dosyaları, robotikte robotun eklemlerini, uzantılarını ve uzantılar ile eklemlerin simüle edilmesi ve kinematik hesapların yapılması için gerekli bilgileri içeren dosyalara verilen isimdir. Temelde XML formatında yazılan bu dosyalar, ana `robot` etiketi içerisinde iki temel etiket içerir: `joint` ve `link`.

4.1 Joint Etiketi

`joint` etiketi içerisinde;

- `<origin>`: Eklemin konumunu (`xyz` attribute'u ile) ve eklemin hangi eksenlerde kaç derece dönebildiğini (`rpy` attribute'u ile) belirtir.
- `<parent>` ve `<child>`: Eklemin hangi `link`leri birbirine bağladığını belirten etiketlerdir.
- `<axis>`: Aslında `<origin>` etiketindeki `rpy` parametresi eklemin hangi açılarda kaç derece dönebildiğini gösterir; fakat `<axis>` etiketi ile hangi yönün `joint` tarafında hareket edilebilir olduğu bool değişkenlerle tekrar ifade edilir.
- `<limit>`: Bazı eklemlerin fiziksel sebeplerle (örneğin uzantının robotun gövdesine çarpması gibi) hareket aralığı kısıtlanır. Bu etiketle eklemlerin hangi aralıkta hareket edebildiği gösterilir. Kullanımı zorunlu değildir.

4.2 Link Etiketi

`link` etiketi içerisinde temelde üç adet alt etiket vardır: `<inertial>`, `<visual>` ve `<collision>`.

- `<inertial>`: Linkin kütle merkezi (`origin` etiketi ile), kütlesi (`mass` etiketi ile) ve farklı yönlerde dönmeye karşı uyguladığı direnci (`inertia` etiketi ile) ifade eden etikettir.
- `<visual>` ve `<collision>`: `visual` etiketi linkin simülasyon içerisindeki görüntüsünün ayarlamalarının yapılması için gerekli iken, `collision` etiketi çarpışma hesaplamalarının yapılması için gerekli parametrelerin belirtildiği etikettir. Her iki etikette de linkin modeli `<geometry>` etiketi içerisindeki `<mesh>` etiketi ile yüklenecek üç boyutlu modelin yolu verilir.

Kullanılan 3 boyutlu dosyalardaki toplam (tüm robottaki) üçgen (triangle) sayısı simülasyonun optimize olması için 150-200 bini geçmemelidir. Modellerin optimize olmaması halinde MeshLab, Blender gibi programlardan optimizasyon yapılabilir. Genelde optimizasyon açısından `collision` meshlerinin daha detaysız olması tercih edilir. Hatta bazı durumlarda doğrudan mesh tanımlamak yerine `box`, `cylinder`, `sphere` gibi etiketlerle basit sınırlar tanımlanır. Diğer etiketlerde olduğu gibi `origin` etiketi ile modelin tam konumu verilir. `visual` etiketinde `material` etiketi ile `color` ve `texture` da belirlenebilir.

4.3 Naim23_urdf

`naim23_urdf`, rover aracımızın kullandığı ana URDF dosyasıdır. İçerisinde 9 `link` ve 8 `joint` tanımlıdır. `joint`lerden biri hareketsiz merkez `joint`i iken, biri gripperı ifade eder. `link`lerden biri rover'ın gövdesi, biri robot kolun çıkış yeri, biri ise `end_effector` yani gripperi ifade eder. Geri kalan `joint` ve `link`ler robot kol içindir; dolayısıyla robot kolumuz 6 eksenden oluşur.

5 SRDF

Semantic robot description file. Robotun hangi kısımlarının hareket ettirilebilir olduğu, hangi kısımlarının doğası gereği çarpışamaz olduğu ve robotun belli bir isme atanmasının kolaylık sağlayacağı pozisyon bilgilerinin gösterildiği XML formatındaki belgelerdir. URDF dosyalarının tamamlayıcısı olmakla beraber, onların yerine geçemez. Robot kolun simülasyon ortamında ve gerçek hayatta çalışabilmesi için her iki dosyanın da workspace içerisinde mevcut olması gerekmektedir. Pek çok bilgiyi içeren pek çok etiket kullanılabilir olsa da, bizim arm24_ws içerisinde kullandığımız naim23_urdf.srdf dosyasında kullanılmış üç ana etiket bulunmaktadır: `<group>`, `<group_state>`, `<disable_collisions>`. Srdf'in Urdf'den farkı kısaca bahsetmek gerekirse, urdf'in simülasyon programlarına entegre edilmiş hali gibi düşünebiliriz.

5.1 group

`group`: Robot kolun hareket ettirilebilir kısmının tanımlandığı ve isimlendirildiği etikettir. İçerisinde bulunan `<chain>` etiketi ile robot kolun hareket etmesi istenen kısmının hangi link ile başladığını ve hangi link ile bittiğini ifade eder.

