

virtual Robotic Laboratory and Learning Materials for ROSin Education Project



Virtual Robotic Laboratory and Learning Materials



Supported by ROSIN - ROS-Industrial Quality-Assured Robot Software Components.
More information: rosin-project.eu



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 732287.



Virtual Robotic Laboratory and Learning Materials for ROSin by Inovasyon Muhendislik Ltd. Sti. is licensed under CC BY-NC-ND 4.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/4.0/>

Sanal Robotik Laboratuvarı ve ROSin için Eğitim Materiyalleri

Eğitim Projesi

Yazarlar:

- **Önsöz:** Dr. Uğur Yayan • Inovasyon Muhendislik Ltd. Sti •
ugur.yayan@inovasyonmuhendislik.com
- **Bölüm 1- 2:** Mustafa Karaca • Inovasyon Muhendislik Ltd. Sti •
mustafa.karaca@inovasyonmuhendislik.com
- **Bölüm 3:** Hakan Gencturk • Inovasyon Muhendislik Ltd. Sti •
hakan.gencturk@inovasyonmuhendislik.com
- **Bölüm 4:** Muhammed Oguz Tas • Eskisehir Osmangazi University •
motas@ogu.edu.tr
- **Bölüm 5:** Sezgin Secil • Eskisehir Osmangazi University • ssecil@ogu.edu.tr

Feragat:

CC-BY-NC-ND Bu lisans, yeniden kullanıcıların içerik oluşturucuya kredi vermesini gerektirir. Yeniden kullanıcıların malzemeyi yalnızca ticari olmayan amaçlarla herhangi bir ortam veya biçimde kopyalamasına ve dağıtmamasına olanak tanır. Diğerleri malzemeyi yeniden karıştırır, uyarlar veya üzerine inşa ederse, değiştirilmiş malzemeyi dağıtamazlar.



Virtual Robotic Laboratory and Learning Materials for ROSin by Inovasyon Muhendislik Ltd. Sti. is licensed under CC BY-NC-ND 4.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/4.0/>

Önsöz

1920'li yıllarda Çekoslavak Karel Cařek'in tiyatro oyununda kullandığı çek dilinde hizmet eden anlamına gelen "robota" kelimesinden türeyen robot kavramı ile insanların hayatlarını kolaylaştıracak robotlar hakkındaki düşüncelere öncülük edilmiştir. Özellikle ülkeler arası ticaret, savaşlar, üretim maliyetinin düşmesi ve uzaya erişme çabaları ile robotik alanında birçok teknolojik yeniliğin ortaya çıkmasında etkili olmuştur. Geliştirilen çeşitli mobil ve endüstriyel robotlar hastanelerde, otellerde ve endüstriyel alanlarda birçok fayda sağlamaktadır. Özellikle 80'li yıllarda günümüze kadar pazarda önemli etkisi olan mobil ve endüstriyel robotların günümüz endüstri 4.0 hareketi ile gelecekte geliştirilecek olan akıllı yapı ekosisteminde anahtar roller üstleneceklerdir. Günümüzde de bu robotların akıllı hale gelmesi için çalışmalar yapılmaktadır. Robotik sistemlerin akıllı hale getirilmesinde olan karmaşıklığın daha basit şekilde anlaşılp, yeni geliştirilen sistemlerin belirli standardlarda hayatı geçmesi için günümüzde birçok yazılım platformu geliştirilmiştir. Bu platformlar arasında Robot işletim sistemi (Robot Operating System : ROS) yüksek popülerliğe erişmiştir. Bu başarının asıl kriterlerinden biriside açık kaynak kodlara sahip olması ve büyük bir topluluk tarafından sürekli desteklenmesidir. Anlaşılması göreceli olarak zor olan ROS platformunun anlaşılması için çeşitli kaynaklar sunulmuştur fakat gelişen teknoloji ve gelişen açık kaynaklı mimaride en son pratikleri öğrenmek önem teşkil etmektedir. Özellikle ülkemizde Linux, Gazebo ve ROS'u kapsamlı şekilde birlikte ele alan, günümüz uygulamalarına sahip bir kaynak bulması çok zordur. Tecrübeli mühendis kadrosu ile en uygun şekilde teori ve pratiği bir araya getirdiğimiz bu kitapta ROS ile ilgili birçok önemli kavram ele alınmaktadır. Bu kitapta takip edilen yol ile emek vermiş her okuyucunun ROS uyumlu yazılım üretici olması hedeflenmektedir. Kitapta anlatılan interaktif konular ile okuyucunun sadece konuları okuyarak geçmesi değil, Okuyucununda kitaptaki örnekler ile birlikte ROS arakatmanına adım atarak içeriği prensipleri ezberlemeden uygulamalı şekilde öğrenerek kalıcı bir ROS uyumlu yazılım sağlayacısı olması hedeflenmektedir. Bu süreçte pratik yapılmadan, klavyeye dokunmadan bir programlama eğitimiminin başarılı olamayacağı düşünüldüğünden kitap doğrudan okuyucuya uygulamalar ile pratik yapması üzerine kurgulanmıştır ve her okuyucunun kodlama ile yakınılığının arttırması sağlanmıştır. Titizlikle hazırlanmış bu kitap ile gelecekteki akıllı platformlara katkı sağlayacak yeni mühendislerin gelişmesi yazarların en büyük hayali olmakta ve okuyucuların bu kitaptan büyük fayda görmesi amaçlanmaktadır.

İçindekiler

| | | |
|--------|--|----|
| 1. | Giriş..... | 10 |
| 2. | Linux Tarihçesi ve Programlamaya giriş | 11 |
| 2.1. | Tarihçe | 11 |
| 2.1.1. | Unix..... | 11 |
| 2.1.2. | GNU | 12 |
| 2.1.3. | Linux | 13 |
| 2.1.4. | Dağıtımlar | 13 |
| 2.1.5. | Paket Yöneticileri..... | 13 |
| 2.1.6. | Ubuntu..... | 14 |
| 2.1.7. | Shell(kabuk) | 14 |
| 2.2. | Ubuntu kurulumu | 14 |
| 2.3. | Ubuntu temelleri | 22 |
| 2.3.1. | Prompt (Komut İstemi)..... | 22 |
| 2.3.2. | Dosya indirme..... | 23 |
| 2.3.3. | Dosya listeleme | 24 |
| 2.3.4. | Dosya Görüntüleme..... | 24 |
| 2.3.5. | Head ve Tale | 25 |
| 2.3.6. | Standart akışlar..... | 26 |
| 2.3.7. | Yönlendirmeler..... | 26 |
| 2.3.8. | Düzenli ifadeler(regex) | 27 |
| 2.4. | Linux Dosya Yapısı | 28 |
| 2.4.1. | Sudo..... | 30 |
| 2.4.2. | Kullanıcılar | 30 |
| 2.4.3. | Dosya izinleri..... | 30 |
| 2.4.4. | Prosesler | 31 |
| 2.4.5. | Paket kurulumları | 32 |
| 2.4.6. | Servisler | 32 |
| 2.4.7. | Metin Editörleri | 32 |
| 2.4.8. | Linux terminalı..... | 33 |
| 2.5. | Python Programlamaya giriş | 34 |
| 2.5.1. | Değişkenler | 35 |

| | | |
|---------|---|----|
| 2.5.2. | Data türleri | 35 |
| 2.5.3. | Koşullu ifadeler..... | 36 |
| 2.5.4. | Fonksiyonlar | 36 |
| 2.5.5. | Döngüler | 37 |
| 2.5.6. | Koleksiyonlar | 38 |
| 2.5.7. | Operatörler..... | 39 |
| 2.5.8. | Asal sayı bulma uygulaması..... | 40 |
| 2.5.9. | Plot uygulaması | 41 |
| 2.5.10. | İkili plot uygulaması..... | 42 |
| 3. | ROS Uygulamaları | 43 |
| 3.1. | ROS Nedir – Ne Değildir?..... | 43 |
| 3.2. | Neden ROS Kullanmalıyız?..... | 44 |
| 3.3. | ROS Sensörler ve ROS Kullanan Firmalar..... | 44 |
| 3.4. | ROS Kurulumu | 45 |
| 3.5. | Linux Temel Kodlar | 48 |
| 3.6. | ROS Mimarisi Giriş | 50 |
| 3.7. | Dosya Sistemi – File System Level | 50 |
| 3.7.1. | Paketler (Packages) | 50 |
| 3.7.2. | Metapaketler (Metapackages) | 52 |
| 3.7.3. | Paket bildirileri (Package Manifests) | 53 |
| 3.7.4. | Mesaj tipleri (Message types) | 54 |
| 3.7.5. | Servis tipleri (Service types) | 55 |
| 3.8. | İşlem Grafiği – Computation Graph Level | 55 |
| 3.8.1. | Düğümler (Nodes) | 55 |
| 3.8.2. | Parametre Servisi (Parameter Service)..... | 56 |
| 3.8.3. | Mesajlar (Messages)..... | 57 |
| 3.8.4. | Konular (Topics)..... | 58 |
| 3.8.5. | Servisler (Services)..... | 59 |
| 3.8.6. | Çantalar (Bags)..... | 60 |
| 3.8.7. | Ana (Master)..... | 60 |
| 3.9. | Topluluk – Community Level | 61 |
| 3.10. | Publisher-Subscriber ve Service Client Yapıları | 62 |
| 3.11. | ROS Araçları | 62 |

| | | |
|---------|---|-----|
| 3.12. | ROS uygulamaları..... | 63 |
| 3.12.1. | Uygulama 1: ROS Ortamının Hazırlanması | 63 |
| 3.12.2. | Uygulama 2: catkin Paketi Oluşturma ve ROS Ortamını Tanıma..... | 66 |
| 3.12.3. | Uygulama 3: TurtleSim | 70 |
| 3.12.4. | Uygulama 4: Mesaj ve Servis Oluşturma | 74 |
| 3.12.5. | Uygulama 5: Publisher-Subscriber Uygulaması..... | 78 |
| 3.12.6. | Uygulama 6: Service-Client Uygulaması..... | 84 |
| 3.12.7. | Uygulama 7: Verileri Kaydetme ve Oynatma | 88 |
| 4. | Gazebo..... | 91 |
| 4.1. | Gazebo nedir?..... | 91 |
| 4.2. | Gazebo Bileşenleri..... | 91 |
| 4.2.1. | Dünya Dosyaları..... | 92 |
| 4.2.2. | Model Dosyaları..... | 93 |
| 4.2.3. | Ortam Değişkenleri..... | 94 |
| 4.2.4. | Gazebo Sunucusu | 95 |
| 4.2.5. | Gazebo İstemcisi..... | 95 |
| 4.2.6. | Eklentiler (plugins) | 96 |
| 4.3. | Gazebo uygulamaları..... | 97 |
| 4.3.1. | Gazebo Kurulumu | 97 |
| 4.3.2. | Gazebo çalıştırılması | 98 |
| 4.3.3. | Çalışma Ortamının Hazırlanması | 98 |
| 4.3.4. | Dünya Oluşturulması | 99 |
| 4.3.5. | Model Oluşturulması | 102 |
| 4.3.6. | Mesh Giydirmeye | 112 |
| 4.3.7. | Sensör Ekleme | 114 |
| 4.4. | ROS ile Kontrol..... | 118 |
| 5. | Movelt | 119 |
| 5.1. | Movelt nedir? | 119 |
| 5.2. | Movelt İçeriği..... | 119 |
| 5.2.1. | Movelt Kurulumu | 120 |
| 5.2.2. | Movelt Setup Assistant..... | 120 |
| 5.2.3. | Movelt ile Planlama ve Rviz ile Gazebo Üzerinde Çalışma Gösterimi | 136 |
| 5.3. | Örnek Uygulama..... | 141 |

Figürler

| | |
|--|----|
| Figure 1 Ken Thompson ve Dennis Ritchie | 11 |
| Figure 2 Unix'in gelişimi ve Unix benzeri sistemler | 12 |
| Figure 3 Mevcut Sürümleri | 15 |
| Figure 4 Rufus Programı | 15 |
| Figure 5 Kurulum Ekranı | 16 |
| Figure 6 Kurulum Hazırlığı | 17 |
| Figure 7 Kurulum Seçenekleri | 18 |
| Figure 8 Bölgesel Saat Ayarı | 19 |
| Figure 9 Klavye Düzen Ekranı | 20 |
| Figure 10 Giriş Ekranı | 21 |
| Figure 11 Kurulum Tamamlanma Mesajı | 21 |
| Figure 12 Örnek Uygulama | 22 |
| Figure 13 örnek Komut istemi (kullanıcı ve bilgisayar adı) | 22 |
| Figure 14 Örnek Uygulama | 23 |
| Figure 15 Örnek wget Uygulaması | 23 |
| Figure 16 Örnek Ls Uygulaması | 24 |
| Figure 17 Örnek Uygulama | 25 |
| Figure 18 Örnek Uygulama | 25 |
| Figure 19 Standard Akış Sistemi | 26 |
| Figure 20 Örnek Uygulama | 26 |
| Figure 21 Örnek Uygulama | 26 |
| Figure 22 Örnek Uygulama | 27 |
| Figure 23 Çalışma Sistemi | 27 |
| Figure 24 Örnek Uygulama | 28 |
| Figure 25 Örnek Uygulama | 28 |
| Figure 26 Linux Kök Dizin Yapısı | 29 |
| Figure 27 Sudo Nedir? | 30 |
| Figure 28 Örnek adduser Uygulaması | 30 |
| Figure 29 Örnek İzin Uygulaması | 31 |
| Figure 30 Terminal Penceresi | 34 |
| Figure 31 Temel Programlama Terimleri | 35 |
| Figure 32 Örnek Asal Sayı Bulma uygulaması | 40 |
| Figure 33 Örnek Plot Uygulaması | 42 |
| Figure 34 Örnek İkili Plot Uygulaması | 43 |
| Figure 35 Boston Dynamics Atlas Robot | 45 |
| Figure 36 EvaRobot | 45 |
| Figure 37 Platform Seçim ekranı | 46 |
| Figure 38 Sürüm Seçim Ekranı | 46 |
| Figure 39 Linux Terminal Ekranı | 48 |
| Figure 40 ROS Dosya Sistemi | 50 |
| Figure 41 CMakeList.txt | 52 |
| Figure 42 ROS Robot Metapaket Dağıtımları | 53 |

| | |
|--|----|
| Figure 43 Package.xml Bölüm ve açıklamaları..... | 54 |
| Figure 44 konum mesajı | 54 |
| Figure 45 TurtleSim Oluşturma servisi | 55 |
| Figure 46 ROS İşlemGrafiği Seviyeleri..... | 55 |
| Figure 47 ROS Mesaj Standardları | 57 |
| Figure 48 ROS Başlık Mesajı | 58 |
| Figure 49 Yayınlayıcı / Dinleyici | 59 |
| Figure 50 RosMaster..... | 61 |
| Figure 51 Topluluk Seviyesi | 62 |
| Figure 52 Publish-Subscribe örneği | 62 |
| Figure 53 Service-Client örneği..... | 62 |
| Figure 54 rqt_graph..... | 63 |
| Figure 55 Rviz | 63 |
| Figure 56 rqt_plot..... | 63 |
| Figure 57 Çalışma Alanı Dosya Yapısı | 64 |
| Figure 59 .Bashrc | 65 |
| Figure 60 Catkin Paketi Oluşturmak | 67 |
| Figure 61 first_script.cpp..... | 68 |
| Figure 62 Catkin derlenmesi..... | 68 |
| Figure 63 roscore ekranı..... | 68 |
| Figure 64 Package.xml..... | 69 |
| Figure 65 rospack Örneği..... | 69 |
| Figure 66 Bütün rospack Bağımlılıkları | 70 |
| Figure 67 Turtlebot Teleoperasyonı | 70 |
| Figure 68 rosservice Listesi..... | 71 |
| Figure 69 TurtleSim oluşturulması | 71 |
| Figure 70 TurtleSim/Spawn servislerinin gösterilmesi | 71 |
| Figure 71 Yeni Servis Çağırılması | 72 |
| Figure 72 Ros Parametre Listesi | 72 |
| Figure 73 Arkaplan Ayarlanması..... | 72 |
| Figure 74 TurtleSim Arkaplan Renk değişim örneği | 73 |
| Figure 75 Servislerin Kapatılması..... | 73 |
| Figure 76 Catkin Derleme | 75 |
| Figure 77 Caktin Derlenmesi | 77 |
| Figure 78 Talker.cpp ve Detayları..... | 78 |
| Figure 79 Listener.cpp ve Detayları..... | 79 |
| Figure 80 Catkin Derlemesi | 80 |
| Figure 81 talker.py ve Detayları..... | 81 |
| Figure 82 Listener.py ve Detayları..... | 82 |
| Figure 83 Catkin Derlemesi | 82 |
| Figure 84 Terminal 2..... | 83 |
| Figure 85 Terminal 3..... | 83 |
| Figure 86 add_two_ints_server.cpp..... | 84 |
| Figure 87 add_two_ints_client.py | 85 |

| | |
|---|-----|
| Figure 88 Catkin Derlemesi | 85 |
| Figure 89 add_two_ints_server.py..... | 86 |
| Figure 90 add_two_ints_client.py..... | 87 |
| Figure 91 Catkin Derlemesi | 87 |
| Figure 92 Terminal 2..... | 88 |
| Figure 93 Terminal 3..... | 88 |
| Figure 94 Terminal 1..... | 88 |
| Figure 95 TurtleSim Teleoperasyonu | 90 |
| Figure 96 Sabit hız verildiğinde TurtleSim | 90 |
| Figure 97 rosbag çalıştırılması | 90 |
| Figure 98 rosbag içeriği | 90 |
| Figure 99 rosbag'ın çalıştırılması | 90 |
| Figure 100 Gazebo Simülasyon platformu | 91 |
| Figure 101 Gazebo Çalışma Ortamı | 100 |
| Figure 102 Model ekleme..... | 101 |
| Figure 103 Model Özellikleri..... | 102 |
| Figure 104 Basit ve Özel şekiller | 104 |
| Figure 105 Model bağlantıları | 105 |
| Figure 106 Bağlantı atalet ve pozisyonları..... | 106 |
| Figure 107 Geometri özellikleri | 107 |
| Figure 108 Geometri alan boyutu | 108 |
| Figure 109 Oluşacak Model | 110 |
| Figure 110 Bağlantı Oluşturma..... | 111 |
| Figure 111 Oluşacak Model | 112 |
| Figure 112 Giydirilmiş Mesh..... | 114 |
| Figure 113 Algılayıcı Gösterimi | 117 |
| Figure 114 Movelt Arayüzü | 121 |
| Figure 115 Movelt Kurulum Asistanı | 122 |
| Figure 116 Self-collision Ayarlaması..... | 123 |
| Figure 117 Robot Link Çiftleri | 123 |
| Figure 118 Sanal Eklem Oluşturma | 124 |
| Figure 119 Planlama grubu oluşturulması..... | 125 |
| Figure 120 Eklemlerin Eklenmesi ve Kaydedilmesi | 126 |
| Figure 121 Seçilenlerin Düzenlenmesi..... | 127 |
| Figure 122 Robot Uzuv seçimi | 128 |
| Figure 123 Uç Uzuv Tanımlanması | 129 |
| Figure 124 Oluşan Konfigürasyon..... | 130 |
| Figure 125 Tool Oluşturulması | 131 |
| Figure 126 Robot Pozisyonu tanımlanması | 132 |
| Figure 127 Uç nokta Tanımlanması | 133 |
| Figure 128 ROS ile kontrol | 134 |
| Figure 129 Paket oluşturma | 135 |
| Figure 130 Rviz arayüzü | 137 |
| Figure 131 Gazebo Arayüzü..... | 137 |

| | |
|--|-----|
| Figure 132 Başlangıç Konum Seçimi | 138 |
| Figure 133 Rastgele Hedef Konum seçimi | 139 |
| Figure 134 Oluşan Plan | 140 |
| Figure 135 Gazebo Üzerinde Görünüm | 140 |
| Figure 136 Örnek Uygulama | 141 |
| Figure 137 Örnek Uygulama | 142 |
| Figure 138 Örnek Uygulama | 142 |

1. Giriş

Sevgili okuyucu,

Bu kitap, robotik sistemlerin programlanmasıında büyük fayda ve kolaylık sağlayan Robot İşletim Sistemi (Robot Operating System : ROS) yazılımını uygulamalı olarak anlatmak, ROS ile ilgili konu ve kavramları uygulamalı şekilde örnekler ile öğrenilerek konulara devam edilmesi amaçlandı. Sunulan başlıkların her biri okuyucunun temelleri sağlam olmasını sağlayıp zorlanmayacak şekilde bir ROS uyumlu yazılım üreticisi olmasını hedeflemektedir.

2000'li yıllarda itibaren robotların aktif şekilde hayatımıza girmesi ile gelecekte önemli özelliklere sahip olacak robotik sistemlerin günümüz akıllı teknolojileri ile sahip olduğu değer sürekli artmaktadır. Günümüzde ve gelecekte büyük önem taşıyan bu konu için yeni değerler üretmek önem teşkil etmektedir. Ayrıca ROS uyumlu yazılımlar konusunda değerler üretmek, bu konuya yeni başlayan kişiler için göreceli olarak zor olabilir. Sürekli kendini yenileyen robotik alanda ROS ile ilgili son teknikleri kapsayan bir kaynak bulmanın zor olduğu günümüzde, sizlere sunduğumuz ders kitabı niteliğine sahip bu kaynak ile Linux işletim sisteminin temellerini oluşturan Unix tarihi ve felsefesi ile başlayacaktır. Linux işletim sisteminin orataya çıkıştı ve gelişmesi ile devam eden ilk bölümler, Ubuntu'nun kurulması, temel komutların öğrenilmesi ile devam edecktir. Sonrasında, Python programlama ilgili önemli uygulamalar yapılacak ve öğrenilen temel python bilgisi ile ROS arakatmanı ile ilgili detaylı ve uygulamalı bir anlatıp sunulacaktır.

Sonrasında ROS ile uyumlu şekilde kullanılabilen ve gelişmiş özelliklere sahip olan Gazebo ile ilgili önemli bilgiler anlatılıp MoveIt ile ilgili detaylı ve örnekli çalışmalar sunulmaktadır. Kitapta sunulan her bölüm bir öncekinin üstüne eklenen bilgiler ile inşa edilmiş ve hiçbir bölüm birbirinden ayrı görülmediğinden kitaptaki bölümlerin verilen sıra ile takip edilmesi önem taşımaktadır. Emek harcayan her okuyucu için bu kitabın bir öğrenme aracı olacağı düşünülmektedir ve eğitimin sonunda her emektar okuyucunun bir ROS Uyumlu yazılım üreticisi olması hedeflenmektedir.

2. Linux Tarihçesi ve Programlamaya giriş

2.1. Tarihçe

2.1.1. Unix

Unix işletim sisteminin bulunması MULTICS'e yani çoklayıcı işletim ve esaplama sistemine dayanmaktadır. MULTICS projesi 1960'ların ortasında General Electric, Massachusetts teknoloji Enstitüsü ve Bell laboratuvarları'nın işbirliği ile başladı ve 1969'da Bell Labaratuvarı projeden çekildi. Bu proje üstünde çalışan kişilerden biri ise Ken Thompson idi. MULTICS'in sahip olduğu büyük pontasiyeli gördü. Ancak MULTICS çok karmaşık bir yapıya sahipti ve Ken Thompson aynı işin daha basit bir şekilde yapılacağını hissetti. 1969 yılında UNICS adlı UNIX'in ilk versiyonunu yazdı. UNICS Uniplex Information and Computing system yani tek taraflı işletim ve hesaplama sistemini temsil etti. İşletim sistemi zamanla değişmiş olsa da ad aynı kalıp Unix'e kısaltıldı.

Ken Thompson ilk C derleyicisini yazan Dennis Ritchie ile birlikte çalıştı. 1973'te C'de Unix çekirdeğini yeniden yazdılar. Ertesi yıl Beşinci Baskı olarak bilinen Unix'in bir versiyonu ilk olarak üniversitelere lisanslandı. 1978'de piyasaya sürülen Yedinci Baskı, iki farklı Unix geliştirme hattı için bir bölme noktası oldu ve SRV4 (system V) ve BSD olarak iki dala bölündü. Farklı versiyonları çıkan bu işletim sistemi farklı kurumlara lisanslanarak ticarileştirilmiştir. Unix tek taraflı hesaplama felsefi asıl başarı kaynağı denilebilir.

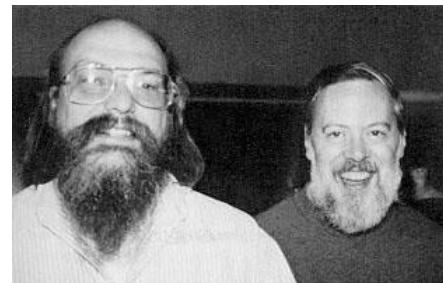


Figure 1 Ken Thompson ve Dennis Ritchie

Unix felsefesi:

- Daha kötü daha iyidir.
- Basit, kullanımı kolay, minimalist, tekrar kullanılabilir yazılımlar.
- 4 kural: Basit, doğru, tutarlı, tamamlayıcı.
- Her bir program bir iş yapsın ama en iyi şekilde yapsın.
- Eski programlara yeni özellik eklemek yerine sıfırdan yenileri yazılsın.

- Bir programın çıktısı bir başka programın girdisi olsun.

Unix felsefesinde her şey bir dosyadır.

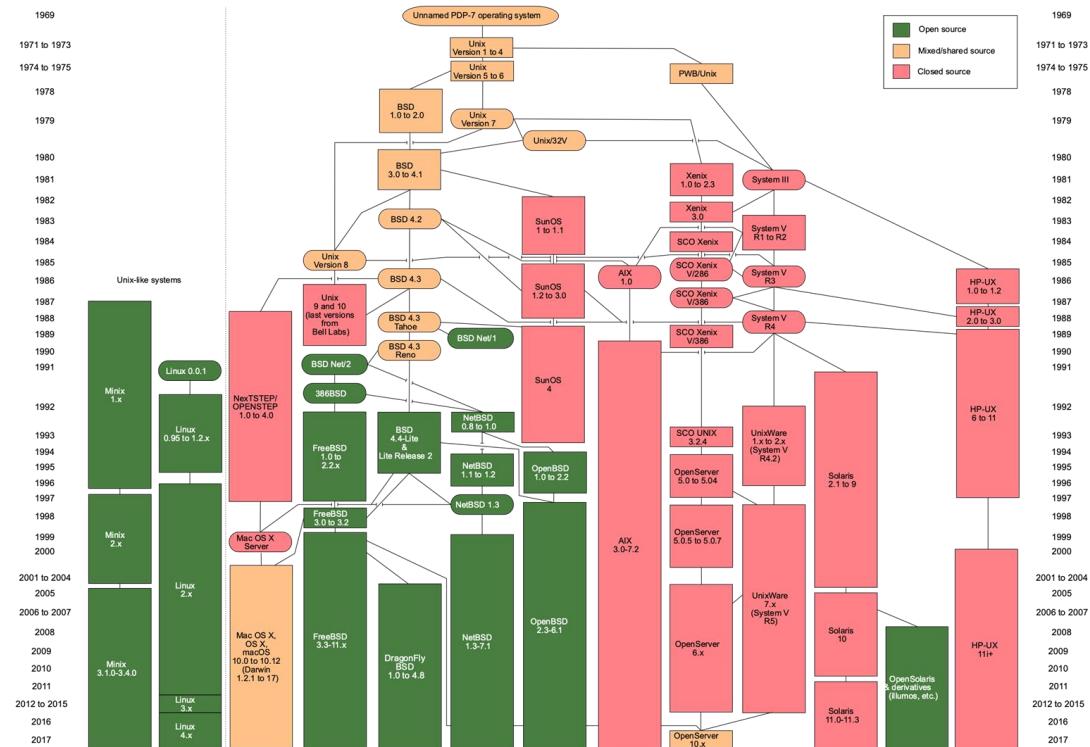


Figure 2 Unix'in gelişimi ve Unix benzeri sistemler

2.1.2. GNU

1983 senesinde Richard Stallman , GNU olarak isimlendirdiği ve UNIX işletim sistemine benzeyen fakat ücretsiz ve açık kaynak kodlu olacak bir işletim sistemi oluşturma hedefini aştı. Bu zor hedefin en önemli gereksinimi bir işletim sisteminin merkezini oluşturan çekirdeğin geliştirilmesi olduğunu biliyordu. GNU Herd olarak adlandırılan bu çekirdeğin geliştirilmesi çok yavaş ilerlediği için, 1990'larda bir üniversite öğrencisi olan Finlandiyalı Linus Torvalds'ın yayınladığı Linux çekirdeği ve GNU parçacıklarının bir araya gelmesi ile Linux/GNU işletim sistemleri ve dağıtımları ortaya çıkmaya başladı. Dolayısıyla Linux işletim sistemi denildiği aman aslında Linux çekirdeği ve bununla birlikle GNU çatısı altında bulunan diğer komponentler anlaşılmaktadır. Bu oluşan yapıya Linux/GNU denilmektedir. 2012 yılından beri Linux çekirdeğinin resmi olarak GNU projesine dahil edilmesiyle tamamlanmış bulunmaktadır.

2.1.3. Linux

Üniversite öğrencisi Linus Torvalds Linux core bulmuş ve version 1.0 1994 yılında çıkmıştır. Linux/GNU ile oluşan bu işletim sistemi genel olarak 6 parçacıkтан oluşur. Bunlar:

- Linux çekirdeği
- GNU araçları ve kütüphaneleri,
- Dokümantasyon,
- Pencere sistemi (En çok kullanılan ve geleneksel olan X Pencere Sistemi, günümüzde Wayland de kullanılmaktadır),
- Masaüstü Çevremi,
- Paket yöneticileri

Linux açık kaynaklı bir lisansa sahip olması, herhangi bir amaç için program çalışma özgürlüğü, programın çalışma mekanizmasının incelenmesi, geliştirilen kodların başkalarına dağıtıma özgürlüğü ile günümüzde önemli bir yere sahiptir.

2.1.4. Dağıtımlar

Linux'un her tür kullanıcıya uyacak farklı versiyonlar sunmaktadır. Bu sürümlere dağıtım/ dağıtımlar denir. Günümüzde 500'den fazla aktif dağıtım vardır. Her dağıtımın masaüstüünü farklı şekilde ele alır. Bazıları çok modern kullanıcı arabirimlerini tercih ederken (GNOME ve Elementary OS's Pantheon gibi), diğerleri daha geleneksel bir masaüstü ortamına bağlı kalır (openSUSE, KDE kullanır).

2.1.5. Paket Yöneticileri

Linux dağıtımında bulunanlardan biriside paket yöneticisi olduğunu söylemiştir. Paket yönetimi, açık kaynak kodlu dağıtımları yapılan her bir projenin her makinada ayrı ayrı derlenmesi ve gereksinimlerinin sağlanmanın zorluğundan dolayı farklı CPU ve ve donanım konfigürasyonları için derlenip bir depoda (repo) saklanmaktadır. Kullanıcılar indirerek kullanabilmektedirler. Linux çekirdeği birçok konfigürasyonu desteklemektedir. Bunlar x86-32, x86-64, ARM-32 ve ARM-64 gibi önemli mimarileri örneklenebilir. Linux dağıtımlarını birbirinden ayıran en önemli parça da paket yönetim sistemidir. Debian ve Ubuntu temelli dağıtımlar *apt*

sistemi ve .deb paketlerini, Arch temelli olanlar pacman yöneticisini, Fedora ve RedHat türevleri .rpm paket yapısını kullanır.

| Dağıtım | Paket sistemi |
|-------------------------|------------------|
| Debian | apt (dpkg, .deb) |
| Red Hat, CentOS, Fedora | yum (.rpm) |
| SUSE | YaST |
| Arch | pacman |
| Pardus | PiSi |

2.1.6. Ubuntu

Ubuntu, Canonical Ltd. şirketi tarafından geliştirilen yaklaşık 450 çalışanı olan özel bir şirket tarafından yazılan Unix tabanlı bir işletim sistemidir. Senede 2 sürüm çıkmaktadır ve bunlar 0.4 ve .10 sürümleri olmaktadır. Bu sürümlerden LTS olan sürümler “Long Term Support” yani uzun süreli 5 yıllık geliştirme desteği almaktadır. Günümüzde 16.04. ve 18.04 sürümleri LTS özelliği bulundurmaktadır.

2.1.7. Shell(kabuk)

Tüm kullanıcı arayüzleri kabuk(Shell) adı verilen bir program aracılığı ile kullanıcı-işletim sistemi haberleşmesi sağlanmaktadır. Bunlar Shell Graphic user interface (GUI) veya Command-line interface (CLI) temelli olabilir. Shell işletim sisteminin hizmetlerine erişilmesini sağlamakta ve Windows işletim sisteminde bilinen cmd.exe programı buna örnektir. Linux'te tarihsel olarak kullanılan kabuklar csh, tcsh, ksh, zsh, ash olarak söylenebilir. Günümüzde en popüler olarak kullanılan Shell GNU Bash'tir. GNU Bash'te komut satırında girilen komutlar bir dosyaya alt alta yazılacak olursa, Bash yorumlayıcısı bu betiği çalıştırabilir.

2.2. Ubuntu kurulumu

Tüm kullanılabilecek ubuntu versiyonları <http://releases.ubuntu.com/> linkinde bulunabilir. İndirilen Ubuntu .iso dosyası Usb belleğe ya da CD'ye yazdırılmalıdır. Yazdırma işlemi için Unetbootin ve ya Rufus programmatic kullanılabilir.

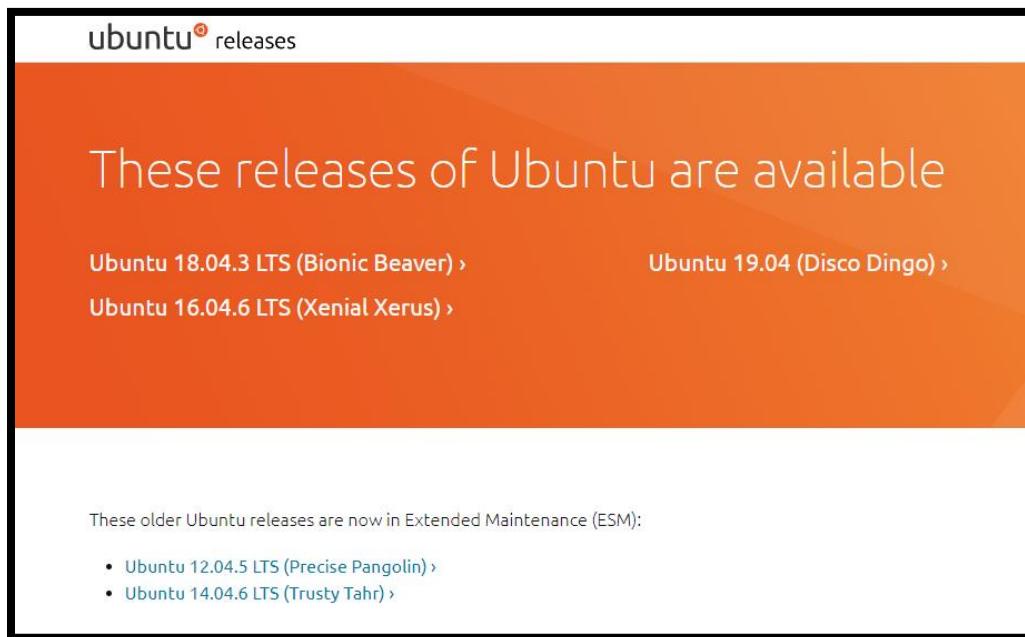


Figure 3 Mevcut Sürümüleri

Ubuntu kurulumunun ikinci aşamasında Rufus programı indirilip aygıt bölümünden harici USB disk seçilir(1). Sonrasında Boot tipi seçilecektir(2). Boot tipi seçildikten sonra .iso dosyasının dizini gösterilir(3). Bu işlemden sonra başlat tuşna basılır ve bellek hazır olduğu zaman kapat tuşuna basılabilir.

Bilgisayar açılırken hazırlanan Usb belleği bilgisayara takın ve F2 ya da F12'ye basarak boot ayarlarından Ubuntu kurulumu için kullanılacak Usb belleği seçin. Çıkan ekranda Ubuntu'yı kurmadan deneyebileceğiniz seçenek vardır:

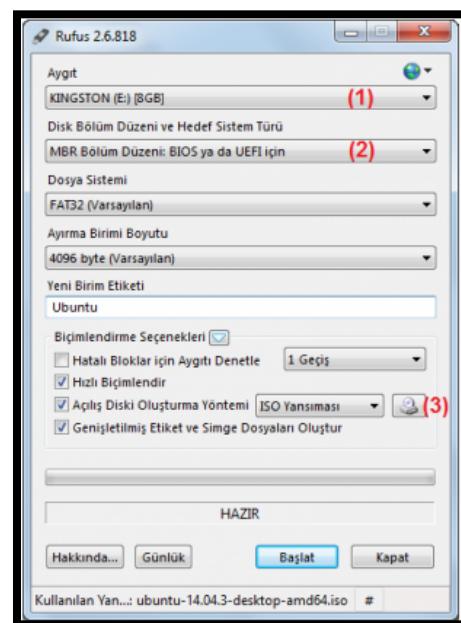


Figure 4 Rufus Programı

- “Ubuntu’yu dene” seçeneği ile Ubuntu sürümü denenip hiç bir kayıt yapmadan çıkılabilir.
- “Ubuntu’yu kur” seçeneği ile veya deneme sırasında Ubuntu kurulumu başlatılabilir.

Seçeneklerle ilgili örnek ekran görüntüsü figür 5 ‘te verilmiştir.



Figure 5 Kurulum Ekranı

Install Ubuntu seçeneğine basılarak Ubuntu kurulumuna başlanabilir. Kurulum sırasında sürümme gelmiş güncellemelerin yüklenmesi ve yan teknolojilerin yüklenmesi için yandaki seçenekler seçilebilir. Bu bölümler zorunluluk içermemektedir. Kullanıcı isterse güncellemeleri kurulum işlemi bittikten sonra da yapabilir. Bu işlemde karşınıza çıkabilecek ekran görüntüsü figür 6’te gösterilmiştir.

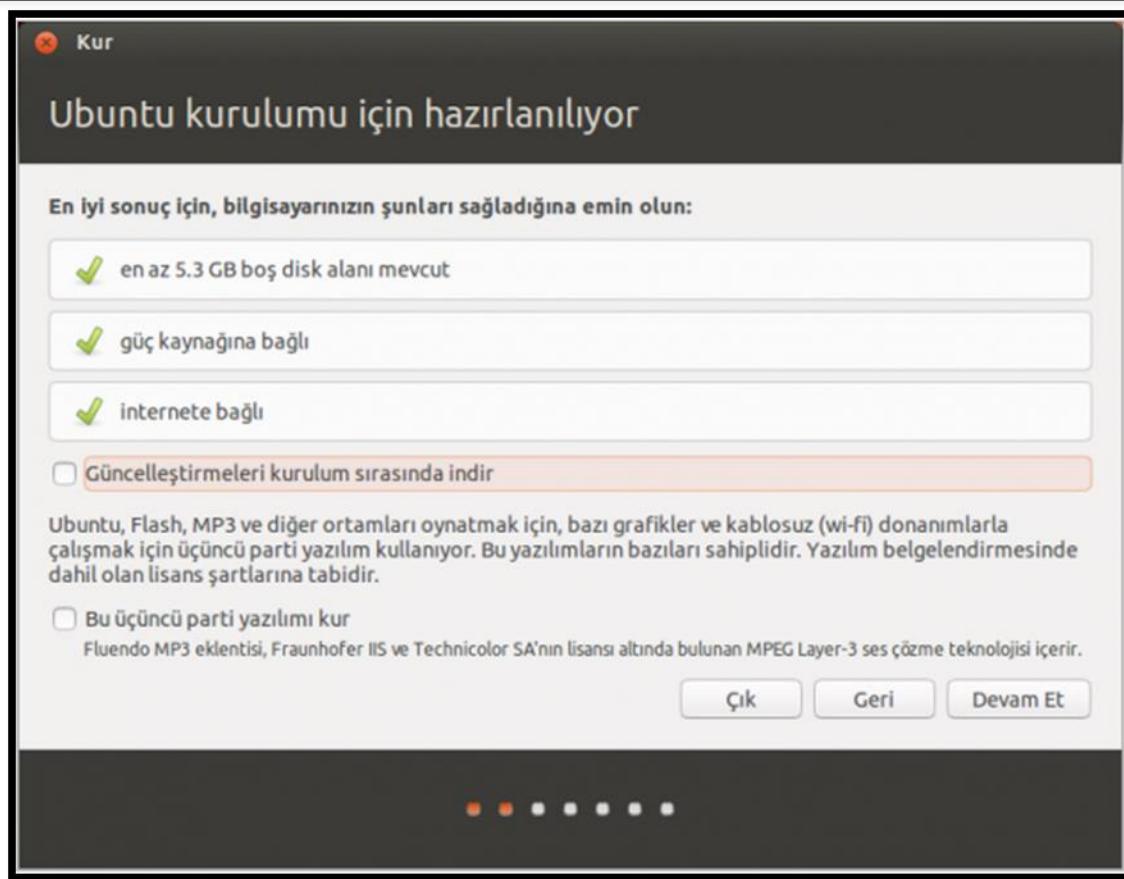


Figure 6 Kurulum Hazırlığı

Figür 7'te olduğu gibi kurulum esnasında karışınıza çıkacak Kurulum Türü penceresinde:

- Windows işletim sistemini Ubuntu ile değiştir: Bu seçenek Windows'u ve bilgisayarlarınızdaki belgeleri siler ve yerine Ubuntu'yu kurar.
- Yeni Ubuntu kurulumunu güvenlik için şifrele: Bu seçenek ile, kuruluma şifre ekleyerek kendimiz dışındakiler için erişimi ve müdahaleyi kontrol altına alabiliriz.
- Yeni Ubuntu kurulumu ile LVM kullan: Bu seçenek ile boyutu genişletilebilir şekilde yapılandırılmış diske kurulum yaparız. Bu özel bir disk gerektirmez. Bu seçimi yaparsak diskimiz ileride verilere zarar vermeden diskimizin boyutunu arttırma özelliğine uygun olarak yapılandırılır.
- Başka bir şey: Bu seçenek diski, kendi ihtiyaçlarınıza özel bir şekilde yapılandırmamanıza imkan verir. Bu seçenekin kullanımı biraz karışiktır ve deneyim gerektirir.
- Eğer bilgisayarınızda herhangi bir işletim sistemi yüklü değil ise ilk seçenek Diski sil ve Ubuntu yükle olacaktır.

