

KOU ROVER
YAZILIM EKİBİ

**ROBOTİK PROJE İÇİN PYQT 6 KULLANARAK ARAYÜZ
GELİŞTİRME VE ROS 2 İLE ENTEGRASYON**

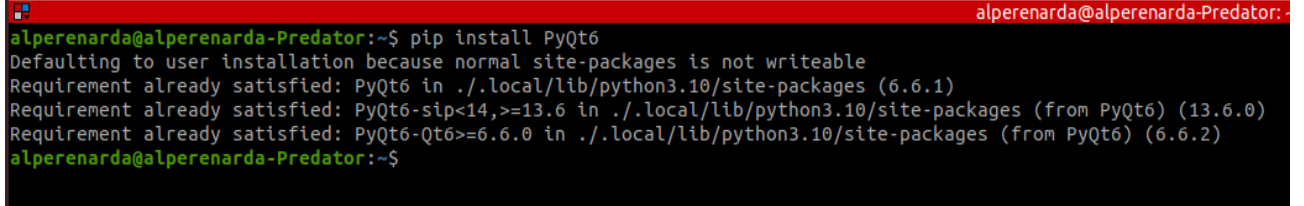
ADEM ALPEREN ARDA

İZMİR 2024

PyQt 6 Kurulumu, Diğer Kütüphaneler İçin ROS 2 Paketi Oluşturma ve Yapılandırma

1. ADIM: Bilgisayardaki terminal (veya Terminator uygulaması) açılır. Terminalde aşağıdaki komut satırı çalıştırılır. Bu komut, pip kullanarak PyQt6 kütüphanesini indirir ve kurar:

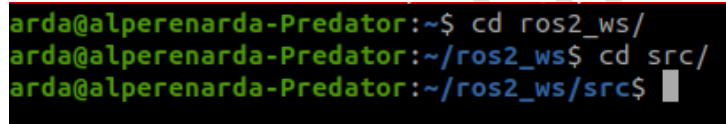
“pip install PyQt6”



```
alperenarda@alperenarda-Predator:~$ pip install PyQt6
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: PyQt6 in ~/.local/lib/python3.10/site-packages (6.6.1)
Requirement already satisfied: PyQt6-sip<14,>=13.6 in ~/.local/lib/python3.10/site-packages (from PyQt6) (13.6.0)
Requirement already satisfied: PyQt6-Qt6>=6.6.0 in ~/.local/lib/python3.10/site-packages (from PyQt6) (6.6.2)
alperenarda@alperenarda-Predator:~$
```

Şekil 1.1. Terminalden pip ile PyQt 6 kütüphanesi kurulumu

2. ADIM: Bilgisayardaki terminal (veya Terminator uygulaması) açılır. Daha önce kurmuş olduğunuz **“ROS 2 Workspace”** ve **“src”** yoluna gidilir.



```
arda@alperenarda-Predator:~$ cd ros2_ws/
arda@alperenarda-Predator:~/ros2_ws$ cd src/
arda@alperenarda-Predator:~/ros2_ws/src$
```

Şekil 1.2. Terminalde ROS 2 Workspace yoluna gitme

3. ADIM: Eğer daha önce arayüz paketi kurulmadıysa yeni paket kurmak için aşağıdaki kod satırı çalıştırılır. Burada dikkat edilmesi gereken nokta, **“PyQt6”** ve **“rclpy”** bağımlılıklarının eklenmesidir. Ayrıca kod satırında paketin ismi tercihe yönelik **“ros_gui_pkg”** olarak yazılmıştır. Farklı proje ihtiyaçlarına göre farklı bir isim kullanılabilir:

“ros2 pkg create ros_gui_pkg --build-type ament_python --dependencies rclpy pyqt6”

```

alperenarda@alperenarda-Predator:~/ros2_ws/src$ ros2 pkg create ros_gui_pkg --build-type ament_pytho
n --dependencies rclpy pyqt6
going to create a new package
package name: ros_gui_pkg
destination directory: /home/alperenarda/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['alperenarda <alperen.arda.adem22@gmail.com>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: ['rclpy', 'pyqt6']
creating folder ./ros_gui_pkg
creating ./ros_gui_pkg/package.xml
creating source folder
creating folder ./ros_gui_pkg/ros_gui_pkg
creating ./ros_gui_pkg/setup.py
creating ./ros_gui_pkg/setup.cfg
creating folder ./ros_gui_pkg/resource
creating ./ros_gui_pkg/resource/ros_gui_pkg
creating ./ros_gui_pkg/ros_gui_pkg/__init__.py
creating folder ./ros_gui_pkg/test
creating ./ros_gui_pkg/test/test_copyright.py
creating ./ros_gui_pkg/test/test_flake8.py
creating ./ros_gui_pkg/test/test_pep257.py

[WARNING]: Unknown license 'TODO: License declaration'. This has been set in the package.xml, but n
o LICENSE file has been created.
It is recommended to use one of the ament license identifiers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
alperenarda@alperenarda-Predator:~/ros2_ws/src$ ls
gui_pkg  my_cpp_pkg  my_py_pkg  ros_gui_pkg  templates
alperenarda@alperenarda-Predator:~/ros2_ws/src$

```

Şekil 1.3. Gerekli bağımlılıklara sahip “ros_gui_pkg” adlı bir paket oluşturulur.

4. ADIM: Paket oluşturulduktan sonra terminalde “ros2_ws” dizinine geçiş yapılır. Yapılan değişiklikleri ROS 2 yazılımında kaydetmek için sıklıkla kullanılan “colcon build” komutu çalıştırılır:

“colcon build --packages-select ros_gui_pkg”

```

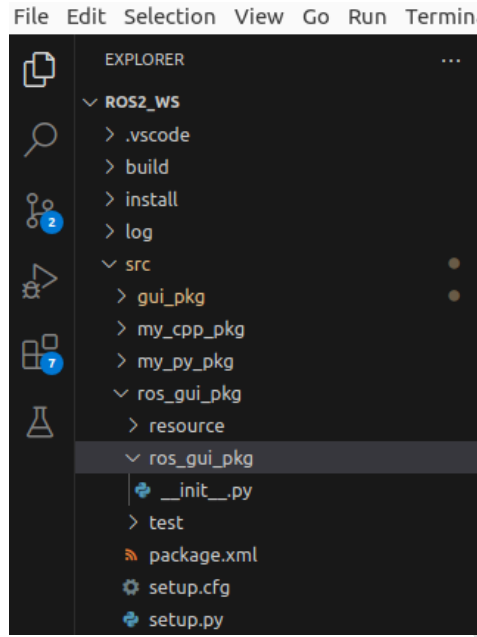
alperenarda@alperenarda-Predator:~$ cd ros2_ws/
alperenarda@alperenarda-Predator:~/ros2_ws$ colcon build --packages-select ros_gui_pkg
Starting >>> ros_gui_pkg
Finished <<< ros_gui_pkg [0.57s]

Summary: 1 package finished [0.74s]
alperenarda@alperenarda-Predator:~/ros2_ws$

```

Şekil 1.4. colcon build komutu ile yapılan işlemleri ROS 2 yazılımı içerisinde kaydetmek

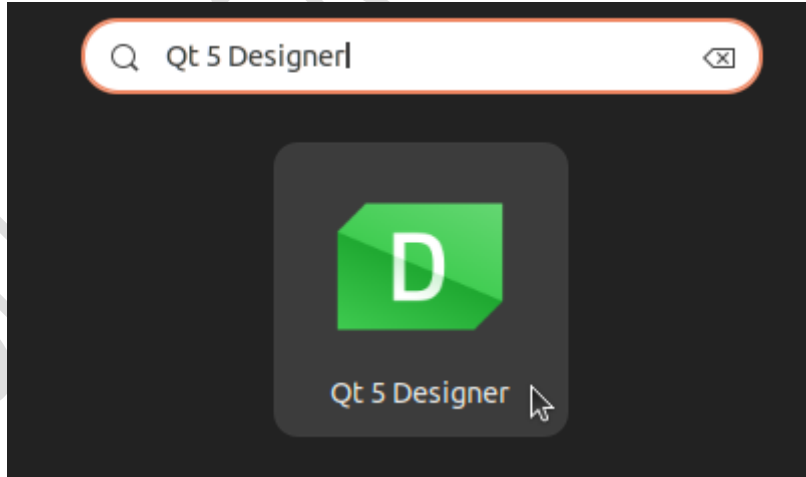
“Visual Studio Code” açıldığında, “ros2_ws/src” yolunda paketin oluşturulmuş olduğu gözlemlenebilir. Bu paketin içinde iki önemli dosya olan “__init__.py” ve “setup.py” dosyalarının bulunduğu dikkat edilmelidir.