5.2 group_state

`group_state`: İçerisinde bulunan `<joint>` etiketleri ile kullanılan jointler için radyan cinsinden bir açı atanır. Var olan bütün jointlere bir açı değeri verilerek robot kol için özel bir pozisyon oluşturulur. `group_state` etiketinin kendi içerisinde isimlendirilen bu özel durum, gerektiğinde launch dosyalarında parametre atamak için kullanılan YAML dosyalarında başlangıç pozisyonu olarak ayarlanabilir veya RViz içerisinde ismi verilen pozisyona bir buton atanarak doğrudan simüle edilebilir. Örneğin bizim SRDF dosyamızda bulunan "home" isimli `group_state`'in joint değerleri ile oynayarak, robot kolumuzun RViz üzerindeki başlangıç pozisyonu ile oynayabilirsiniz.

5.3 disable_collisions

`disable_collisions`: Çarpışması durumunda program tarafından görmezden gelinmesi istenen link ve jointleri ifade etmek için kullanılır. Bunun istenmesinin pek çok sebebi olabilir ancak temelde iki sebebi vardır: Belirtilen linklerin ve jointlerin zaten birbirine bağlı olan, art arda gelen jointler ve linkler oluşu; belirtilen joint ve linklerin birbirine dokunmasının zaten matematiksel olarak mümkün olmaması. Etiketin kullanılmasının sebebi `reason` parametresi ile ifade edilir.

6 arm23_config

Bu paket, workspace'teki genel konfigürasyon ayarlarının yapıldığı ve robotun simülasyonda çalışması için gerekli olan dosyaların bulunduğu pakettir.

6.1 demo_gazebo.launch

`demo_gazebo.launch`, RViz ve Gazebo simülasyon ortamlarını açan ana `launch` dosyasıdır. Dosya ilk çalıştırıldığında RViz açılır. RViz açıldıktan sonra **space** tuşuna basılırsa Gazebo da açılır.

Temelde bu `launch` dosyası içerisinde iki ayrı `launch` dosyası çağırılır:

- **gazebo.launch**: Gazebo'nun açılması için gerekli dosyaların çağrıldığı `launch` dosyasıdır. `empty_world.launch` dosyası ile boş bir Gazebo ortamı oluşturulur. Simülasyonun başta kapalı olup **space** tuşu ile tekrar açılmasının sebebi, bu dosyada `pause` parametresinin açık olarak atanmış olmasıdır. Ayrıca, `ros_controllers.launch` dosyası ile diğer simülasyon ayarlarını yapacak dosyalar da çağrılır.
- **demo.launch**: RViz, temelde bu `launch` dosyası ile çağrılır. Başta atanan bir grupta robot simülasyon ortamında çalıştırılırken gerekli olan `node`'ların ve parametrelerin yüklenmesi sağlanır. Bu dosya, üç ayrı `launch` dosyasını daha çağırır:
  - `move_group.launch`: Robot kolun MoveIt ayarlarının yapıldığı ve gerekli hesaplama `node`'larının çağrıldığı dosyadır.

- `moveit_rviz.launch`: RViz'in, MoveIt yapılandırmasıyla uyumlu şekilde başlatılmasını sağlar.
- `default_warehouse_db.launch`: Veritabanı yönetimini sağlayan `launch` dosyasıdır.

fake_controllers.yaml: `demo.launch` dosyasında çağrılan bu `YAML` dosyası, simülasyon ortamı için gerekli ayarların yapılmasını sağlar. `initial` altındaki `pose:` parametresi, SRDF'te tanımlanmış belirli pozisyonlardan biri olmalıdır. Bu değişken, robotun simülasyon başlatıldığındaki başlangıç pozisyonunu belirler.

7 arm23_control

7.1 Amaç

`arm23_control` paketi, robot kolun donanım arayüzü ve temel kontrol işlevlerini sağlar; robot kolun kullanılmasını sağlayan ana pakettir. Robot kol motorları ile ROS arasında haberleşme altyapısını yönetir ve donanım kontrolcülerini çalıştırır.

7.2 Dosya Yapısı

```
arm23_control/
 CMakeLists.txt
 config/
    controllers.yaml       % Kontrolcü ayarları
    fake_controllers.yaml  % Simülasyon için sahte kontrolcüler
 include/arm23_control/
    arm23_hw_interface.h   % Donanım arayüzü sınıf tanımı
 launch/
    control_hw.launch      % Donanım arayüzü ve controller manager başlatma
    control_tel.launch     % Teleop için launch (Joystick vs.)
 rviz/
    rk_rviz.rviz           % RViz konfigürasyonu
 script/
    rk23_serial.py         % Seri port haberleşme Python node'u
    çeşitli test scriptleri
 src/
    arm23_hw_interface.cpp % Donanım arayüzü implementasyonu
    arm23_hw_main.cpp      % Donanım arayüzü node main dosyası
 package.xml
```

7.3 Önemli Not

Not: Moteus hakkındaki kodlar ve launch dosyası hala workspace içerisinde yer almakla beraber artık kullanılmamaktadır.

7.4 Launch Dosyaları ve Kullanım Sırası

Robot kolun kontrolünü başlatmak için tipik çalışma adımları:

1. `roslaunch arm23_control control_hw.launch`

   Robot kolumuz temel olarak üç ana launch dosyası kullanılarak çalıştırılır: `control_hw.launch`, `control_tel.launch` ve `joy_launch`.

   `control_hw.launch` dosyası çalıştırıldığında aşağıdaki işlemler gerçekleştirilir:

   - Robotun URDF dosyası parametre sunucusuna yüklenir.
   - Gömülü sistemle iletişim kuran ve donanım arayüzünü yöneten `arm23_hw_interface` node'u başlatılır.