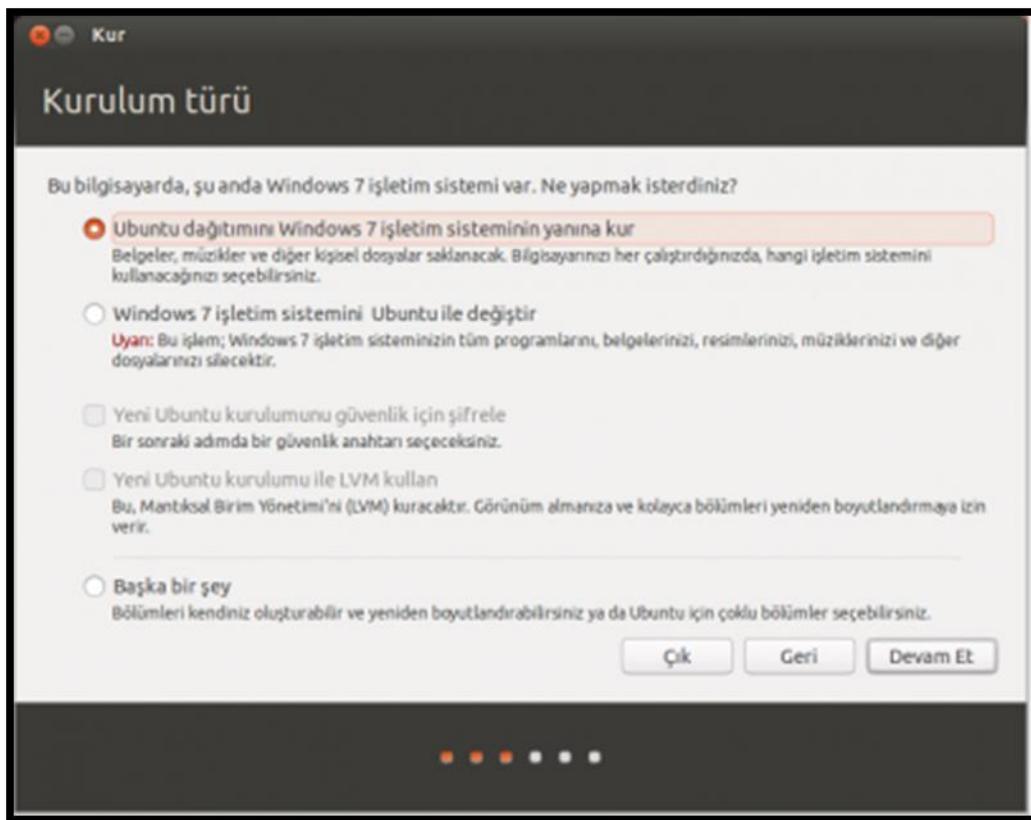


Figure 7 Kurulum Seçenekleri

Bu işlemden sonra karşımıza “Neredesiniz?” sorusunu soran figür 8’da gösterilen ekran ile karşılaşılacaktır. Burada bulunduğuğunuz ülkenin saat dilimini seçebilirsiniz. Türkiye seçeneği için İstanbul seçilebilir.



Figure 8 Bölgesel Saat Ayarı

Kurulumun bir sonraki adımda "Klavye düzeni" belirtme ekranı ile karşılaşılacaktır. Burada normal şartlarda klavye düzenimiz otomatik olarak belirleneceğinden genellikle herhangi bir işlem yapmamız gerekmekz. Eğer otomatik olarak belirlenememe durumu olursa sol alt kısmda yer alan 'Klavye Düzenini Algıla' düğmesi de klavyenin algılanmasında kullanılabilir. Burada istenilen tuşlara basıldığı taktirde klavye düzeniniz algılanacaktır. Ayrıca, Bir klavye dili seçtikten sonra burada yer alan kutucuğa klavyeden bazı Türkçe karakterler girerek söz konusu dilin doğruluğunu yani klavyenizle uyumluluğunu kontrol edebilirsiniz.

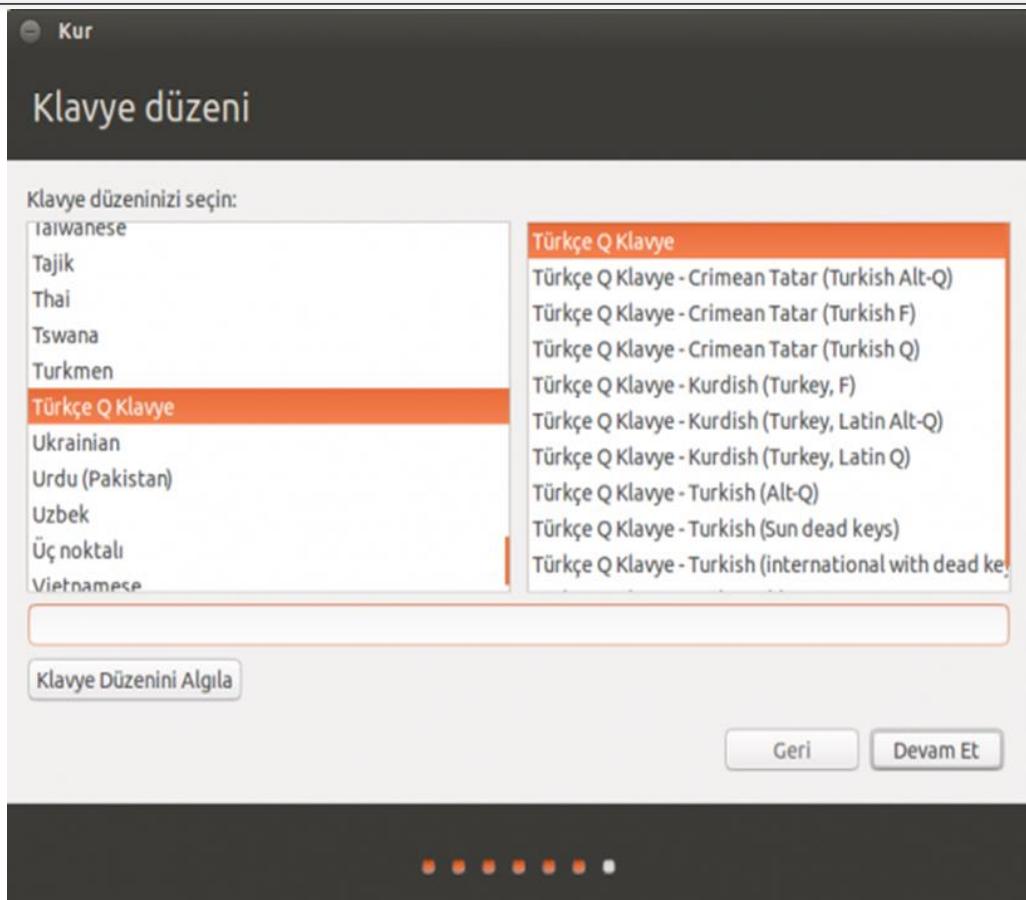


Figure 9 Klavye Düzen Ekranı

Bir sonraki adımda figür 10 de görülen "Kimsiniz?" bölümü gelecek, kullanıcı adı ve parola belirlenecektir. Bu aşamada bilgisayarımızın adını ve kullanıcı hesabımızın adını gireriz. Belirleyeceğimiz parola en az 6 karakter uzunluğunda olur ve harf, rakam, ?*- vb karakterler içerirse Ubuntu kurulumu parolamızı beğenir ve 'İyi Parola' der.

- **Otomatik giriş yap:** Ubuntu'yu her açığınızda kullanıcı parolanızı girmeden doğrudan giriş yapmak istiyorsanız bu seçeneği işaretleyebilirsiniz. Bu seçeneği işaretlemeseniz bile Ubuntu'yu kuruktan sonra Ayarlar üzerinden de otomatik kullanıcı girişini etkinleştrebilirsiniz.
- **Giriş yapmak için parola iste:** Üstteki seçeneğin tersidir. Öntanımlı olarak bu seçenek işaretli gelir.
- **Ev dizinimi şifrele:** Bu seçeneği işaretlenerek /Home isimli klasörün altındaki kullanıcı ev dizinimiz şifrelenerek bizim dışımızdaki erişimler için kontrol altına alınabilir.

Ardından 'Devam Et' düğmesini kullandığımızda 'Sistem Kuruluyor' mesajı ile gelen "Kur" ekranı, kurulumu tamamlamak üzere işleme başlar.

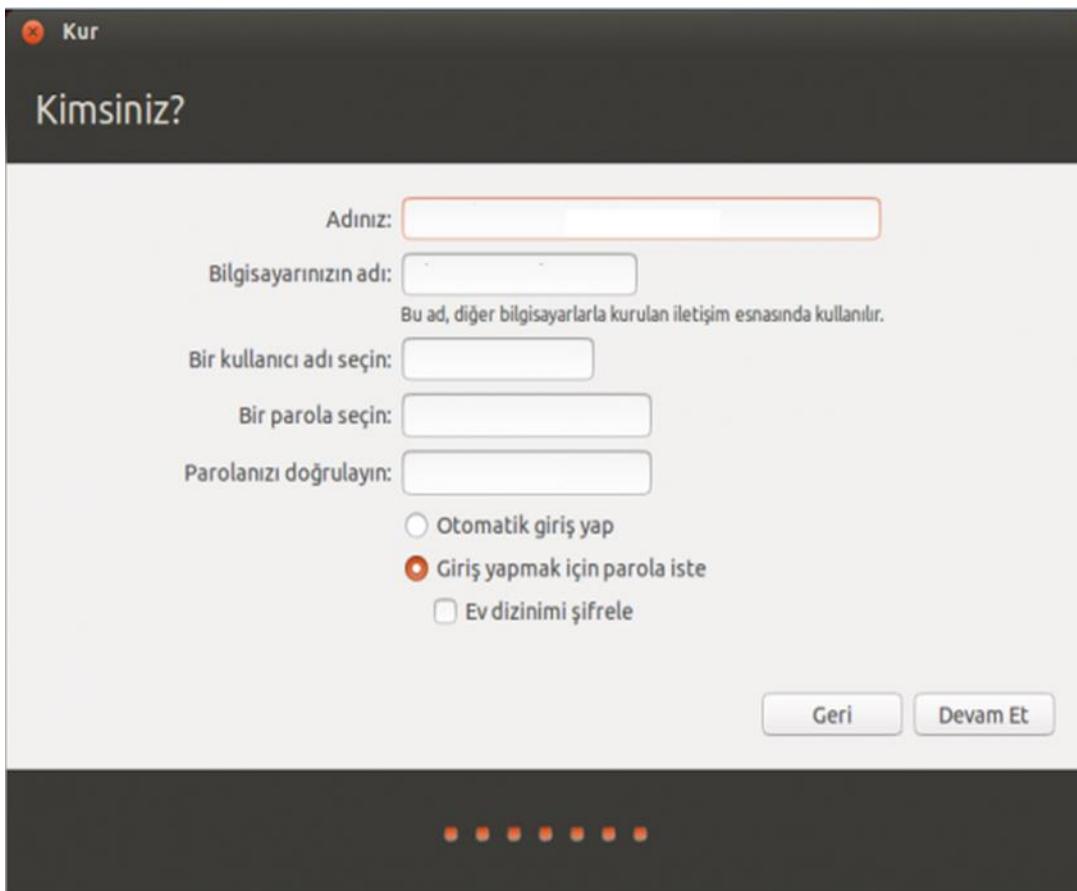


Figure 10 Giriş Ekranı

Bu işlemden sonra kurulumun bitmesini bekleyin. Kurulum tamamlandığında “Kurulum Tamamlandı” mesajını alırız. “Şimdi Yeniden Başlat” dedikten sonra artık Ubuntu'yu kullanmaya başlayabiliriz.

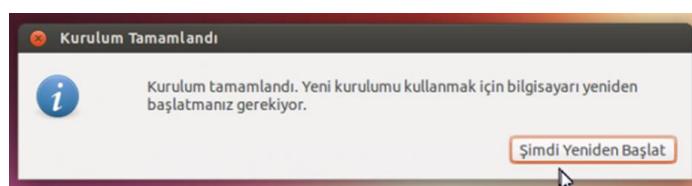


Figure 11 Kurulum Tamamlanması Mesajı

2.3. Ubuntu temelleri

2.3.1. Prompt (Komut İstemi)

Komut istemi Kullanıcıların girdiği komutları alan ve işlem yapması için işletim sistemine gönderen bir programdır. Komutların sonuçlarını ve çıktısını kendi üzerinde gösterebilir. Yani, işletim sistemi ile kullanıcı arasındaki bir köprüün olmasını sağlar. Komut istemine örnek olarak figür 13'de "yonetici" kullanıcısına ait komut istemi verilmiştir.

```
yonetici@yoneticipc:~$ whoami
yonetici
yonetici@yoneticipc:~$ hostname
yoneticipc
yonetici@yoneticipc:~$ lscpu
yonetici@yoneticipc:~$ uname
Linux
yonetici@yoneticipc:~$ uname -r
5.0.0-29-generic
```

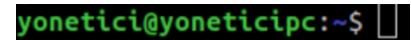


Figure 13 örnek Komut istemi (kullanıcı ve bilgisayar adı)

whoami: Sisteme giriş yaparken yazdığınız kullanıcı isminizi verir.

Hostname: Makinanın konak ismini verir.

lscpu: CPU, iş parçacığı, çekirdek, soket ve Düzungün Olmayan Bellek Erişimi (NUMA) düğümlerinin sayısını bildiren cpu bilgilerini gösterir.

uname: sistem bilgilerinin gösterildiği ekrandır burada belirli bilgileri yazdırmak için uname -e ek olarak:

-a, --all

- bilinmiyorsa omit -p ve -i hariç tüm bilgileri aşağıdaki sırayla yazdır.

-s, --kernel-name

- Kernel ismi yazılır.

-r, --kernel-release

- çekirdek sürümünü yazdır

-v, --kernel-version

- kernel versiyonu yazılır.

- m, --machine
 - makine donanım adını yazdır
- o, --operating-system
 - işletim sistemini yazdır
- help
 - yardım bölümünü gösterir ve çıkar
- version
 - sürüm bilgi çıktısı alır ve çıkar.

```
$ mkdir siirler  
$ cd siirler  
$ pwd  
/home/yonetici/siirler
```

mkdir: mkdir komutu yeni bir dizin oluşturmak için kullanılmaktadır.

Cd: “change directory” anlamına gelen bir dizine gitmek için kullanılan komuttur. büyük / küçük harfe duyarlıdır ve klasörün adını tam olarak yazılması gerekmektedir.

Figure 14 Örnek Uygulama

Pwd: hangi dosyanın içinde olduğu bilinmek istenirse bu komut kullanılabilir.

Örnek olarak figür 14 de kullanıcının “mkdir” ile siirler klasörü oluşturup “cd” komutu ile bu klasöre girdiği ve “pwd” ile kontrol ettiği görülmektedir.

2.3.2. Dosya indirme

Wget: dosya indirmek için kullanılan komuttur. Örnek bir uygulama için figür 15’te örnek bir indirme adresi verilmiştir.

```
$ wget http://188.132.181.140/otuzbes.txt  
$ wget http://188.132.181.140/handuvar.txt  
$ wget http://188.132.181.140/sessiz.txt
```

Figure 15 Örnek wget Uygulaması

2.3.3. Dosya listeleme

Ls: klasörde bulunan dosyaların listelenmesini sağlar. -1 komutu eklenerek ayrı satırlarda gösterilmesi sağlanabilir ve -l komutu ile detaylı bilgilere ulaşılabilir. Örnek uygulama figür 16'da gösterilmiştir.

```
$ ls  
handuvar.txt  otuzbes.txt  sessiz.txt  
  
$ ls -1  
handuvar.txt  
otuzbes.txt  
sessiz.txt  
  
$ ls -l  
-rw-r--r-- 1 yonetici yonetici 6662 Sep 23 01:46  
handuvar.txt  
-rw-r--r-- 1 yonetici yonetici 1163 Sep 21 02:09  
otuzbes.txt  
-rw-r--r-- 1 yonetici yonetici 528 Sep 23 01:49  
sessiz.txt
```

Figure 16 Örnek Ls Uygulaması

2.3.4. Dosya Görüntüleme

More: bir metin dosyasını bir seferde görüntülemek için kullanılır. Her seferinde 1 sayfa görüntülenir. Sonraki sayfaya geçmek için boşluğa basılır.

Less: more ile nerdeyse aynı olmakla birlikte “more” komutunda mümkün olmayan aşağı ve yukarı sayfaya geçiş yapılmasını sağlayan navigasyona sahiptir.

Cat(Concatenation): birden fazla dosyayı birleştirmek ve sonucu ekrana yazdırınmak için kullanılabilir.

Bu komutların kullanımı için örnek uygulama figür 17'ye bakılabilir.

```
$ more otuzbes.txt  
$ less handuvar.txt  
$ cat sessiz.txt  
$ cat otuzbes.txt handuvar.txt sessiz.txt  
$ cat *
```

Figure 17 Örnek Uygulama

2.3.5. Head ve Tale

Head: bir dosyanın ilk on satırını görüntüler. -n komutu ile istenilen satırdan sonraki bölümleri gösterir.

Tale: dosyanın son bölümünü görüntüler.

Kullanımları ile ilgili örnek uygulama figür 18'de gösterilmiştir.

```
$ head otuzbes.txt  
$ tale handuvar.txt  
$ head -n 5 sessiz.txt
```

Figure 18 Örnek Uygulama

2.3.6. Standart akışlar

Bilgisayar programlamada önceden belirlenmiş girdi ve çıktılar için standard giriş, standard çıkış ve standard hatayi belirten çıktılar bulunur. Standard input için "0", standard çıkış için "1" ve standard hata için "2" çıktısı belirlenmiştir. Bu sistem figür 19'da gösterilmiştir.

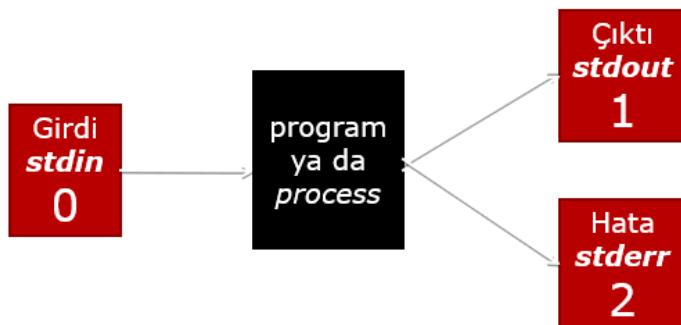


Figure 19 Standard Akış Sistemi

2.3.7. Yönlendirmeler

>: komuta sonucu çıktı sağlamak için kullanılır.

```
$ cat otuzbes.txt handuvar.txt >
ikisiir.txt

$ more ikisiir.txt
```

Figure 20 Örnek Uygulama

<: komuta girdi sağlamak için kullanılır.

```
$ cat < handuvar.txt
```

Figure 21 Örnek Uygulama

|: bir komutun sonucunu başka bir komutta kullanmak için kullanılır. Figür 22 ve figür 23'de nasıl kullanılacağı ve nasıl çalıştığı gösterilmiştir.

```
$ echo "Benim Siirim" | cat sessiz.txt -
```

Figure 22 Örnek Uygulama



Figure 23 Çalışma Sistemi

2.3.8. Düzenli ifadeler(regex)

Düzenli ifadeler karmaşık kalıpların eşleşmesinde kullanılan, verinin arayışına yardımcı özel karakterleri temsil eder.

| Sembol | Açıklama |
|--------|---|
| . | Herhangi bir karakterin yerini alır |
| ^ | string başlangıcıyla eşleşir |
| \$ | string sonuya eşleşir |
| * | Önceki karakterin sıfır veya daha fazla katıyla eşleşir |
| \ | Özel karakterleri temsil eder |
| () | Grup halindeki düzenli ifadeleri temsil eder |
| ? | yalnız bir karakterle eşleşir |

Örnek verecek olursa figür 24 ve figür 25 e bakılabilir.

```
$ grep "^B" sessiz.txt
```

Bicare gonuller! Ne giden son gemidir bu!
Bilmez ki giden sevgililer donmeyecekler.
Bir cok gidenin her biri memnun ki yerinden,
Bir cok seneler gecti; doneн yok seferinden.

Figure 24 Örnek Uygulama

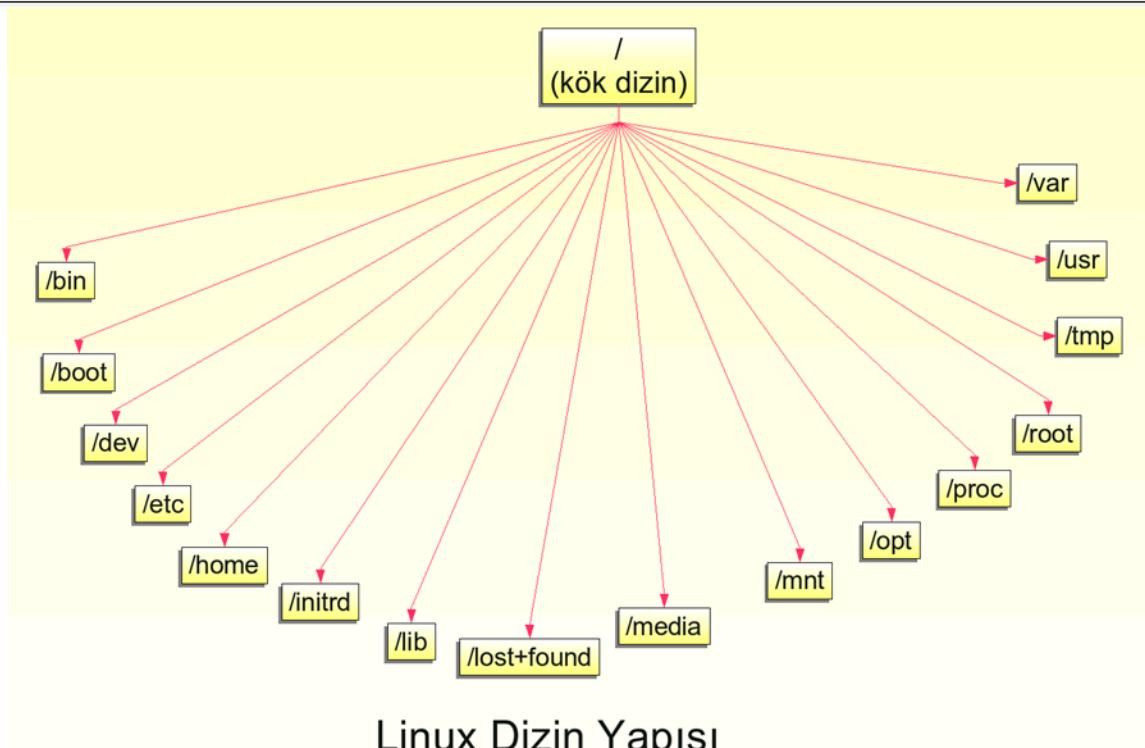
```
$ grep '!$' handuvar.txt
```

Yalniz arabacinin dudaginda bir islik!
Artik bahtin aciktir, uzun etme, arkadas!
Basucumda gordugum su satirlarla yandim!
Ey Marasli Seyhoglu, evliyalar adagi!
Bahtina lanet olsun asmadinsa bu dagi!
Donmeyen yolculara aglayan yasli yollar!

Figure 25 Örnek Uygulama

2.4. Linux Dosya Yapısı

Unix baş aşağı duran bir ağaç gibi düşünülebilir. Bu ağaçın en tepesinde “root” yani ağaçın kökü bulunmaktadır. Figür 26’da gösterildiği gibi bütün dizinler root'a dayanmaktadır.



Linux Dizin Yapısı

Figure 26 Linux Kök Dizin Yapısı

Root altındaki temel klasörler:

| Dizin | Açıklama |
|--------|---|
| /bin | Olması şart komut dosyalarını içerir |
| /boot | Başlangıç için gerekli dosyaları bulundurur |
| /dev | Donanım dosyaları vardır |
| /etc | Sistem ayarlarını barındırır |
| /lib | Kütüphane dosyaları ve kernel modülleri bulunur |
| /media | Kaldırılabilir aygıtların (CD-Rom, Flash bellek vs...) sisteme eklendiği klasördür. |
| /mnt | Bir dosya sistemini geçici olarak eklemek için kullanılır. |
| /opt | Ekstra programların kurulması içindir |

| | |
|-------|---|
| /sbin | Sistem yöneticiyle ilgili çalıştırılabilir dosyaları tutar. |
| /srv | Sistemin sunduğu hizmetlerle alakalıdır |
| /tmp | Geçici dosyaları tutmak içindir |
| /usr | İkincil bir hiyerarşî |
| /var | Değişken verileri saklar |

2.4.1. Sudo

Sudo: “super user do” anlamına gelen root olmayan kullanıcıların normalde süper kullanıcı izinleri gerektiren komutları çalıştırmasına izin verir.

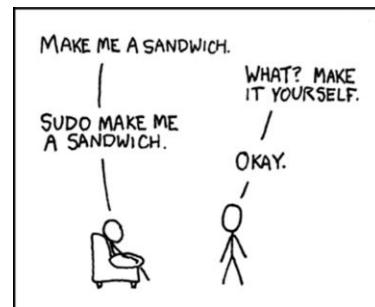


Figure 27 Sudo Nedir?

2.4.2. Kullanıcılar

Root: süper kullanıcı olarak tanımlanır. Sistemdeki süper kullanıcı tanımlı bir kök adıdır. Bu kullanıcının sistemde herhangi bir değişiklik yapma hakkı vardır. Sudo komutu, getirdiği güvenlik risklerini ortadan kaldırmak ve önce bazı işlemleri kök düzeyinde çalıştırırmak için kullanılır.

Sisteme yeni bir kullanıcı eklemek için **useradd** veya **adduser** komutları kullanılabilir. **Adduser** komutunun kullanılması önerilir çünkü yüksek düzeyde çalışır. (Örnek için figür 28'e bakınız.)

```
$ man adduser
```

Figure 28 Örnek adduser Uygulaması

2.4.3. Dosya izinleri

Her dosyanın bir sahibi (owner) bir de grubu (group) vardır.

Bir dosya için şu kullanıcılar için haklar tanımlıdır:

- User (u)
- Group (g)
- Others (o)

Tanımlanabilir haklar şunlardır:

- Read (okuma) – 4
- Write (yazma) – 2
- Execute (çalıştırma) - 1

Örnek olarak “4:2:1” verildiğinde (r:w:x) ifadesi tanımlanmış olur yani kullanıcı okuma yazma ve çalışma yetkilerine sahip olur. Örnek uygulama figür 29’da verilmiştir.

```
$ ls -l  
-rw-r--r-- 1 yonetici yonetici 6662 Sep 23 01:46  
handuvar.txt  
  
-rw-r--r-- 1 yonetici yonetici 1163 Sep 21 02:09  
otuzbes.txt  
  
-rw-r--r-- 1 yonetici yonetici 528 Sep 23 01:49  
sessiz.txt  
  
$ chmod go-w handuvar.txt  
$ chmod u+x otuzbes.txt
```

Figure 29 Örnek İzin Uygulaması

2.4.4. Prosesler

ps: Processes anlamına gelir. Linux sistem üzerinde çalışan işlemleri listeler.

ps -aux | grep <işlem_ismi>: Linux üzerinde çalışan belirli işlemi veya işlemleri getirir.

kill -9 <İşlem_ID'si>: Linux üzerinde çalışan ve üstteki komut ile ID'si bulunan işlemi öldürmeyi sağlar. İlk çalışan proses (1 numaralı) init'dir.

2.4.5. Paket kurulumları

wget '<indirilecek_url>' : Bilgisayarda bulunulan klasöre, internet üzerinde belirtilen dosyayı indirir.

sudo apt-get install <paket_ismi>: Repository'lerde paketi arar ve bilgisayara kurar.

sudo apt-get remove <paket_ismi>: Bilgisayarda paketi arar ve bilgisayardan siler.

sudo apt-get update: sources.list dosyasında kayıtlı olan repositorylerin bilgisini bilgisayara alır.

sudo apt-get upgrade: Bilgisayarda bulunan paketlerin güncellemesini yapar.

apt-cache search <paket_ismi>: İlgili paketi repositorylerde arar ve getirir.

sudo chmod <izin_tipi> <klasor_veya_dosya_ismi>: İlgili klasör veya dökümana dosya izinleri verir. İzin tipi olarak 777 kullanılması, Read-Write-Execute anlamına gelir.

2.4.6. Servisler

sudo service <servis_ismi> start: Linux bünyesinde çalışan servisi başlatır.

sudo service <servis_ismi> stop: Linux bünyesinde çalışan servisi sonlandırır.

sudo service <servis_ismi> restart: Linux bünyesinde çalışan servisi yeniden başlatır.

Başlatılabilir servisler:

- FTP
- Web server (NGINX, Apache)
- Database'ler (MySql, PostgreSQL)
- SSHD (Uzak Bağlantı)

2.4.7. Metin Editörleri

Metin editörleri kod yazmak, yapılandırma dosyaları gibi metin dosyalarını düzenlemek, kullanıcı talimat dosyaları oluşturmak ve daha pek çok şey için kullanılabilir. Linux'ta metin düzenleyici, grafik kullanıcı arabirimi (GUI) ve komut satırı metin düzenleyicileri (CLI) olmak üzere iki türdür.

CLI tabanlı:

- pico, nano
- vi, vim
- Emacs

GUI tabanlı:

- pico, nano
- vi, vim
- Emacs

2.4.8. Linux terminali

Kullanıcıların komut yazabileceği ve metin yazdırabileceği bir arayüz sağlar. Linux sunucunuza SSH yaptığınızda, yerel bilgisayarınızda çalıştırığınız ve komutları yazdığınız program bir terminaldir. Terminal için bazı kısa yollar:

Terminal: Ctrl + Alt + T

Kopyalama: Ctrl + Shift + C

Yapıştır: Ctrl + Shift + V

İşlemi Durdurma: Ctrl + C

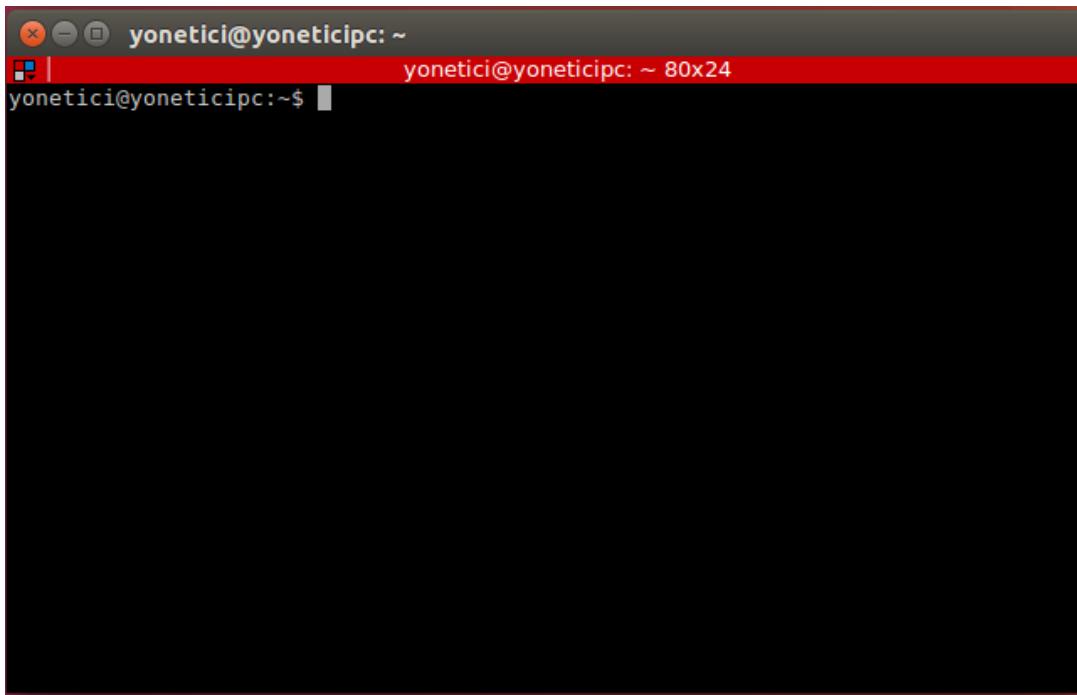


Figure 30 Terminal Penceresi

Figür 30'da gösterilen örnek komut pencersinde “`yonetici@yoneticipc:`” yazmaktadır. **Yönetici** kullanıcı adını temsil ederken **yoneticipc** bilgisayar adını temsil etmektedir.

2.5. Python Programlamaya giriş

Python kodu çalıştırırmak istenirse terminalden doğrula dosyanın olduğu dizine gidilir, kodu çalıştırırmak için "python dosya_adı.py" yazılabilir ama ROS konusunda üretken olmak için programlama dilinin bilinmesi şarttır. Temel terimler figür 31'de verilmiştir.

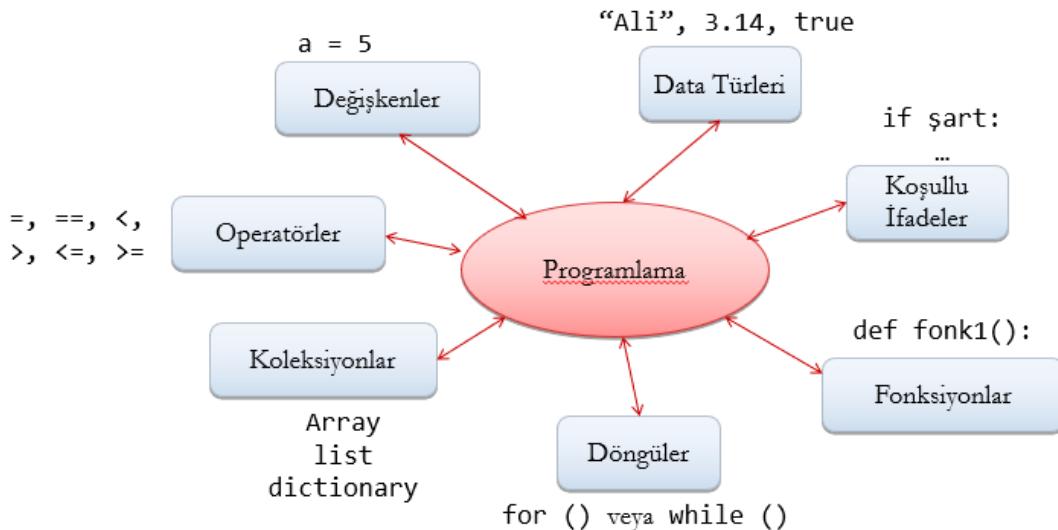


Figure 31 Temel Programlama Terimleri

2.5.1. Değişkenler

Örnek olarak değişkenler:

- a = 5
- b = 'Deneme'
- c, d = 10, 'Deneme2'

2.5.2. Data türleri

Data türleri olarak tam sayılar (integer), ondalıklı sayılar (floating), metinler (string) ve doğru/yanlış (bool) bulunur. Bunlar:

Integer: 0,1,2,3,...,-1,-2,-3....

Float: 1.234, 0.94, 1009.75, -100.32 ...

String: "Ali", "Beşiktaş", "Türkiye"

Bool: True, False

Olarak örneklendirilebilir.

2.5.3. Koşullu ifadeler

Koşul durumlarını belirtmek için üç adet deyimden yararlanmaktadır:

- if
- elif
- else

Koşullu ifadelerin kullanım şekli aşağıda verilmiştir.

```
if koşul_1:  
    sonuç_1  
elif koşul_2:  
    sonuç_2  
elif koşul_3:  
    sonuç_3  
else:  
    sonuç_4
```

2.5.4. Fonksiyonlar

Basit bir fonksiyon tanımlanıçak olursa:

```
def basitFonksiyon():  
    print('Deneme')
```

tanımlanan fonksiyon aşağıdaki şekilde çağrıralırsa:

```
basitFonksiyon()
```

Terminal ekranına “Deneme” yazısı görülür. Başka bir toplama işlemi tanımlanırsa:

```
def toplama(a, b):  
    sonuc = a + b  
    return sonuc
```

Bu fonksiyonun çağrılmaması:

```
sonuc = toplama(5, 10)
```

Şeklinde olacaktır.

2.5.5. Döngüler

Diğer programlama dillerinin çoğundaki döngüler, programlayıcının başlangıç ve bitiş noktalarını belirleyebileceğی sayısal bir sıradan geçerken, döngüler için Python'da sıralı herhangi bir veri türünde yineleme ile gerçekleştirilir. Örnek döngüler ve kullanım şekilleri:

```
while koşul:  
    ...  
  
for i in range(3, 20):  
    print(i)  
  
for harf in metin:  
    print(harf)
```

Olarak verilebilir. Başka bir uygulama yapılacak olursa:

```
parola = input("parola giriniz: ")
```

```
if not parola:  
    pass
```

Bu uygulamada eğer kullanıcı parolayı boş geçerse pass komutu sayesinde hiçbir şey yapamayacağı anlamına gelir. Break komutu için:

```
while True:  
    sayı = input("Bir sayı girin: ")  
    if sayı == "iptal":  
        break
```

Eğer kullanıcı iptal yazarsa döngüden çıkarılır. Continue komutu için:

```
for val in "string":  
    if val == "i":  
        continue
```

Continue komutu, koşulu sağladığı durumda döngüden çıkmadan atlamak için kullanılmaktadır.

2.5.6. Koleksiyonlar

Liste: Bir liste tanımlanırsa:

```
Liste = [50, 60, 'Merhaba', True, 99]
```

Bu listenin çıktıları:

```
print(Liste[2])          # 'Merhaba'  
print(Liste[1:3])        # 60, 'Merhaba'  
print(len(Liste))        # 5
```

Olacaktır. `liste.append("...")` ile yeni eleman eklenebilir ve `liste.sort()` ile sıralama yapılabilir.

Sözlük:örnek bir sözlük yazılsrsa:

```
sozluk = {  
    'isim': 'Ayse',  
    'sehir': 'Eskisehir',  
    'yas': 22  
}
```

Ayşe ismini yazdırırmak için:

```
print(sozluk['isim'])      # Ayse
```

Komutu kullanılabilir.