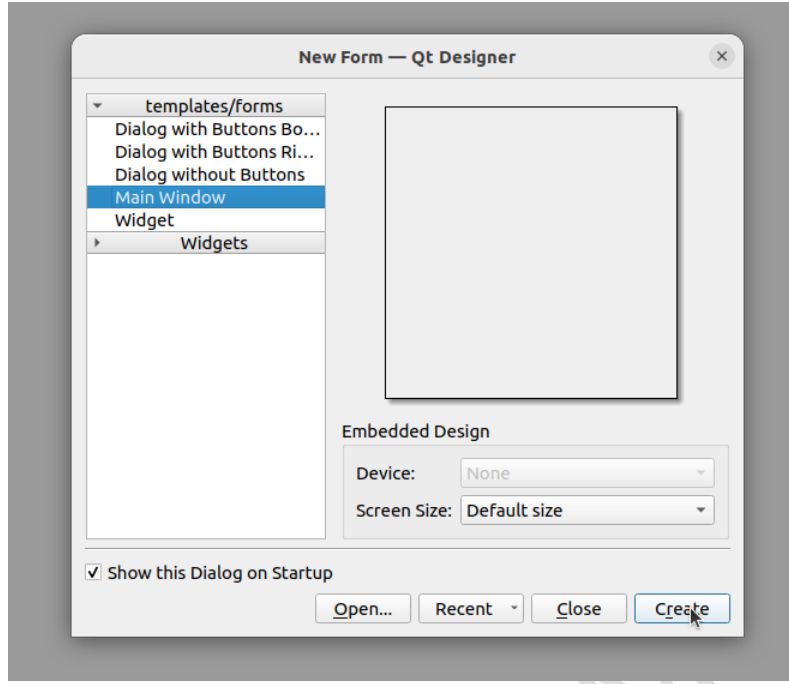
Şekil 1.5. Oluşturulan paketin Visual Studio Code üzerinde görünümü

Qt 5 Designer Kullanarak Arayüzün Görsel Temellerinin Oluşturulması

1. ADIM: ROS 2 kurulumu ile otomatik olarak yüklenen Qt 5 Designer uygulaması başlatılır. Uygulama açıldıktan sonra proje oluşturmak için “*New Form – Qt Designer*” menüsündeki “*templates/forms*” sekmesinden “*Main Window*” seçeneği seçildikten sonra “*Create*” butonuna tıklanır ve yeni proje oluşturulmuş olur.



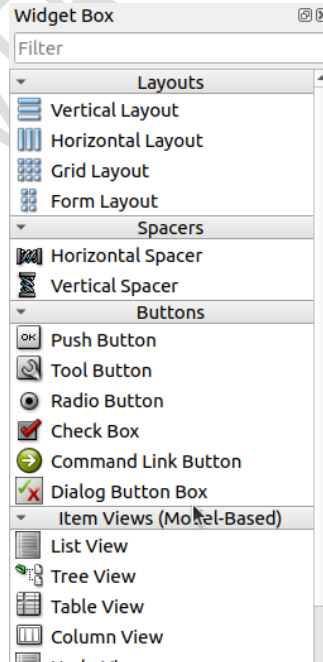
Şekil 2.1. Qt 5 Designer



Şekil 2.2. Uygulama başlangıcında yeni proje oluşturma

2. ADIM: Yeni oluşturulan “*untitled*” adlı dosya, kaydedilip isim verilmeden önce bileşenler incelenecektir ve gerekli olan nesneler örnek arayüze eklenecektir.

Bileşenler(Nesneler): Nesneler(Objects), Qt 5 Designer programının sol tarafındaki “*Widget Box*” penceresinde bulunurlar ve imleç ile “*Main Window*” nesnesine sürüklenip kullanılabilirler.