- Robotun başlangıç pozisyonu ve kontrol parametrelerinin tanımlı olduğu `controllers.yaml` dosyası yüklenir. Eğer başka kontrolcüler de kullanılmak isteniyorsa bu yaml dosyasına tanımlanması gerekmektedir.
- Planlama ve yürütme işlevlerini sağlayan MoveIt'e ait `move_group.launch` dosyası dahil edilir.

2. `roslaunch arm23_control control_tel.launch`

   Teleoperasyon (joystick vb.) için node başlatılır. Bu node, `arm23_teleop` paketinden çağırılan `new_teleop.py` node'udur. Robot kolun manuel kontrolü bu aşamada sağlanır. Aynı zamanda `rk23_serial.py` node'u da bu launch dosyasında çağırılmaktadır.

7.5 Dosyaların Kısa Açıklamaları ve Donanım Haberleşme Yapısı

- `controllers.yaml`
  - **Görevi:** ROS kontrol altyapısı için gerekli kontrolcü konfigürasyonlarını sağlar.
  - **İçerik:**
    - `joint_state_controller`: Eklemlerin mevcut pozisyonlarını yayınlar.
    - `manipulator_controller`: Eklemlere pozisyon komutu uygular ve trajektori takibi yapar.
  - **Frekans:** 300 Hz kontrol frekansı ile çalışır.
- `arm23_hw_interface.cpp` Robot kolun gömülü sistemle iletişimini sağlayan ana C++ implementasyonudur. ROS ile robot donanımı arasında köprü görevi görür.
  - **Görevi:** Donanım soyutlama katmanını oluşturarak ROS kontrol altyapısı ile donanım arasında veri alışverişini sağlar.
  - **read():** `joints_feedback` topic'i üzerinden gelen verileri `feedback()` fonksiyonu ile işler.
  - **write():** ROS kontrolcülerinden gelen hedef pozisyon komutlarını `joint_commands` topic'ine yayınlar.
  - **reset_srv():** Robot kolun referans sıfır pozisyonunu günceller.
  - **enforceLimits():** Eklem hareket sınırlarını kontrol eder, aşımı engeller.
  - **Not:** Donanımla doğrudan seri port haberleşmesi içermez; bu görev başka bir node tarafından yürütülür.
- `arm23_hw_main.cpp`
  - **Görevi:** `arm23_hw_interface` sınıfını kullanarak donanım döngüsünü başlatır.
  - **Yapısı:** `ros_control_boilerplate` üzerinden node başlatımı, döngü yönetimi ve ROS sistemine entegrasyon sağlar.
- `rk23_serial.py`
  - **Görevi:** Gömülü sistem ile doğrudan seri port haberleşmesini sağlar.
  - **Gelen veri akışı:** Donanımdan alınan pozisyon ve voltaj bilgileri parse edilerek `joints_feedback` topic'ine yayınlanır.
  - **Giden komut akışı:** `joint_commands` topic'inden alınan pozisyon komutları seri port üzerinden donanıma gönderilir (örn. `SxxxxF` formatında).
  - **Kontrol bayrakları:** `ALLOW_WRITE`, `VELOCITY_MODE` gibi değişkenler, haberleşme ve komut gönderme modlarını belirler.
  - **Joystick verisi:** Joystick verileri `pos_command` ve `velocity_command` değişkenlerinde tutulur.
- **ROS Topic Katmanı (Ara Katman)**

- **Görev:** `arm23_hw_interface.cpp` ve `rk23_serial.py` arasındaki veri alışverişini ROS topic'leri üzerinden sağlar.
- **Komut Akışı (Yukarıdan Aşağıya):**
  `MoveIt` → `JointTrajectoryController` → `arm23_hw_interface` → `joint_commands` → `rk23_serial.py` → `Gömülü Sistem`
- **Geri Besleme Akışı (Yukarıya):**
  `Gömülü Sistem` → `rk23_serial.py` → `joints_feedback` → `arm23_hw_interface`

- `rk_rviz.rviz` RViz için kaydedilmiş konfigürasyon dosyasıdır.
  - **Görevi:** Robot kolun görselleştirilmesini ve hareket planlama arayüzlerinin otomatik yüklenmesini sağlar.
  - **İçerik:** Izgara (grid), robot modeli, planlama arayüzleri, kamera açıları gibi RViz bileşenlerinin konfigürasyonları.
  - **Kullanım:** RViz başlatıldığında bu dosya yüklenerek görsel ortam oluşturulur.

Gömülüyle Haberleşme: Gerçek donanım ile **doğrudan seri port haberleşmesi yapan tek katman** `rk23_serial.py` dosyasıdır. C++ tabanlı `arm23_hw_interface.cpp` dosyası yalnızca ROS topic'leri aracılığıyla bu node ile haberleşir. Bu yapı, sistemi hem modüler hem de ölçeklenebilir hale getirir. `rk23_serial.py` dosyası gömülü taraf ile SF mesajları vasıtasıyla haberleşir.