2.5.7. Operatörler

Python'da temelde 7 adet operatör bulunmaktadır. Bunlar:

| Operatör | Açıklama |
|----------|------------------------|
| < | küçüktür |
| <= | küçüktür ya da eşittir |
| > | büyüktür |
| >= | büyüktür ya da eşittir |
| == | eşittir |
| != | eşit değildir |
| is | nesne eşitliği |

| | |
|--------|------------------------|
| is not | olumsuz nesne eşitliği |
|--------|------------------------|

2.5.8. Asal sayı bulma uygulaması

```

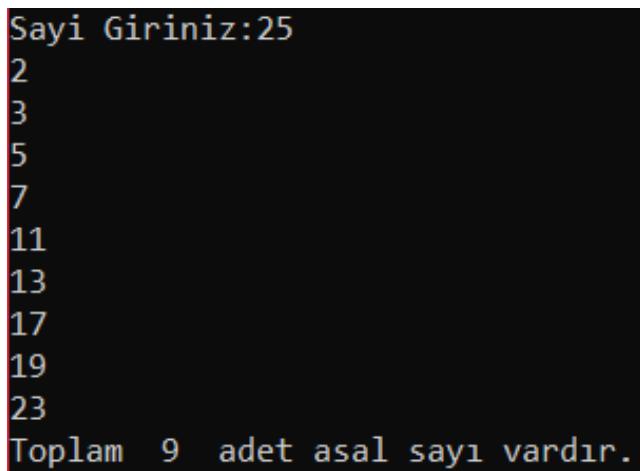
sayı = int(input("Sayı Giriniz:"))
sayac = 0

for i in range(2, (sayı + 1)):
    kontrol = True
    for j in range(2, i):
        if(i % j == 0):
            kontrol = False
            break
    if kontrol:
        print(i)
        sayac += 1

print("Toplam ", sayac, " adet asal sayı vardır.")

```

Çıktı olarak figür 32'de örnek bir çıktı gösterilmiştir.



```

Sayı Giriniz:25
2
3
5
7
11
13
17
19
23
Toplam 9 adet asal sayı vardır.

```

Figure 32 Örnek Asal Sayı Bulma uygulaması

2.5.9. Plot uygulaması

Matplotlib kütüphanesini indirmek için:

- Pip install matplotlib

Pip indirmek için:

- sudo apt install python-pip

komutları yazılmalıdır.

```
import matplotlib.pyplot as plt      # matplotlib kütüphanesi
import etme

x = [1, 2, 3]                      # x ekseni degerleri
y = [2, 4, 1]                      # y ekseni degerleri

plt.plot(x, y, "--r", label = "Line 1") # noktalarla plot
cizmek icin

plt.xlabel('X - Ekseni')            # x ekseni ismi
plt.ylabel('Y - Ekseni')            # y ekseni ismi
plt.title('Plot Ornegi')           # plota baslik eklemek
icin

plt.show()                          # plot gostermeye fonksiyonu
```

Bu uygulamada figür 33'de verilen şekilde bir çıktı beklenmektedir.

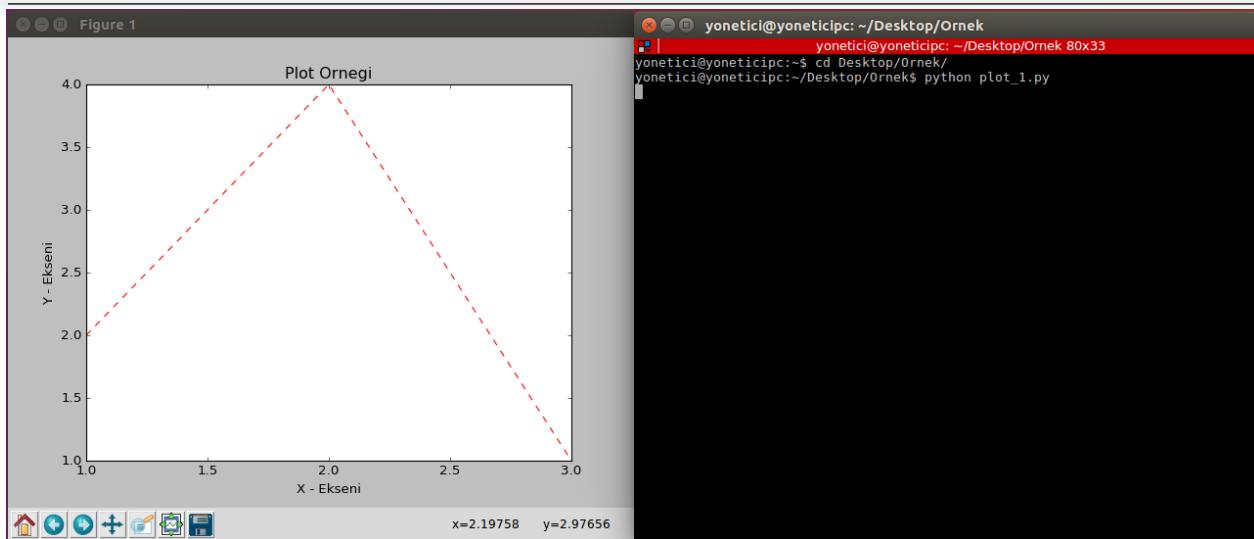


Figure 33 Örnek Plot Uygulaması

2.5.10. İkili plot uygulaması

Aşağıdaki kod eklenirse:

```
import matplotlib.pyplot as plt

x1 = [1,2,3]
y1 = [2,4,1]
plt.plot(x1, y1, "--r", label = "line 1")

x2 = [1,2,3]
y2 = [4,1,3]
plt.plot(x2, y2, "-.g", label = "line 2")

plt.xlabel('x - Eksen')
plt.ylabel('Y - Eksen')
plt.title('İkili Plot Ornegi')

plt.legend()
```

plt.show()

Bu uygulamada figür 34'de verilen şekilde bir çıktı beklenmektedir.

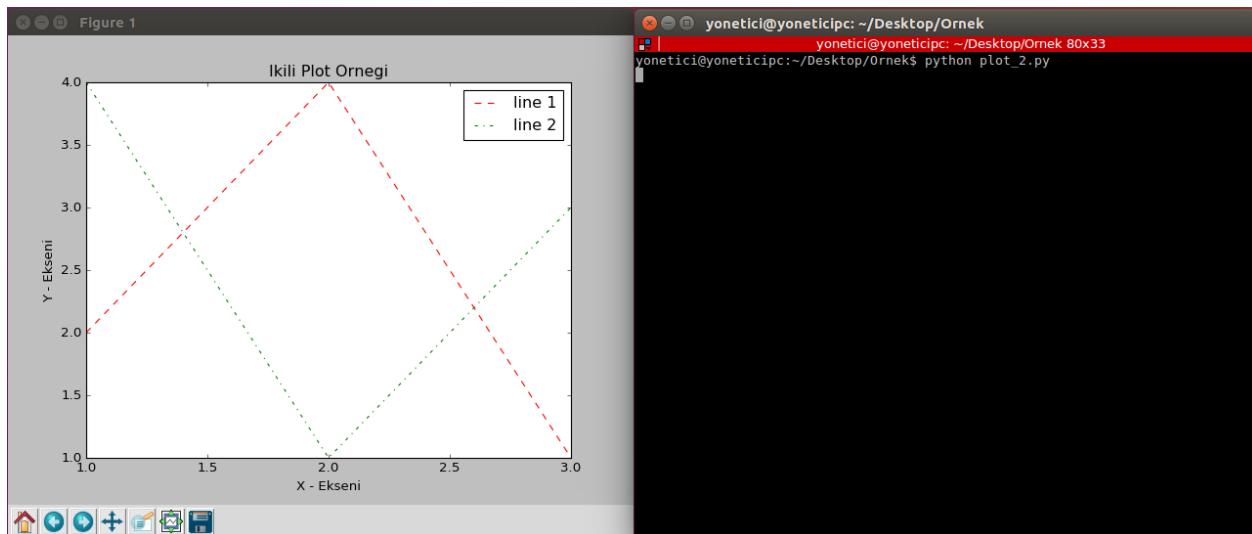


Figure 34 Örnek İkili Plot Uygulaması

3. ROS Uygulamaları

3.1. ROS Nedir – Ne Değildir?

ROS açılımı Robot Operating System açık kaynaklı bir meta işletim sistemidir. Donanımdan soyutlama, düşük seviyeli aygit kontrolü, yaygın olarak kullanılan işlevselligi gerçekleştirmeye, prosesler arasında mesajlaşma ve paket yönetimi işlemlerinin yapılabildiği bir robotik arakatman görevi görmektedir. ROS'un haberleşme topolojisi ucbirimden ucbirime ağ yapısına dayanmaktadır. Bu sayede heterojen bir ağ ile birbirine bağlı olan bilgisayarların haberleşmesinde ortaya çıkan yavaşlıktan kaynaklı veri kayıpları önlenmektedir. ROS yapı olarak düğüm, mesaj, konu ve servis olmak üzere dört ana konseptten oluşmaktadır. Düğüm, proseslerin gerçekleştirildiği yerdir. Bu düğümler arasındaki haberleşme mesajlar üzerinden gerçekleşmektedir. Mesajlar konular üzerinden akmaktadır. Konular yayinci/abone yapısında haberleşmektedir. Bundan dolayı konular asenkron haberleşme sağlamaktadır. Senkron bir haberleşme için ise konuların yerine servisler kullanılmaktadır.

ROS:

- Bir programlama dili değildir.
- Bir kütüphane değildir.
- İşletim sistemi değildir.
- Entegre geliştirme ortamı değildir.
- Bünyesinde servisler ve çeşitli makroları çalıştırın açık kaynaklı robotlar için meta işletim sistemidir.

3.2. Neden ROS Kullanmalıyız?

ROS bünyesinde bir çok faydalı özelliği bulundurmaktadır. Bunlara örnek verilirse:

- Dil bağımsız yapı (C++, Python, Lisp, Java, Lua)
- Modüler çalışma, parametreler, mesajlar ve servisler sayesinde programlara ani müdahale imkanı
- Node/topic ile kolay ve sistematik veri aktarımı
- Birçok sensör, motor ve robot platformu için sürücü desteği
- Açık kaynak kodlu
- Haritalama, konumlandırma ve algılama amaçlı hazır algoritma, kütüphane ve paketler
- Aktif topluluk
- Hızlı test etme
- Donanım soyutlaması
- Görselleştiriciler

Oluđu söylenebilir.

3.3. ROS Sensörler ve ROS Kullanan Firmalar

Ros bünyesinde birçok sensör kullanma imkanı sunar.

ROS tarafından desteklenen farklı sensörler bulunmaktadır. Bunlar:

- tek boyutta mesafe bulucular (1D telemetre)
- 2D telemetre
- 3D Sensörler (telemetre ve RGB-D kameralar)
- Ses / Konuşma Tanıma sensörleri
- Çevre sensörleri (örn. ultrasonik rüzgar sensörü)
- Kuvvet / Tork / Dokunmatik Sensörler
- Hareket yakalama
- Poz Tahmini (GPS / IMU)
- Güç kaynağı
- RFID okuyucu
- Radar sensörleri



Figure 35 Boston Dynamics Atlas Robot

Günümüzde birçok firma robot gelişiminde ROS arakatmanı kullanmaktadır. Bu şirketlerden bazıları:

- ABB Robotics
- Clearpath Robotics
- Doosan Robotics
- Fanuc
- Fetch Robotics
- iRobot
- Sony (Aibo robot dog)



Figure 36 EvaRobot

Inovasyon Mühendislik Bünyesinde ürettiği bütün robotlar ROS uyumlu olarak çalışmaktadır. Bu robotlar Arasında akıllı tekerlekli sandalye (ATEKS) eğitim ve araştırma robotu (EvaRobot) ve endüstriyel kullanım için ürettiği robotlar vardır.

3.4. ROS Kurulumu

ROS kurulumunun genellikle debian paketleri ile yapılması tavsiye edilmektedir. Debian paketlerinin alternatifi ROS kaynak kodu kullanılarak kurulumun yapılmasıdır fakat bu tavsiye edilmemektedir. ROS Kinetic Kame dağıtımları Xenial (Ubuntu 16.04) ve Willy (Ubuntu 15.10) sürümlerini desteklemektedir.

Supported:



Ubuntu Willy amd64 i386
Xenial amd64 i386 armhf arm64

Source installation

Experimental:



OS X (Homebrew)



Gentoo



OpenEmbedded/Yocto



Debian Jessie amd64 arm64

Unofficial Installation Alternatives:



Single line install ROS Kinetic on Ubuntu

Figure 37 Platform Seçim ekranı

| | | | |
|-----------------------------------|---------------------|---|--------------------------|
| ROS Melodic Morenia (Recommended) | May 23rd, 2018 |  | May, 2023 (Bionic EOL) |
| ROS Lunar Loggerhead | May 23rd, 2017 |  | May, 2019 |
| ROS Kinetic Kame | May 23rd, 2016 |  | April, 2021 (Xenial EOL) |
| ROS Jade Turtle | May 23rd, 2015 |  | May, 2017 |
| ROS Indigo Igloo | July 22nd, 2014 |  | April, 2019 (Trusty EOL) |
| ROS Hydro Medusa | September 4th, 2013 |  | May, 2015 |
| ROS Groovy Galapagos | December 31, 2012 |  | July, 2014 |
| ROS Fuerte Turtle | April 23, 2012 |  | – |
| ROS Electric Emys | August 30, 2011 |  | – |
| ROS Diamondback | March 2, 2011 |  | – |
| ROS C Turtle | August 2, 2010 |  | – |
| ROS Binx Turtle | March 2, 2010 |  | – |

Figure 38 Sürüm Seçim Ekranı

Kurulum için verilen adresten sürüm figür 37 ve Figür 38 takip edilerek ROS kinetic Kame seçilerek yüklenebilir.

<http://wiki.ros.org/kinetic/Installation>

Eğer kaynak koddan kurulum yapılmak istenirse aşağıdaki bağlantı kullanabilir:

<http://wiki.ros.org/kinetic/Installation/Source>

Bu aşamada yapılacaklar aşağıda verilmiştir.

Bilgisayarın, settings.ros.org adresinden yazılımı kabul edecek şekilde ayarlanması:

```
sudo sh -c 'echo "deb  
http://packages.ros.org/ros/ubuntu  $(lsb_release -  
sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Anahtarları ayarlama:

```
sudo apt-key adv --keyserver  
'hkp://keyserver.ubuntu.com:80' --recv- key  
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Debian paketinin güncellliğinden emin olma:

```
sudo apt-get update
```

Tam sürüm kurulum yapma ROS,rqt,rviz,robot-generic kütüphaneler, 2D/3D simülatörler ...):

```
sudo apt-get install ros-kinetic-desktop-full
```

Rosdep in başlatılması ve güncellenmesi:

```
sudo rosdep init  
rosdep update
```

Ortam kurulumu:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

Derleme paketleri için bağımlıkların yüklenmesi:

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

3.5. Linux Temel Kodlar

Linux terminal ile ilgili temel kodlar:

- Yeni terminal açma: Ctrl + Alt + T
- yonetici@yoneticipc:
 - yonetici: kullanıcı adı
 - yoneticipc: bilgisayar adı
- Kopyalama: Ctrl + Shift + C
- Yapıştır: Ctrl + Shift + V
- İşlemi Durdurma: Ctrl + C

Ayrıca bir önceki konuda değinilen ve ROS üzerinde işlem yaparken lazım olabilecek Linux temel komutları aşağıda verilmiştir:

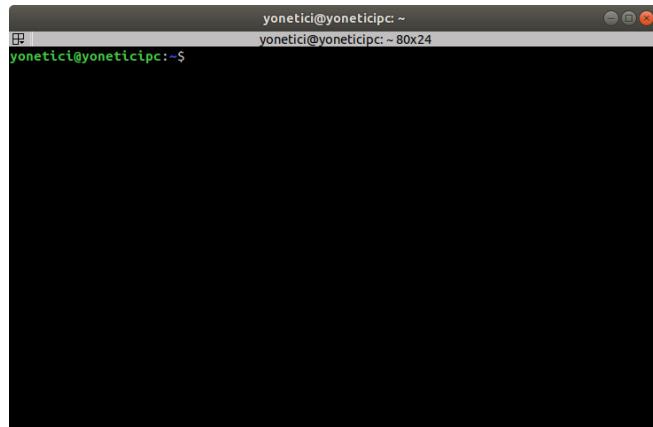


Figure 39 Linux Terminal Ekranı

- **cd** : Change Directory anlamına gelir. Belirtilen yoldaki klasöre giriş yapılır.
- **ls** : List anlamına gelir. Girilen klasördeki dosya ve dökümanları listeler.
- **mkdir <klasor_adi>**: Make Directory anlamına gelir. Bulunulan yolda klasör oluşturur.
- **mkdir -p <klasore_giden_yol>/<klasor_adi>**: Klasör oluşturma işlemini tüm yol üzerinde gerçekler. Eğer olmayan klasörler varsa iç içe o klasörleri oluşturur.
- **rm <silinecek_dosya>**: Remove anlamına gelir. Bir dosyayı silmek için kullanılır.
- **rm -rf <silinecek_dosya>**: Recursive Remove anlamına gelir. Birden fazla dosyayı silmek için kullanılır.
- **mv <tasinacak_dosya> <tasinacak_klasor>**: Dosya taşıma işlemi yapar. Rename işlemi de mv komutu kullanılarak yapılabilir.

- **cp <kopyalanacak_dosya> <kopyalanacak_klasor>**: Kopyalama işlemi yapar.
- **wget '<indirilecek_url>'** : Bilgisayarda bulunulan klasöre, internet üzerinde belirtilen dosyayı indirir.
- **cd** : Change Directory anlamına gelir. Belirtilen yoldaki klasöre giriş yapılır.
- **ls** : List anlamına gelir. Girilen klasördeki dosya ve dökümanları listeler.
- **mkdir <klasor_adi>**: Make Directory anlamına gelir. Bulunulan yolda klasör oluşturur.
- **mkdir -p <klasore_giden_yol>/<klasor_adi>**: Klasör oluşturma işlemini tüm yol üzerinde gerçekler. Eğer olmayan klasörler varsa iç içe o klasörleri oluşturur.
- **rm <silinecek_dosya>**: Remove anlamına gelir. Bir dosyayı silmek için kullanılır.
- **rm -rf <silinecek_dosya>**: Recursive Remove anlamına gelir. Birden fazla dosyayı silmek için kullanılır.
- **mv <tasinacak_dosya> <tasinacak_klasor>**: Dosya taşıma işlemi yapar. Rename işlemi de mv komutu kullanılarak yapılabilir.
- **cp <kopyalanacak_dosya> <kopyalanacak_klasor>**: Kopyalama işlemi yapar.
- **wget '<indirilecek_url>'** : Bilgisayarda bulunulan klasöre, internet üzerinde belirtilen dosyayı indirir.
- **history**: Terminaldeki kod geçmişini gösterir.
- **clear**: Terminaldeki kodları siler.
- **ps** : Processes anlamına gelir. Linux sistem üzerinde çalışan işlemleri listeler.
- **ps -aux | grep <islem_ismi>**: Linux üzerinde çalışan belirli işlemi veya işlemleri getirir.
- **kill -9 <islem_ID'si>** : Linux üzerinde çalışan ve üstteki komut ile ID'si bulunan işlemi öldürmeyi sağlar.
- **udo lsusb** : Linux sistem üzerinde kayıtlı ve çalışan USB cihazlarını listeler.
- **cat** : Bir dosyanın içeriğini terminal ekranına bastırır.
- **pwd** : Bir dosyanın yolunu terminal ekranına bastırır.
- **echo <degisken>** : Linux terminali üzerinde kayıtlı global değişkenlerin ve sonradan tanımlanan değişkenlerin ekrana bastırılmasını sağlar.
- **<editör_ismi> <dosya>** : Linux'da bir dosyayı düzenlemeyi sağlar.
- **setxkbmap tr** : Klavyeyi türkçe karakterli hale getirir.

- **`g++ hello_world.cpp -o hello_world`** : GNU C Compiler aracılığı ile C/C++ dosyalarını derler.
- **`./hello_world`** : Executable dosya çalıştırma

3.6. ROS Mimarisi Giriş

ROS mimarisi 3 bölüme ayrılmıştır:

- **Dosya Sistemi (Filesystem Level)**: ROS'un dahili olarak nasıl oluşturulduğunu, klasör yapısını ve çalışması gereken minimum dosya sayısını açıklamak için bir grup kavramı içerir.
- **İşlem Grafiği (Computation Graph Level)**: ROS'un işlemler ve sistemler arasındaki iletişimini nasıl kullandığını açıklayan kavramları içerir.
- **Topluluk (Community Level)**: Geliştiriciler arasında bilgi, algoritma ve kod paylaşmaya yönelik bir dizi araç ve kavram içerir.

3.7. Dosya Sistemi – File System Level

Ros paket sistemi 5 bölümden oluşmaktadır:

- Paketler (Packages)
- Metapaketler (Metapackages)
- Paket bildirileri (Package Manifests)
- Mesaj tipleri (Message types)
- Servis tipleri (Service types)

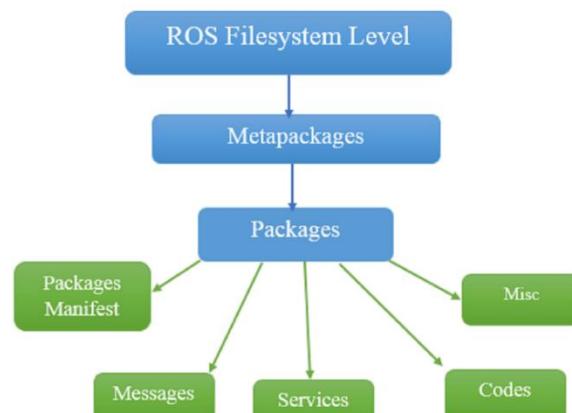


Figure 40 ROS Dosya Sistemi

3.7.1. Paketler (Packages)

Bir paket, işlemleri (düğümleri), ROS'a bağlı kütüphaneleri, veri kümelerini, yapılandırma dosyalarını veya faydalı herhangi bir şeyi içerebilir. Paketlerin oluşturulma amacı kodları ufak parçalara bölgerek yeniden kullanılabilir olmasını

sağlamaktır. Paketler, ROS'ta yazılımın düzenli olmasını sağlayan ana birimlerdir. Paketler genellikle tipik dosya ve klasörlerden oluşur:

- **include/paket_adı/**: Bu dizinde, ihtiyaç duyduğumuz kütüphanelerin (libraries) başlık (header) dosyaları bulunur.
- **msg/**: Standart olarak bulunan mesaj tiplerinden farklı bir mesaj tipi oluşturulacaksa bu klasörün içinde oluşturulmalıdır.
- **scripts/**: Python gibi direk çalıştırılabilir betik (script) dilleri ile yazılan kodlar bu klasör içinde yer almmalıdır.
- **src/**: Programların kaynak dosyalarının (source files) bulunduğu yerdir.
- **srv/**: Servis (service) dosyalarının bulunduğu yerdir.
- **CMakeLists.txt**: Derleyiciye verilen emirleri ve daha birçok yapı bilgisini içeren CMake derleme (built) dosyasıdır.
- **package.xml**: Paketlerin bildiri dosyasıdır.

ROS paketler oluşturmamıza, düzenlememimize ve paketler ile çalışmamıza yardımcı olabilecek araçlara sahiptir:

- **rospack**: Paketleri bulma ve paketler hakkında bilgi edinmeye yarar.
- **roscreate-pkg**: Yeni paket oluşturmaya yarar.
- **rosmake**: Paketleri derlemeye (compile) yarar.
- **rosdep**: Paketlerin bağımlılıklarını (dependencies) yüklemeye yarar.
- **catkin_create_pkg**: Yeni paket oluşturmaya yarar.

ROS'un paketler ile paketlerin klasörleri ve dosyaları arasında dolaşmamızı ve taşıma yapmamızı sağlayan rosbash isimli bir paketi vardır. rosbash aracında desteklenen bazı komutlar:

- **rospack**: Paketleri bulma ve paketler hakkında bilgi edinmeye yarar.
- **roscreate-pkg**: Yeni paket oluşturmaya yarar.
- **rosmake**: Paketleri derlemeye (compile) yarar.
- **rosdep**: Paketlerin bağımlılıklarını (dependencies) yüklemeye yarar.
- **catkin_create_pkg**: Yeni paket oluşturmaya yarar.

Paket mutlaka CMakeLists.txt dosyasını içermelidir. Bu dosya catkin e kodların nasıl ve nereye yükleneceğini bildirir. CMakeLists.txt dosyası aşağıdaki formatı izlemelidir, aksi halde paketler doğru şekilde oluşturulamaz.

- **Required CMake Version**
(cmake_minimum_required)
- **Package Name** (project())
- **Find other CMake/Catkin packages needed for build**
(find_package())
- **Enable Python module support**
(catkin_python_setup())
- **Message/Service/Action**
- **Generators** (add_message_files(), add_service_files(), add_action_files())
- **Invoke message/service/action generation** (generate_messages())
- **Specify package build info export** (catkin_package())
- **Libraries/Executables to build**
(add_library()/add_executable()/target_link_libraries())
- **Tests to build** (catkin_add_gtest())
- **Install rules** (install())

```

Satır numaralandırımı açıkla

1 # Get the information about this package's buildtime dependencies
2 find_package(catkin REQUIRED
3   COMPONENTS message_generation std_msgs sensor_msgs)
4
5 # Declare the message files to be built
6 add_message_files(FILES
7   MyMessage1.msg
8   MyMessage2.msg
9 )
10
11 # Declare the service files to be built
12 add_service_files(FILES
13   MyService.srv
14 )
15
16 # Actually generate the language-specific message and service files
17 generate_messages(DEPENDENCIES std_msgs sensor_msgs)
18
19 # Declare that this catkin package's runtime dependencies
20 catkin_package(
21   CATKIN_DEPENDS message_runtime std_msgs sensor_msgs
22 )
23
24 # define executable using MyMessage1 etc.
25 add_executable(message_program src/main.cpp)
26 add_dependencies(message_program ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
27
28 # define executable not using any messages/services provided by this package
29 add_executable(does_not_use_local_messages_program src/main.cpp)
30 add_dependencies(does_not_use_local_messages_program ${catkin_EXPORTED_TARGETS})

```

Figure 41 CMakeList.txt

3.7.2. Metapaketler (Metapackages)

Metapaketler paketleri organize olarak çalıştırılmaya yarar (çoklu paket kümесini basitçe grupperdir). Meta paketler, ROS'ta yalnızca package.xml dosyası içeren özel paketlerdir.

Örnek: robot metapaketinin içeriği **paketler**: [control_msgs, diagnostics, executive_smach, filters, geometry, joint_state_publisher, kdl_parser, kdl_parser_py, robot_state_publisher, urdf, urdf_parser_plugin, xacro]

```
sudo apt-get install ros-$distro-robot
```

```
sudo apt-get install ros-$distro-actionlib ros-$distro-angles ros-$distro-bond_core ros-$distro-catkin ros-$distro-class_loader ros-$distro-cmake_modules ros-$distro-common_msgs ros-$distro-console_bridge ros-$distro-control_msgs ros-$distro-diagnostics ros-$distro-dynamic_reconfigure ros-$distro-executive_smach ros-$distro-filters ros-$distro-genlisp ros-$distro-gennodejs ros-$distro-genpy ros-$distro-geometry ros-$distro-message_generation ros-$distro-message_runtime ros-$distro-nodelet_core ros-$distro-pluginlib ros-$distro-robot_model ros-$distro-robot_state_publisher ros-$distro-ros ros-$distro-ros_comm ros-$distro-rosbag_migration_rule ros-$distro-rosconsole_bridge ros-$distro-roscpp_core ros-$distro-rosgraph_msgs ros-$distro-roslisp ros-$distro-rospack ros-$distro-std_msgs ros-$distro-std_srvs ros-$distro-xacro
```

Figure 42 ROS Robot Metapaket Dağıtımları

3.7.3. Paket bildirileri (Package Manifests)

Paket bildirileri (**package.xml**), bir paket hakkında: adı, sürümü, açıklaması, lisans bilgileri, bağımlılıkları ve dışa aktarılan paketler gibi diğer bilgileri içeren bir dosyadır. Bu dosyanın oluşturulmasının sebebi paket yüklenmesinin ve dağıtımının kolaylaştırılmasıdır.

Paket içinde bulunan bölümler için detaylı bilgiler figür 42'de verilmiştir.

```

<package format="2">
  <name>foo_core</name> Paketin adı
  <version>1.2.4</version> Paketin sürüm numarası (3 noktayla ayrılmış tam sayı olması gereklidir)
  <description>
    This package provides foo capability.
  </description>
  <maintainer email="ivana@willowgarage.com">Ivana Bildbotz</maintainer> Paketi südüren kişiler
  <license>BSD</license> Kodun yayınlandığı yazılım lisansları (ör. GPL, BSD, ASL).

  <url>http://ros.org/wiki/foo_core</url>
  <author>Ivana Bildbotz</author>

  <buildtool_depend>catkin</buildtool_depend> Build Tool Dependencies, bu paketin kendisini oluşturmak için ihtiyaç duyduğu derleme sistemi araçlarını belirtir.

  <depend>roscpp</depend> Depend, Paketin bağlı olduğu paketleri
  <depend>std_msgs</depend>

  <build_depend>message_generation</build_depend> Build Dependencies, bu paketi oluşturmak için hangi paketlerin gerekli olduğunu belirtir.

  <exec_depend>message_runtime</exec_depend> Execution Dependencies, bu pakette kod çalıştırılmak için hangi paketlerin gerekli olduğunu belirtir.

  <test_depend>python-mock</test_depend> Test Dependencies, ünite testleri için ek bağımlılıklar belirler.

  <doc_depend>doxygen</doc_depend> Documentation Tool Dependencies, bu paketin dokümantasyon oluşturmak için ihtiyaç duyduğu dokümantasyon araçlarını belirtir.
</package>

```

Figure 43 Package.xml Bölüm ve açıklamaları

3.7.4. Mesaj tipleri (Message types)

Mesaj dosyası **msg**/ klasöründe ve .**msg** uzantısına sahip olmalıdır.
(my_package/msg/MyMessageType.msg).

[geometry_msgs/Pose Message](#)

File: [geometry_msgs/Pose.msg](#)

Raw Message Definition

```

# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation

```

Compact Message Definition

```

geometry_msgs/Point position
geometry_msgs/Quaternion orientation

```

Figure 44 konum mesajı

3.7.5. Servis tipleri (Service types)

Servis dosyası srv/ klasöründe ve .srv uzantısına sahip olmalıdır (my_package/srv/MyServiceType.srv).

turtlesim/Spawn Service

File: [turtlesim/Spawn.srv](#)

Raw Message Definition

```
float32 x
float32 y
float32 theta
string name # Optional. A unique name will be created and returned if this is empty
-- string name
```

Compact Message Definition

```
float32 x
float32 y
float32 theta
string name
string name
```

Figure 45 TurtleSim Oluşturma servisi

3.8. İşlem Grafiği – Computation Graph Level

ROS işlem grafik seviyesi toplam 7 bölümden oluşur bunlar:

- Düğümler (Nodes)
- Parametre Servisi (Parameter Service)
- Mesajlar (Messages)
- Konular (Topics)
- Servisler (Services)
- Çantalar (Bags)
- Ana (Master)

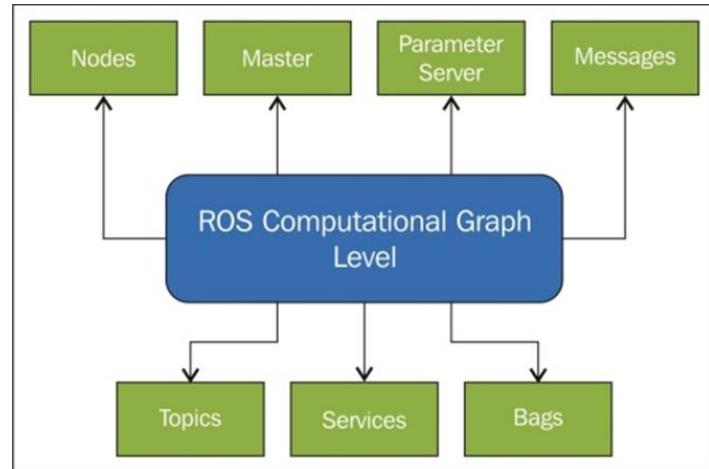


Figure 46 ROS İşlem Grafiği Seviyeleri

Bölümlerinden oluşmaktadır:

3.8.1. Düğümler (Nodes)

Düğümler, hesaplama yapan işlemlerilerdir. Bir düğüm, C++ için roscpp ve Python için rospy gibi farklı kütüphaneler kullanılarak yazılabılır. ROS'taki düğümleri kullanmak bize hata toleransı sağlar ve sistemi ve işlevleri basitleştirerek kodu ve işlevleri ayırrı. Bir düğümün sistemde benzersiz bir adı olması gereklidir. ROS düğümlerinin güçlü bir özelliği, düğümü başlatırken parametreleri (düğüm adı, konu adı, vb) değiştirebilme özelliğidir. Değiştirme işlemi ile, kod yeniden derlenmeden düğüm yapılandırılabilir, böylece farklı senaryolara kolayca adapte edilebilir.

- Düğümdeki konu adını değiştirmeye örnek:
 - rosrun book_tutorials tutorialX topic1:=/level1/topic1
- Düğümdeki parametreleri değiştirmeye örnek:
 - Rosrun book_tutorials tutorialX _param:= 9.0

ROS, nodelets adında başka bir düğüm türüne sahiptir. Bu özel düğümler, tek bir işlemde birden fazla düğümü çalıştırılmak üzere tasarlanmıştır. Bununla düğümler ağa aşırı yüklenmeden daha verimli iletişim kurabilirler. Nodelets özellikle aktarılan veri hacminin çok yüksek olduğu kamera sistemleri ve 3D sensörler için kullanışlıdır.

Not: Sistemde her şeyi yapan büyük bir düğüme sahip olmak yerine, yalnızca tek bir işlevsellik sağlayan birçok düğüme sahip olmak daha verimlidir.

ROS, düğümleri işlemek ve bilgiler vermek için rosnode aracına sahiptir. rosnode aracında desteklenen bazı komutlar:

- **rosnode info node_name:** Düğümlarındaki bilgileri yazdırır.
- **rosnode kill node_name:** Çalışan bir düğümü sonlandırır.
- **rosnode list:** Aktif düğümleri listeler.
- **rosnode machine hostname:** Belirli bir makinede çalışan düğümleri listeler.
- **rosnode ping node_name:** Düğüme olan bağlantıyı test eder.

3.8.2. Parametre Servisi (Parameter Service)

Parametreler ile, çalışan düğümleri yapılandırmak veya bir düğümün çalışma parametrelerini değiştirmek mümkündür.ROS, Parametre Server ile çalışmak için rosparam aracına sahiptir. rosparam aracında desteklenen bazı komutlar:

- **rosparam list:** Sunucudaki tüm parametreleri listeler.
- **rosparam get parameter:** Bir parametrenin değerini alır.
- **rosparam set parameter:** Bir parametrenin değerini ayarlar.
- **rosparam delete parameter:** Bir parametreyi siler.
- **rosparam dump file:** Parametre sunucusunu bir dosyaya kaydeder.
- **rosparam load file:** Parametre sunucusunda bir dosyayı (parametreleriyle birlikte) yükler.

3.8.3. Mesajlar (Messages)

Düğümler birbirleriyle mesajlar yoluyla haberleşir. Bir mesaj diğer düğümlere bilgi sağlayan veri içerir. Bir mesaj tip ve isim olmak üzere iki parçadan oluşmaktadır. Kendi mesaj tipimizi oluşturabiliriz.

In ROS, you can find a lot of standard types to use in messages, as shown in the following table list:

| Primitive type | Serialization | C++ | Python |
|----------------|-------------------------------|---------------|----------------|
| bool (1) | unsigned 8-bit int | uint8_t (2) | bool |
| int8 | signed 8-bit int | int8_t | int |
| uint8 | unsigned 8-bit int | uint8_t | int (3) |
| int16 | signed 16-bit int | int16_t | int |
| uint16 | unsigned 16-bit int | uint16_t | int |
| int32 | signed 32-bit int | int32_t | int |
| uint32 | unsigned 32-bit int | uint32_t | int |
| int64 | signed 64-bit int | int64_t | long |
| uint64 | unsigned 64-bit int | uint64_t | long |
| float32 | 32-bit IEEE float | float | float |
| float64 | 64-bit IEEE float | double | float |
| string | ascii string (4) | std::string | string |
| time | secs/nsecs signed 32-bit ints | ros::Time | rospy.Time |
| duration | secs/nsecs signed 32-bit ints | ros::Duration | rospy.Duration |

Figure 47 ROS Mesaj Standardları

Başlıklar (Headers) ROS'ta özel bir tiptir. Zaman çizelgesi, mesajların kimden geldiğini öğrenebilmemizi sağlayan numaralandırma sistemi ve dizi numarasına sahiptir.

std_msgs/Header Message

File: std_msgs/Header.msg

Raw Message Definition

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
#uint32 stamp
# * stamp.sec: seconds (stamp.sec) since epoch (In Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (In Python the variable is called 'nsecs')
# time-handling sugar is provided by the client library
time stamp
#frame this data is associated with
string frame_id
```

Compact Message Definition

```
uint32 seq
time stamp
string frame_id
```

Figure 48 ROS Başlık Mesajı

ROS mesaj tanımını ve mesaj tipinin belirtildiği kaynak dosyasını görebilmemizi sağlayan rosmsg isimli bir araca sahiptir. rosmsg aracında desteklenen bazı komutlar:

- **rosmsg show:** Bu mesajın alanlarını görüntüler.
- **rosmsg list:** Tüm mesajları listeler
- **rosmsg package:** Belirli paketteki tüm mesajları listeler.
- **rosmsg packages:** Mesajı olan tüm paketleri listeler.
- **rosmsg users:** Mesaj türünü kullanan kod dosyalarını arar.

3.8.4. Konular (Topics)

Mesajlar, yayinallyama / abone olma semantигine sahip bir taşıma sistemi aracılığıyla yönlendirilir. Bir düğüm, belirli bir konu yayinallyarak bir mesaj gönderir. Konu, mesajın içeriğini tanımlamak için kullanılan bir addır. Belirli bir veri türüyle ilgilenen bir düğüm uygun konuya abone olacaktır. Karışıklıktan kaçınmak için konu adlarının benzersiz olması önemlidir. Tek bir konu için birden fazla eşzamanlı yayinci ve abone olabilir ve tek bir düğüm birden fazla konuya yayın yapabilir ve / veya abone olabilir.

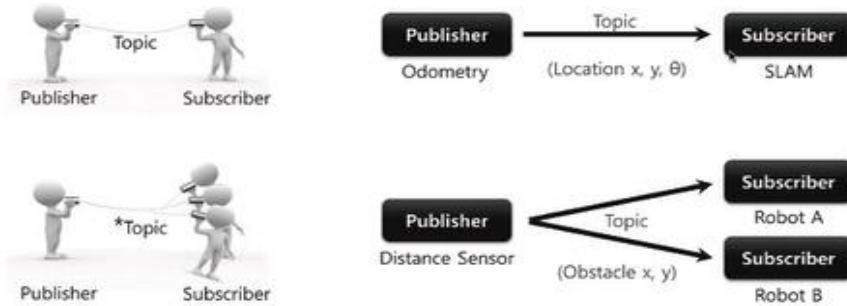


Figure 49 Yayınlayıcı / Dinleyici

ROS, rostopic denilen konularda çalışacak bir araca sahiptir. rostopic aracında desteklenen bazı komutlar :

- **rostopic bw/topic:** Konunun kullandığı bant genişliğini gösterir.
- **rostopic echo/topic:** Mesajları ekrana basar.
- **rostopic find message_type:** Türlerine göre konuları bulur.
- **rostopic hz/topic:** Konunun yayılanma oranını gösterir.
- **rostopic info/topic:** Mesaj türü, yayıcıları ve aboneleri gibi konu hakkındaki bilgileri yazdırır.
- **rostopic list:** Aktif konular hakkında bilgi yazdırır.
- **rostopic pub/topic type args:** Konuya ilgili verileri yayınlar. İstediğimiz konuda, doğrudan komut satırından veri oluşturmamızı ve yayılmamamızı sağlar.
- **rostopic type/topic:** Konu türünü, yani yayınladığı mesaj türünü yazdırır.

3.8.5. Servisler (Services)

Yayınlama / abone olma modeli çok esnek bir iletişim paradigmasıdır, ancak çoktan çoga, tek yönlü taşımacılığı genellikle dağıtılmış bir sistemde istenen istek / cevap etkileşimleri için uygun değildir. Talep / cevap, bir çift mesaj yapısı tarafından tanımlanan servisler vasıtasyyla yapılır: biri istek (request) için, diğeri yanıt (response) için. Sağlayıcı bir isim altında bir servis sunar ve bir müşteri talep mesajını gönderip cevabı bekleyerek servisi kullanır. ROS servislerle çalışmak için rossrv ve rosservice komut satırı araçlarına sahiptir. Bu araçlarda desteklenen bazı komutlar:

- **rosservice call/service args:** Servisi verilen argümanlarla çağırır.
- **rosservice find msg-type:** Servis tipine göre servis bulur.

- **rosservice info/service:** Servislarındaki bilgileri yazdırır.
- **rosservice list:** Aktif servisleri listeler.
- **rosservice type/servis:** Servis tipini yazdırır.

3.8.6. Çantalar (Bags)

Çanta, ROS tarafından oluşturulan, tüm mesajların, konuların, servislerin ve diğer bilgilerin tüm bilgilerini kaydetmek ve daha sonra oynatmak için .bag biçiminde oluşturulan bir dosyadır. Çantalar, toplanması zor olabilen ancak algoritmalar geliştirmek ve test etmek için gerekli olan sensör verileri gibi verileri depolamak için önemli bir mekanizmadır. Çanta dosyalarını kullanmak için ROS'ta kullanılabilecek araçlar:

- **rosbag:** İstenilen verileri kaydetmek, oynatmak ve gerçekleştirmek için kullanılır.
- **rqt_bag:** Verileri grafik ortamda görselleştirmek için kullanılır.

3.8.7. Ana (Master)

ROS'taki düğümlerin birbirleriyle iletişim kurmasını kolaylaştıran kısmına ROS master denir. ROS Master, İşlem Grafiğinin geri kalanına arama sağlar. Master olmadan, düğümler birbirlerini bulamaz, mesaj alışverişinde bulunamaz veya servis çağrıları yapamazlar. Herhangi bir ROS düğümünü çalıştırmadan önce, ROS Master ve ROS parametre sunucusunu başlatmalıyız. ROS Master ve ROS parametre sunucusunu roscore adlı tek bir komut kullanarak başlatabiliriz. Karşımıza çıkan metinlerde:

- Birinci bölümde, ROS düğümlerinden gelen günlükleri toplamak için ~/.ros/log klasörünün içinde bir günlük dosyası oluşturulur. Bu dosya hata ayıklama amacıyla kullanılabilir.
- İkinci bölümde, roscore adlı bir ROS başlatma dosyası başlatılır. Bu bölüm port içindeki ROS parametre sunucusunun adresini gösterir.
- Üçüncü bölümde, rosdistro ve rosversion gibi parametreleri görüntülenmektedir.

- Dördüncü bölümde, rosmaster düğümünün daha önce ortam değişkeni olarak tanımladığımız ROS_MASTER_URI kullanılarak başlatıldığı görülmektedir.
- Beşinci bölümde, rosout düğümünün başladığı görülmektedir.