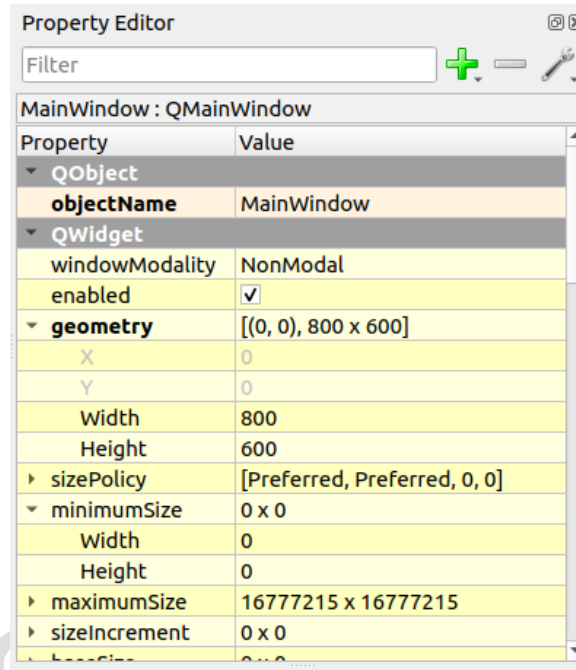


Şekil 2.3. Widget Box

1) Main Window: Temel penceredir. İsmi veya özellikleri sağ tarafta bulunan “*Property Editor*” penceresinden değiştirilebilir.



Şekil 2.4. Main Window Nesnesi



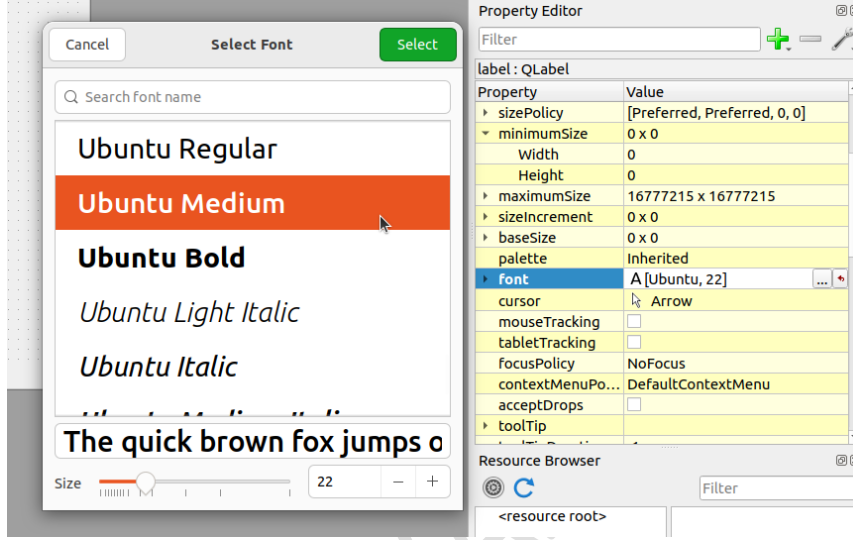
Şekil 2.5. Property Editor Penceresi

Her nesne için bulunan **“Property Editor”** penceresinde; nesne adı, yükseklik-genişlik, yazı tipi gibi onlarca özellik üstüne çift tıklanarak ayarlanabilir. Bu yazıda her nesnenin en çok kullanılan özelliği için birer örnek verilecektir. Her nesne için anlatılan özellikler bu yazıda yeterli olmamakla beraber diğer ayrıntılı özelliklerin incelenmesi için **“Property Editor”** penceresine daha ayrıntılı göz atılabilir.

2) **Label:** Arayüze yazı ve resim eklenmesi için kullanılır. **“Property Editor”** penceresi üzerinden; nesne adı, yazı tipi, yazı büyüklüğü ayarlanabilir. Resim yolu eklenebilir. Yazı tipi seçimi ve yazı tipi büyüklüğü seçimi için **“Property Editor”** penceresindeki **“font”** özelliğinin yanındaki üç noktaya tıklanır ve Şekil 2.7. deki pencere açılır. Buradan gerekli işlemler yapıp **“Select”** tuşuna tıklanır.