7.6 Özet

`arm23_control` paketi, robot kol donanımını ROS ekosistemine entegre eder. Seri port haberleşme ve donanım arayüzü node'ları sayesinde robot eklemlerinin konum bilgileri alınır ve komutlar motorlara iletilir. `moteus` tabanlı eski sistem artık kullanılmamaktadır; ilgili dosyalar temizlenerek paket güncel ve yönetilebilir tutulmalıdır. Üst seviye kontrol için teleoperasyon ve planlama paketleri ile birlikte çalışır.

8 Arm23 Teleop

Bu proje, bir robot kolun **MoveIt** kullanılarak **joystick ile teleoperasyon kontrolünü** sağlar.
Teleoperation (teleop), bir robotun uzaktan manuel olarak kontrol edilmesidir. Bu kontrol; klavye, joystick veya GUI gibi cihazlarla yapılabilir.

8.1 Proje Yapısı

8.1.1 Klasör Yapısı

```
arm23_teleop/
 src/
    teleop_server.cpp       # C++ ile yazılmış teleop düğümü (MoveIt tabanlı)
 scripts/
    new_teleop.py           # Ana joystick kontrolü (kontrol modları burada)
    gripper_teleop.py       # Gripper joystick kontrolü (seri port üzerinden)
    gripper_v2.py           # UDP tabanlı gripper kontrolü
    ik_solution.py          # Temel IK test dosyası
    ik_solution_modular.py  # Harf bazlı hedef pozisyon kontrolü
    start_usb_cam.py        # USB kamera başlatıcısı
 CMakeLists.txt
 package.xml
```

8.2 Launch Dosyaları

8.2.1 Tanımlar

- **joy.launch**: `joy_node` çalıştırılır. Joystick verileri `/joy` topic'i üzerinden alınır ve `/joy_rk` olarak yeniden yönlendirilir.
- **teleop.launch**: `new_teleop.py` çalıştırılır. Joystick verilerini okuyarak robot kola uygun komutlar üretir.
- **teleop_server.launch**: `teleop_server.cpp` dosyasındaki C++ düğüm başlatılır. Cartesian mod kontrolü sağlar.

8.2.2 Çalıştırma Sırası

[language=bash] 1. Joystick baglanir: roslaunch arm23$_t$*eleopjoy.launch*
2. Teleoperation baslatilir: roslaunch arm23$_t$*eleopteleop.launch*
3. Gerekirse C++ node baslatilir: roslaunch arm23$_t$*eleopteleop$_s$erver.launch*

8.3 Kontrol Modları

8.3.1 new_teleop.py

Joystick yardımıyla robot kolun üç farklı modda kontrolünü sağlar:

Joint Modu (Eklemsel Kontrol)

- Joystick eksenleri ve butonlarına göre her bir eklem için hız değeri belirlenir.
- Bu hızlar `"S123456F"` formatında bir string'e dönüştürülerek gönderilir.
- Her rakam bir joint'i temsil eder (1–6 arası).
  - 5: nötr
  - 1–4: negatif yön
  - 6–9: pozitif yön
- Örnek: `S579135F` → 6 eklemin her biri farklı yön/hızda çalışır.

Cartesian Modu

- Joystick'ten gelen lineer ve açısal hız komutları `TwistStamped` formatında alınır.
- Bu hız komutları uygun topic üzerinden sistemin kontrol bileşenlerine iletilir.
- Gelen hız bilgileri entegre edilerek hedef pozisyon sürekli güncellenir.
- Güncel hedef pozisyona karşılık gelen eklem hızları, ters kinematik ve planlama modülleri tarafından hesaplanır.
- Hesaplanan eklem hızları, kontrolör aracılığıyla robota uygulanarak uç efektör joystick yönünde sürekli hareket ettirilir.

Pose Modu

- Robot kolu, ROS üzerinden gönderilen hedef eklem pozisyonlarına (örneğin MoveIt veya teleop arayüzünden) geçer.
- Her bir eklem için hedef pozisyon (radyan cinsinden) belirlenir ve `joint_commands` topic'i üzerinden iletilir.
- Python arayüzünde (`rk23_serial.py`), bu pozisyonlar belirli aralıklarda (örneğin $[-3.14, 3.14]$) `interp` fonksiyonu ile $[0, 9999]$ aralığına ölçeklenir.
- Her eklem için elde edilen tam sayı değeri string'e çevrilir ve `"SxxxxxxF"` formatında bir SF mesajı oluşturulur (S: başlangıç, F: bitiş, x: her eklem için 0–9999 arası değer).
- SF mesajı seri port üzerinden gömülü sisteme gönderilir; gömülü sistem bu değerleri hedef pozisyon olarak yorumlayıp motorları sürer.
- Gömülü sistemden gelen pozisyon verileri de aynı şekilde paketlenmiş olarak gelir, Python arayüzünde tekrar orijinal aralıklara (`interp` ile) çevrilip ROS'a geri besleme olarak yayınlanır.

8.3.2 gripper_teleop.py

- Joystick kullanılarak gripper'a ait lazer ve mıknatıs gibi fonksiyonlar kontrol edilir.
- Seri port üzerinden `S<val><magnet><laser>F` formatında komutlar gönderilir.
- Ayrıca voltaj verileri alınarak ROS topic'i olarak yayın yapılır.

8.3.3 ik_solution_modular.py

- `dictionary_topic` üzerinden harf karşılığı hedef pozisyonlar alınır.
- Her hedef için MoveIt'in `GetPositionIK` servisi ile ters kinematik çözüm yapılır.
- Hedef pozisyona ulaşıldığında `press_button` sinyali gönderilir.