```

robot@robot-VirtualBox:~$ roscore
... logging to /home/robot/.ros/log/a3a8e160-e1ae-11e4-b7be-0800273c354c/roslaunch-robot-Virtu
alBox-2138.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://robot-VirtualBox:42377/
ros_comm version 1.11.10

```

1

2

```

SUMMARY
=====

PARAMETERS
* /rosdistro: indigo
* /rosversion: 1.11.10

```

3

4

```

NODES
auto-starting new master
process[master]: started with pid [2183]
ROS_MASTER_URI=http://robot-VirtualBox:11311/

setting /run_id to a3a8e160-e1ae-11e4-b7be-0800273c354c
process[rosout-1]: started with pid [2196]
started core service [/rosout]

```

5

5

Figure 50 RosMaster

3.9. Topluluk – Community Level

- Dağıtımlar (Distributions):** ROS Dağıtımları, yükleyebileceğiniz sürümlü yiğinların koleksiyonlarıdır.
- Depolar (Repositories):** ROS, farklı kurumların kendi robot yazılım bileşenlerini geliştirip yayinallyabilecekleri bir kod deposu sunar.
- ROS Wiki:** ROSlarındaki bilgileri belgeleyen ana forumdur.
- Mail Listesi:** Ros-users e-posta listesi, ROS'taki yeni güncellemelerin yanı sıra ROS yazılımı hakkında sorular soran bir forum olan birincil iletişim kanalıdır.
- ROS Answers:** ROS ile ilgili sorularınızı cevaplamak için soru-cevap sitesidir.
- Blog:** <http://www.ros.org/news> , fotoğraflar ve videolar dahil olmak üzere düzenli güncellemeler sağlar.

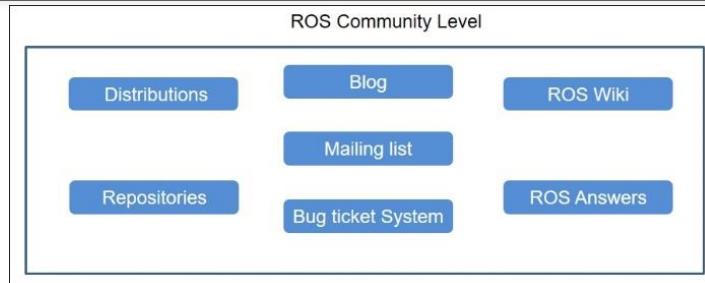


Figure 51 Topluluk Seviyesi

3.10. Publisher-Subscriber ve Service Client Yapıları

Publish-Subscribe: Bu iletişim modeli, yayımcının açıkça alıcıları belirtmeden ya da amaçlanan alıcıların bilgisine sahip olmadan mesajın yayınlanmasını gerektirir. Abone yayınlanan mesajlardan ilgili olanları kaydeder.

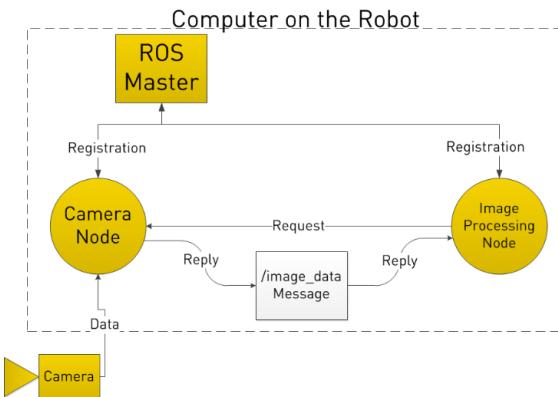


Figure 52 Publish-Subscribe örneği

Service-Client: Tek seferlik iletişim sağlayan ve müşterinin istek gönderip, sunucunun geriye yanıt döndürdüğü iletişim modelidir. Robotun özel bir görev yapması istendiğinde (örneğin A noktasından B noktasına yol bulmasında) kullanılır.

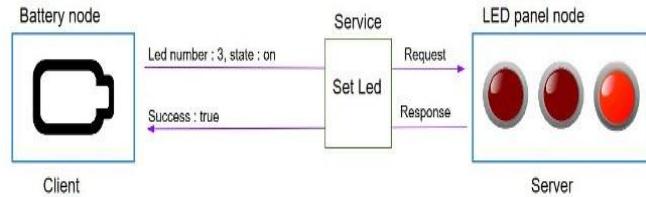


Figure 53 Service-Client örneği

3.11. ROS Araçları

ROS, mesajları incelemek ve hata ayıklamak için çeşitli GUI ve komut satırı araçlarına sahiptir. Bunlardan bazıları:

- **rviz:** ROS konularından ve parametrelerinden 2D ve 3D değerlerini görselleştirmek için ROS'ta mevcut 3D görselleştiricilerden biridir.
- **rqt_plot:** ROS konuları biçimindeki skaler değerleri çizmek için bir araçtır.
- **rqt_graph:** ROS düğümleri arasındaki bağlantı grafiğini görselleştirir.

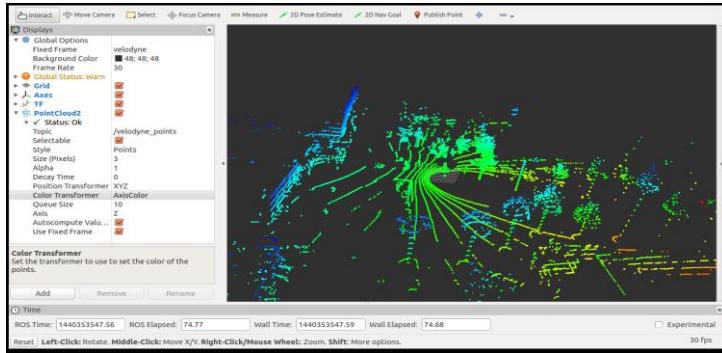


Figure 55 Rviz

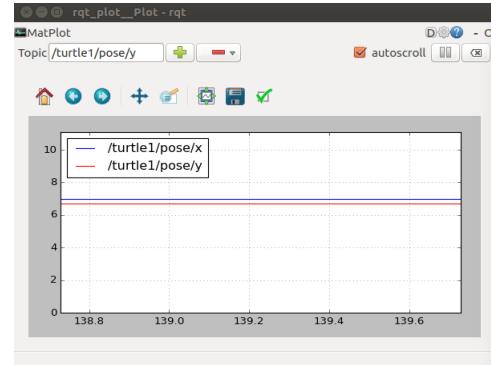


Figure 56 rqt_plot

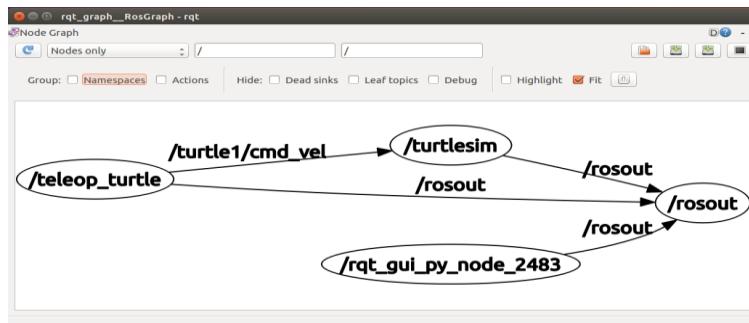


Figure 54 rqt_graph

3.12. ROS uygulamaları

3.12.1. Uygulama 1: ROS Ortamının Hazırlanması

Çalışma alanı paketleri içeren bir klasördür. Bu paketler kaynak dosyaları içerir. Çeşitli paketler aynı anda derlenmek (merkezleştirmek) istendiğinde kullanışlıdır.

- **Kaynak alan (src):** Kaynak alana (src klasörü), paketler, projeler, vb. konulur. Bu alanda ayrıca CMakeLists.txt dosyası bulunur.

- **Derleme alanı (build):** derleme klasöründe cmake ve catkin, paketler ve projeler için önbellek bilgilerinin, yapılandırmasını ve diğer ara dosyaları saklar.
- **Geliştirme alanı (devel):** Derlenmiş programları korumak ve programları kurulum aşaması olmadan test etmek için kullanılır.

Ortamı kontrol edelim:

```
printenv | grep ROS
```

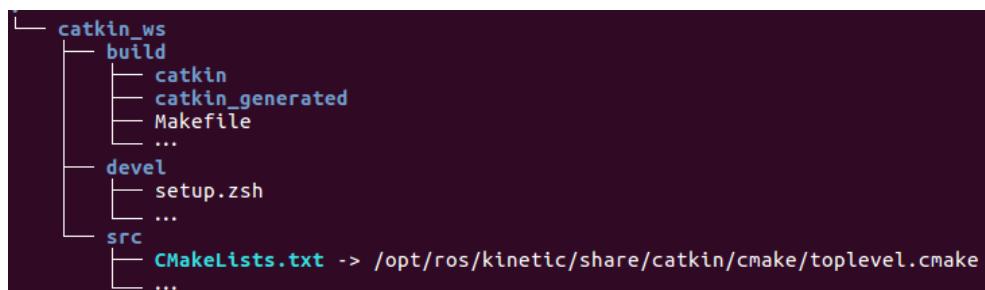


Figure 57 Çalışma Alanı Dosya Yapısı

Konfigürasyon ayarlarını her seferinde yapmamak için:

```
gedit ~/.bashrc
```

Gedit editöründe açılan ekrana aşağıdaki kodlar eklenir.

```
<source /opt/ros/kinetic/setup.bash>
<source /home/<kullanıcı_adı>/ros_ws/devel/setup.bash>
```

```
bashrc (-/)-gedit
Open ▾ 🗑 CMakeLists.txt x .bashrc

case $TERM in
xterm*|rxvt*)
    PS1="\[\e[0;5;${debian_chroot:+($debian_chroot)}\u@\h: \w\]\]$PS1"
    ;;
*)
    ;;
esac

# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $(($?)) = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*\s*/;/g;s/\|/;/g'\')$*'"$([ $(($?)) = 0 ] && echo alerts:///\''")'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

source /opt/ros/kinetic/setup.bash
source ~/home/moguztas/ros_ws/devel/setup.bash
export ROS_HOSTNAME=moguztas
export ROS_MASTER_URI=http://localhost:11311
export ROS_PACKAGE_PATH=~/home/moguztas/ros_ws/src
```

Figure 58 .Bashrc

Not: bashrc'de yapılan değişikliklerin önceden açılan terminalde algılanabilmesi için bash komutu ile terminal yenilenmelidir.

Çalışma ortamı oluşturmak için:

```
mkdir -p ~/ros_ws/src  
cd ~/ros_ws/  
catkin_make
```

Not: catkin make ile aynı işe yapabilecek kod blogu:

```
cd ~/ros_ws  
cd src  
catkin_init_workspace  
cd ..  
mkdir build
```

```
cd build  
cmake ../src -DCMAKE_INSTALL_PREFIX=../install -  
DCATKIN_DEVEL_PREFIX=../devel  
make
```

Not: ROS_PACKAGE_PATH yapılandırma değişkeninin konumu için terminal ekranına aşağıdaki kod yazılır.

```
echo $ROS_PACKAGE_PATH  
/home/(KULLANICI_ADINIZ)/ros_ws/src:/opt/ros/kinetic/s  
hare
```

3.12.2. Uygulama 2: catkin Paketi Oluşturma ve ROS Ortamını Tanıma

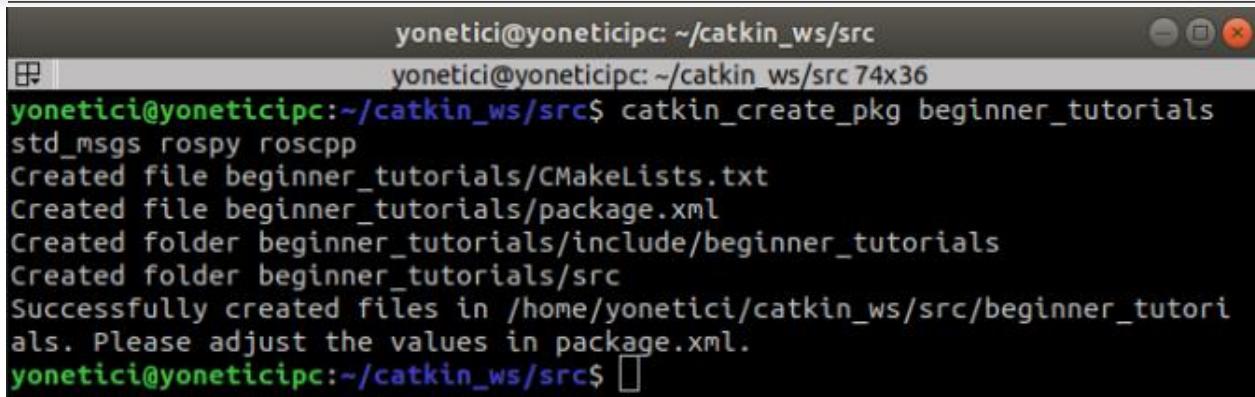
İlk olarak, ros çalışma alanında src dizinine gidilir.

```
cd ~/ros_ws/src
```

catkin_create_pkg komutu ile beginner_tutorials isimli std_msgs, roscpp ve rospy a bağlı olan bir paket oluşturulur:

```
catkin_create_pkg beginner_tutorials std_msgs rospy  
roscpp
```

Not: Bu beginner_tutorial isimli package.xml ve bir CMakeLists.txt dosyası içeren bir dosya oluşturacaktır. CMakeLists.txt dosyası kısmen catkin_create_pkg komutu tarafından doldurulmuştur.



```
yonetici@yoneticipc: ~/catkin_ws/src
yonetici@yoneticipc: ~/catkin_ws/src 74x36
yonetici@yoneticipc:~/catkin_ws/src$ catkin_create_pkg beginner_tutorials
std_msgs rospy roscpp
Created file beginner_tutorials/CMakeLists.txt
Created file beginner_tutorials/package.xml
Created folder beginner_tutorials/include/beginner_tutorials
Created folder beginner_tutorials/src
Successfully created files in /home/yonetici/catkin_ws/src/beginner_tutorials.
Please adjust the values in package.xml.
yonetici@yoneticipc:~/catkin_ws/src$
```

Figure 59 Catkin Paketi Oluşturmak

beginner tutorials klasörü içerisinde ne olduğuna bakmak için:

```
cd beginner_tutorials
ls
```

Klasörün içinde Cmakelist.txt, include ve src klasörlerinin olacağı görülecektir.
beginner_tutorials src dosyasına giderek burada basit bir kod yazalım:

```
cd src
gedit first_script.cpp
```

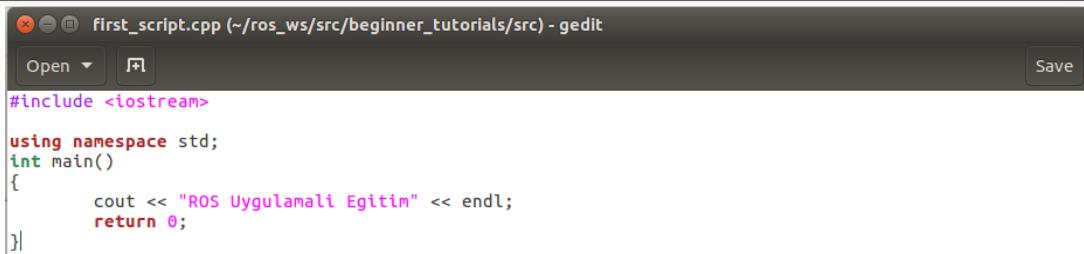
CMakeLists.txt dosyasında yazdığımız kodun çalıştırılabilir olmasını sağlayalım:

```
cd ..
gedit CMakeLists.txt
```

Çalışma alanımızı derleyelim.

```
cd ~/ros_ws/
catkin_make
```

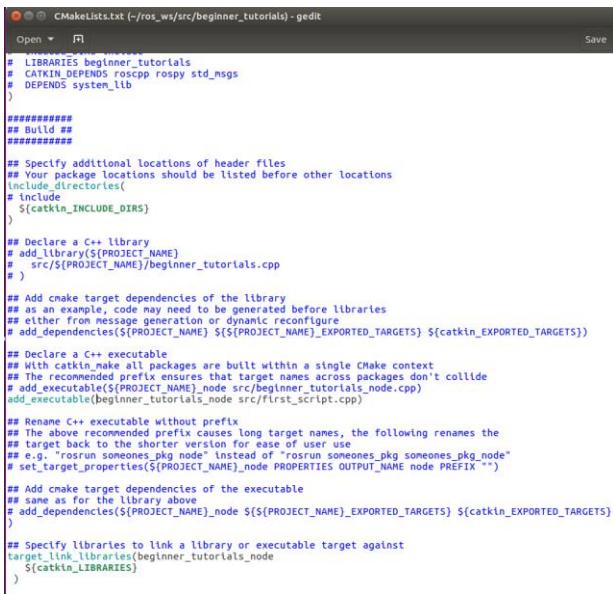
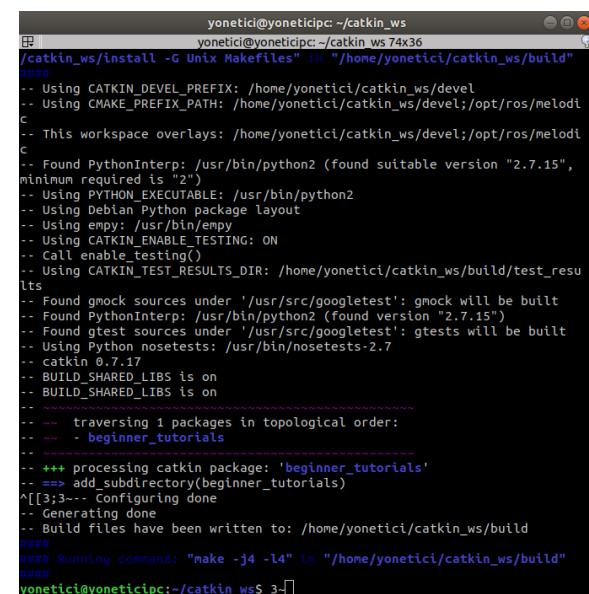
Not: Derleme tamamlandığında src klasöründe build,devel ve src alt klasörleri kurulmuş olacak,paket kullanıma hazır hale gelecektir.



```
#include <iostream>

using namespace std;
int main()
{
    cout << "ROS Uygulamali Egitim" << endl;
    return 0;
}
```

Figure 61 first_script.cpp

```
yoneticici@yoneticicpc:~/catkin_ws
[catkin_ws] catkin_ws$ /catkin_ws/install -G Unix Makefiles" in "/home/yoneticici/catkin_ws/build"
#####
-- Using CATKIN_DEVEL_PREFIX: /home/yoneticici/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/yoneticici/catkin_ws/devel;/opt/ros/melodic
-- This workspace overlays: /home/yoneticici/catkin_ws/devel;/opt/ros/melodic
-- Found PythonInterp: /usr/bin/python2 (found suitable version "2.7.15",
minimum required is "2")
-- Using PYTHON_EXECUTABLE: /usr/bin/python2
-- Using Debian Python package layout
-- Using empv: /usr/bin/empv
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/yoneticici/catkin_ws/build/test_results
-- Found gmock sources under '/usr/src/googletest': gmock will be built
-- Found PythonInterp: /usr/bin/python2 (found version "2.7.15")
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.17
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-----
-- traversing 1 packages in topological order:
-- - beginner_tutorials
-----
--+ processing catkin package: 'beginner_tutorials'
--=> add_subdirectory(beginner_tutorials)
^[[3;--- Configuring done
-- Generating done
-- Build files have been written to: /home/yoneticici/catkin_ws/build
#####
##### Running command: "make -j4 -l4" in "/home/yoneticici/catkin_ws/build"
#####
yoneticici@yoneticicpc:~/catkin_ws$ 3-
```

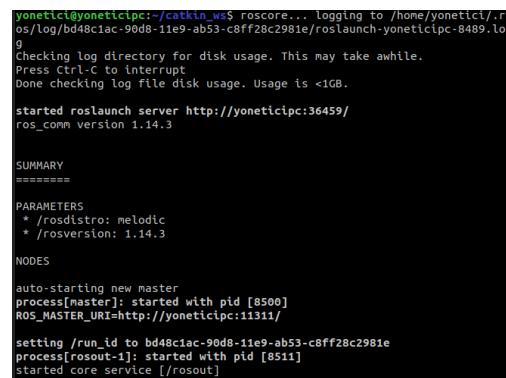
Figure 60 Catkin derlenmesi.

Yazılan kodun çalıştırılması için iki terminal açılır. İlk terminalde aşağıdaki kod çalıştırılır.

roscore

Düger terminalde ise oluşturulan ros paketi çalıştırılır.

rosrun beginner_tutorials beginner_tutorials_node



```
yoneticici@yoneticicpc:~/catkin_ws$ roscore... logging to /home/yoneticici/.ros/log/bd48ciac-90d8-11e9-ab53-c0ff28c2981e/roslaunch-yoneticicpc-8489.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file usage. Usage is <1GB.

started roslaunch server http://yoneticicpc:36459/
ros_comm version 1.14.3

SUMMARY
========
PARAMETERS
  * /rosdistro: melodic
  * /rosversion: 1.14.3

NODES
auto-starting new master
process[master]: started with pid [8500]
ROS_MASTER_URI=http://yoneticicpc:11311/
setting /run_id to bd48ciac-90d8-11e9-ab53-c0ff28c2981e
process[rosout-1]: started with pid [8511]
started core service [/rosout]
```

Figure 62 roscore ekranı

rospack aracı ile paketteki bağımlılıkları görüntülemek için:

```
rospack depends1 beginner_tutorials
```

Bu bağımlılıklar package.xml dosyasında da listelenmektedir.

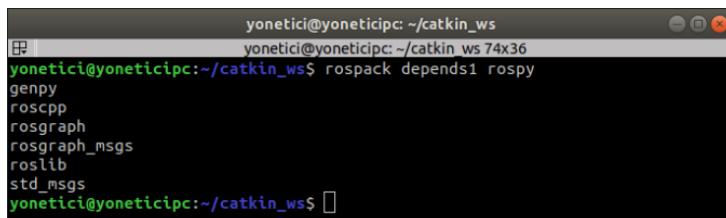
```
roscd beginner_tutorials  
gedit package.xml
```

```
<buildtool_depend>catkin</buildtool_depend>  
<build_depend>roscpp</build_depend>  
<build_depend>rospy</build_depend>  
<build_depend>std_msgs</build_depend>  
<build_export_depend>roscpp</build_export_depend>  
<build_export_depend>rospy</build_export_depend>  
<build_export_depend>std_msgs</build_export_depend>  
<exec_depend>roscpp</exec_depend>  
<exec_depend>rospy</exec_depend>  
<exec_depend>std_msgs</exec_depend>  
  
<!-- The export tag contains other, unspecified, tags -->  
<export>  
<!-- Other tools can request additional information be placed here -->  
</export>  
</package>
```

Figure 63 Package.xml

Dolaylı bağımlılıklar rospack aracılıyla görüntülenebilir. Örneğin rospy a bağlı bağımlılıkları görmek için:

```
rospack depends1 rospy
```



```
yonetici@yoneticipc:~/catkin_ws  
yonetici@yoneticipc:~/catkin_ws$ rospack depends1 rospy  
genpy  
roscpp  
rosgraph  
rosgraph_msgs  
roslib  
std_msgs  
yonetici@yoneticipc:~/catkin_ws$
```

Figure 64 rospack Örneği

Paketteki tüm bağımlılıkları görmek için:

```
rospack depends beginner_tutorials
```

```
yonetici@yoneticipc: ~/catkin_ws
yonetici@yoneticipc: ~/catkin_ws 74x36
yonetici@yoneticipc:~/catkin_ws$ rospack depends beginner_tutorials
cpp_common
rostime
rospp_traits
rospp_serialization
catkin
genmsg
genpy
message_runtime
gencpp
geneus
gennodejs
genlisp
message_generation
rosbuild
rosconsole
std_msgs
rosgraph_msgs
xmlrpcpp
rospp
rosgraph
ros_environment
rospack
roslib
rospy
yonetici@yoneticipc:~/catkin_ws$
```

Figure 65 Bütün rospack Bağımlılıkları

3.12.3. Uygulama 3: TurtleSim

TurtleSim uygulaması için:

```
sudo apt-get install ros-kinetic-ros-tutorials
```

ROS Master’ı başlatmak için:

```
roscore
```

TurtleSim’i çalıştırılmak için:

```
rosrun turtlesim
turtlesim_node
```

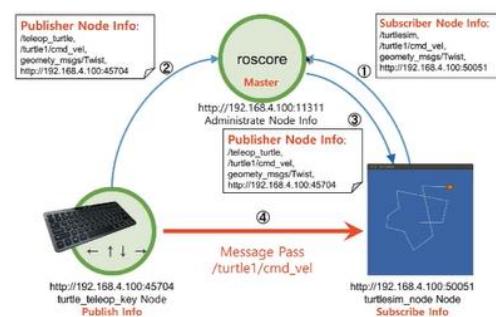


Figure 66 Turtlebot Teleoperasyonu

Klavye ile kontrol için:

```
rosrun turtlesim turtle_teleop_key
```

Altta ki kodları sırasıyla yazınız. Her bir kod için örnek çıktı verilmiştir.

```
rosservice list
```

```
yonetici@yoneticipc:~/catkin_ws$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/rostopic_10158_1560766441502/get_loggers
/rostopic_10158_1560766441502/set_logger_level
/rostopic_9686_1560765920693/get_loggers
/rostopic_9686_1560765920693/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
yonetici@yoneticipc:~/catkin_ws$
```

Figure 67 rosservice Listesi

```
rosservice type /spawn
```

```
yonetici@yoneticipc:~$ rosservice type /spawn
|turtlesim/Spawn
yonetici@yoneticipc:~$
```

Figure 68 TurtleSim oluşturulması

```
rossrv show turtlesim/Spawn
```

```
yonetici@yoneticipc:~$ rossrv show turtlesim/Spawn
float32 x
float32 y
float32 theta
string name
...
string name
yonetici@yoneticipc:~$
```

Figure 69 TurtleSim/Spawn servislerinin gösterilmesi

```
rosservice call /spawn 3 3 0 new_turtle
```

```
yonetici@yoneticipc:~$ rosservice call /spawn 3 3 0 new_turtle
name: "new_turtle"
yonetici@yoneticipc:~$
```

Figure 70 Yeni Servis Çağırılması

```
rosparam list
```

```
yonetici@yoneticipc:~$ rosparam list
/background_b
/background_g
/background_r
/rosdistro
/roslaunch/uris/host_moguztas_35969
/rosversion
/run_id
yonetici@yoneticipc:~$
```

Figure 71 Ros Parametre Listesi

```
rosparam get /background_b
```

```
yonetici@yoneticipc:~$ rosparam get /background_b
255
yonetici@yoneticipc:~$
```

Figure 72 Arkaplan Ayarlanması

```
rosparam set /background_b 10
rosparam get /background_b 10
```



Figure 73 TurtleSim Arkaplan Renk değişim örneği

```
rosservice call /clear
```

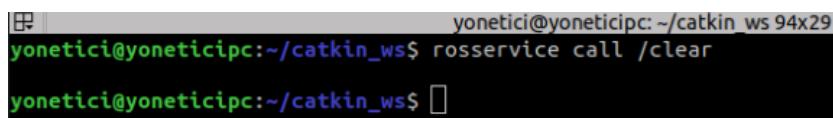


Figure 74 Servislerin Kapatılması

3.12.4. Uygulama 4: Mesaj ve Servis Oluşturma

Mesaj oluşturma:

Çalışma alanındaki beginner_tutorials klasörüne msg klasörü oluşturalım.

```
roscd beginner_tutorials  
mkdir msg
```

Mesaj dosyamızı oluşturalım ve terminalde oluşturduğumuz dosyayı gösterelim.

```
echo "int64 num" > msg/Num.msg  
rosmsg show beginner_tutorials/Num
```

package.xml dosyamızı düzenleyelim.

```
roscd beginner_tutorials  
gedit package.xml  
<build_depend>message_generation</build_depend>  
<exec_depend>message_runtime</exec_depend>
```

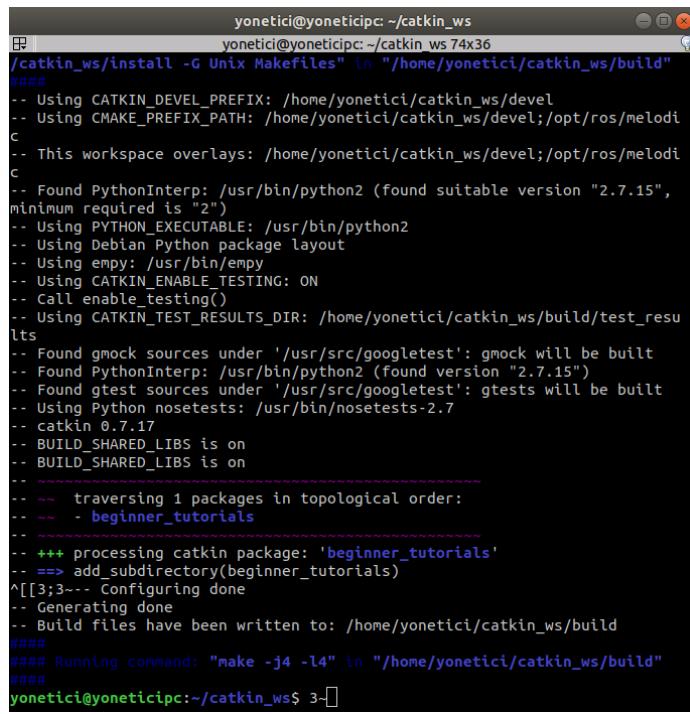
CMakeLists.txt dosyamızı aşağıdaki şekilde düzenleyelim.

```
gedit CMakeLists.txt  
find_package(catkin REQUIRED COMPONENTS  
...  
message_generation  
)  
  
catkin_package(  
...  
CATKIN_DEPENDS message_runtime ...  
...  
)
```

```
add_message_files(
    FILES
    Num.msg
)
generate_messages(
    DEPENDENCIES
    std_msgs
)
```

Çalışma alanını derleyin.

```
cd ~/ros_ws
catkin_make
```



```
yoneticici@yoneticipc: ~/catkin_ws
[catkin_ws/install -G Unix Makefiles" in "/home/yoneticici/catkin_ws/build"
#####
-- Using CATKIN_DEVEL_PREFIX: /home/yoneticici/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/yoneticici/catkin_ws/devel;/opt/ros/melodic
C
-- This workspace overlays: /home/yoneticici/catkin_ws/devel;/opt/ros/melodic
C
-- Found PythonInterp: /usr/bin/python2 (found suitable version "2.7.15",
minimum required is "2")
-- Using PYTHON_EXECUTABLE: /usr/bin/python2
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/yoneticici/catkin_ws/build/test_results
lts
-- Found gmock sources under '/usr/src/googletest': gmock will be built
-- Found PythonInterp: /usr/bin/python2 (found version "2.7.15")
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.17
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- ~~~~~
-- ~~~ traversing 1 packages in topological order:
-- ~~~ - beginner_tutorials
-- ~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
^[[3;3--- Configuring done
-- Generating done
-- Build files have been written to: /home/yoneticici/catkin_ws/build
#####
##### Running command: "make -j4 -l4" in "/home/yoneticici/catkin_ws/build"
#####
yoneticici@yoneticipc:~/catkin_ws$ 3-
```

Figure 75 Catkin Derleme

Servis Oluşturma:

Çalışma alanındaki beginner_tutorials klasörüne srv klasörü oluşturalım.

```
roscd beginner_tutorials  
mkdir srv
```

srv dosyamızı oluşturalım ve terminalede oluşturduğumuz dosyayı gösterelim

```
roscp rospy_tutorials AddTwoInts.srv  
srv/AddTwoInts.srv  
rossrv show beginner_tutorials/AddTwoInts
```

Not: srv dosyaları da iki bölüm içermesi hariç msg dosyaları gibidirler.

package.xml dosyamızı düzenleyelim.

```
roscd beginner_tutorials  
gedit package.xml  
<build_depend>message_generation</build_depend>  
<exec_depend>message_runtime</exec_depend>
```

CMakeLists.txt dosyamızı aşağıdaki şekilde düzenleyelim.

```
gedit CMakeLists.txt  
find_package(catkin REQUIRED COMPONENTS  
...  
message_generation  
)  
  
catkin_package(  
...  
CATKIN_DEPENDS message_runtime ...  
...  
)
```

```

add_service_files(
FILES
AddTwoInts.srv
)
generate_messages(
DEPENDENCIES
std_msgs
)

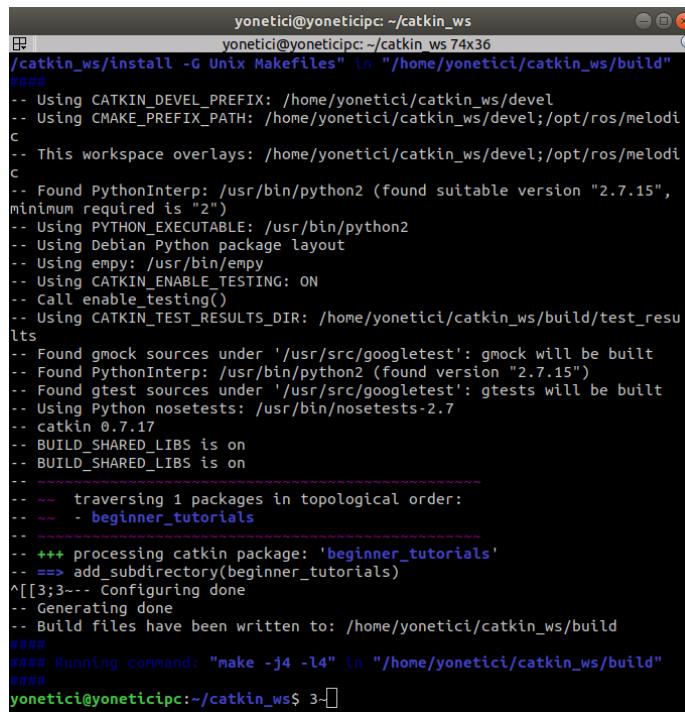
```

Çalışma alanını derleyin.

```

cd ~/ros_ws
catkin_make

```



```

yonetici@yoneticipc: ~/catkin_ws
yonetici@yoneticipc: ~/catkin_ws 74x36
[catkin_ws/install -G Unix Makefiles" in "/home/yonetici/catkin_ws/build"
#####
-- Using CATKIN_DEVEL_PREFIX: /home/yonetici/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/yonetici/catkin_ws/devel;/opt/ros/melodic
-- This workspace overlays: /home/yonetici/catkin_ws/devel;/opt/ros/melodic
-- Found PythonInterp: /usr/bin/python2 (found suitable version "2.7.15",
minimum required is "2")
-- Using PYTHON_EXECUTABLE: /usr/bin/python2
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/yonetici/catkin_ws/build/test_results
-- Found gmock sources under '/usr/src/googletest': gmock will be built
-- Found PythonInterp: /usr/bin/python2 (found version "2.7.15")
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.17
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-----
--   traversing 1 packages in topological order:
--   - beginner_tutorials
-----
--   +++ processing catkin package: 'beginner_tutorials'
--   ==> add_subdirectory(beginner_tutorials)
[[3;3--- Configuring done
-- Generating done
-- Build files have been written to: /home/yonetici/catkin_ws/build
#####
##### Running command: "make -j4 -l4" in "/home/yonetici/catkin_ws/build"
#####
yonetici@yoneticipc:~/catkin_ws$ 3-
```

Figure 76 Catkin Derlenmesi

3.12.5. Uygulama 5: Publisher-Subscriber Uygulaması

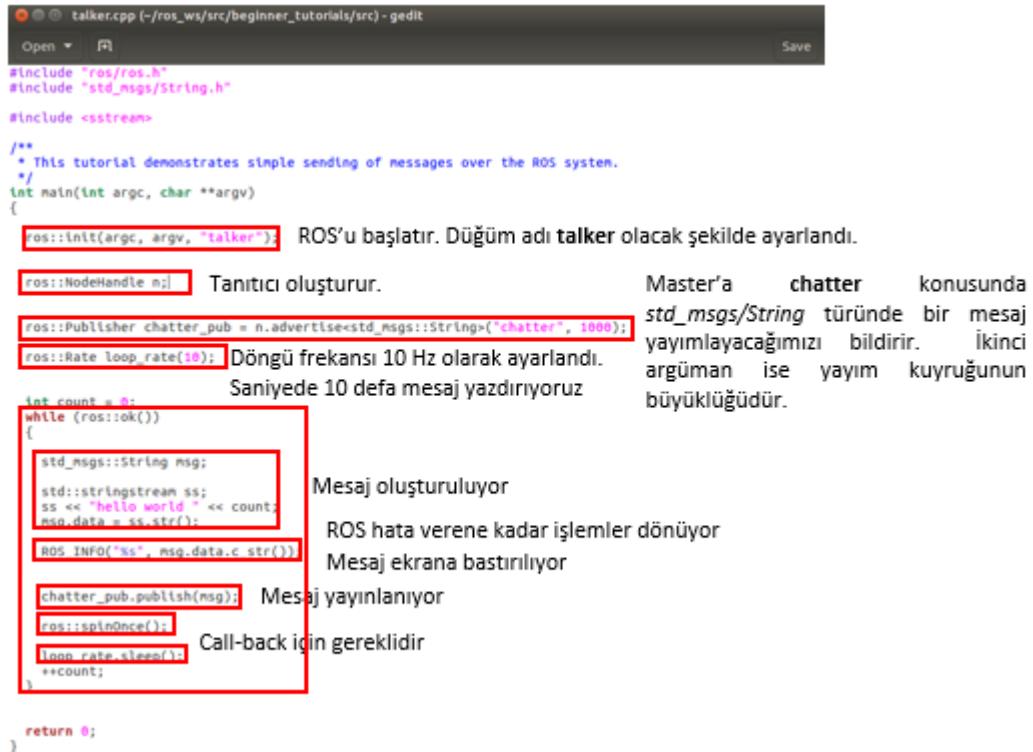
C++ için:

beginner_tutorials altındaki src klasörüne gidelim.

```
roscd beginner_tutorials/src
```

Publisher dosyamızı oluşturalım.

```
gedit talker.cpp
```



```
#include <ros/ros.h>
#include <std_msgs/String.h>

#include <iostream>

/**
 * This tutorial demonstrates simple sending of messages over the ROS system.
 */
int main(int argc, char **argv)
{
    ros::init(argc, argv, "talker"); ROS'u başlatır. Düğüm adı talker olacak şekilde ayarlandı.

    ros::NodeHandle n; Tanıtıcı oluşturulur.