Şekil 2.6. Label nesnesi



Şekil 2.7. Label nesnesinin font 'unun değiştirilmesi

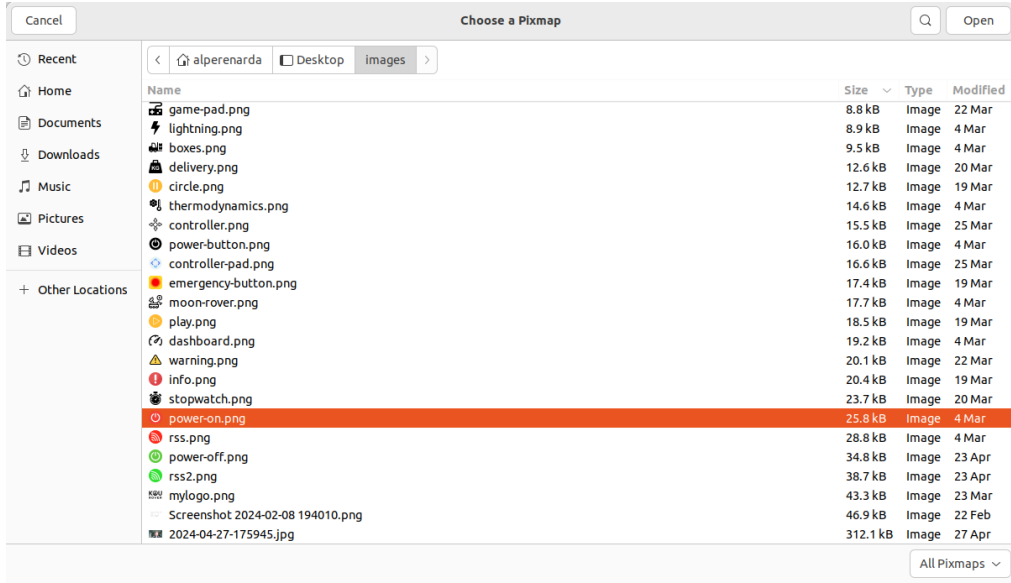
Font değiştirmenin yanı sıra Label nesnesi yoluyla resim ekleme yöntemi gösterilecektir.

Bunun için yeni bir Label eklenir ve ardından üstüne tek tıklanır. **“Property Editor”** penceresinde bulunan **“QLabel”** bölümündeki **“pixmap”** özelliğine tek tıklanır ve o özelliğin sağında beliren **“aşağı ok”** simgesine tıklanır. Tıklandıktan sonra **“Choose Resource”** ve **“Choose File”** seçenekleri belirir. **“Choose File”** seçeneği seçilir.



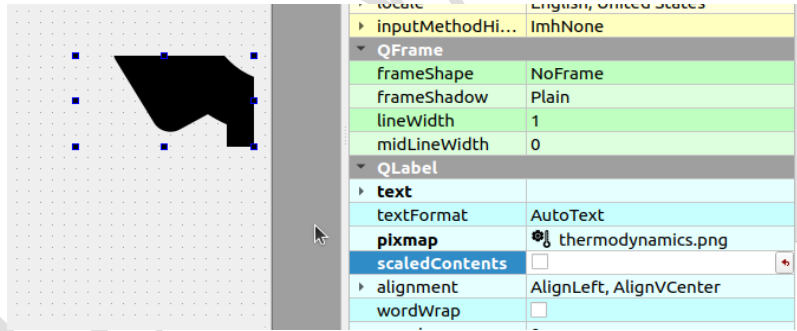
Şekil 2.8. Label nesnesine dosya yolundan resim eklenmesi

“Choose File” seçeneği seçildikten sonra Şekil 2.9. daki pencere ekrana gelir. Buradaki pencerede bilgisayara daha önce indirilen resmin resim yolu bulunur ve resmin üstüne çift tıklanır.