8.3.4 start_usb_cam.py

- Sistemdeki USB kamerayı başlatır.
- Görüntü işleme ya da kullanıcı arayüzü uygulamaları için yardımcıdır.

8.4 teleop_server.cpp

8.4.1 Görevleri

- MoveIt ile bağlantı kurar (`move_group`, `planning_scene`, `robot_model`).
- Başlangıçta uç efektör pozisyonunu alır ve hedef olarak tanımlar.
- `/servo_server/delta_twist_cmds` topic'inden gelen hız bilgilerini pozisyona entegre eder. Direkt olarak hız komutlarını anlık pozisyonlarına ekleyerek(pseudo integration) hedef pozisyon oluşturur.
- Hedef pozisyon için ters kinematik çözüm yapılır.
- Geçerli çözüm bulunduysa robot eklemleri bu pozisyona yönlendirilir.
- Her yeni pozisyon bir `tf` dönüşümü olarak da yayınlanır (`/world → /target`). Bu sayede anlık olarak kartezyen eksenlerde kontrol edilmiş olur.
- `/servo_server/reset_target` servisi ile mevcut pozisyon tekrar hedef yapılabilir.
- Bu metodun uygulanabilmesi için motor sürücülerin hassas bir şekilde pozisyon kontrol edebilmesi gerekir.

8.4.2 Güvenlik Özelliği

- Hedef eklem açısı ile mevcut açı arasındaki fark belirli bir sınırı aşarsa (`threshold = 0.8 rad`), komut gönderilmez.

8.5 Komut ve Formatlar

8.5.1 S-F Formatı ve Komut Yapısı

- `S` başlangıç karakteridir.
- 6 karakter (1-6. eklemler) her biri bir eklemi temsil eder.
- Her rakam:
  - 5: nötr (hareketsiz)
  - 1–4: negatif yönde hız artışı
  - 6–9: pozitif yönde hız artışı
- `F` bitiş karakteridir.

8.5.2 Mod Geçiş Tuşları

Mod	Joystick Tuşu	Kod Etkisi
IDLE	Sağ yön (axes[6]== 1) + button[6]	`CURRENT_STATE = "IDLE"`
JOINT	Yukarı yön (axes[7]== 1) + button[6]	`CURRENT_STATE = "JOINT"`
POSE	Aşağı yön (axes[7]== -1) + button[6]	`CURRENT_STATE = "POSE"`

8.6 ROS İletişim ve Parametreler

8.6.1 ROS İletişim Haritası

- **Topic:** `/joy_rk` — Joystick verileri alınır.
- **Topic:** `/servo_server/delta_twist_cmds` — Cartesian kontrol mesajları.
- **Topic:** `velocity_commands` — Joint mod komut string'i.
- **Service:** `/servo_server/reset_target` — Hedef sıfırlama servisi.
- **Service:** `/hardware_interface/allow_write` — Yazma izni verilir.
- **Service:** `/hardware_interface/velocity_mode` — Velocity kontrolü.

8.6.2 Launch Parametreleri

- `dev`: Joystick cihazı (ör. `/dev/input/js1`)
- `deadzone`: Küçük joystick hareketlerini yok saymak için eşik değeri.
- `autorepeat_rate`: Tuş basılı tutulduğunda yayınlanma hızı (Hz).
- `coalesce_interval`: Mesajların birleştirilme zaman aralığı.

8.7 Sistem Akış Diyagramı

Joystick → `/joy` → `/joy_rk` → `new_teleop.py` → (mod'a göre) → `velocity_commands` veya `delta_twist_cmds` → `teleop_server.cpp` (IK çözümü) → `/manipulator_controller/command/` → Robot

8.8 Sonuç

Bu yapı ile bir robot kol, joystick yardımıyla **eklemsel**, **cartesian** veya **hazır pozisyon** modlarında etkili şekilde kontrol edilebilir. MoveIt altyapısı sayesinde, ters kinematik çözümler ve çarpışma kontrolleri sistemin esnekliğini artırır.

9 bio_ik Paketi – Kısa Özet

bio_ik, robotlar için ileri düzey *inverse kinematics (IK)* hesaplamalarını hızlı ve esnek şekilde yapan, **ROS tabanlı** bir kütüphanedir. Genellikle *MoveIt* ile birlikte kullanılır ve robotun eklem konumlarını, verilen uç efektör hedeflerine göre otomatik olarak hesaplar.

9.1 Temel Amaç

Robotun uç efektörünü istenen pozisyon ve oryantasyona getirmek için gerekli eklem açılarını bulur. Özellikle çok serbestlik dereceli (DOF) robotlar ve karmaşık kinematik yapılar için uygundur.

9.2 Paket Yapısı ve Dosyalar

Yol / Dosya	İçerik
`include/bio_ik/`	Kütüphanenin genel API'si ve başlık dosyaları.
`src/`	Kütüphanenin asıl kaynak kodları (genellikle değiştirilmez).
`CMakeLists.txt` & `package.xml`	Derleme ve bağımlılık tanımları için gerekli ROS dosyaları.
`README.md`	Kurulum, temel kullanım ve örnek konfigürasyonlar.
`doc/`	Ek dokümantasyon ve görseller (algoritma örnekleri).
`test/` (*varsa*)	Örnek testler.