    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000); Master'a chatter konusunda std_msgs/String türünde bir mesaj yaymayıcağızı bildirir. İkinci arguman ise yayım kuyruğunun büyülügüdür.

    ros::Rate loop_rate(10); Döngü frekansı 10 Hz olarak ayarlandı. Saniyede 10 defa mesaj yazdırıyoruz

    int count = 0;
    while (ros::ok())
    {
        std_msgs::String msg;
        std::stringstream ss;
        ss << "Hello world " << count;
        msg.data = ss.str();
        ROS_INFO("%s", msg.data.c_str()); Mesaj oluşturuluyor

        chatter_pub.publish(msg); ROS hata verene kadar işlemler dönüyor

        ros::spinOnce(); Mesaj ekrana bastırılıyor

        ros::Duration loop_rate.sleep(); Mesaj yayılanıyor

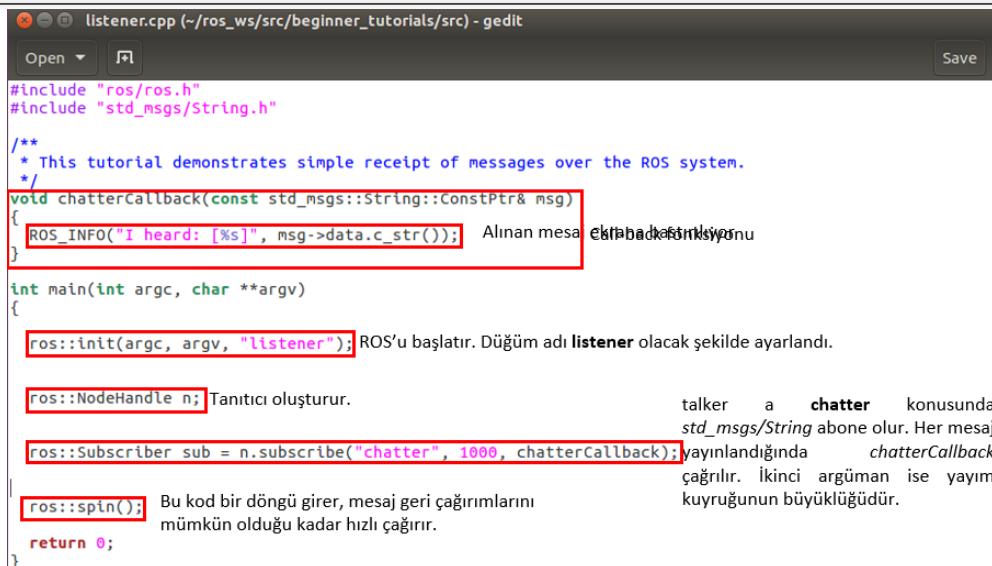
        ros::spinOnce(); Call-back için gereklidir
        ++count;
    }

    return 0;
}
```

Figure 77 Talker.cpp ve Detayları

Subscriber dosyamızı oluşturalım.

```
gedit listener.cpp
```



```

#include <ros/ros.h>
#include <std_msgs/String.h>

/*
 * This tutorial demonstrates simple receipt of messages over the ROS system.
 */
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("I heard: [%s]", msg->data.c_str()); Alinan mesaj ekranda gösterilir.
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "listener"); ROS'u başlatır. Düğüm adı listener olacak şekilde ayarlandı.

    ros::NodeHandle n; Tanıtıcı oluşturur.

    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback); talker a chatter konusunda std_msgs/String abone olur. Her mesaj yayınlandığında chatterCallback çağrılır. İkinci argüman ise yayım hızıdır.

    ros::spin(); Bu kod bir döngü girer, mesaj geri çağrımlarını mümkün olduğu kadar hızlı çağrıır.

    return 0;
}

```

Figure 78 Listener.cpp ve Detayları

CMakeLists.txt dosyamızı düzenleyelim.

```

roscd beginner_tutorials
gedit CMakeLists.txt

```

Çalışma alanını derleyin.

```

cd ~/ros_ws
catkin_make

```

```

yoneticici@yoneticipc: ~/catkin_ws
yoneticici@yoneticipc: ~/catkin_ws 74x36

yoneticici@yoneticipc:~$ cd ~/catkin_ws/
yoneticici@yoneticipc:~/catkin_ws$ catkin_make
Base path: /home/yoneticici/catkin_ws
Source space: /home/yoneticici/catkin_ws/src
Build space: /home/yoneticici/catkin_ws/build
Devel space: /home/yoneticici/catkin_ws/devel
Install space: /home/yoneticici/catkin_ws/install
#####
##### Running command: "cmake /home/yoneticici/catkin_ws/src -DCATKIN_DEVEL_PREFIX=/home/yoneticici/catkin_ws/devel -DCMAKE_INSTALL_PREFIX=/home/yoneticici/catkin_ws/install -G Unix Makefiles" in "/home/yoneticici/catkin_ws/build"
#####
-- Using CATKIN_DEVEL_PREFIX: /home/yoneticici/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/yoneticici/catkin_ws/devel;/opt/ros/melodic
-- This workspace overlays: /home/yoneticici/catkin_ws/devel;/opt/ros/melodic
-- Found PythonInterp: /usr/bin/python2 (found suitable version "2.7.15",
minimum required is "2")
-- Using PYTHON_EXECUTABLE: /usr/bin/python2
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/yoneticici/catkin_ws/build/test_results
-- Found gmock sources under '/usr/src/googletest': gmock will be built
-- Found PythonInterp: /usr/bin/python2 (found version "2.7.15")
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Using Python nosetests: /usr/bin/nosetests-2.7
-- catkin 0.7.17
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- 

```

Figure 79 Catkin Derlemesi

Python için:

beginner_tutorials klasörüne gidip scripts klasörü oluşturalım..

```

roscd beginner_tutorials/src
mkdir scripts
cd scripts

```

Publisher dosyamızı oluşturalım.

```
gedit talker.py
```

```

talker.py (~/ros_ws/src/beginner_tutorials/scripts) - gedit
Open Save
#!/usr/bin/env python

## Simple talker demo

import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True) talker düğümü oluşturuluyor.
    rate = rospy.Rate(10) # 10hz Döngü frekansı 10 Hz olarak ayarlanıyor.
    while not rospy.is_shutdown(): Zaman alınıyor
        hello_str = "hello world %s" % rospy.get_time() Mesaj oluşturuluyor
        rospy.loginfo(hello_str) Mesaj ekrana bastırılıyor
        pub.publish(hello_str) Mesaj yayınlanıyor.
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass

```

Master'a **chatter** konusunda *std_msgs/String* türünde bir mesaj yaymayıcağızı bildirir. Üçüncü argüman ise yayım kuyruğunun büyüklüğündür.

`rospy.init_node('talker', anonymous=True)` talker düğümü oluşturuluyor.

`rate = rospy.Rate(10) # 10hz` Döngü frekansı 10 Hz olarak ayarlanıyor.

`while not rospy.is_shutdown():` Zaman alınıyor

`hello_str = "hello world %s" % rospy.get_time()` Mesaj oluşturuluyor

`rospy.loginfo(hello_str)` Mesaj ekrana bastırılıyor

`pub.publish(hello_str)` Mesaj yayınlanıyor.

`rate.sleep()` ROS hata verene kadar işlemler dönüyor

Figure 80 talker.py ve Detayları

Subscriber dosyamızı oluşturalım.

gedit listener.py

```
#!/usr/bin/env python
## Simple listener demo

import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + ' I heard %s', data.data)
    Call-back fonksiyonu
    Alınan mesaj ekrana bastırılıyor

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously
    rospy.init_node('listener', anonymous=True)  listener düğümü oluşturuluyor.

    rospy.Subscriber('chatter', String, callback) Konuya abone olunuyor.

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin() Düğüm sonlanan kadar işlemler devam ediyor.

if __name__ == '__main__':
    listener()
```

Figure 81 Listener.py ve Detayları

Dosyalarımızı çalıştırılabilir yapalım.

*1s
chmod +x listener.py*

Çalışma alanını derleyin.

```
cd ~/ros_ws
```

Figure 82 Catkin Derlemesi

catkin_make

Uygulamayı çalıştırın:

- Terminal 1:
 - Roscore
- Terminal 2:
 - (C++) rosrun beginner_tutorials talker
 - (Python) rosrun beginner_tutorials talker.py
- Terminal 3:
 - (C++) rosrun beginner_tutorials listener
 - (Python) rosrun beginner_tutorials listener.py

```
moguztas@moguztas:~/ros ws$ rosrun beginner_tutorials talker
[ INFO] [1567209367.884622634]: hello world 0
[ INFO] [1567209367.984790365]: hello world 1
[ INFO] [1567209368.0847753585]: hello world 2
[ INFO] [1567209368.184757122]: hello world 3
[ INFO] [1567209368.284752723]: hello world 4
[ INFO] [1567209368.384781284]: hello world 5
[ INFO] [1567209368.484788813]: hello world 6
[ INFO] [1567209368.584788005]: hello world 7
[ INFO] [1567209368.684791028]: hello world 8
[ INFO] [1567209368.784744496]: hello world 9
[ INFO] [1567209368.884766553]: hello world 10
[ INFO] [1567209368.984751599]: hello world 11
[ INFO] [1567209369.084771260]: hello world 12
[ INFO] [1567209369.184755739]: hello world 13
[ INFO] [1567209369.284732317]: hello world 14
[ INFO] [1567209369.384752488]: hello world 15
[ INFO] [1567209369.484772846]: hello world 16
[ INFO] [1567209369.584798485]: hello world 17
[ INFO] [1567209369.684763541]: hello world 18
[ INFO] [1567209369.784781674]: hello world 19
[ INFO] [1567209369.884801013]: hello world 20
[ INFO] [1567209369.984798947]: hello world 21
[ INFO] [1567209370.084707976]: hello world 22
[ INFO] [1567209370.184756887]: hello world 23
[ INFO] [1567209370.284791587]: hello world 24
[ INFO] [1567209370.384794507]: hello world 25
```

Figure 83 Terminal 2

```
moguztas@moguztas:~/ros ws$ rosrun beginner_tutorials listener.py
[INFO] [1567209575.599204]: /listener_16717_1567209575367I heard hello world 165
[INFO] [1567209575.699208]: /listener_16717_1567209575367I heard hello world 166
[INFO] [1567209575.799234]: /listener_16717_1567209575367I heard hello world 167
[INFO] [1567209575.899221]: /listener_16717_1567209575367I heard hello world 168
[INFO] [1567209575.999226]: /listener_16717_1567209575367I heard hello world 169
[INFO] [1567209576.099204]: /listener_16717_1567209575367I heard hello world 170
[INFO] [1567209576.199212]: /listener_16717_1567209575367I heard hello world 171
[INFO] [1567209576.299228]: /listener_16717_1567209575367I heard hello world 172
[INFO] [1567209576.399248]: /listener_16717_1567209575367I heard hello world 173
[INFO] [1567209576.499223]: /listener_16717_1567209575367I heard hello world 174
[INFO] [1567209576.599137]: /listener_16717_1567209575367I heard hello world 175
[INFO] [1567209576.699225]: /listener_16717_1567209575367I heard hello world 176
[INFO] [1567209576.799226]: /listener_16717_1567209575367I heard hello world 177
[INFO] [1567209576.899234]: /listener_16717_1567209575367I heard hello world 178
[INFO] [1567209576.999163]: /listener_16717_1567209575367I heard hello world 179
[INFO] [1567209577.099210]: /listener_16717_1567209575367I heard hello world 180
[INFO] [1567209577.199121]: /listener_16717_1567209575367I heard hello world 181
[INFO] [1567209577.299158]: /listener_16717_1567209575367I heard hello world 182
[INFO] [1567209577.399200]: /listener_16717_1567209575367I heard hello world 183
[INFO] [1567209577.499195]: /listener_16717_1567209575367I heard hello world 184
[INFO] [1567209577.599238]: /listener_16717_1567209575367I heard hello world 185
[INFO] [1567209577.699100]: /listener_16717_1567209575367I heard hello world 186
[INFO] [1567209577.799165]: /listener_16717_1567209575367I heard hello world 187
[INFO] [1567209577.899168]: /listener_16717_1567209575367I heard hello world 188
[INFO] [1567209577.999260]: /listener_16717_1567209575367I heard hello world 189
```

Figure 84 Terminal 3

3.12.6. Uygulama 6: Service-Client Uygulaması

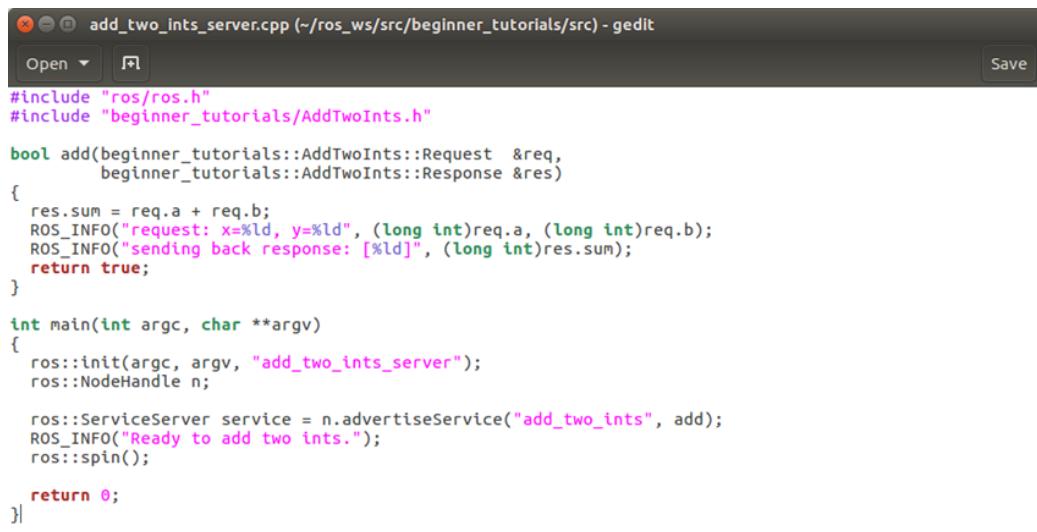
C++ için:

beginner_tutorials altındaki src klasörüne gidelim.

```
roscd beginner_tutorials/src
```

Server Dosyamızı oluşturalım.

```
gedit add_two_ints_server.cpp
```



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

bool add(beginner_tutorials::AddTwoInts::Request &req,
          beginner_tutorials::AddTwoInts::Response &res)
{
    res.sum = req.a + req.b;
    ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
    ROS_INFO("sending back response: [%ld]", (long int)res.sum);
    return true;
}

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

    ros::ServiceServer service = n.advertiseService("add_two_ints", add);
    ROS_INFO("Ready to add two ints.");
    ros::spin();
}

return 0;
}
```

Figure 85 add_two_ints_server.cpp

client dosyamızı oluşturalım.

```
gedit add_two_ints_client.cpp
```

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_two_ints_client");
    if (argc != 3)
    {
        ROS_INFO("usage: add_two_ints_client X Y");
        return 1;
    }

    ros::NodeHandle n;
    ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
    beginner_tutorials::AddTwoInts srv;
    srv.request.a = atol(argv[1]);
    srv.request.b = atol(argv[2]);
    if (client.call(srv))
    {
        ROS_INFO("Sum: %ld", (long int)srv.response.sum);
    }
    else
    {
        ROS_ERROR("Failed to call service add_two_ints");
        return 1;
    }

    return 0;
}
```

Figure 86 add_two_ints_client.py

CMakeLists.txt dosyamızı düzenleyelim.

```
roscd beginner_tutorials  
gedit CMakeLists.txt
```

Çalışma alanını derleyin.

```
cd ~/ros_ws  
catkin_make
```

Figure 87 Catkin Derlemesi

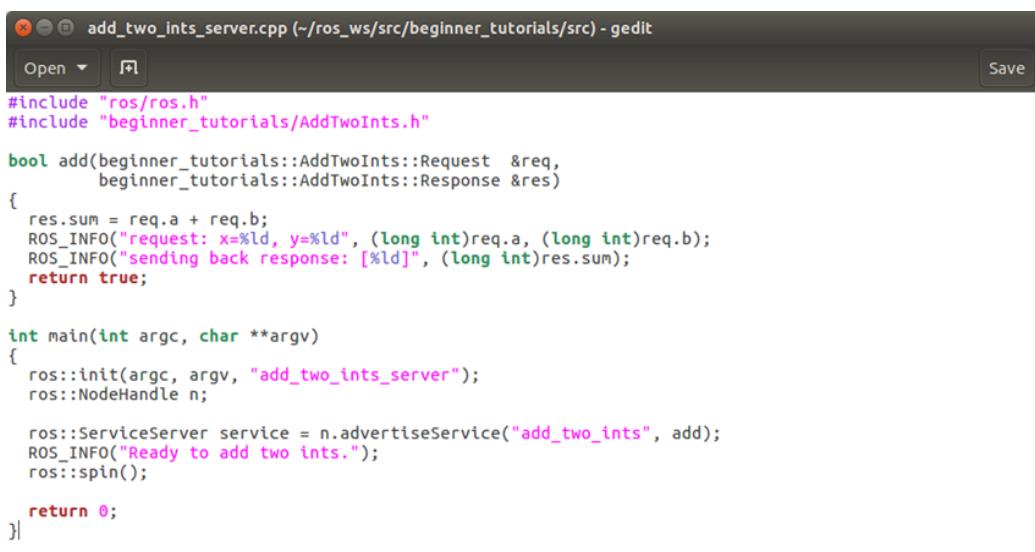
Python için:

beginner_tutorials altındaki src klasörüne gidelim.

```
roscd beginner_tutorials/src
```

Server dosyamızı oluşturalım.

```
gedit add_two_ints_server.cpp
```



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

bool add(beginner_tutorials::AddTwoInts::Request &req,
          beginner_tutorials::AddTwoInts::Response &res)
{
    res.sum = req.a + req.b;
    ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
    ROS_INFO("sending back response: [%ld]", (long int)res.sum);
    return true;
}

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

    ros::ServiceServer service = n.advertiseService("add_two_ints", add);
    ROS_INFO("Ready to add two ints.");
    ros::spin();

    return 0;
}
```

Figure 88 add_two_ints_server.py

Client dosyamızı oluşturalım.

```
gedit add_two_ints_client.cpp
```

```

add_two_ints_client.cpp (~/ros_ws/src/beginner_tutorials/src) - gedit
Open Save
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "add_two_ints_client");
    if (argc != 3)
    {
        ROS_INFO("usage: add_two_ints_client X Y");
        return 1;
    }

    ros::NodeHandle n;
    ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
    beginner_tutorials::AddTwoInts srv;
    srv.request.a = atol(argv[1]);
    srv.request.b = atol(argv[2]);
    if (client.call(srv))
    {
        ROS_INFO("Sum: %ld", (long int)srv.response.sum);
    }
    else
    {
        ROS_ERROR("Failed to call service add_two_ints");
        return 1;
    }

    return 0;
}

```

Figure 89 add_two_ints_client.py

Dosyalarımızı çalıştırılabilir yapalım.

```

chmod +x add_two_ints_server.py
chmod +x add_two_ints_client.py
ls

```

Çalışma alanını derleyin.

```

cd ~/ros_ws
catkin_make

```

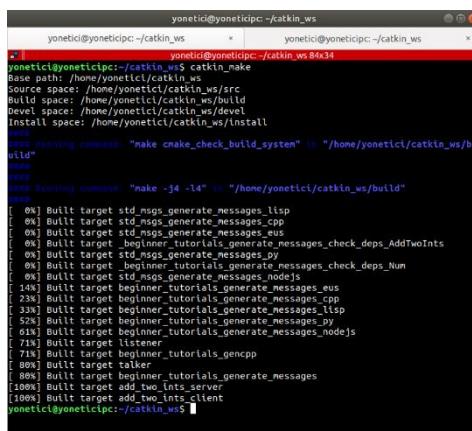


Figure 90 Catkin Derlemesi

Uygulamayı çalıştırın:

- Terminal 1:
 - Roscore
- Terminal 2:
 - (C++) rosrun beginner_tutorials add_two_ints_server
 - (Python) rosrun beginner_tutorials add_two_ints_server.py
- Terminal 3:
 - (C++) rosrun beginner_tutorials add_two_ints_server
 - (Python) rosrun beginner_tutorials add_two_ints_server.py

```
roscore http://yoneticipc:11311/ 84x7
process[master]: started with pid [7299]
ROS_MASTER_URI=http://yoneticipc:11311/
setting /run_id to 3b3944da-91c2-11e9-a9be-c8ff28c2981e
process[rosout-1]: started with pid [7310]
started core service [/rosout]
```

Figure 93 Terminal 1

```
yonetici@yoneticipc: ~/catkin_ws 84x4
[ INFO] [1560859913.257761091]: Ready to add two ints.
[ INFO] [1560859976.429118549]: request: x=5, y=7
[ INFO] [1560859976.429148988]: sending back response: [12]
```

Figure 91 Terminal 2

```
yonetici@yoneticipc: ~/catkin_ws 84x19
yonetici@yoneticipc:~/catkin_ws$ rosrun beginner_tutorials add_two_ints_client 5 7
[ INFO] [1560859976.429264741]: Sum: 12
yonetici@yoneticipc:~/catkin_ws$
```

Figure 92 Terminal 3

3.12.7. Uygulama 7: Verileri Kaydetme ve Oynatma

Roscore'u çalıştırın

```
roscore
```

TurtleSim'i açın.

```
rosrun turtlesim turtlesim_node
```

Klavye kontrol düğümünü açın.

```
rosrun turtlesim turtle_teleop_key
```

beginner_tutorials klasörü altına bagfiles isimli klasör açın.

```
mkdir ~/bagfiles  
cd ~/bagfiles
```

Yayınlanan tüm konuları kaydetmek için:

```
roscd beginner_tutorials/src
```

Bazı konuları kaydetmek için:.

```
gedit add_two_ints_server.cpp
```

Robot figürlerde görüldüğü gibi klavye yardımıyla hareket ettirilebilir.

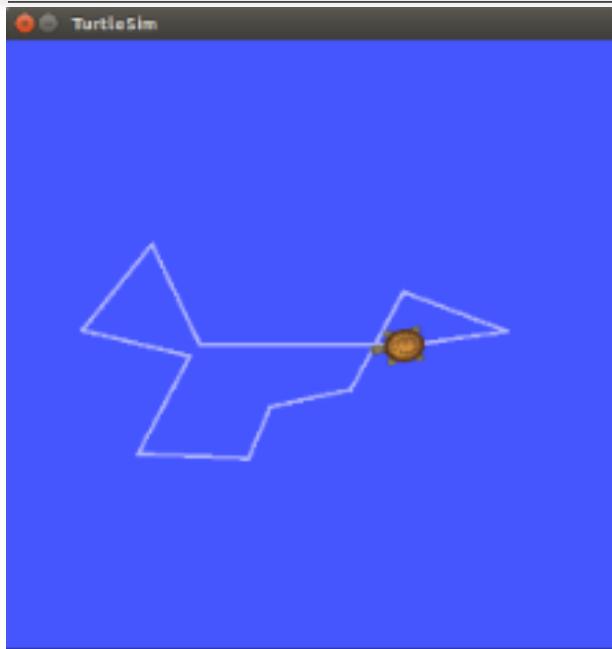


Figure 94 TurtleSim Teleoperasyonu

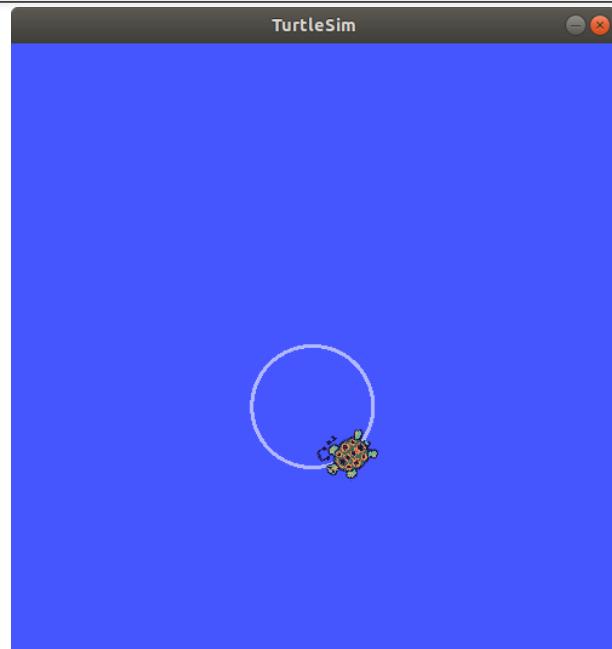


Figure 95 Sabit hız verildiğinde TurtleSim

Bag dosyasının içeriği control edilirse.

```
rosbag info bag_file
```

```
yonetici@yoneticipc:~/catkin_ws$ cd
yoneticici@yoneticipc:~/bagfiles$ rosbag record -a
[ INFO] [1560861409.542054607]: Recording to 2019-06-18-15-36-49.bag.
[ INFO] [1560861409.543464238]: Subscribing to /turtle1/color_sensor
[ INFO] [1560861409.545428123]: Subscribing to /turtle1/cmd_vel
[ INFO] [1560861409.547594081]: Subscribing to /rosout
[ INFO] [1560861409.550016397]: Subscribing to /rosout_agg
[ INFO] [1560861409.552390785]: Subscribing to /turtle1/pose
```

Figure 96 rosbag çalıştırılması

```
yonetici@yoneticipc:~/bagfiles$ rosbag info 2019-06-18-15-36-49.bag
path: 2019-06-18-15-36-49.bag
version: 2.0
duration: 53.4s
start: Jun 18 2019 15:36:49.55 (1560861409.55)
end: Jun 18 2019 15:37:42.97 (1560861462.97)
size: 463.0 KB
messages: 6654
compression: none [1/1 chunks]
types: rosgngh_msgs/Log [acffd30cd6bde30f120938c17c593fb]
       turtlesim/Color [353891e354491c51aae32df673fb446]
       turtlesim/Pose [863b248d5016ca62ea2e895ae5265cf9]
topics: /rosout 4 msgs
       : rosgngh_msgs/Log (2 connections)
       : /turtle1/color_sensor 3325 msgs
       : turtlesim/color
       : /turtle1/pose 3325 msgs
       : turtlesim/Pose
```

Figure 97 rosbag içeriği

Bag Dosyasını oynatalım

```
yonetici@yoneticipc:~/ros_ws/src/beginner_tutorials/bagfiles$ rosbag play 2019-08-31-03-21-38.bag
[ INFO] [1567211137.277837787]: Opening 2019-08-31-03-21-38.bag
Waiting 0.2 seconds after advertising topics... done.
Hit space to toggle paused, or 's' to step.
[RUNNING] Bag Time: 1567210939.989698 Duration: 41.603941 / 76.436171
```

Figure 98 rosbag'in çalıştırılması

-O argümanı rosbag record komutuna subset.bag isimli bir dosyaya sadece bu iki konuyu takip edip yazmasını söyler.

4. Gazebo

4.1. Gazebo nedir?

Robotların iç ve dış ortamlarda simülasyonu için gürbüz fizik motoru, yüksek kalitede grafikler ve grafik arayüzü ile gerekli altyapıyı sunar.

Gazebo:

- ODE, Bullet, Simbody ve DART gibi yüksek performansa sahip fizik motorlarının kullanımını,
- OGRE sayesinde ortamların gerçekçi bir şekilde oluşturulmasını,
- Sensör ve sensör verilerinin kullanımını,
- Mevcut robot modellerinin kullanımını veya kendi robot modelini oluşturma imkanını



Figure 99 Gazebo Simülasyon platformu

Sağlamaktadır.

4.2. Gazebo Bileşenleri

Gazebo bünyesinde gerçekleştirilen bir simülasyon aşağıdaki öğeleri içermektedir:

- Dünya dosyaları
- Model dosyaları
- Ortam Değişkenleri
- Gazebo sunucusu
- Gazebo istemcisi

- Eklentiler (Plugins)

4.2.1. Dünya Dosyaları

Dünya dosyaları, simülasyon ortamındaki;

- robot,
- sensör,
- ışık,
- nesne

gibi ortam elemanlarını tanımlamak için kullanılmaktadır.

Bu dosyalar SDF isimli tanımlama biçimini kullanılarak oluşturulmakta ve genel itibariyle .world uzantısına sahip olmaktadır.

Çeşitli dünya dosyaları Gazebo kurulumuyla beraber gelmekte ve

- <install_path>/share/gazebo-<version>/worlds

İçerisinde yer almaktadır.

Aynı zamanda SDF tanımlama biçimini kullanılarak kullanıcı tanımlı dünya dosyaları oluşturmak da mümkündür.

Örnek bir dünya dosyası olan “empty.world” aşağıdaki gibi bir içeriye sahiptir.

```
<?xml version="1.0" ?>
<sdf version="1.5">
    <world name="default">
        <!-- A global light source -->
        <include>
            <uri>model://sun</uri>
```

```
</include>
<!-- A ground plane -->
<include>
    <uri>model://ground_plane</uri>
</include>
</world>
</sdf>
```

4.2.2. Model Dosyaları

Dünya dosyaları gibi SDF tanımlama biçimi kullanmaktadır ve sadece

```
<model>
    ...
</model>
```

şeklinde tag içermektedir.

Bu dosyaların kullanımındaki amaç, modellerin tekrar kullanılabilirliğini sağlamak ve dünya dosyalarını basitleştirmektir.

Oluşturulan bir model, dünya dosyası içerisinde aşağıdaki gibi kullanılabilmektedir.

```
<include>
    <uri>model://model_file_name</uri>
</include>
```

Simülasyon ortamı için, online veritabanında yer alan veya eski Gazebo versiyonlarında kurulumla beraber gelen model dosyaları kullanılabileceği gibi, kullanıcı tarafından tanımlanan model dosyaları da kullanılabilmektedir.

4.2.3. Ortam Değişkenleri

Gazebo, dosyaları konumlandırip erişebilmek ve sunucu ile istemci arasındaki ilişkiyi kurabilmek adına çeşitli ortam değişkenleri kullanmaktadır. Kurulumla beraber ortam değişkenleri derlenmiş olarak gelir.

Varsayılan ortam değişkenleri aşağıdaki shell script (.sh dosyası) içerisinde yer almaktadır.

- <install_path>/share/gazebo/setup.sh

Gazebo'nun dikkate alacağı ortam değişkenleri değiştirmek istenirse ilgili shell script dosyası

- source <install_path>/share/gazebo/setup.sh

ile kaynak gösterilerek, istege bağlı olarak modifiye edilmelidir.

Örnek bir script dosyası içeriği aşağıda gösterilmektedir.

```
export GAZEBO_MASTER_URI=http://localhost:11345
export
GAZEBO_MODEL_DATABASE_URI=http://gazebosim.org/models
    export GAZEBO_RESOURCE_PATH=/usr/share/gazebo-
7:/usr/share/gazebo_models:${GAZEBO_RESOURCE_PATH}
    export GAZEBO_PLUGIN_PATH=/usr/lib/x86_64-linux-
gnu/gazebo-7/plugins:$ {GAZEBO_PLUGIN_PATH}
    export GAZEBO_MODEL_PATH=/usr/share/gazebo-
7/models:${GAZEBO_MODEL_PATH}
    export
LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/lib/x86_64-
linux-gnu/gazebo- 7/plugins
```

4.2.4. Gazebo Sunucusu

Gazebo sunucusu, Gazebo'nun tüm işlem yükünü üstlenen kısmıdır. Kendisine verilen bir dünya dosyasını ayırtırır ve ardından bir fizik ve sensör motoru kullanarak dünyayı simüle eder.

Gazebo sunucusu herhangi bir grafik arayüzü içermemektedir.

Gazebo sunucusu terminalde

- gzserver

komutuyla yalnız olarak başlatılabilceği gibi

- gzserver worlds/empty.world

komutunda olduğu gibi bir dünya dosyası (bu örnekte Gazebo ile birlikte gelen bir dünya dosyası) ile birlikte de başlatılabilir.

4.2.5. Gazebo İstemcisi

Gazebo istemcisi, çalışan bir Gazebo sunucusuna bağlanarak ve ortama ait elemanların görselleştirilmesi görevini üstlenmektedir.

Aynı zamanda kullanıcıya çalışan bir simülasyon üzerinde değişiklik yapma fırsatı sunmaktadır.

Gazebo sunucusu, terminalde

- gzclient

komutuyla yalnız olarak başlatılabilirmektedir.

4.2.6. Eklentiler (plugins)

Eklentiler, ortak kütüphane dosyası olarak derlenen ve simülasyona eklenen kod parçaları olarak ele alınmaktadır. Eklentilerin, standart C++ sınıfları üzerinden tüm Gazebo işlevselligine erişmeleri mümkündür.

Eklentiler, Gazebo'nun her yönüyle kontrol edilmesine imkan tanımakta ve çalışan bir sisteme eklenip çıkarılabilir hale getirilmektedir.

Programlı bir şekilde simülasyonda bir şeyle değiştirmek istendiğinde (modelleri hareket ettirmek, belirli koşullar sağlandığında yeni model eklemek vb.), Gazebo için hızlı bir arayüz istendiğinde, oluşturulan eklenti başkaları için de faydalı olabilecek şekilde paylaşmak istendiğinde eklentiler kullanılabilir.

Mevcut olarak altı tane eklenti türü bulunmaktadır.

- Dünya
- Model
- Sensör
- Sistem
- Görsel
- Arayüz

Her eklenti türü farklı bir Gazebo bileşeni tarafından yönetilmektedir. Örneğin, bir model eklentisi Gazebo'daki belirli bir modele eklenmekte ve bu modeli kontrol etmektedir.

Eklenti türü, istenilen işlevselligi göre seçilmelidir. Örneğin;

- Fizik motoru, ortam aydınlatması vb. dünya özelliklerini kontrol etmek için bir dünya eklentisi,
- Eklemlerin ve modelin durumunu kontrol etmek için bir model eklentisi,
- Sensör bilgilerini elde etmek ve sensör özelliklerini kontrol etmek için ise bir sensör eklentisi

Kullanılabilir. Eklentiler, aşağıdaki şekilde terminal komut satırından yüklenebileceği gibi;

- `gzserver -s <plugin_filename>`

SDF dosyası içerisinde aşağıdaki şekilde belirtilerek de;

```
<plugin name="gazebo_ros_control"
filename="libgazebo_ros_control.so">
    <robotNamespace>/MYROBOT</robotNamespace>
</plugin>
```

Yüklenebilirler.

4.3. Gazebo uygulamaları

4.3.1. Gazebo Kurulumu

Gazebo, ROS tarafından kullanılan bağımsız bir projedir. Genellikle, her ROS sürüm döngüsünün başında mevcut olan en yeni Gazebo sürümü tam olarak entegre ve desteklenecek şekilde resmi olarak seçilmektedir.

Ubuntu üzerinde Gazebo yüklemek için paketler kullanılabilir. Gazebo paketlerine ev sahipliği yapan iki ana kaynak (packages.ros.org ve packages.osrfoundation.org) bulunmaktadır.

- packages.ros.org üzerinde
 - ROS Indigo: Gazebo 2.x
 - ROS Kinetic: Gazebo 7.x
 - ROS Lunar: Gazebo 7.x
 - ROS Melodic: Gazebo 9.x
- packages.osrfoundation.org üzerinde
 - gazebo 7.x (gazebo7 paket adıyla)
 - gazebo 8.x (gazebo8 paket adıyla)

- gazebo 9.x (gazebo9 paket adıyla)

paketleri yer almaktadır. Bu kaynaklardan biri, Gazebo yüklemek için kullanılabilir.

ROS'un belirli bir sürümünü çalıştırması gereken ve Gazebo ROS ile ilgili tüm paketleri kullanıma hazır şekilde kullanmak isteyen kullanıcılar için, packages.ros.org kaynağında yer alan Gazebo sürümünün kullanılması önerilmektedir. Örneğin ROS Kinetic, Gazebo'nun 7.x sürümünü barındırmakta ve kullanmaktadır.

İhtiyaç duyulduğu takdirde önerilen seçeneklerin dışında belirli bir Gazebo ve ROS versiyonunu birlikte kullanmak mümkündür (uyumluluk sorunları oluşabilir). Ancak bu durumda Gazebo ile ilgili herhangi bir ROS Ubuntu paketi, ROS dağıtım kaynağından kullanılamamaktadır.

Paketlerin eşdeğeri OSRF kaynağından yüklenebilir, fakat diğer bütün yazılımlar catkin_workspaces kullanılarak kaynaktan oluşturulmalıdır.

Ayrıca mevcut ROS kurulumuyla beraber gelen bir Gazebo versiyonu mevcutsa, farklı bir Gazebo versiyonu kurulmak istendiğinde ilk olarak bu versiyon kurulumunun kaldırılması gerekebilir.

4.3.2. Gazebo çalıştırılması

Terminalden

- gazebo

komutu çalıştırarak Gazebo başlatılabilir. Grafik arayüzünde de görülebileceği üzere, Gazebo istemci sayesinde çalışan simülasyonu belirli bir ölçüde modifiye etmek mümkündür.

4.3.3. Çalışma Ortamının Hazırlanması

ROS-Gazebo çalışmalarını gerçekleştirebilmek adına Catkin çalışma ortamı hazırlanabilir.

Daha önceden oluşturduğumuz “catkin_workspace” üzerine:

```
mkdir -p ~/catkin_ws/src  
cd ~/catkin_ws/src  
catkin_init_workspace  
cd ~/catkin_ws  
catkin_make
```

komutlarıyla gerçekleştirilecek örnek çalışma için paketimizi oluşturalım.

4.3.4. Dünya Oluşturulması

Simülasyon ortamında kullanılacak dünyayı oluştururken, Gazebo veritabanında yer alan hazır dünya dosyalarından yararlanılabildiği gibi, kullanıcı kendi dünyasını da oluşturabilmektedir.

Dünya dosyası;

- grafik arayüzü aracılığıyla,
- elle SDF formatına uygun tanımlamalar yaparak

oluşturulabilir.

Grafik arayüzü aracılığıyla dünya oluşturmak için terminalden aşağıdaki komutları sırasıyla çalıştırarak Gazebo’yu başlatalım

```
cd  
gazebo
```

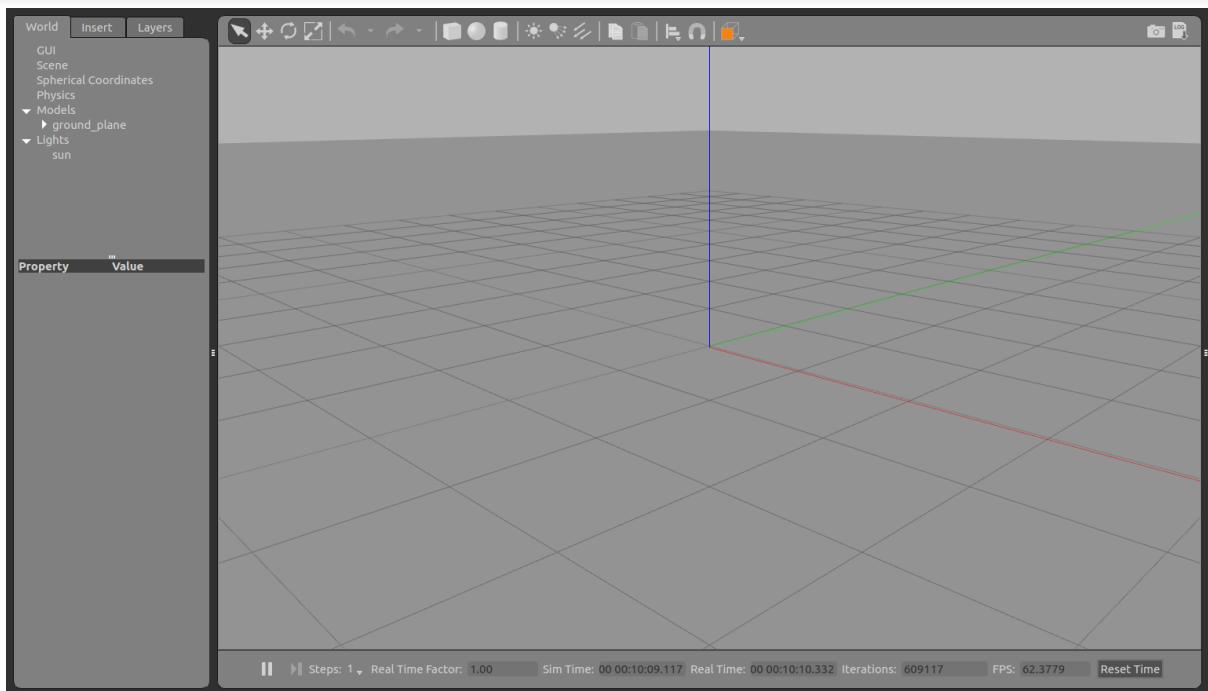


Figure 100 Gazebo Çalışma Ortamı

Sol tarafta yer alan World kısmında da görüldüğü üzere ortamda model olarak “ground_plane”, Lights olarak “sun” yer almaktadır. Yeni bir model eklemek için aşağıda görüldüğü üzere “Simple Shapes” kısmından basit nesneler ekleneceği gibi, “Insert” kısmında yer alan model veritabanından da model eklenebilir.