Şekil 2.9. “Choose File” seçeneğini seçtikten sonra gelen pencere

Ardından Label nesnesine resim yüklendiği görülür. Bu işlemlerin sonucunda eklediğimiz resim Label nesnesinin boyutlarına sığmayabilir. Bu sorunun çözülmesi için tekrardan Label nesnesinin üstüne tıklanır. Ekranın sağ tarafındaki **“Property Editor”** penceresinde bulunan **“QLabel”** bölümündeki **“scaledContents”** özelliğinin yanındaki kutuya tek tıklanarak özellik aktif edilir. Label nesnesinin köşelerindeki noktaları sürükleyerek resmin büyüklüğü ayarlanabilir.



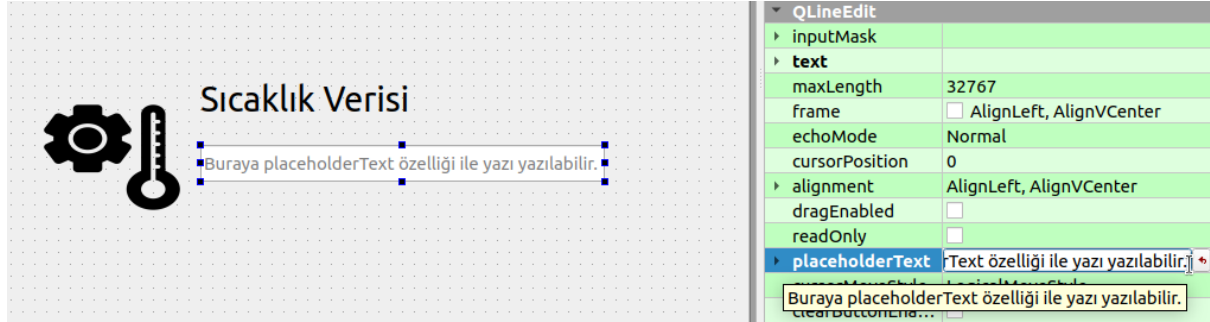
Şekil 2.10. “scaledContents” özelliği pasif iken



Şekil 2.11. “scaledContents” özelliğinin aktif edilmesi

3) LineEdit: Arayüze girdi(input) girilmesi veya sayısal verilerin gösteriminin kolaylığı için kullanılır. **“Property Editor”** penceresi üzerinden; nesne adı, font, lineEdit büyüklüğü, lineEdit başlangıç yazısı, lineEdit yazısı gibi özellikler ayarlanabilir. Bu nesne için örnek olarak başlangıç yazısı eklenmesi gösterilecektir.

Öncelikle, daha önce belirtildiği gibi QLineEdit nesnesi, **"Widget Box"** penceresinden **"MainWindow"** penceresine sürüklenir. Label nesnesinde uygulanan adımlar gibi **"font"** özelliğindeki üç noktaya tek tıklanır. Açılan pencerede uygun font büyüklüğü seçilip **"Select"** seçeneğine tıklanır. Ardından QLineEdit başlangıç yazısının eklenmesi için **"QLineEdit"** bölümündeki **"placeholderText"** özelliğine tıklanıp uygun metin yazılır. Son olarak klavyedeki Enter tuşuna basılır ve QLineEdit başlangıç yazısı eklenmiş olur.