9.3 Kullanımda Dikkat Edilecekler

Kinematik Ayarları MoveIt konfigürasyonunda `kinematics.yaml` dosyasında solver olarak:

```
right_arm:
  kinematics_solver: bio_ik/BioIKKinematicsPlugin
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.005
  kinematics_solver_attempts: 1
```

seçilmelidir.

Değiştirilebilecek Dosyalar `config/kinematics.yaml` — Robotunuzun kinematik parametrelerini burada ayarlarsınız.

10 inverse_kinematics Paketi Özeti

inverse_kinematics, önceki kullanıcıların yazdığı kodlardan oluşuyor.

10.1 Script Dosyaları (`/script` klasörü)

- `key_teleop.py`: Klavye ile robot kontrolü (`q/a`: X, `w/s`: Y, `e/d`: Z eksenleri)
- `forward.py`: İleri kinematik hesaplamaları için script
- `joint_traject.py`: Eklem yörüngesi planlama ve kontrolü
- `command.py`: Robot komut kontrolü
- `test.py`, `deneme.py`, `lastone.py`: Test ve deneme amaçlı scriptler

10.2 Paket Yapılandırma Dosyaları

- `CMakeLists.txt`: Derleme konfigürasyonu
- `package.xml`: ROS paket bağımlılıkları ve meta verileri

10.3 Önemli Noktalar

- `/keyboard` topic'i üzerinden kontrol komutları iletilir.
- Scriptler test/geliştirme amaçlı olduğundan, kullanırken dikkatli olunmalıdır.
- Çalıştırmadan önce gerekli ROS node'larının aktif olduğundan emin olunmalıdır.

Not: Bu paket, özelleştirilmiş robot kontrolü için yazılmış olup, MoveIt'in hazır IK çözücülerine alternatif veya ek çözümler sunar.

11 fiducials Klasörü ve Alt Paketleri Detaylı Özeti

Bu klasör, marker tabanlı lokalizasyon ve haritalama için birden fazla ROS paketini içerir. Her alt klasörün işlevi ve önemli dosyaları aşağıda özetlenmiştir.

11.1 Ana Klasör: `fiducials/`

11.1.1 Amaç

Marker haritalama, robotun konumunu belirleme, ana algoritmalar ve genel dokümantasyon.

11.1.2 Önemli Dosyalar

- `README.md`: Sistemin genel açıklaması, kullanım ve troubleshooting.
- `LICENSE`, `CHANGELOG.rst`: Lisans ve değişiklik geçmişi.
- `.clang-format`, `.gitignore`: Kod stil ve git ayarları.
- Kamera kalibrasyon dosyaları (ör. `Logitech-C920.yaml`): Marker algılamada kullanılan kameranın parametreleri.

11.1.3 Kullanıcı Tarafından Değiştirilebilecekler

- Kamera kalibrasyon dosyaları.
- Launch/config dosyaları.

11.2 Alt Paket: `aruco_detect/`

11.2.1 Amaç

ArUco marker'larının algılanması ve pozisyonlarının çıkarılması.

11.2.2 Önemli Dosyalar

- `src/aruco_detect.cpp`: Marker algılama, pozisyon ve oryantasyon çıkarımı, ROS topic'lerine veri yayını. Algoritmanın ana kodu.
- `package.xml`, `CMakeLists.txt`: ROS bağımlılık ve derleme ayarları.
- `setup.py` (varsa): Python paket kurulumu.
- `CHANGELOG.rst`: Paket değişiklik geçmişi.

11.2.3 Kullanıcı Tarafından Değiştirilebilecekler

- Algılama hassasiyeti.
- Marker boyutu.
- Ignore listesi gibi parametreler (launch/config dosyaları veya node parametreleri üzerinden).

Not: Algoritma kodu genellikle sabit kalır, parametrelerle özelleştirilir.

11.3 Alt Paket: `fiducial_msgs/`

11.3.1 Amaç

Marker ile ilgili özel ROS mesaj tiplerini tanımlar.

11.3.2 Önemli Dosyalar

- `msg/` klasörü: `Fiducial.msg`, `FiducialArray.msg`, `FiducialTransform.msg` gibi mesaj tanımları.
- `package.xml`, `CMakeLists.txt`: ROS bağımlılık ve derleme ayarları.
- `CHANGELOG.rst`: Paket değişiklik geçmişi.

11.3.3 Kullanıcı Tarafından Değiştirilebilecekler

- Mesaj tanımları.
- Yeni marker tipleri veya ek veri taşımak için mesajlara yeni alanlar eklenebilir.

11.4 Alt Paket: `fiducials/fiducials/`

11.4.1 Amaç

Marker haritalama ve robotun konumunu belirleme algoritmaları ile sistemin başlatılması için gerekli launch dosyaları.

11.4.2 Önemli Dosyalar

- Ana kodlar: Marker'lar arası ilişkilerden harita oluşturur, robotun konumunu hesaplar.
- Launch dosyaları: Sistemi başlatmak için.
- `package.xml`, `CMakeLists.txt`: ROS bağımlılık ve derleme ayarları.
- `CHANGELOG.rst`: Paket değişiklik geçmişi.