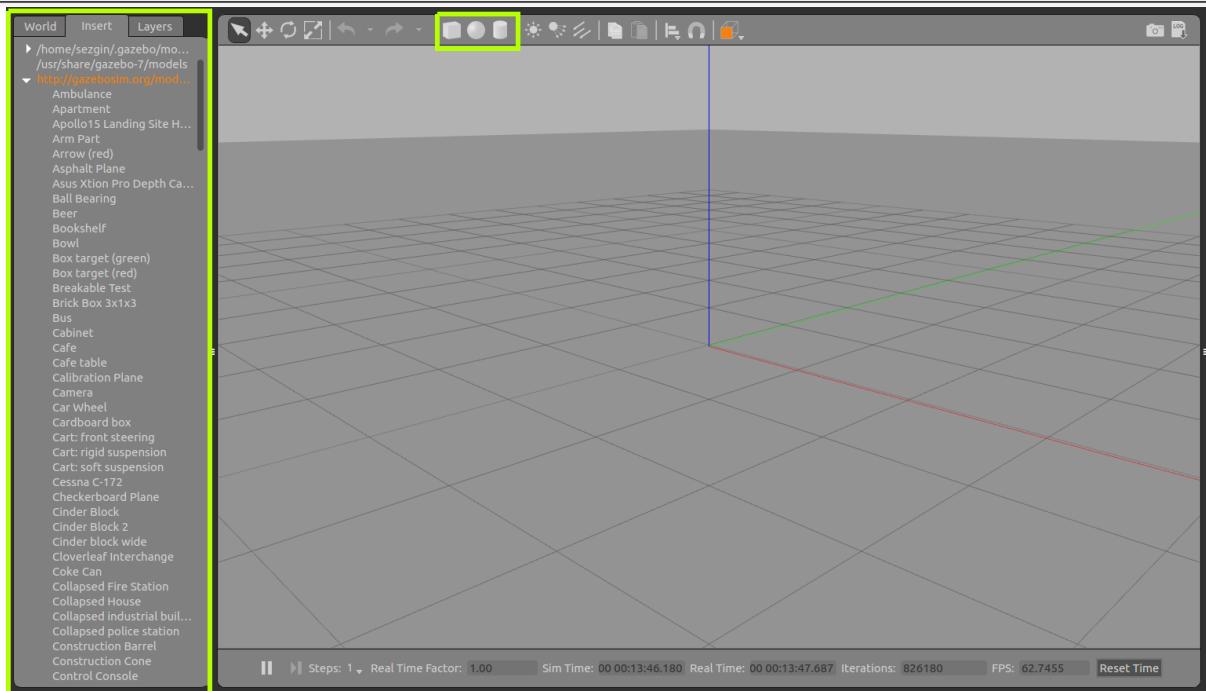


Figure 101 Model ekleme

Eklenen modeller üzerinde değişiklik yapılmak istendiğinde üst araç çubuğuunda yer alan “Translation” seçeneğiyle modelin konumu, “Rotation” kısmıyla modelin duruşu, “Scale” seçeneğiyle de modelin boyutu değiştirilebilir. Aynı şekilde ilgili model seçilmiş haldeyken sol kısımda yer alan seçeneklerden modelin konumu ve duruşu değiştirilebilir.

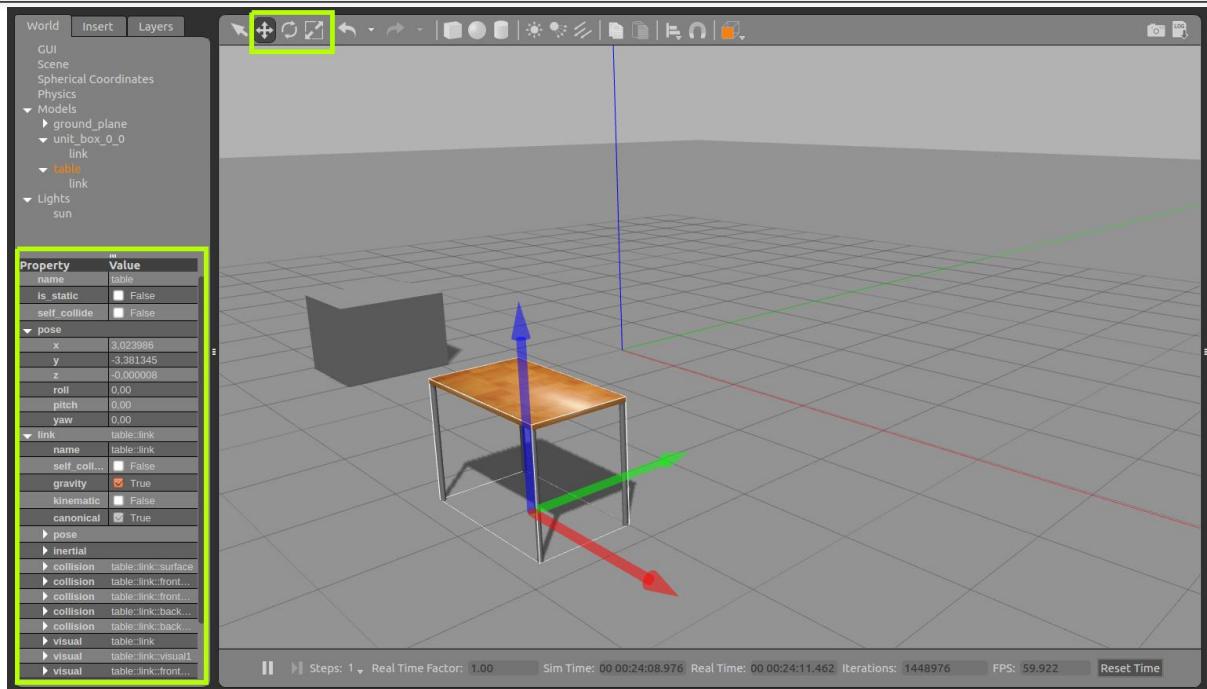


Figure 102 Model Özellikleri

Dünya istenildiği gibi oluşturulduğu takdirde, “File” -> “Save World As” seçenekinden kaydedilebilir. Bu çalışmada dünyamızı, “~/catkin_ws/src/myrobot/worlds” klasörüne, “first_world_gui.sdf” ismiyle kaydedebiliriz.

Gazebo’yu kapatıktan sonra oluşturduğumuz dünyayı

```
cd
cd catkin_ws/src/myrobot/worlds/
gazebo first_world_gui.sdf
```

komutlarıyla görebiliriz.

4.3.5. Model Oluşturulması

Simülasyon ortamında kullanılacak modeller için dünya oluştururken olduğu gibi model dosyası da;

- grafik arayüzünün içерdiği “Model Editor” aracılığıyla,
- elle SDF veya URDF, URDF.XACRO formatına uygun tanımlamalar yaparak,
oluşturulabilir.

SDF formatı kullanılarak oluşturulan dosyalar doğrudan Gazebo ile kullanılabilirken, URDF ve URDF.XACRO formatındaki dosyalar ekstra işleme gerek duyabilmektedir.

Grafik arayüzünde yer alan “Model Editor” aracılığıyla model oluşturmak için;

Terminalden aşağıdaki komutları sırasıyla çalıştırarak Gazebo’yu başlatalım

```
cd  
gazebo
```

Araç çubuğunda yer alan “Edit” -> “Model Editor” kısmından model editörünü açalım.

“Model Editor” açıldığında “Links” yani model uzvu eklemek için birkaç seçenekimiz bulunmaktadır. Solda yer alan “Insert” kısmından, “Simple Shapes” ile basit geometrik parçaları modele ekleyebilirken, “Custom Shapes” seçeneğiyle elimizde bulunan “Mesh” parçalarını modele ekleyebiliriz. Gazebo7, “Mesh” dosyası olarak .svg, .dae ve .stl dosya türlerini desteklemektedir.

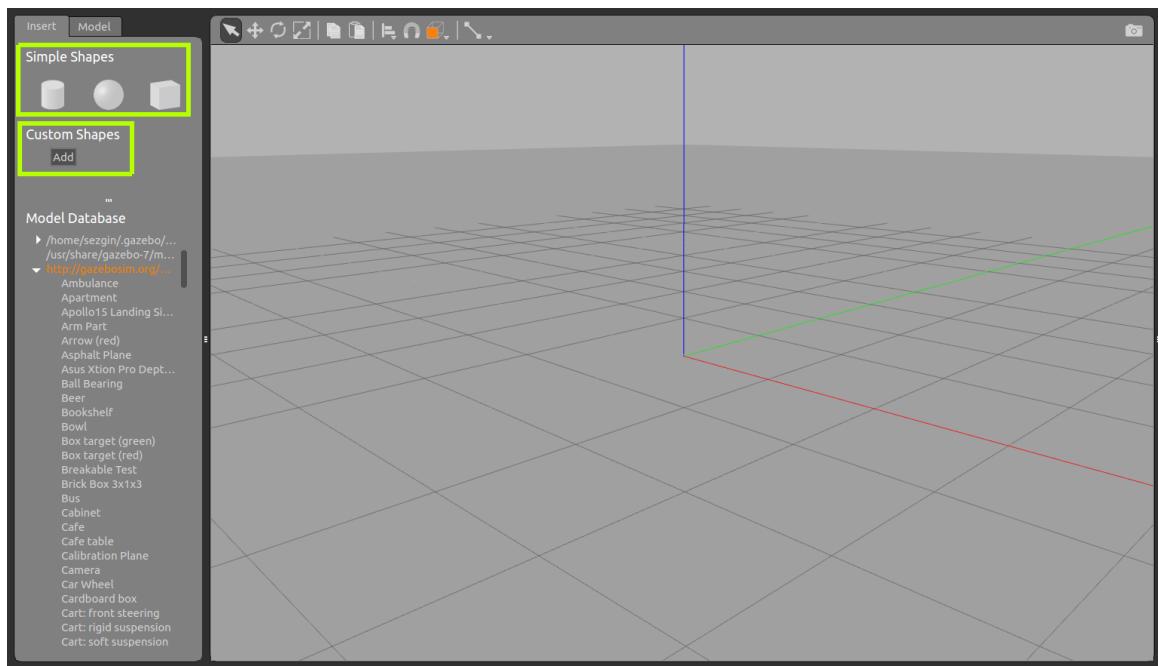


Figure 103 Basit ve Özel şekiller

İlk olarak bir tane “Box” ve daha sonra dört tane “Cylinder” ekleyerek toplamda beş “Link”i modele ekleyelim. Bu uzuqlar eklenme sırasına göre link_0, link_1, link_2, link_3 ve link_4 olarak isimlendirilecektir. Daha sonra eklediğimiz “Link” parçalarını solda yer alan “Model” kısmından görebiliriz. Aynı kısımdan model adını, yani “Model Name” kısmını “first_robot_gui” olarak değiştirelim.

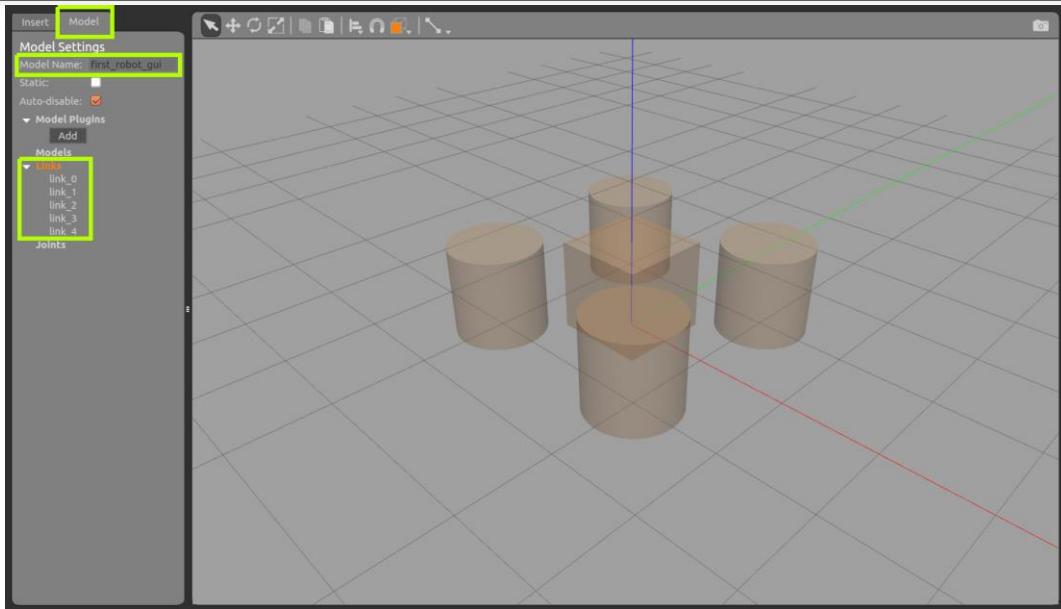


Figure 104 Model bağlantıları

Parçaların özelliklerini değiştirmek için öncelikle eklediğimiz “Box” (ekleniş sırasına göre link_0 adını alacaktır) şekline sağ tuşla tıklayarak “Open Link Inspector” seçeneğine tıklayalım.

link_0 için “Visual” ve “Collision” kısımlarında “Geometry” özelliklerini aşağıdaki şekildeki gibi ayarlayıp, parçamızın hem görsel, hem de katı çarpışma modelindeki boyutlarını değiştirerek 1x2x0.5 m’lik bir “Box” elde edelim.

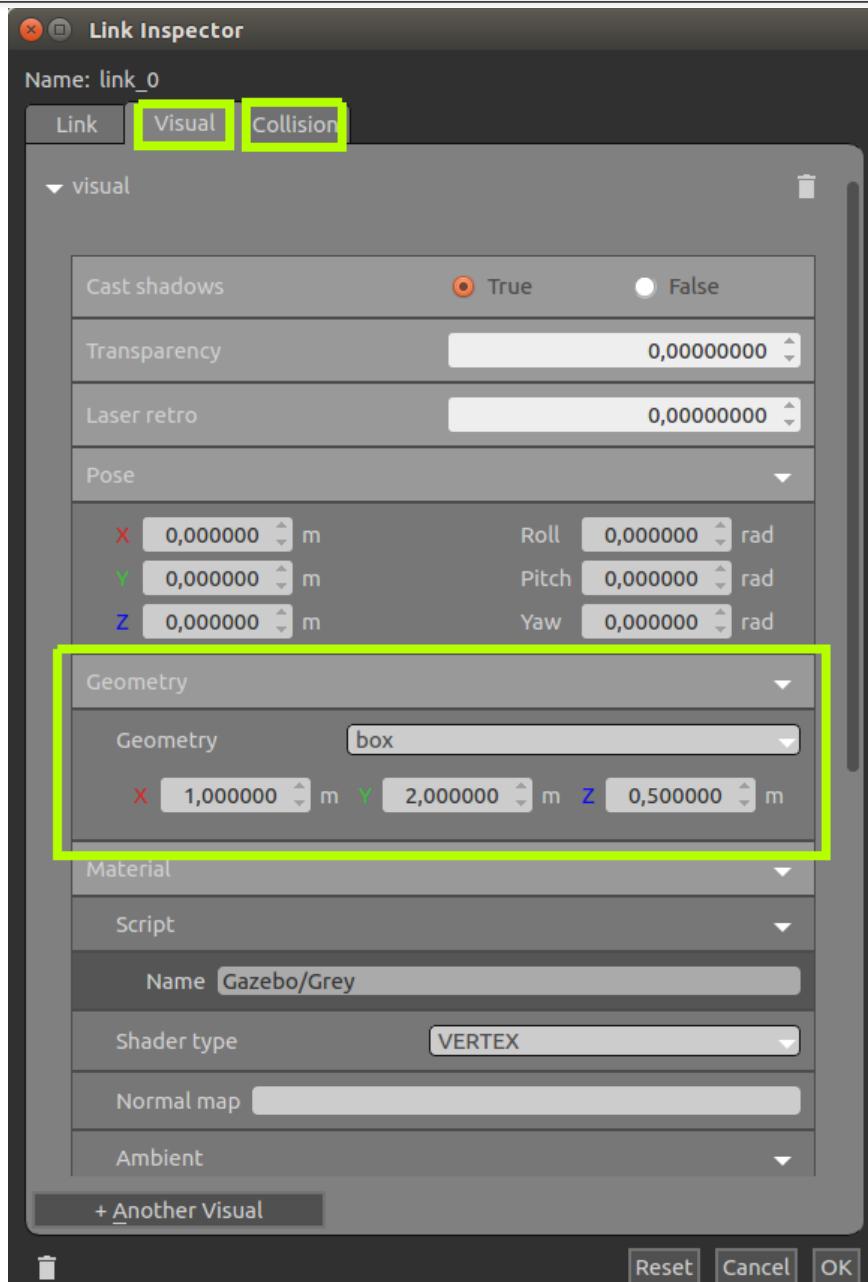


Figure 105 Bağlantı atalet ve pozisyonları

Yine link_0 için bu sefer "Link" kısmında yer alan "Pose" seçeneğinden, "Box" parçasının merkez noktasının duruşunu simülasyon ortamının orjinine göre z ekseninde 0.5m yukarıda bulunacak şekilde aşağıdaki gibi ayarlayalım. "Inertial" kısmında yer alan "Pose" kısmına ise atalet ile alakalı özelliklerde bir değişiklik olmayacağı için herhangi bir değişiklik yapmayağım.

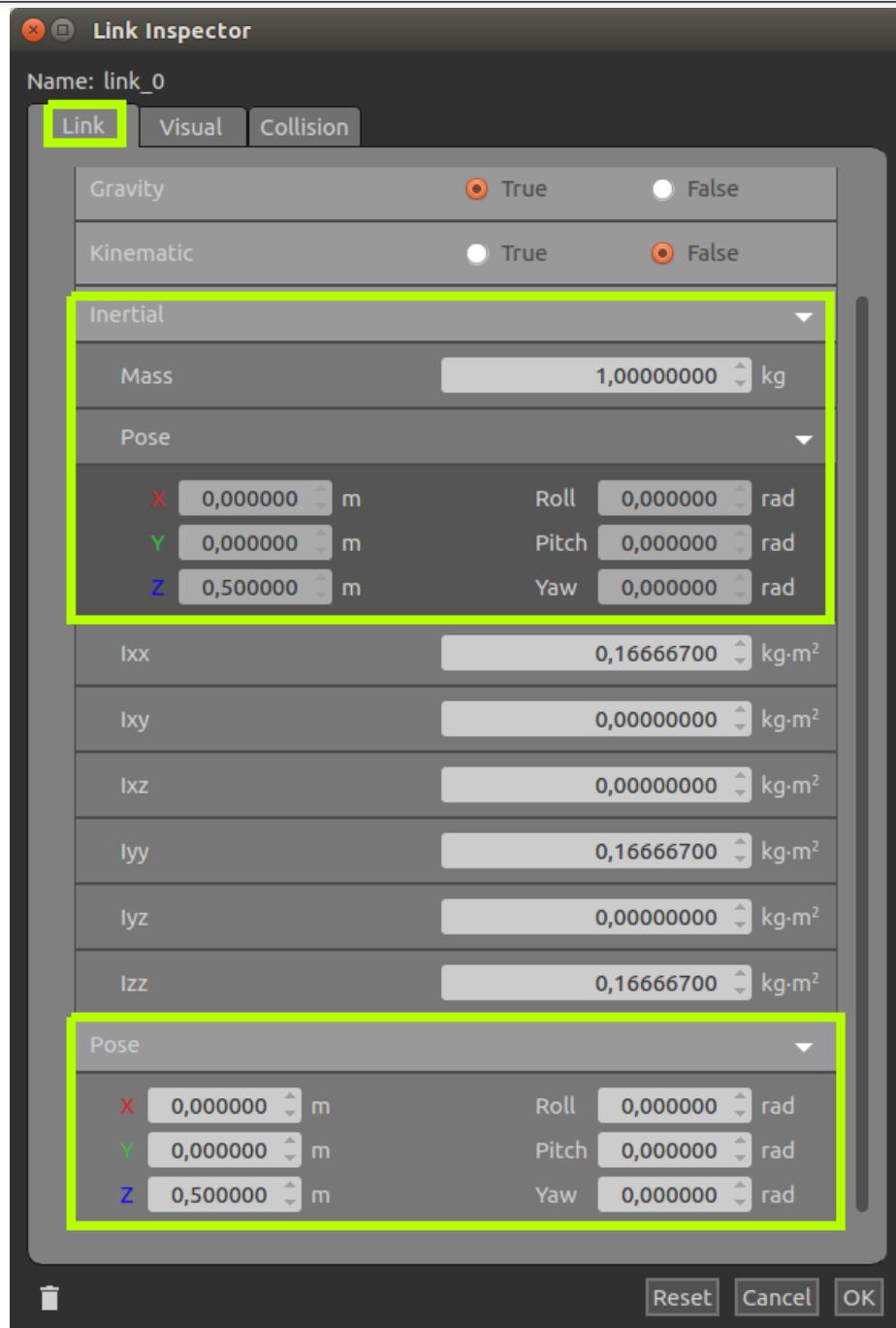


Figure 106 Geometri özellikleri

Daha sonra ortama link_1, link_2, link_3 ve link_4 olarak eklemiş olduğumuz her bir “Cylinder” parçası için “Visual” ve “Collision” kısımlarındaki “Geometry” alanından boyutları 0.5x0.5m olarak ayarlayalım.

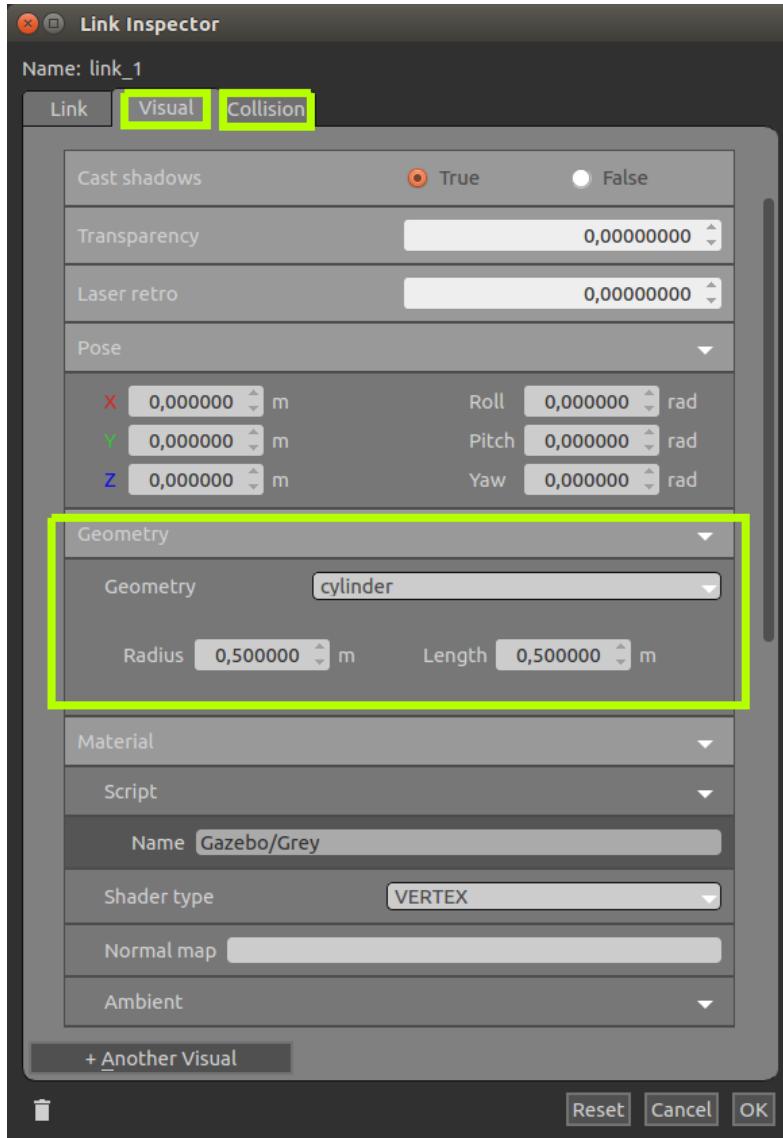


Figure 107 Geometri alan boyutu

Son olarak da “Link” kısmında yer alan “Pose” seçeneklerinden, her bir link için ayrı ayrı aşağıdaki ayarları yapalım.

Her birinin “Pose” kısımlarındaki “Pose” bölümünden “Roll”, “Pitch” ve “Yaw” değerleri aynı şekilde sırasıyla “0,000000”, “1,570000” ve “0,000000” olacak.

“Pose” kısımlarındaki “Pose” değerlerinde yer alan “X”, “Y” ve “Z” değerleri ise;

- link_1 için;
 - X : -0,750000
 - Y : -1,000000
 - Z: 0,500000
- link_2 için;
 - X : -0,750000
 - Y : 1,000000
 - Z: 0,500000
- link_3 için;
 - X : 0,750000
 - Y : 1,000000
 - Z: 0,500000
- link_4 için;
 - X : 0,750000
 - Y : -1,000000
 - Z: 0,500000

olacaktır. “Link” parçalarını doğru bir şekilde ayarladığınızda simülasyonda oluşan görüntü aşağıdaki şekildeki gibi olacaktır.

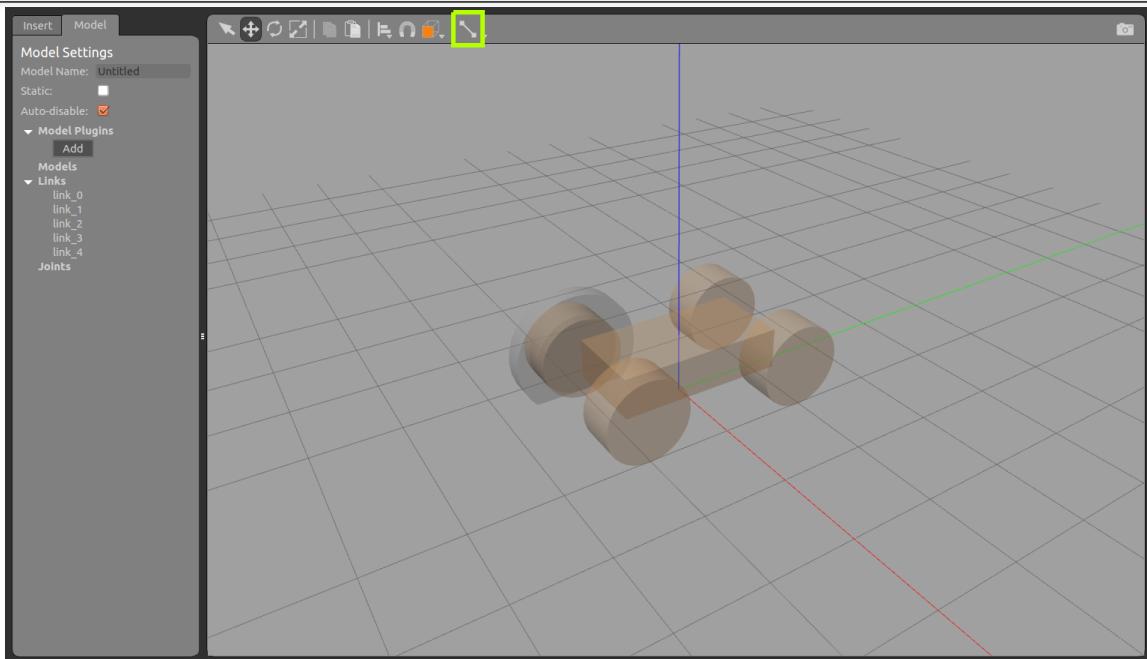


Figure 108 Oluşacak Model

Modelimize “Joints” yani eklem eklemek için ise üst şekilde gösterildiği gibi üst kısımda yer alan araç çubuğuundan “Joint” butonuna tıklayabiliriz. Burada eklem tipini, üst ve alt link seçimini ve eklem ekseni seçebilmekteyiz. Bütün ayarları yaptıktan sonra “Create” seçeneğiyle “Joint” oluşturabiliriz. Bu işlemi her bir alt link için yapacağımızdan bu çalışmada toplamda dört tane “Joint” oluşturmamız gerekmektedir.

“Joint types” her bir eklem için “Revolute”,

“Link Selections” kısmındaki “Parent” her bir eklem için “link_0”,

“Link Selections” kısmındaki “Child” ilgili eklem için “link_x” (x sırasıyla 1,2,3,4),

“Joint axis” her bir eklem için “Z”

olacak şekilde ayarlayıp, her bir ayardan sonra “Create” diyerek bir diğer eklemin ayarlarına geçelim.

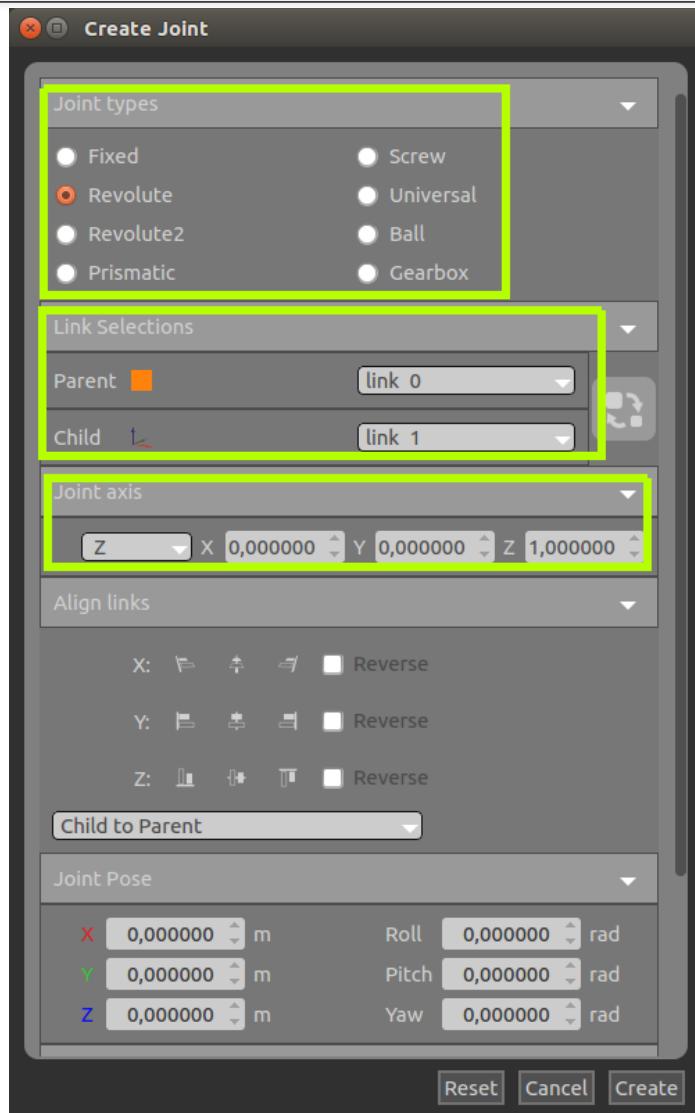


Figure 109 Bağlantı Oluşturma

“Joints” ayarlarını doğru bir şekilde yaptığımızda simülasyonda oluşan görüntü aşağıdaki şekildeki gibi olacaktır.

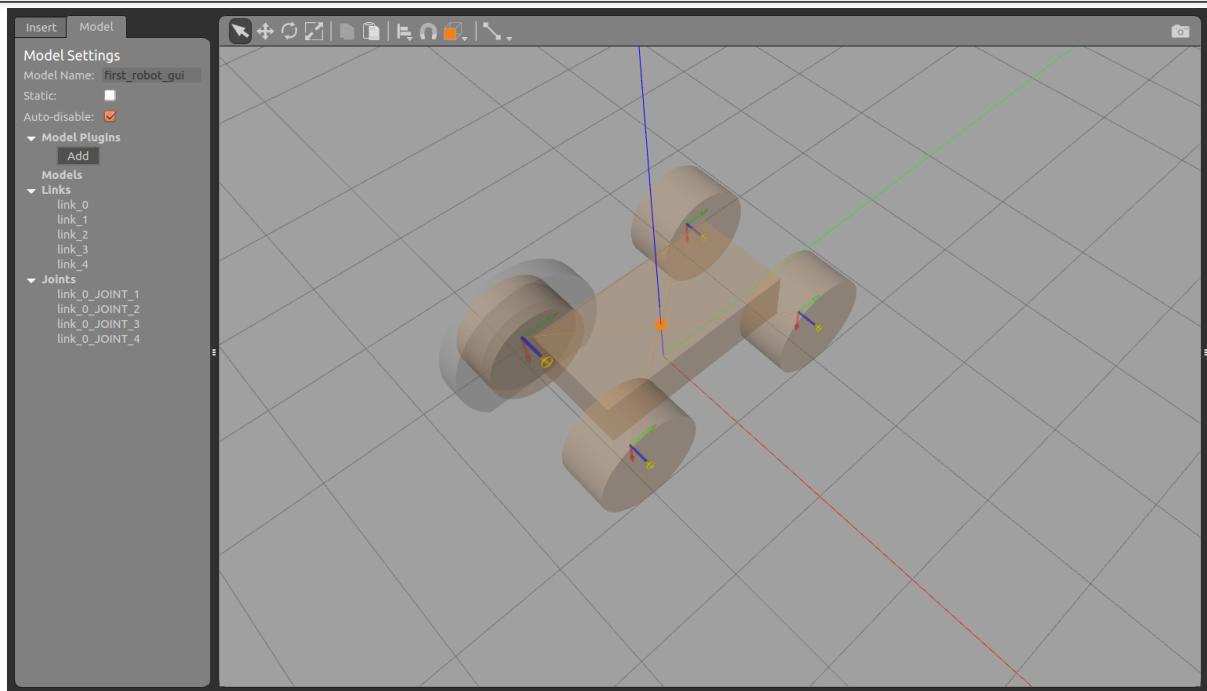


Figure 110 Oluşacak Model

Modeli doğru bir şekilde oluşturabildiysek, modelimizi kaydedebiliriz. Kayıt işlemi modelimiz için bir dizin, SDF dosyası ve konfigürasyon dosyası oluşturacaktır. Kayıt yapabilmek için “File” -> “Save As” seçeneklerini seçelim. Kayıt seçeneklerini;

Model Name: first_robot_gui

Location: .gazebo/models/first_robot_gui

şeklinde ayarlayıp kaydedelim. Daha sonra “Model Editor” arayüzünden, “File” -> “Exit Model Editor” seçeneklerini kullanarak çıkışım. Oluşturduğumuz robot, simülasyon ortamına eklenmiş olarak bulunacaktır.

4.3.6. Mesh Giydirmeye

Modeller oluşturulurken <visual> etiketinde bir model dosyası tanımlanarak model üzerine kalıp giydirilebilir.

<visual> etiketinde tanımlama yapıldığı takdirde, bu durumdan modelin sadece görsel özelliği etkilenmektedir.

Bunun için aşağıdaki kısımda görülen “Chassis” linkine ait “Visuals” etiketini;

```
<!--visual>
    <origin xyz="0.0 0.0 0.1" rpy="0.0 0.0 0.0" />
    <geometry>
        <box size="0.40 0.20 0.10"/>
    </geometry>
</visual-->
```

“comment” olacak şekilde değiştirerek, altına aşağıdaki parçayı ekleyelim;

```
<visual>
    <origin xyz="0.0 0.0 0.2" rpy="0.0 0.0 0.0"
    />
    <geometry>
        <mesh
            filename="model://pioneer2dx/meshes/chassis.dae"
            scale="1.1 0.75 1.5"/>
    </geometry>
</visual>
```

Güncellediğimiz first_robot.urdf.xacro modelini;

```
cd
cd ~/catkin_ws
roslaunch myrobot first_robot.launch
```

komutlarıyla Gazebo üzerinde görebiliriz.

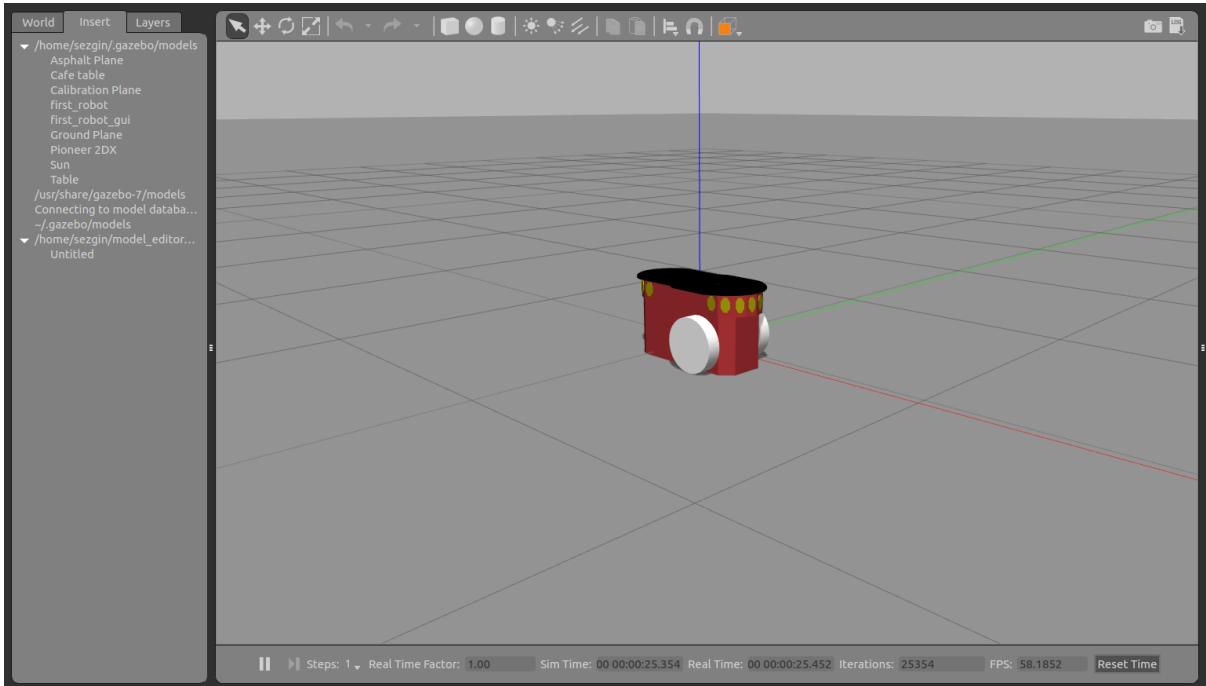


Figure 111 Giydirilmiş Mesh

4.3.7. Sensör Ekleme

Modeller oluşturulurken, model dosyasında `<link>`, `<joint>` ve gerekiğinde `.urdf` dosyaları için `<gazebo reference>` tanımlamaları yaparak modele sensör eklenebilmektedir.

Oluşturduğumuz `first_robot.urdf.xacro` modeline lazer sensörü eklemek için;

```
gedit
~/catkin_ws/src/myrobot/urdf/first_robot.urdf.xacro
```

komutlarıyla modelimizi güncelleerek sensör ekleyelim.

Bunun için dosyanın sonundaki “`</robot>`” etiketinden önce aşağıdaki parçayı ekleyelim;

```

<link name="laser">
    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <box size="0.1 0.1 0.1"/>
        </geometry>
    </collision>

    <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh
filename="model://hokuyo/meshes/hokuyo.dae"/>
        </geometry>
    </visual>

    <inertial>
        <mass value="1e-5" />
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <inertia ixx="1e-6" ixy="0" ixz="0"
iyx="1e-6" iyz="0" izz="1e-6" />
    </inertial>
</link>

<joint name="hokuyo_joint" type="fixed">
    <axis xyz="0 0 0" />
    <origin xyz="0.2 0 0.3" rpy="0 0 0"/>
    <parent link="chassis"/>
    <child link="laser"/>
</joint>

```

```

<gazebo reference="laser">
    <sensor type="gpu_ray"
name="head_hokuyo_sensor">
        <pose>0 0 0 0 0 0</pose>
        <visualize>true</visualize>
        <update_rate>40</update_rate>
        <ray>
            <scan>
                <horizontal>
                    <samples>720</samples>
                    <resolution>1</resolution>
                    <min_angle>-
3.1416</min_angle>

                    <max_angle>3.1416</max_angle>
                </horizontal>
            </scan>
            <range>
                <min>0.5</min>
                <max>5.0</max>
                <resolution>0.1</resolution>
            </range>
            <noise>
                <type>gaussian</type>
                <mean>0.0</mean>
                <stddev>0.01</stddev>
            </noise>
        </ray>

        <plugin
name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_gpu_laser.so">

```

```
<robotNamespace>first_robot</robotNamespace>
    <topicName>sensor/Laser</topicName>
    <frameName>laser</frameName>
    </plugin>
</sensor>
</gazebo>
```

Güncellediğimiz first_robot.urdf.xacro modelini;

```
cd
cd ~/catkin_ws
roslaunch myrobot first_robot.launch
```

komutlarıyla Gazebo üzerinde görebiliriz.