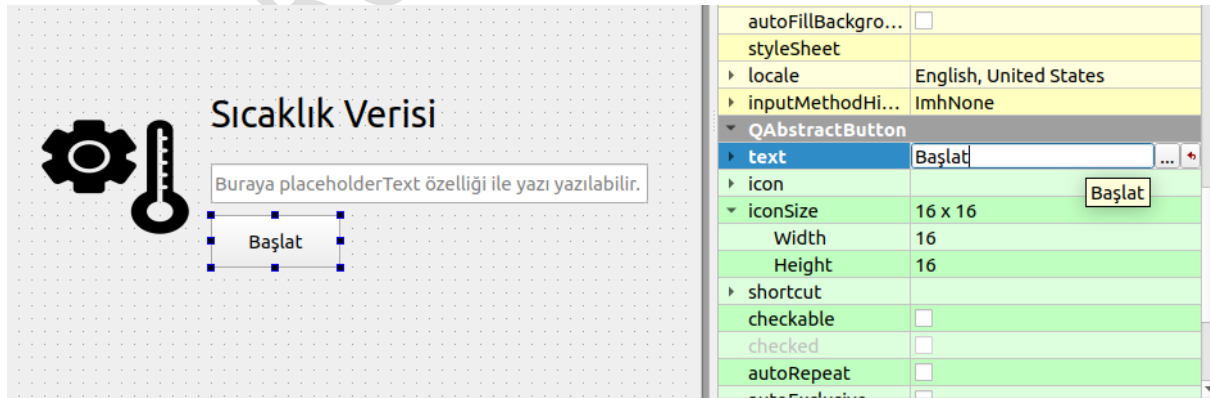


Şekil 2.12. "placeholderText" özelliği ile QLineEdit nesnesine başlangıç metni ekleme

LineEdit nesnesinin diğer özellikleri, gösterilen yordamlar gibi eklenebilir ve özelleştirilebilir.

Push Button: Push Button genel olarak belirli bir görevin yerine getirilmesi, bir komutun çalıştırılması veya başka bir ekranla ya da işlemle etkileşime geçilmesi amacıyla kullanılır. **"Property Editor"** penceresi üzerinden; butonun adı, fontu, butonun büyüklüğü, buton üzerindeki yazı, butonun iconu ve butonun tıklanma olayı gibi özellikler ayarlanabilir. Bu nesne için örnek olarak butona ikon ve yazı eklenmesi gösterilecektir.

Öncelikle, daha önce belirtildiği gibi Push Button nesnesi, **"Widget Box"** penceresinden MainWindow penceresine sürüklenir. Ardından Push Button metninin eklenmesi için **"QAbstractButton"** bölümündeki **"text"** özelliğine tıklanıp uygun metin yazılır. Son olarak klavyedeki Enter tuşuna basılır ve Push Button nesnesine metin eklenmiş olur. Push Button nesnesinin köşelerindeki noktaları sürükleyerek butonun büyüklüğü ayarlanabilir.

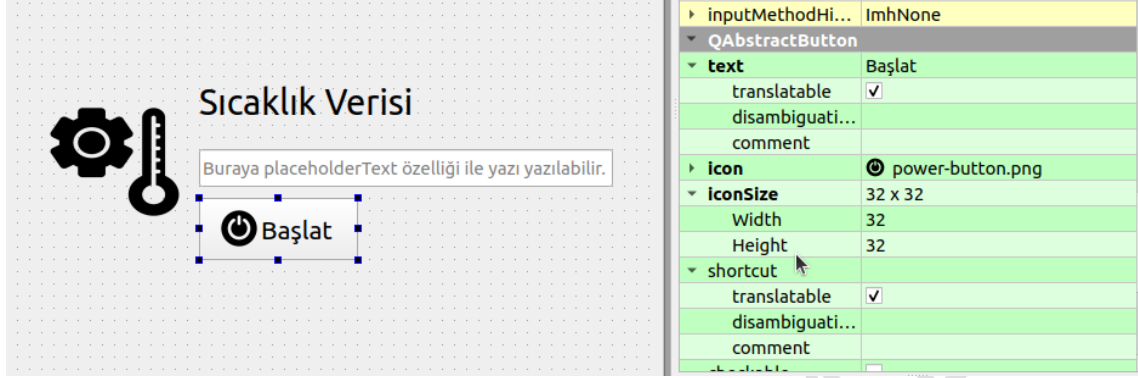


Şekil 2.13. Push Button yazı ekleme

Butona ikon eklenmesi için **"QAbstractButton"** bölümündeki **"icon"** özelliğine tıklanır ve o özelliğin sağında beliren **"aşağı ok"** simgesine tıklanır. Tıklandıktan sonra **"Choose Resource"**, **"Choose File"** ve **"Set Icon From Theme"** seçenekleri belirir. **"Choose File"**

seçeneği seçilir. **“Choose File”** seçeneği seçildikten sonra Şekil 2.9. daki pencere ekrana gelir. Buradaki pencerede bilgisayara daha önce indirilen resmin resim yolu bulunur ve resmin üstüne çift tıklanır.