11.4.3 Kullanıcı Tarafından Değiştirilebilecekler

- Launch/config dosyaları.
- Haritalama parametreleri.

11.5 Diğer Dosya ve Klasörler

11.5.1 Kamera Kalibrasyon Dosyaları

- Dosya formatı: `*.yaml`.
- Kameranın iç parametreleri, marker algılamanın doğruluğu için önemlidir.

11.5.2 Launch/Config Dosyaları

- Marker boyutu (`fiducial_len`).
- Ignore listesi (`ignore_fiducials`).
- Dictionary seçimi (`dictionary`).

11.5.3 Mesaj Tanımları

- Marker verisinin ROS üzerinden iletilmesini sağlar.

11.6 Kısaca Değiştirilebilecekler

- Kamera kalibrasyon dosyaları (`*.yaml`).
- Launch/config dosyaları (algılama hassasiyeti, marker boyutu, ignore listesi, dictionary seçimi).
- Mesaj tanımları (`fiducial_msgs/msg/`).

Not: Algoritma kodları genellikle sabit kalır. Sistem parametre ve ayar dosyalarıyla özelleştirilir.

11.7 Sonuç

Bu yapı sayesinde, robotun kullandığı kamera veya marker boyutları değişirse sadece ilgili kalibrasyon ve parametre dosyalarını düzenlemek yeterlidir. Kodun büyük kısmı hazırdır ve genellikle değiştirilmez; ihtiyaca göre parametre/ayar dosyaları düzenlenir.

12 ros_control_boilerplate

ros_control_boilerplate, PickNikRobotics tarafından geliştirilmiş, ROS tabanlı robot projelerinde gerçek veya simülasyon ortamlarında donanım arayüzü (`hardware_interface`) ile kontrol altyapısı (`controller_manager`) arasında bağlantıyı sağlayan bir şablon (*boilerplate*) pakettir. Kendi başına bir kontrolcü değildir, ancak ROS'un resmi kontrolcüleri (`JointPositionController`, `JointTrajectoryController` vb.) ile çalışacak şekilde tasarlanmıştır.

Amacı:

- Robotun donanımına erişip eklem durumlarını (`joint states`) okuyarak ROS kontrol altyapısına sunmak.
- Kontrolcülerden gelen komutları (örneğin hedef pozisyonlar) donanıma iletmek.
- Simülasyon ve gerçek donanım arasında hızlı geçiş sağlamak.
- Donanım protokolüne bağımlı olmayan esnek bir şablon sunmak.

Çalıştığı Durumlar:

- **Gerçek robot üzerinde:** Donanımınız kendi protokolü (CANBus, USB-serial, Ethernet) üzerinden kontrol ediliyorsa, `read()` ve `write()` metodları robotunuza uygun hale getirilerek çalıştırılabilir.
- **Simülasyon ortamında:** `sim_control_mode` parametresiyle simülasyon moduna geçerek Gazebo veya başka bir simülasyon altyapısıyla kullanılabilir.
- **Kontrolcü testi:** Gerçek robot olmadan kontrolcü davranışlarını gözlemlemek ve PID parametrelerini ayarlamak için kullanılabilir.

Paket Yapısı ve Dosyalar:

```
include/ros_control_boilerplate/ros_control_boilerplate.h
src/ros_control_boilerplate.cpp
launch/ros_control_boilerplate.launch
config/controllers.yaml
scripts/controller_to_csv
scripts/csv_to_controller
scripts/keyboard_teleop
resources/
rrbot_control/
```

include/ros_control_boilerplate/ros_control_boilerplate.h

- `RosControlBoilerplate` sınıfını tanımlar.
- Metodlar: `init()`, `read()`, `write()`, `update()`.
- Özelleştirme: `joint_names`, kontrol modları (`position`/`velocity`/`effort`), donanım bağlantısı.

src/ros_control_boilerplate.cpp

- Donanım döngüsünü (`read()` → `controller_manager.update()` → `write()`) yönetir.
- Özelleştirme: `read()` ve `write()` kodları, donanımınızın API'sine göre doldurulmalıdır.

launch/ros_control_boilerplate.launch

- Donanım arayüzünü başlatır, URDF'yi ve kontrolcüleri yükler.
- Parametreler: `robot_description`, `controllers.yaml`, `loop_hz`, `sim_control_mode`.

config/controllers.yaml

- Her ekleme uygun kontrolcü ve PID değerleri atanır.
- Örnek kontrolcüler: `JointPositionController`, `JointStateController`.

scripts/

- `controller_to_csv`: Kontrolcü komutlarını CSV'ye kaydeder.
- `csv_to_controller`: CSV'den okur ve robotu sürer.
- `keyboard_teleop`: Klavye ile manuel kontrol.

rrbot_control/

- RRBot için örnek bir donanım konfigürasyonu ve simülasyon şablonu sunar.
- Kullanıcı bunu kendi robot modeliyle değiştirebilir.

Çalışma Akışı:

1. URDF ve kontrolcü parametreleri yüklenir.
2. Donanım arayüzü başlatılır, eklemler ve kontrol modları yapılandırılır.
3. Döngü başlar: `read()` → `controller_manager.update()` → `write()`.
4. Eklem durumları ROS topic'lerinde yayınlanır, komutlar donanıma gönderilir.