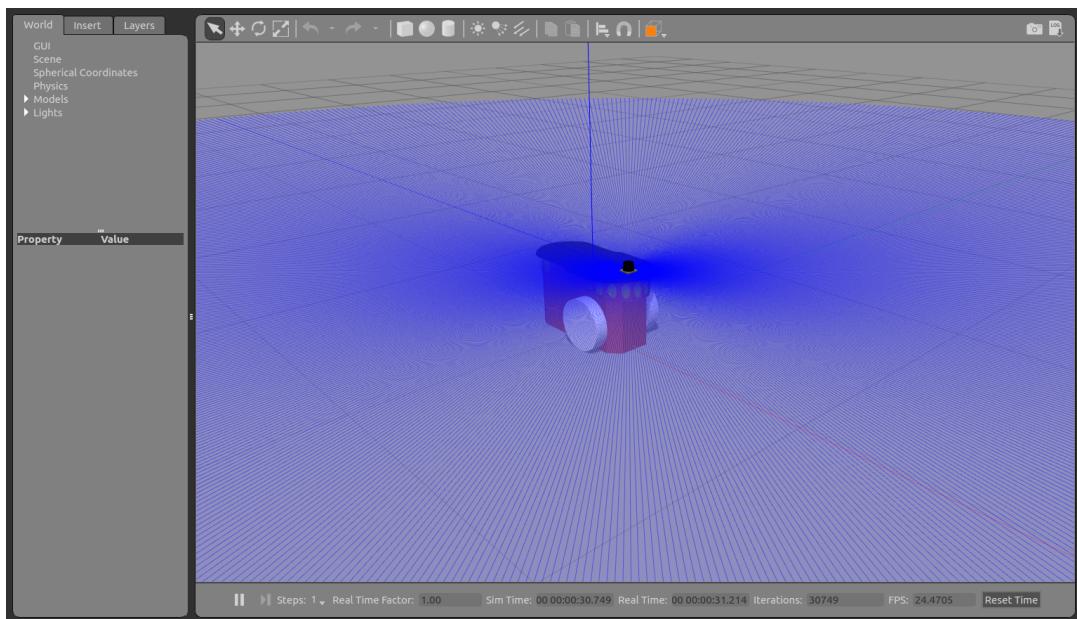


Figure 112 Algılayıcı Gösterimi

4.4. ROS ile Kontrol

Oluşturduğumuz modeli simülasyon ortamında ROS ile kontrol edebilmek için öncelikle ilgili paketlerin kurulu olması gerekmektedir. Örneğin ROS Kinetic için;

```
sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control
```

komutuyla ilgili gazebo_ros_pkgs indirilebilir. Ardından bu çalışmanın başında yapıldığı gibi bir catkin_workspace ortamı hazırlanmalıdır.

ROS ve Gazebo ortamı uygun bir şekilde oluşturulup yapılandırıldı ise

```
rosrun gazebo_ros gazebo
```

komutu başarılı bir şekilde Gazebo çalıştıracaktır.

Halihazırda çalışmanın önceki kısımlarında gösterildiği gibi

```
roslaunch myrobot first_robot.launch
```

komutuyla oluşturduğumuz launch dosyasından robotumuzu Gazebo ortamında ortaya çıkarmamız da ROS ile kontrol için yapılması gerekenler arasında yer almaktadır.

5. Movelt

5.1. Movelt nedir?

Movelt, robot manipülasyonu için yaygın olarak kullanılmakta;

- gelişmiş uygulamalar geliştirmek,
- yeni tasarımları değerlendirmek,
- entegre ürünler oluşturmak

İçin kullanımı kolay bir robotik platform sağlamaktadır.

5.2. Movelt İçeriği

Movelt bir platform olarak içerdiği imkanlar sayesinde;

- hareket planlama
- robot manipülasyonu
- ters kinematik çözümlemeleri
- robot kontrolü
- 3-B algılama
- çarışma kontrolü

uygulamaları için kullanılabilmektedir.

“Rviz Motion Planning Plugin” adlı eklentiyle birlikte engellerin bulunduğu ortamlarda çeşitli planlama algoritmalarının denenmesi mümkün olmaktadır.

“OMPL”, “CHOMP” ve “STOMP” gibi çeşitli planlama kütüphaneleri ile birlikte literatürde yer alan güncel planlama algoritmalarını kullanma imkanı sağlanmaktadır.

“MoveIt Setup Assistant” adlı konfigürasyon sihirbazı sayesinde ise herhangi bir robottu adım adım yapılandırmak veya önceden yapılandırılmış popüler yapıları kullanmak mümkün olmaktadır.

Gazebo, ROS Control ve MoveIt birbirine entegre edilerek ise güçlü bir robot geliştirme platformu elde edilebilmektedir.

5.2.1. MoveIt Kurulumu

ROS başarılı bir şekilde kurulduysa ve MoveIt henüz kurulu değil ise kurulum Kinetic için pre-built binaries kullanılarak aşağıdaki komutla gerçekleştirilebilir;

```
sudo apt install ros-kinetic-moveit
```

Bu aşamadan sonra ise catkin_workspace oluşturulabilir. Bu çalışmanın önceki aşamalarında oluşturulan workspace, bu çalışma için de kullanılabilir. Mevcut çalışma ortamını ayarlamak için aşağıdaki komutları sırasıyla çalıştıralım;

```
cd ~/catkin_ws/src/  
mkdir myrobot_moveit_config  
cd myrobot_moveit_config/
```

5.2.2. MoveIt Setup Assistant

“MoveIt Setup Assistant”, MoveIt ile kullanılacak herhangi bir robottu yapılandırmak için kullanılan bir kullanıcı arayüzüdür. Bu arayüzün kullanımı sonucunda, Semantik Robot Açıklama Formatı (SRDF) dosyası ve MoveIt pipeline içerisinde kullanılmak üzere diğer gerekli konfigürasyon dosyaları oluşturulmaktadır.

Bir robota ait konfigürasyon paketini oluşturabilmek için öncelikle “MoveIt Setup Assistant”ı başlatalım;

```
roslaunch moveit_setup_assistant  
setup_assistant.launch
```

ve ardından “Create New MoveIt Configuration Package” butonuna tıklayalım.

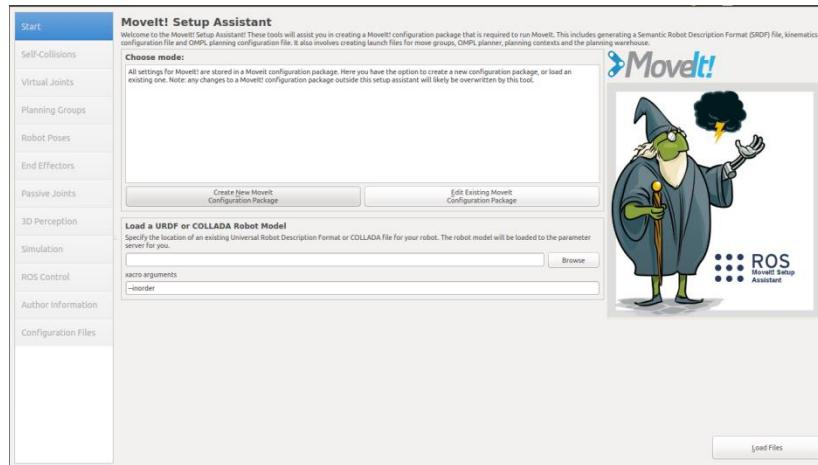


Figure 113 MoveIt Arayüzü

Açılan sayfada robotumuza ait .urdf veya .urdf.xacro dosyasını seçip, “Load Files” butonuna tıklayalım. Bu işlemlerden sonra robotumuz arayüzün sağ tarafında gözükecektir. Sol tarafta ise gerçekleştireceğimiz konfigürasyon için takip edeceğimiz aşamalar yer alacaktır.

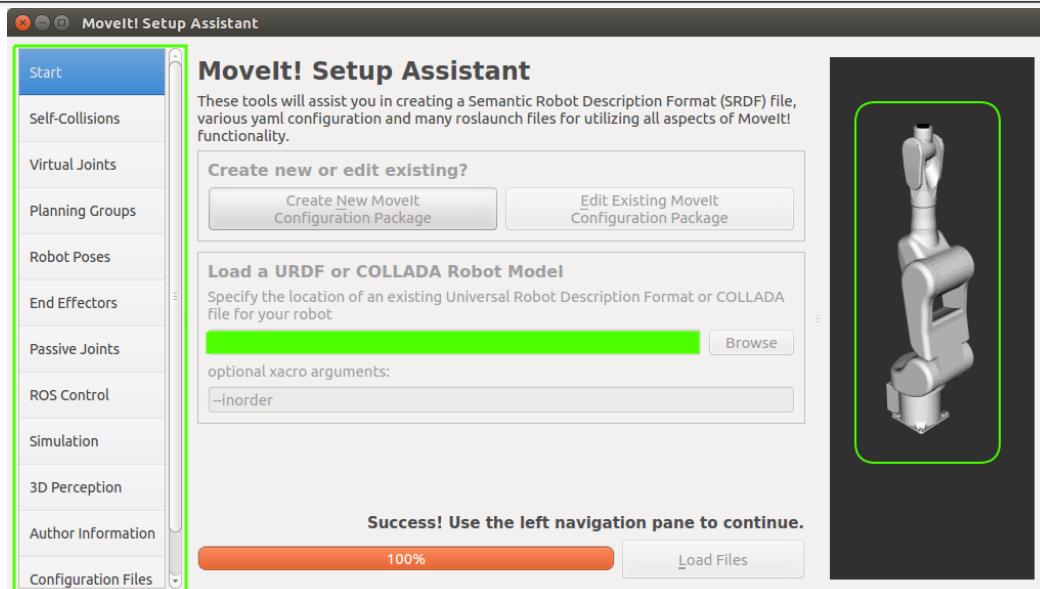


Figure 114 MoveIt Kurulum Asistanı

Konfigürasyon için ilk olarak sol taraftan “Self-Collisions” seçeneğine tıklayalım. Açılan sayfada, kendi kendine çarışma kontrolü optimize edilecektir. Bu işlem, hareket planlama süresini azaltmayı amaçlamaktadır. Gerçekleştirmek için “Sampling Density” istenildiği şekilde ayarlanıp, “Generate Collision Checking” seçeneğine tıklayalım. “Sampling Density”, kendi kendine çarışma kontrolü için kaç rastgele robot konumunun belirleneceğini göstermektedir. Bu değer yüksek tutulursa bu aşamada gerçekleştirilecek işlem süresi uzayabilir, ancak elde edilen sonuçlar daha hassas olabilir.

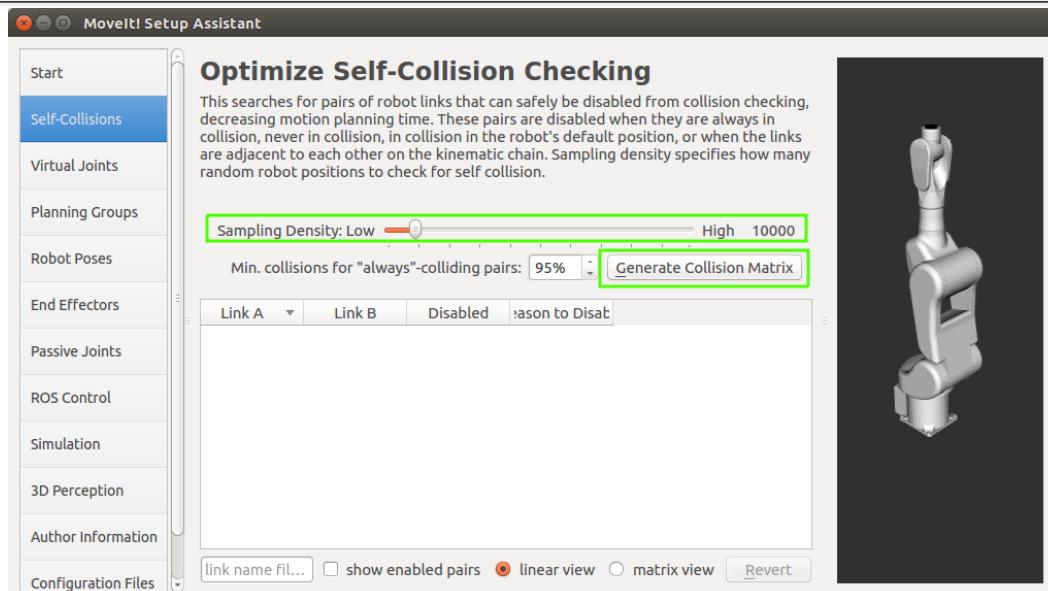


Figure 115 Self-collision Ayarlaması

Çarpışma matrisinin üretilmesinden sonra, aşağıdaki gibi asla çarpışma halinde olmayan robot link çiftleri veya bitişik robot link çiftleri belirlenecektir.

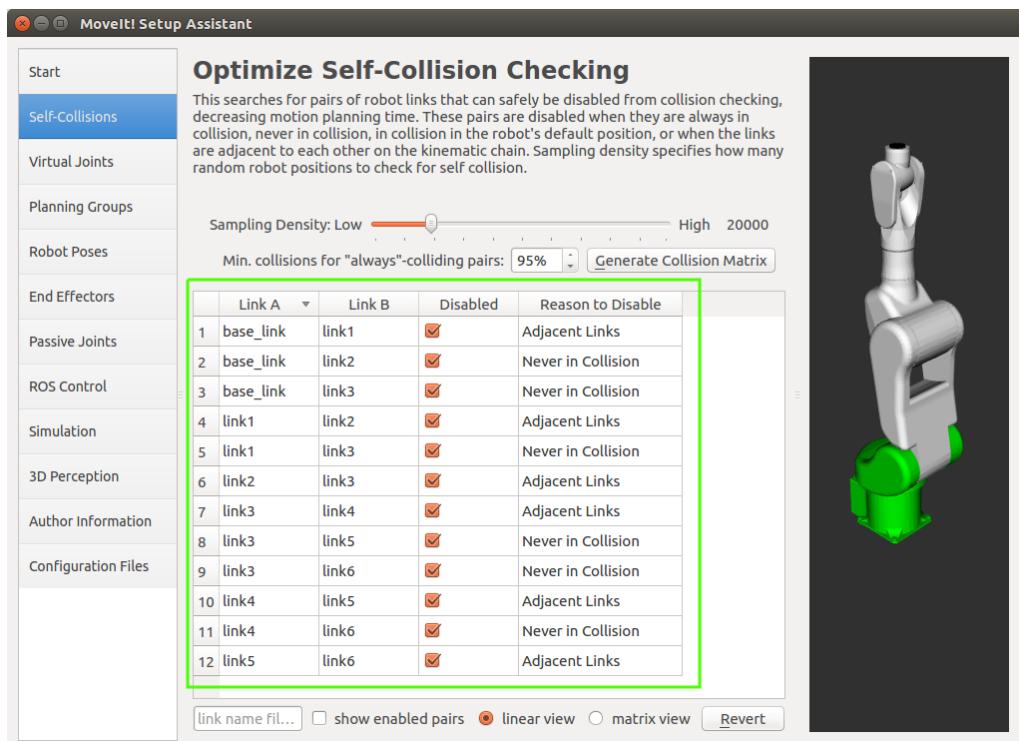


Figure 116 Robot Link Çiftleri

Sonraki aşamada “Virtual Joints” seçeneğine tıklayarak robottu dünyaya bağlamak için sanal bir eklem tanımlayabiliriz. Bunun için “Virtual Joints” seçeneğine tıkladıktan sonra, “Add Virtual Joint” seçeneğine tıklayalım ve aşağıdaki gibi sanal bir eklem oluşturalım.

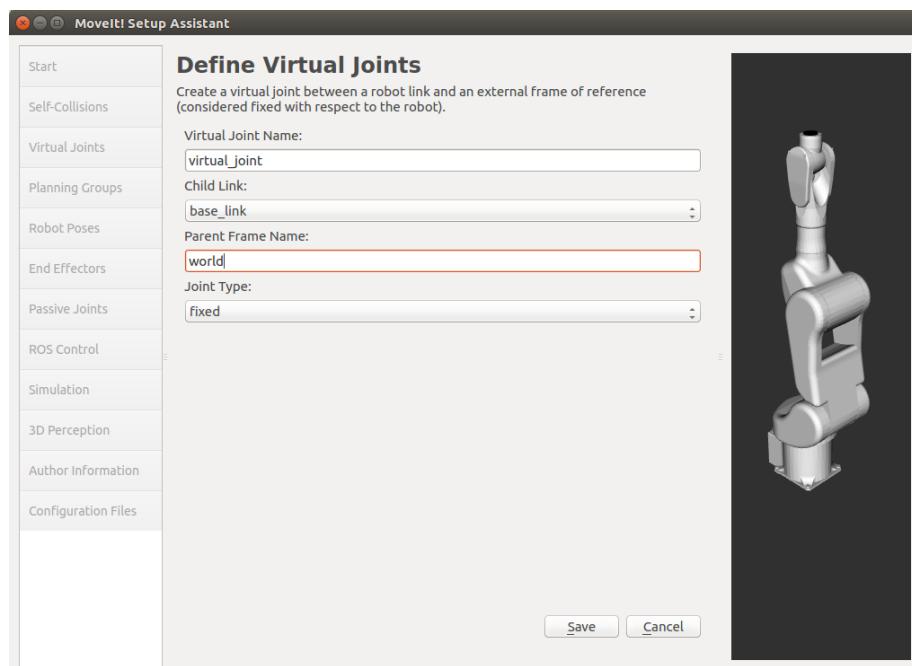


Figure 117 Sanal Eklem Oluşturma

Bunun için aşağıdaki seçenekleri;

Virtual Joint Name: virtual_joint

Child Link: base_link

Parent Frame Name: world

Joint Type: Fixed

şeklinde belirleyebiliriz.

Bundan sonraki aşamada “Planning Group” belirlenebilir. “Planning Group”, bir robot kolunun ne olduğunu veya robotun üç noktasının tanımlanması gibi robotun farklı bölgelerini anlamsal olarak tanımlamak için kullanılmaktadır. Bunun için

“Planning Group” seçeneğine tıkladıktan sonra, “Add Group” seçeneğine tıklayalım ve aşağıdaki gibi bir planlama grubu oluşturalım.

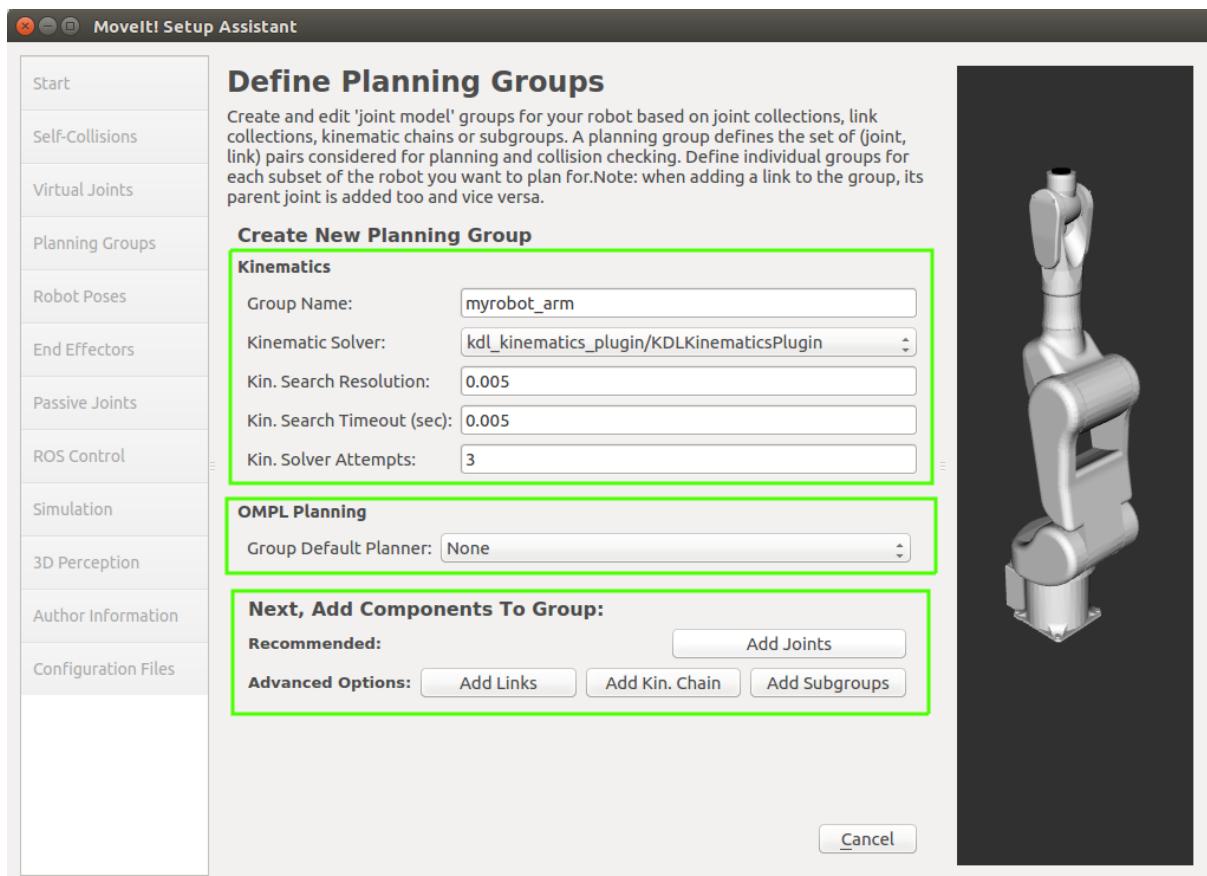


Figure 118 Planlama grubu oluşturulması

“Kinematics” kısmını aşağıdaki gibi;

Group Name: myrobot_arm

Kinematic Solver: kdl_kinematics_plugin/KDLKinematicsPlugin

Kin.Search Resolution: 0.005

Kin.Search Timeout (sec): 0.005

Kin. Solver Attempts: 3

tanımlayabiliriz. Grup adı, planlamanın gerçekleştirileceği grubu tanımlamak için kullanılmaktadır. KDL kinematik eklentisi, Orocó KDL paketi tarafından sağlanan

sayısal ters kinematik çözümünü kullanmaktadır. Bu eklenti, MoveIt tarafından sağlanan varsayılan kinematik eklentisidir. Bunun dışında güçlü bir özel ters kinematik çözümü istenirse, kullanıcı tarafından özel olarak IKFast MoveIt eklentisi oluşturulabilir.

“OMPL Planning” kısmında varsayılan planlayıcı için istenilen hareket planlama algoritması seçilebileceği gibi seçilmeden de bırakılabilir.

“Next, Add Components to Group” kısmında ise önce “Add Joints” seçeneğine tıklayarak eklemleri aşağıdaki gibi ekleyebilir ve “Save” seçeneğine tıklayabiliriz.

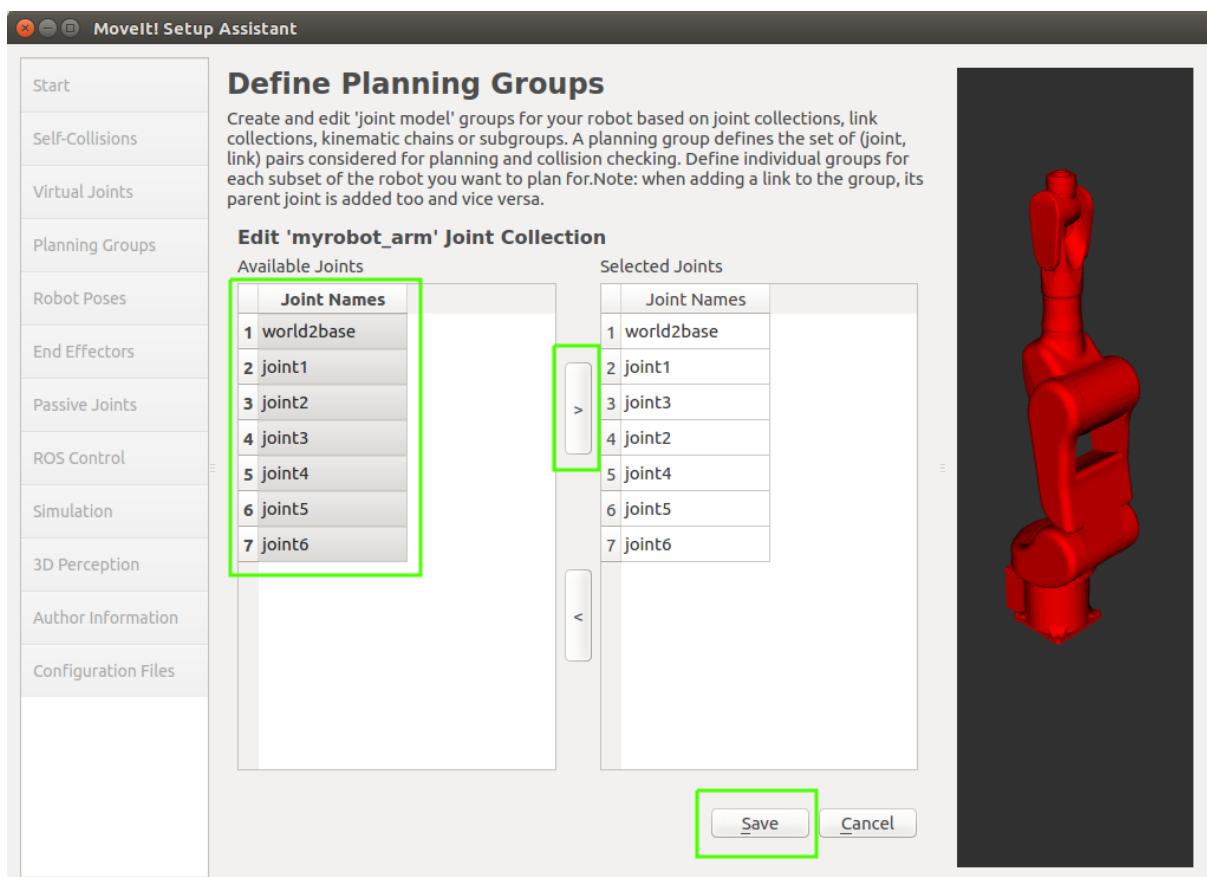


Figure 119 Eklemlerin Eklenmesi ve Kaydedilmesi

Eklemlerin seçilmesinden sonra da gelen ekranda aşağıdaki gibi önce “Link” seçeneğini, daha sonra ise “Chain” seçeneğini ayarlayabiliriz. Bunun için ilgili seçeneği seçip “Edit Selected” butonuna tıklayabiliriz.

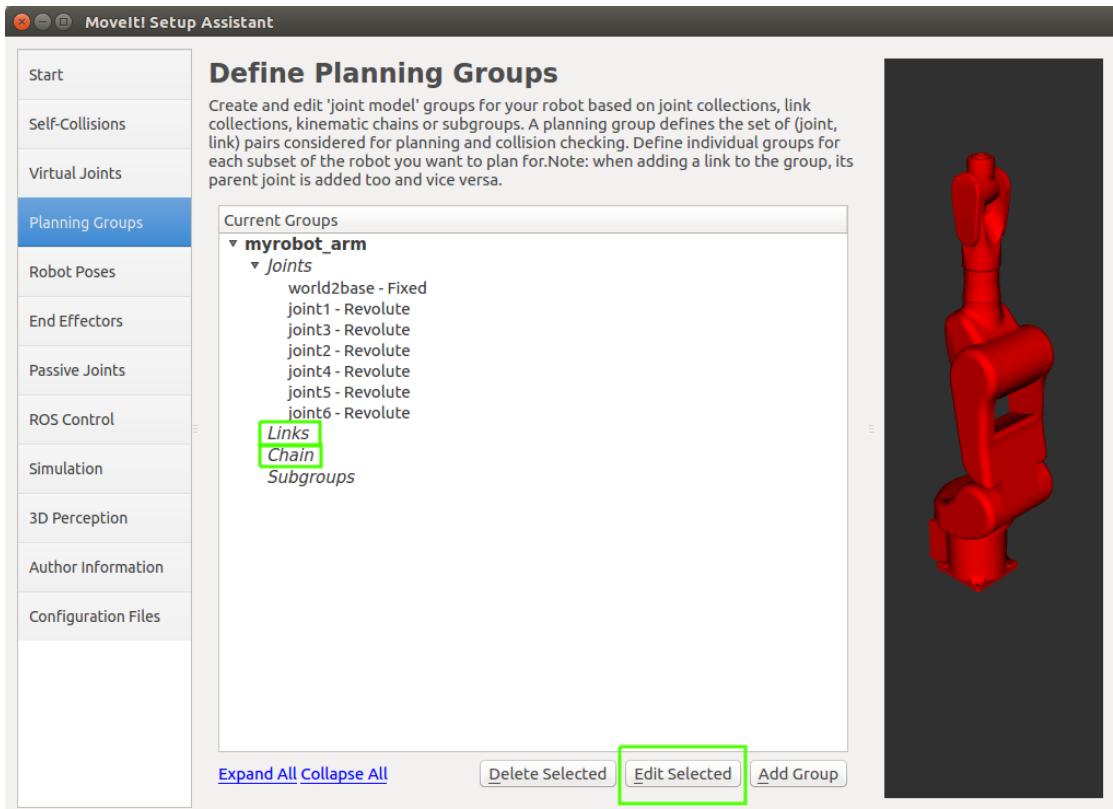


Figure 120 Seçillenlerin Düzenlenmesi

“Link” seçeneği için aşağıdaki gibi robot uzuvlarını seçerken;

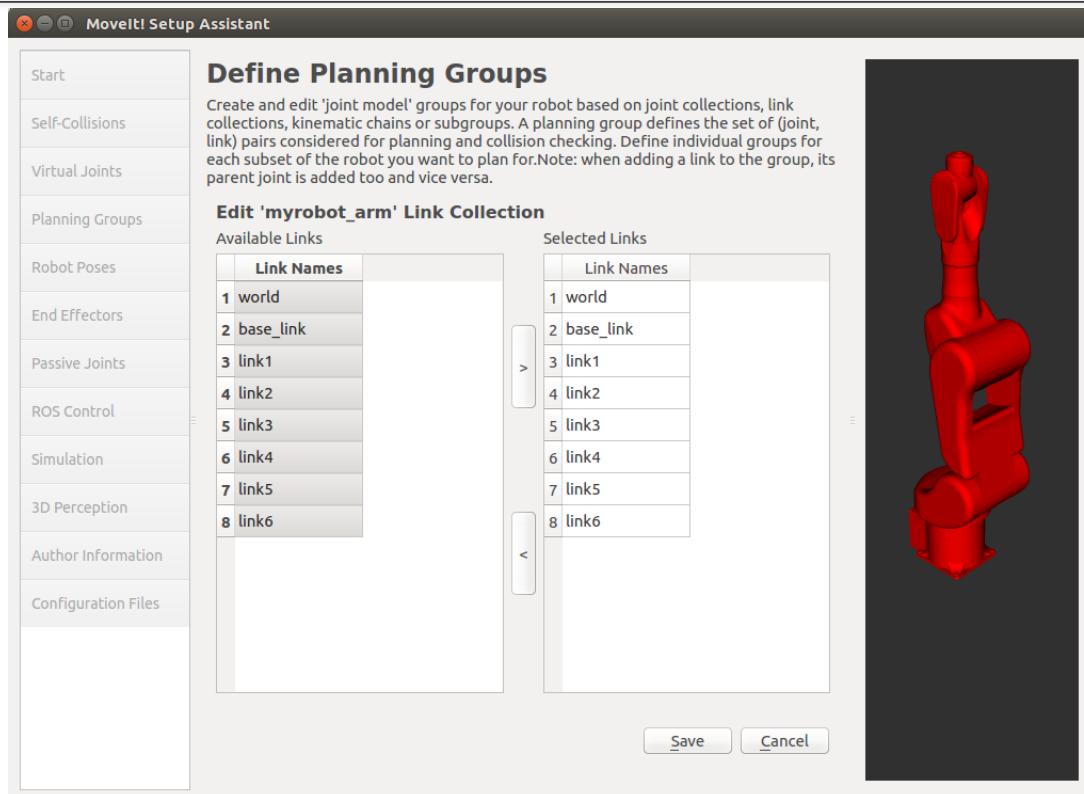


Figure 121 Robot Uzuv seçimi

“Chain” seçeneğinde ise aşağıdaki gibi robota ait kinematik zincirinde başlangıç uzvunu ve uç uzvu tanımlayabiliriz;

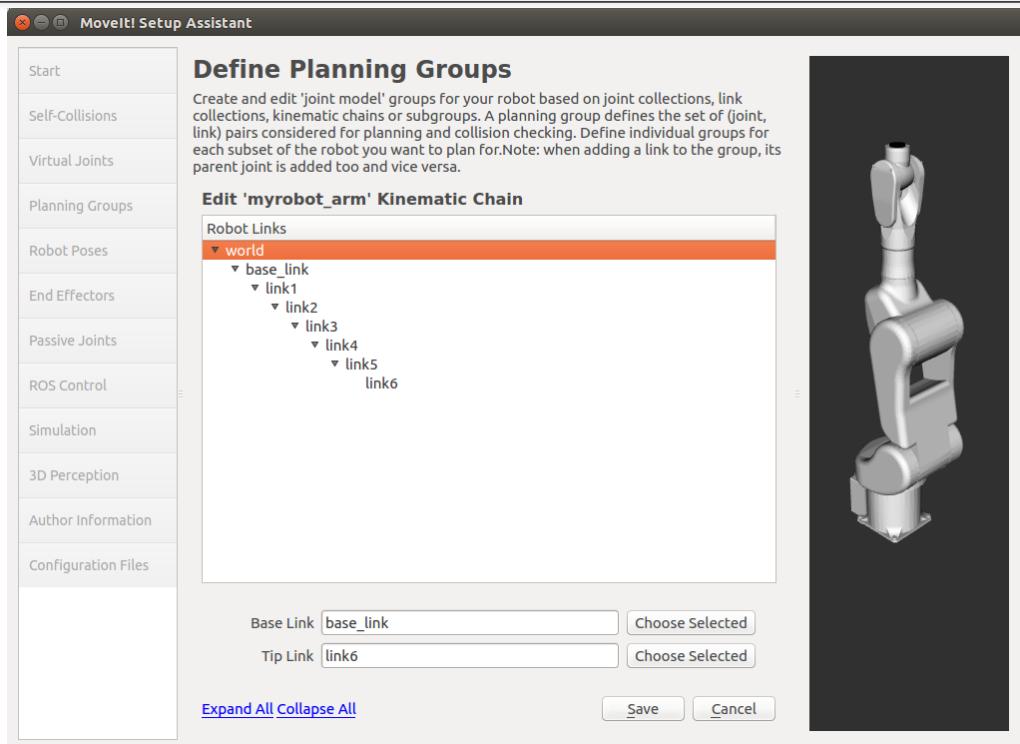


Figure 122 Uç Uzuv Tanımlanması

Nihayetinde planlama grubu için aşağıdaki gibi bir konfigürasyon elde ederiz.

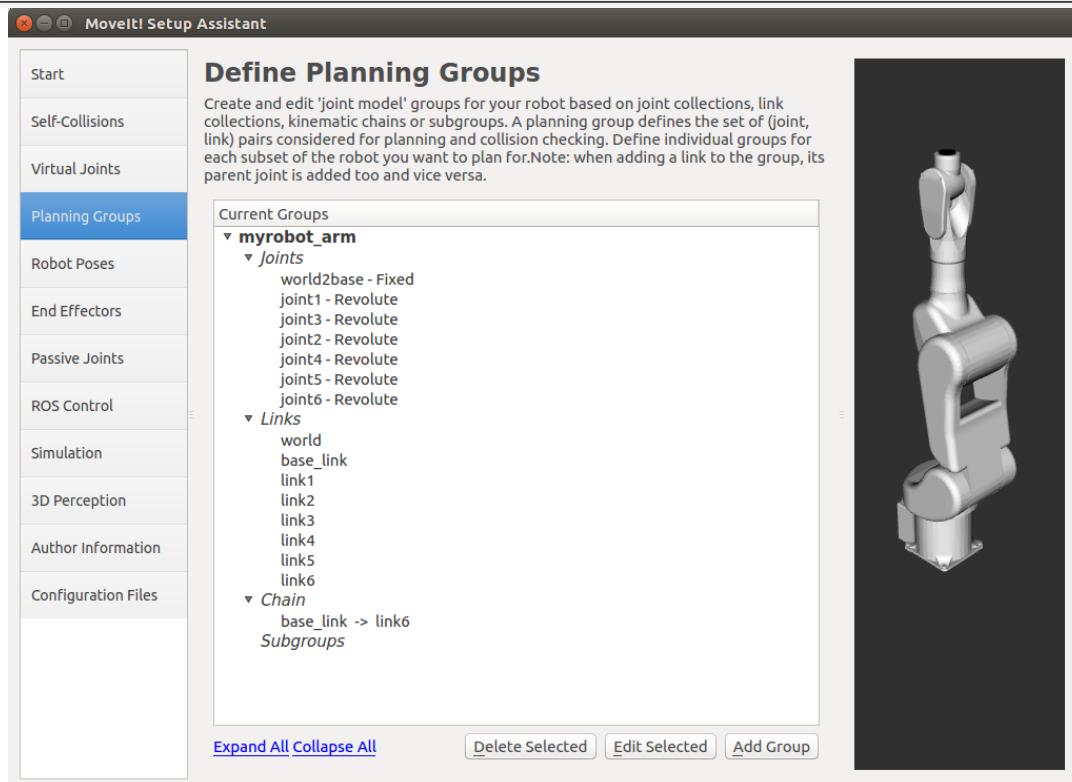


Figure 123 Oluşan Konfigürasyon

Aynı şekilde robot ucunda “tool” olarak tanımlayabileceğimiz bir grup daha tanımlayabiliriz ve aşağıdaki gibi “tool” adında bir grup daha oluşturabiliriz.

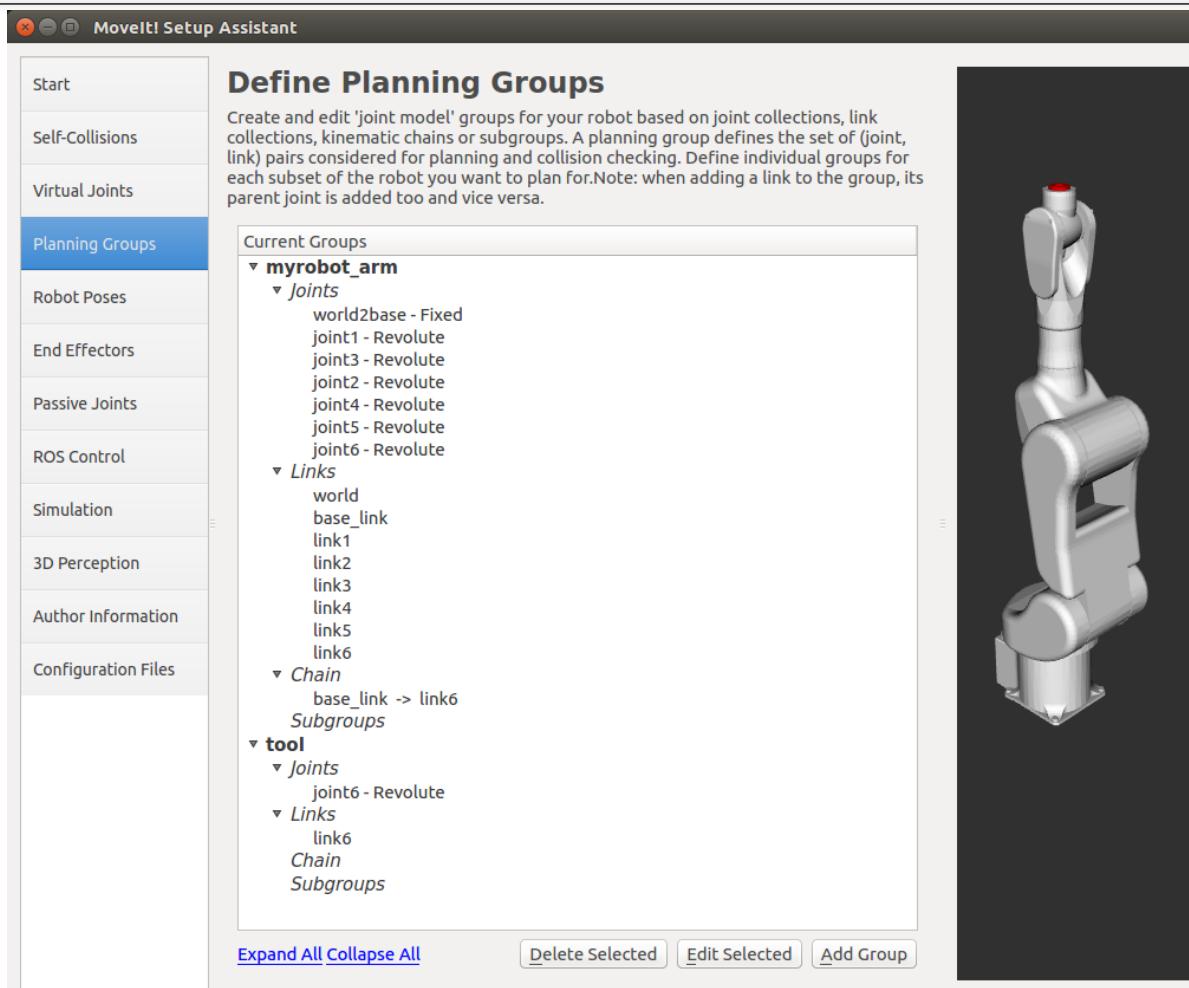


Figure 124 Tool Oluşturulması

Bunun ardından, “Robot Poses” kısmından yapılandırmaya belirli sabit robot duruşlarını eklemek de mümkündür. Örneğin robot için “home” olarak ifade edebileceğimiz bir ilk duruş tanımlayabiliriz. Bunun için “Add Pose” seçeneğini tıklayarak aşağıdaki gibi;

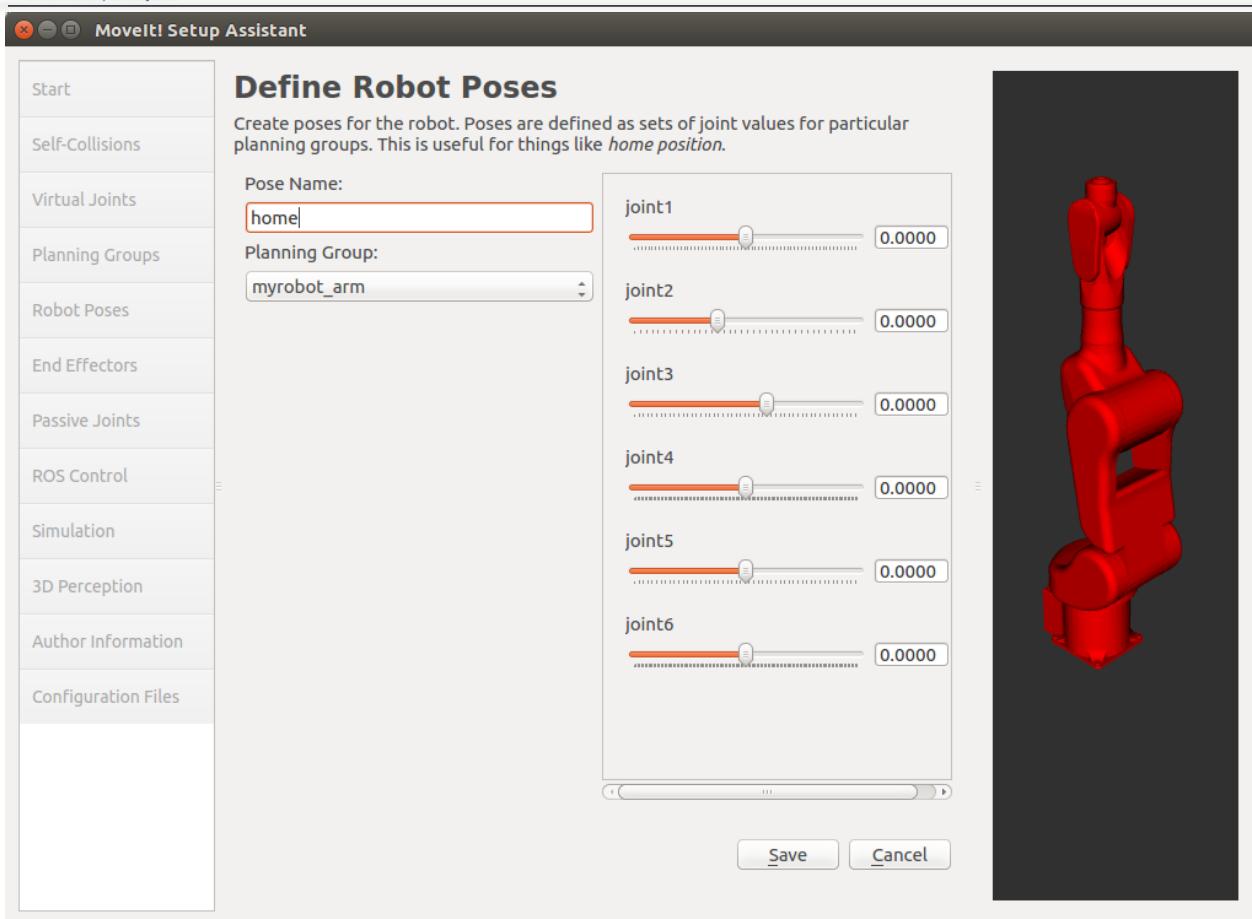


Figure 125 Robot Pozisyonu tanımlanması

belirleyebileceğimiz bir duruşu, ilgili “Pose Name” kısmını “home” olarak doldurup “Save” butonuna tıklayarak bir duruş tanımlayabiliriz.

Bir sonraki aşamada ise “End Effectors” kısmından iki önceki aşamada “tool” olarak belirlediğimiz grubu özel bir uç noktası grubu olarak belirleyebilmekteyiz. Bunun için “Add End Effector” seçeneğine tıklayarak;

End Effector Name: ee

End Effector Group: tool

Parent Link: link6

şeklinde uç noktasını tanımlayabiliriz.

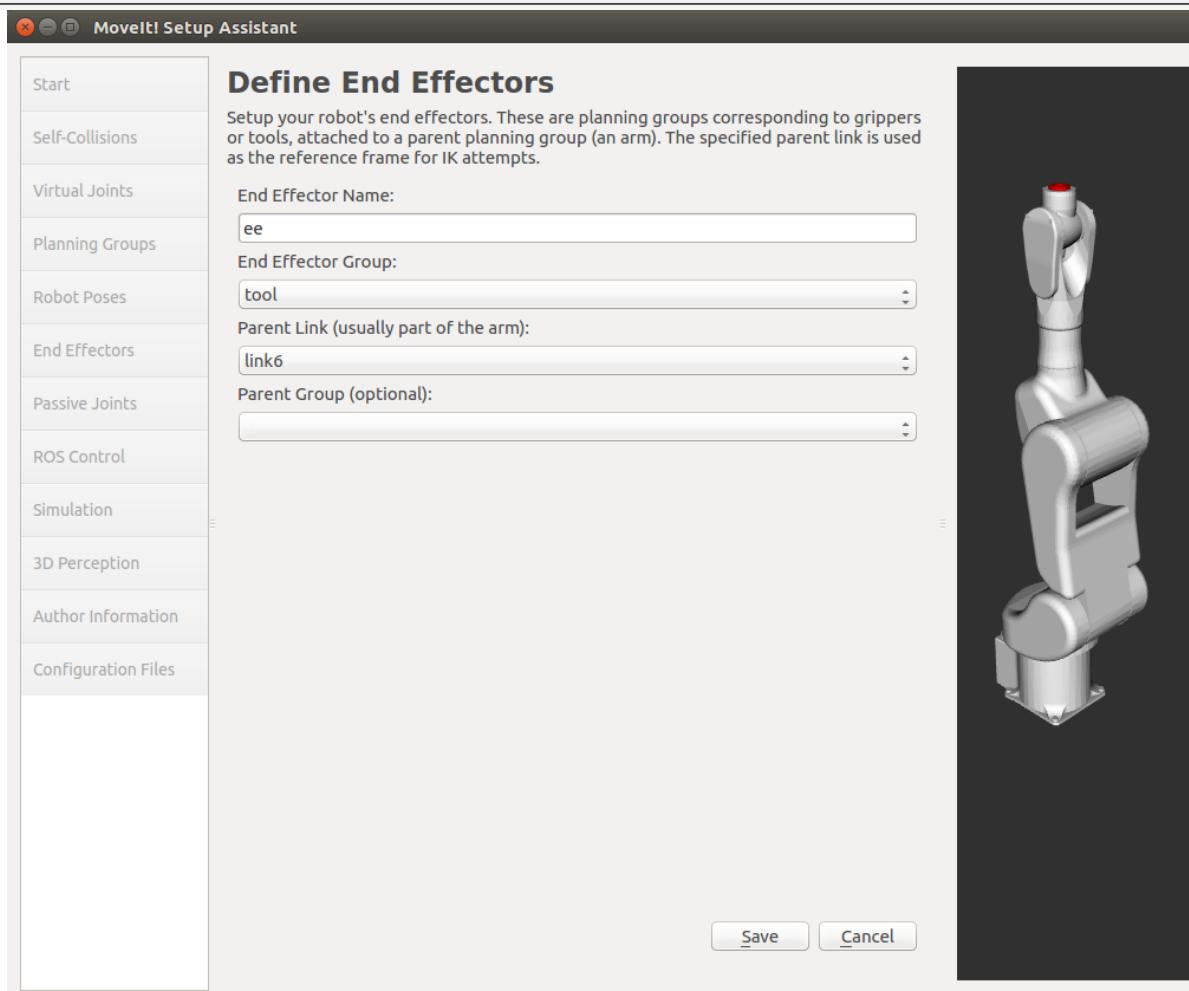


Figure 126 Uç nokta Tanımlanması

“Passive Joints” kısmı ise bir robotta bulunabilecek pasif eklemleri belirlemek için kullanılmaktadır. Bu sayede pasif eklemler direkt olarak kontrol edilemeyecekleri için (kinematik olarak), bu eklemler için planlama yapılmasına gerek kalmayacaktır. Bu çalışmada kullanılan robot modelinde böyle bir eklem bulunmadığı için herhangi bir işlem yapılmasına gerek yoktur.

“ROS Control” kısmında ise robotun eklemlerini harekete geçirmek üzere simüle edilmiş kontrolörler otomatik olarak oluşturulabilir. Bu kısımda “Auto Add FollowJointsTrajectory Controllers For Each Planning Group” seçeneğine tıklayarak bu işlem aşağıdaki gibi gerçekleştirilebilir. Kontrolör türü olarak başka bir tür seçilmek istendiğinde, ilgili kontrolöre tıklanıp, “Edit Selected” seçeneğine tıklanarak tür değiştirilebilir.

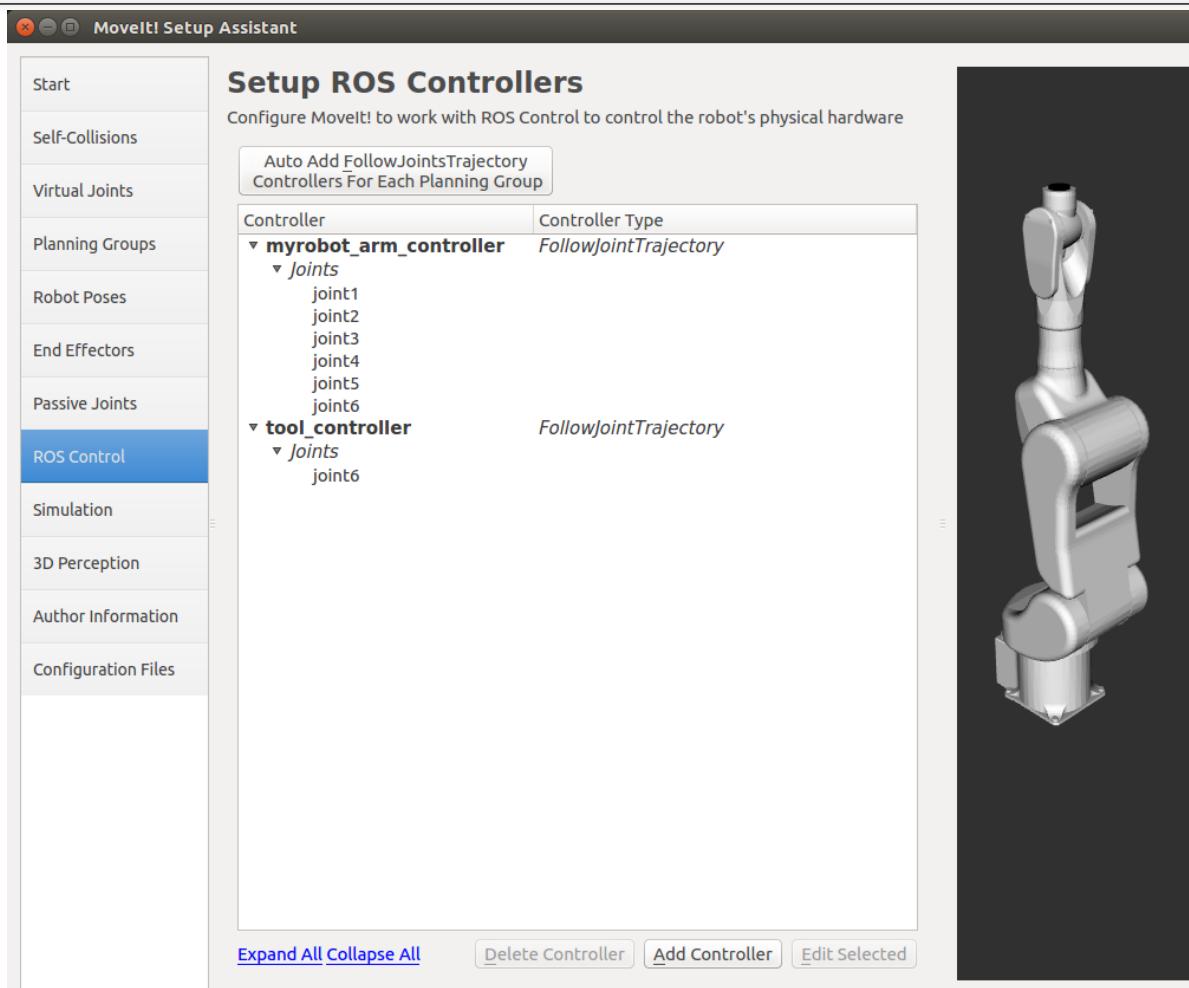


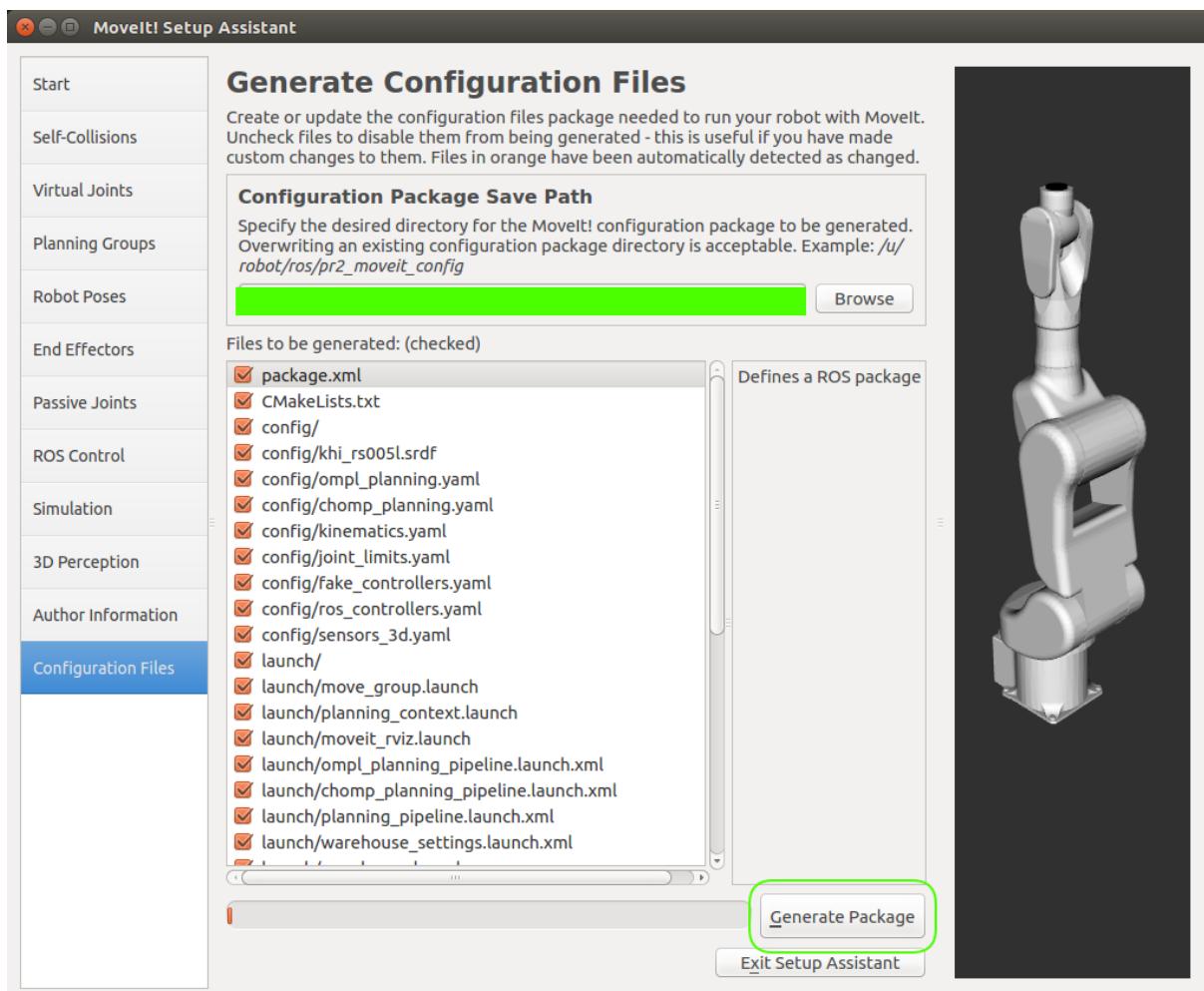
Figure 127 ROS ile kontrol

“Simulation” kısmında ise ROS Control ve MoveIt’ın Gazebo ile uyumlu şekilde çalışabilmesi için URDF dosyasında yapılması gereken değişiklikler otomatik olarak üretilmektedir. Yapılması gereken değişiklikler yeşil renkte gösterilmektedir. İstenirse bu değişiklikler kopyalama yoluyla ilgili URDF dosyasına eklenebilmektedir.

“3D Perception” kısmında ise YAML yapılandırma dosyasının parametrelerini ayarlayarak 3B sensörleri yapılandırmak mümkündür. Bu çalışmada kullanılan robot modelinde böyle bir sensör bulunmadığı için herhangi bir işlem yapılmasına gerek yoktur.

“Author Information” kısmından ise paketi oluşturan kişiye ait bilgiler girilebilmektedir.

Son olarak “Configuration Files” kısmında ise konfigürasyonu tamamlayabilmekteyiz. Bunun için aşağıdaki gibi konfigürasyonun kaydedileceği yolu seçip, “Generate Package” seçeneğine tıklayabiliriz.



Bu aşamaların sonunda, MoveIt ile planlama için kullanılacak konfigürasyon dosyaları oluşturulmaktadır.

5.2.3. Movelt ile Planlama ve Rviz ile Gazebo Üzerinde Çalışma Gösterimi

Movelt paketi doğru bir şekilde oluşturulduğunda robottu hem Rviz üzerinde, hem de Gazebo üzerinde ortaya çıkararak;

- “Rviz Motion Planning Plugin” eklentisi sayesinde hareket planlama yapabilmekte
- Oluşan planı çalıştırarak robotun durumu Gazebo üzerinde gözlemlenebilmektedir.

Movelt paketi doğru bir şekilde oluşturulduğunda;

```
rosrun moveit_config demo_gazebo.launch
```

komutuyla robottu hem Rviz üzerinde, hem de Gazebo üzerinde ortaya çıkararak “Rviz Motion Planning Plugin” eklentisi sayesinde hareket planlama yapabilmekte ve oluşan planı çalıştırarak robotun durumunu Gazebo üzerinde gözlemeylebilmekteyiz (Bu işlemi gerçekleştirebilmek için ayrı launch dosyaları da tanımlayabiliriz).

Bu işlemlerin sonucunda Rviz üzerinde aşağıdaki gibi;

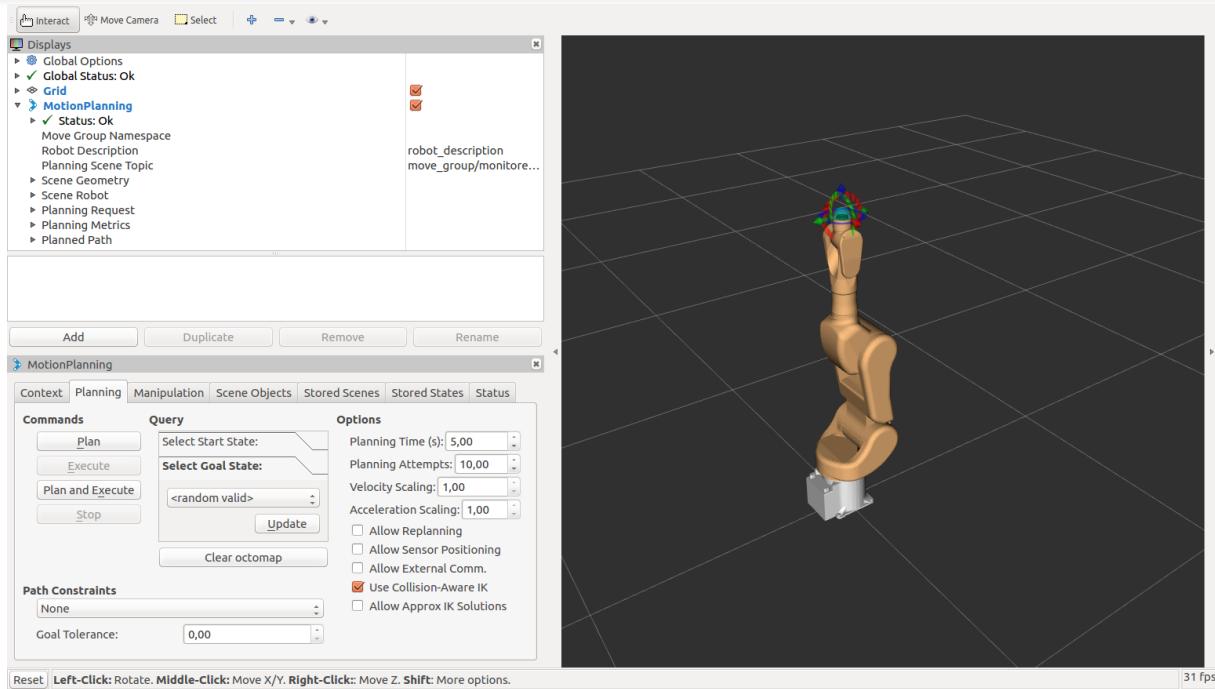


Figure 129 Rviz arayüzü

Gazebo üzerinde ise aşağıdaki gibi;

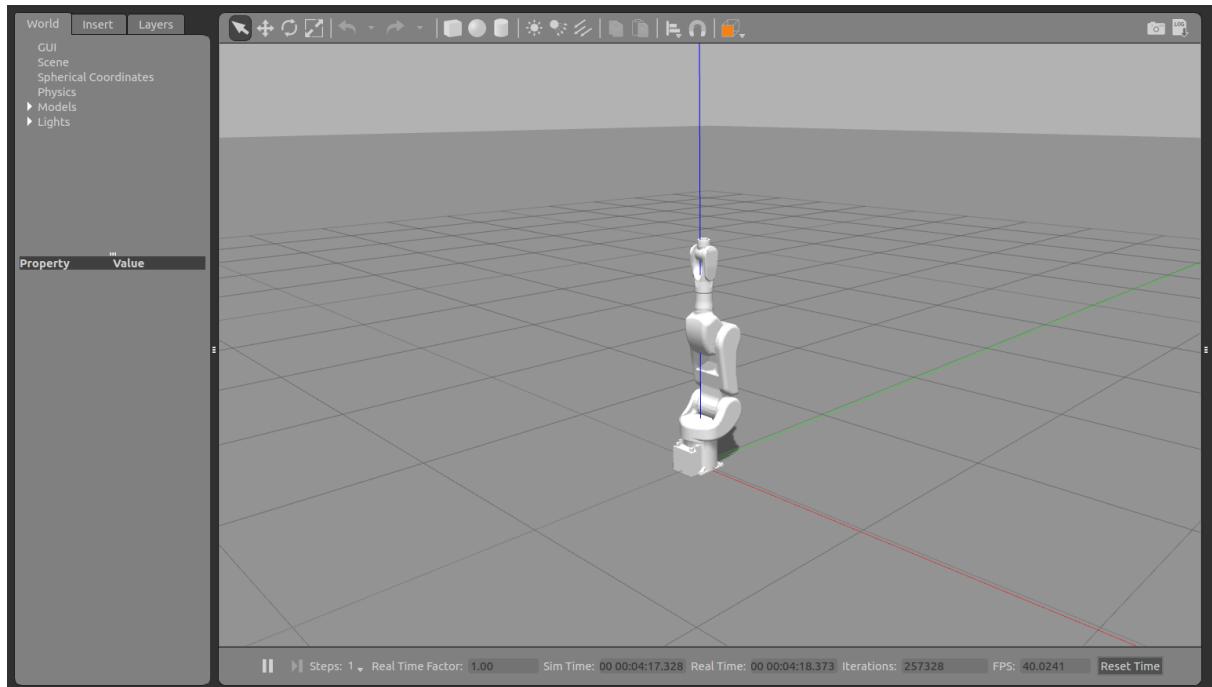


Figure 130 Gazebo Arayüzü

bir durum elde ederiz.

Örnek bir hareket planlama için öncelikle “Select Start State” seçeneğini “current” şeklinde seçerek “Update” butonuna tıklayabiliriz. Bu sayede planlama için başlangıç konumunu mevcut durusu olarak seçmiş oluruz.

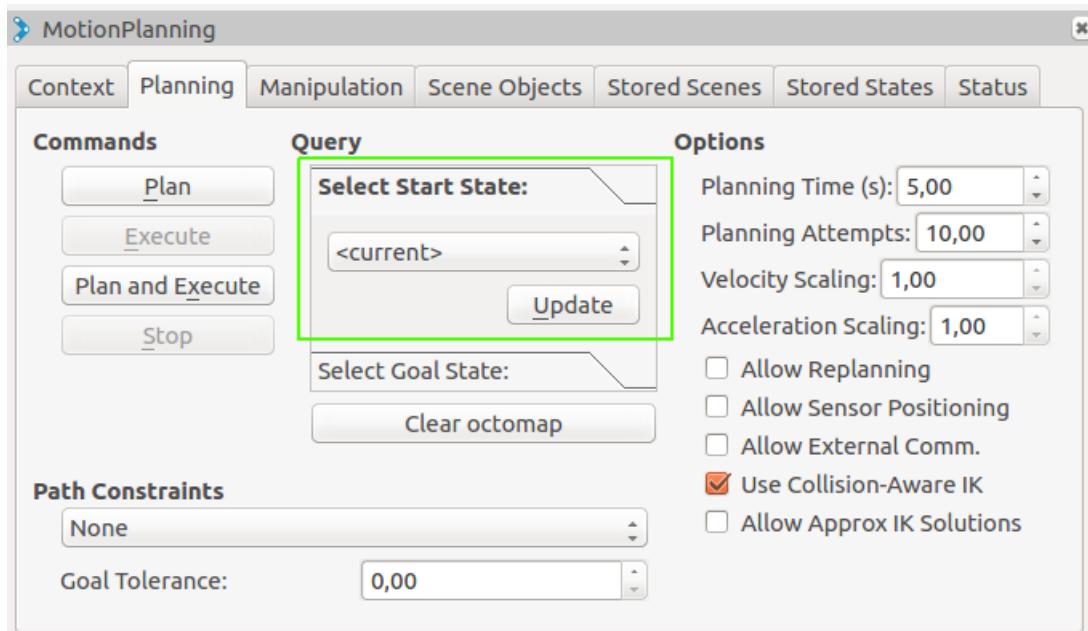


Figure 131 Başlangıç Konum Seçimi

Daha sonra ise “Select Goal State” seçeneğini “random valid” şeklinde seçerek “Update” butonuna tıklayabiliriz. Bu sayede ise planlama için hedef konumunu rastgele geçerli bir duruş olarak seçmiş oluruz.

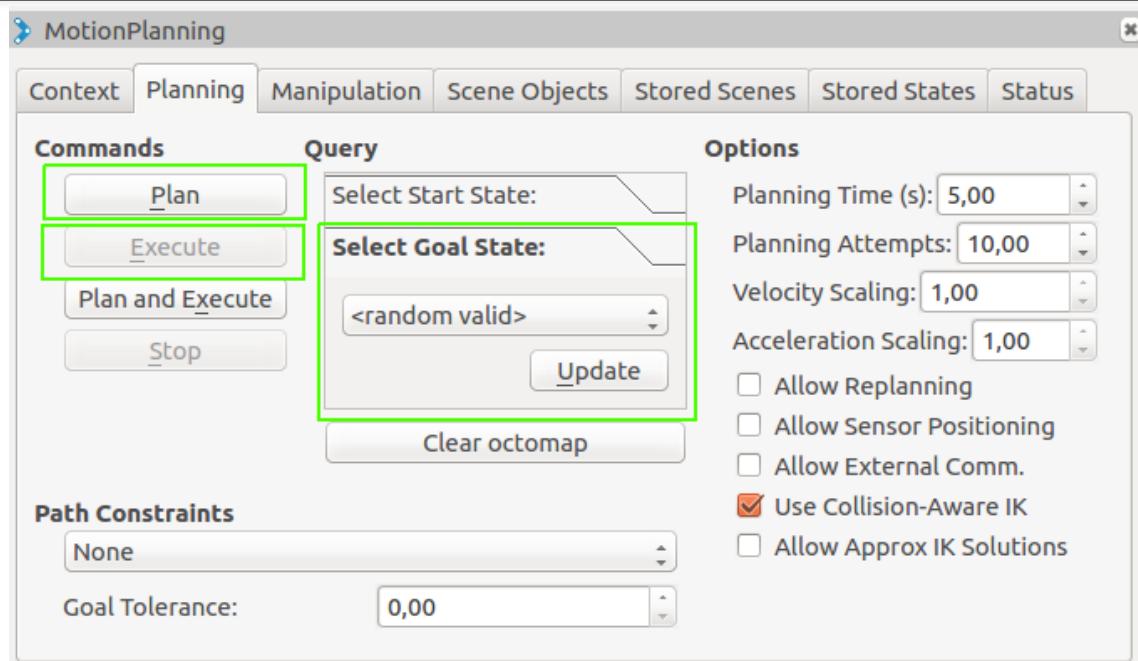


Figure 132 Rastgele Hedef Konum seçimi

Bu durumun ardından önce “Plan” seçeneğine tıklayarak planlamanın yapılmasını, daha sonra ise “Execute” seçeneğini seçerek planın uygulanması sağlanmaktadır.

“Plan” sonrasında Rviz üzerinde oluşan görüntü aşağıdaki gibi olurken;

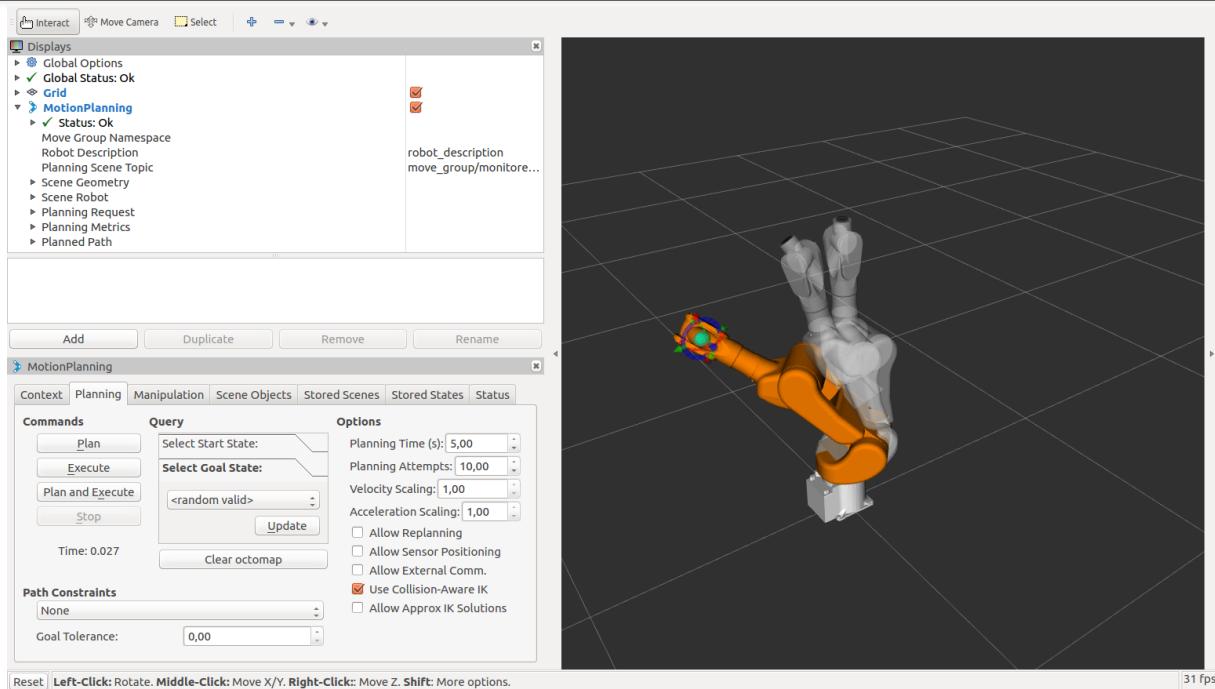


Figure 133 Oluşan Plan

Planlanan yörüngenin kol kontrolörüne “Execute” ile gönderilmesiyle beraber Gazebo üzerinde robotun hedef konumuna ulaşması aşağıdaki gibi görülmektedir;



Figure 134 Gazebo Üzerinde Görünüm

5.3. Örnek Uygulama

Örnek olarak aşağıda gösterildiği gibi bir başlangıç duruşuna (yeşil) ve hedef duruşuna (turuncu) sahip bir robot için hareket planı oluşturulsun.

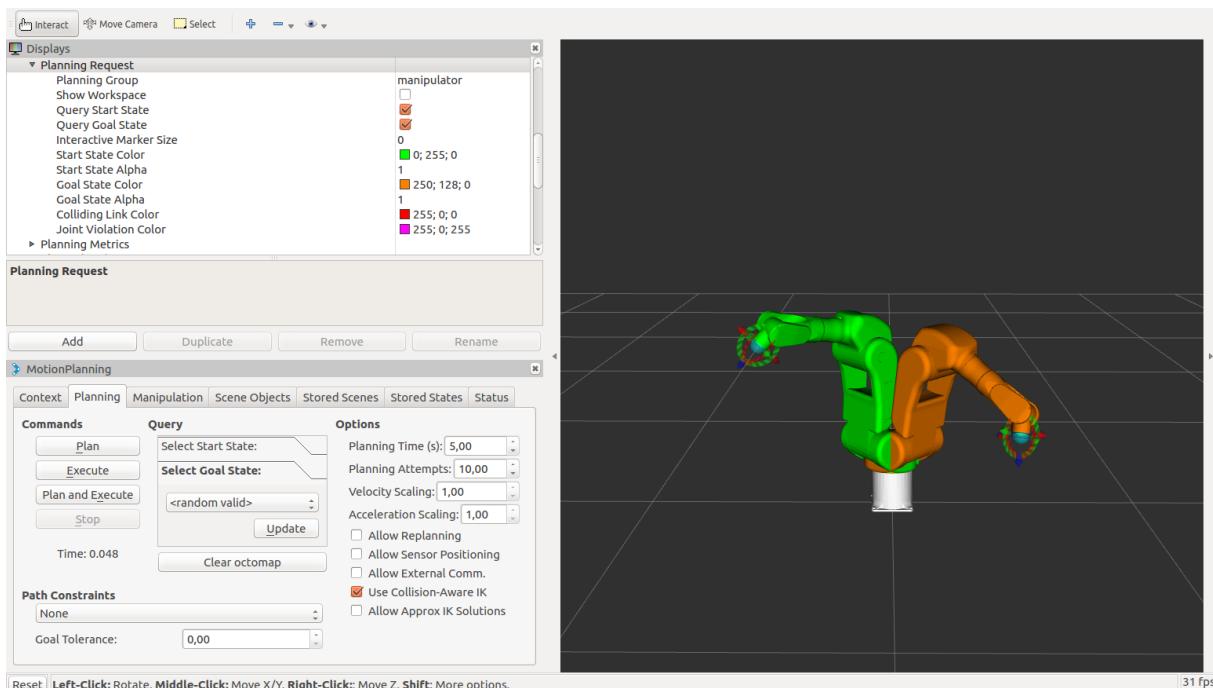


Figure 135 Örnek Uygulama

Hareket planı oluşturulduğunda oluşan yörüngenin belirli anlarındaki robot duruşları (beyaz) Rviz üzerinde aşağıdaki gibi gözükecektir. Burada “Show Trail” seçeneği seçilerek, “Trail Step Size” değeri “6” olarak belirlenmiştir.

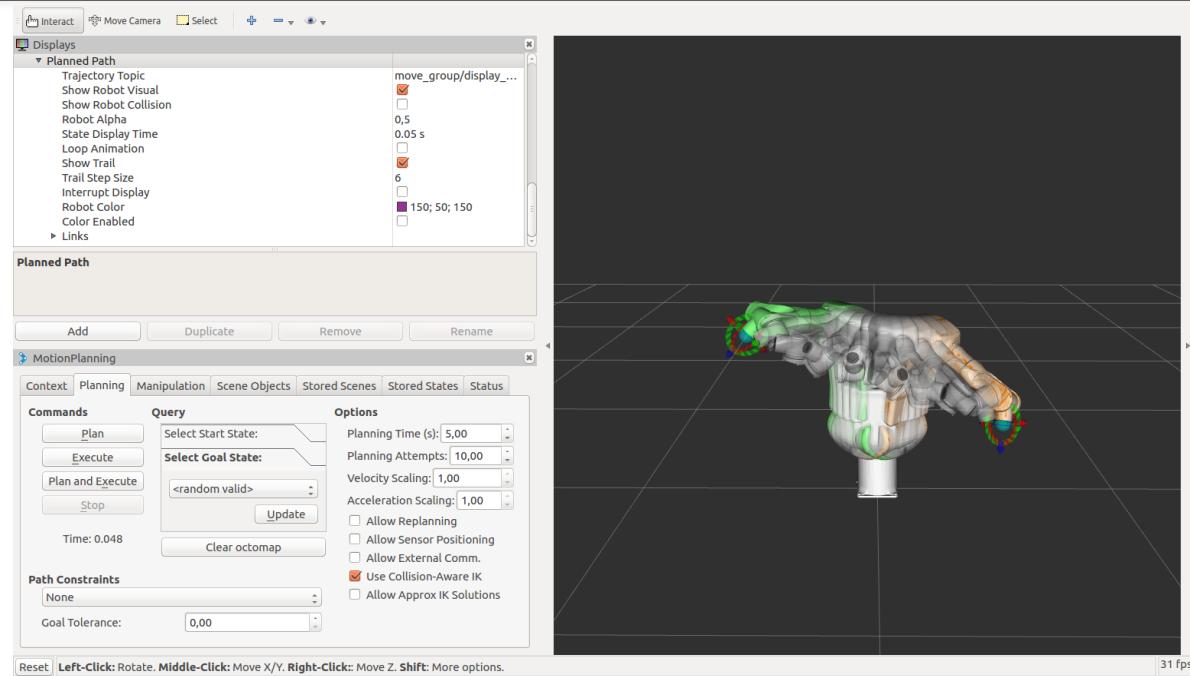


Figure 136 Örnek Uygulama

Rviz üzerinde “Execute” komutunu uyguladığımızda, Gazebo üzerinde yer alan modelin başlangıç konumundan hedef konumuna gittiği görülecektir.



Figure 137 Örnek Uygulama