Neden Kontrolcü Değil?

- Bu paket, kontrolcülerin kendisini sağlamaz. Kontrolcüler ROS'un resmi paketlerinden yüklenir.
- **Görevi:** Donanımı *kontrol edilebilir* hale getirmek ve ROS kontrolcülerine doğru veriyi sağlamak.
- Kontrolcülerin seçimi (`position`/`velocity`/`effort`) ve PID ayarları `controllers.yaml` dosyasında belirlenir.

Özelleştirme Noktaları:

- `read() / write()`: Donanım protokolünüze göre.
- `joint_names` ve sayısı: URDF ile uyumlu olmalı.
- Kontrolcü tipi: Her eklem için doğru kontrolcü seçilmeli.
- PID ayarları: Robot dinamiklerine göre.
- Loop frekansı: Robot tepki süresine uygun.

Özet: *ros_control_boilerplate* paketi, gerçek robot donanımı ile ROS kontrol altyapısı arasında köprü kuran bir şablondur. Kendi başına bir kontrolcü değildir; kontrolcülerle çalışmak için donanım arayüzünü hazır hale getirir. Özellikle `read()/write()` metodları, eklem isimleri, kontrolcü tipleri ve PID parametreleri doğru şekilde yapılandırılarak, hem simülasyon hem de gerçek robot üzerinde güvenilir bir kontrol sistemi kurulabilir.

13 rosparam_shortcuts Kütüphanesi

`rosparam_shortcuts`, ROS projelerinde parametreleri kolay ve güvenli biçimde yüklemek için kullanılan bir C++ kütüphanesidir. Özellikle karmaşık tiplerde (örneğin `Pose`, `Isometry3d`, `vector`) parametre okuma işlemlerini basitleştirir.

Bu kütüphane GitHub üzerinden açık kaynak olarak paylaşılmıştır: https://github.com/PickNikRobotics/rosparam_shortcuts

Proje Yapısı ve Dosyalar

1. Ana Dizin

```
rosparam_shortcuts/
 include/
    rosparam_shortcuts/
        rosparam_shortcuts.h
        deprecation.h
 src/
    rosparam_shortcuts.cpp
    example.cpp
 config/
    example.yaml
 launch/
    example.launch
 CMakeLists.txt
 package.xml
 CHANGELOG.rst
```

2. rosparam_shortcuts.h
Kütüphane arayüzünü içerir. Parametre alma işlemi için farklı veri tiplerini destekleyen `get()` fonksiyonları tanımlıdır. Ayrıca dönüşüm ve hata yönetimi için yardımcı fonksiyonlar bulunur.

3. rosparam_shortcuts.cpp
Tüm `get()` fonksiyonlarının C++ tanımlamaları burada yer alır. Parametre bulunamazsa hata verir. Desteklenen türler: `bool`, `int`, `double`, `std::string`, `std::vector`, `ros::Duration`, `Eigen::Isometry3d`, `geometry_msgs::Pose`.

4. example.cpp
Kütüphanenin nasıl kullanılacağını gösteren örnek node'dur. Tüm parametreleri yükler ve terminale yazdırır. Kullanıcılar bu örnek üzerinden kendi projelerinde benzer işlemleri uygulayabilir.

5. example.yaml
YAML formatında parametrelerin tanımlandığı yapılandırma dosyasıdır. Örnek:

```
control_rate: 100.0
param1: 20
param4: [1, 1, 1, 3.14, 0, 0]
param7: [1, 1, 1, 3.14, 0, 0]
```

6. example.launch
Hem `example.yaml` dosyasındaki parametreleri ROS parametre sunucusuna yükler hem de örnek düğümü başlatır: [language=XML] ¡launch¿ ¡rosparam file="$(findrosparam_shortcuts)/config/example.yaml"command "load"/ >< $nodename = "example"pkg = "rosparam_shortcuts"type = "example"output = "screen"/ ><$ $/launch >$

7. package.xml
ROS bağımlılıklarının tanımlandığı dosyadır. Gerekli bağımlılıklar:

- `roscpp`
- `cmake_modules`
- `eigen_conversions`
- `roslint`

8. CMakeLists.txt
CMake yapılandırmasını içerir. Kütüphane ve örnek düğümün derlenmesi için gerekli ayarlar tanımlıdır.

9. CHANGELOG.rst
Paketin zaman içerisindeki değişiklik geçmişini içerir. GitHub üzerinden takip edilebilir.

Kullanım Şekli

Kütüphane ile parametreleri almak için: [language=C++] rosparam$_s hortcuts :: get("example", nh, "control_rate", control_r$ $get("example", nh, "pose_param", pose)$;

Eksik parametre varsa şu şekilde kapatılabilir: [language=C++] rosparam$_s hortcuts :: shutdownIfError("example", er$

Derleme ve Çalıştırma

[language=bash] cd /arm24$_w s-maincatkin_m akesourcedevel/setup.bashroslaunchrosparam_s hortcutsexample.launch

Sonuç

`rosparam_shortcuts`, ROS projelerinde parametre okuma işlemini standartlaştırır, hatalara karşı güvenli hale getirir ve kullanıcıya temiz bir yapı sunar. Karmaşık tiplerle çalışmak isteyen geliştiriciler için zaman kazandıran bir çözümdür.