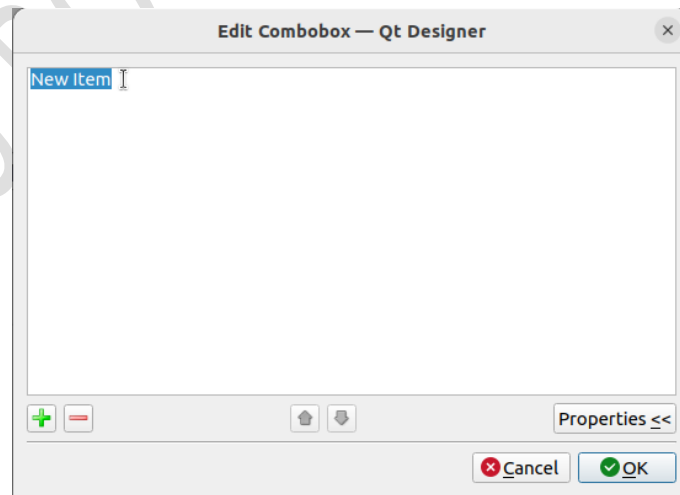
Push Button nesnesinin diğer özellikleri, gösterilen yordamlar gibi eklenebilir ve özelleştirilebilir.



Şekil 2.14. Push Button ikon ekleme

Combo Box: ComboBox, kullanıcıların açılır listeden bir öğe seçmelerine olanak tanıyan bir arayüz bileşenidir. Kullanıcılar, listeyi açarak mevcut seçeneklerden birini seçebilir veya kullanıcı tarafından giriş yapılabilir. **“Property Editor”** penceresi üzerinden; Combo Box'ın adı, fontu, boyutu, açılır liste öğeleri, varsayılan seçili öğe ve kullanıcı girişi izinleri gibi özellikler ayarlanabilir. Bu nesne için örnek olarak Combo Box'a açılır liste öğelerinin eklenmesi gösterilecektir.

Öncelikle, daha önce belirtildiği gibi Combo Box nesnesi, **“Widget Box”** penceresinden MainWindow penceresine sürüklenir. Ardından Combo Box'a açılır liste öğelerinin eklenmesi için Combo Box'a çift tıklanır ve açılan pencerenin sol altındaki **“artı”** ikonuna tıklanır. Combo Box nesnesine yeni öğe eklenmiş olur.



Şekil 2.15. Edit Combobox penceresi

“New Item” yazısının üstüne çift tıklayarak istenilen yazı yazılır ve bu işlem gerekli öğe sayısına göre tekrar edilir. Eklenmek istenen öğeler bittikten sonra sağ alttaki “OK” butonuna tıklanır ve öğeler kaydedilir.

Radio Button: Radio Button, genellikle kullanıcıların bir grup seçenek arasından yalnızca birini seçmesine olanak tanıyan bir arayüz bileşenidir. Bir grup Radio Button içindeki sadece bir seçenek aktif olabilir. **“Property Editor”** penceresi üzerinden; butonun adı, fontu, butonun büyüklüğü, buton üzerindeki yazı ve butonun varsayılan olarak seçili olup olmadığı gibi özellikler ayarlanabilir.



Şekil 2.16. Radio Button nesneleri

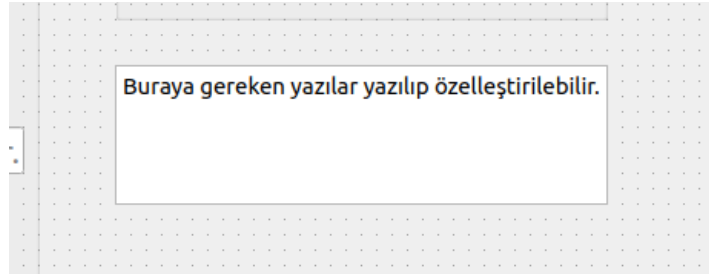
Group Box: Group Box, kullanıcı arayüzünde ilgili bileşenleri bir arada gruplamak için kullanılır. Bu bileşen, içindeki bileşenleri görsel olarak ayırarak daha düzenli ve anlaşılır bir arayüz oluşturulmasına yardımcı olur. Group Box, başlık ve çerçeve içerir ve içinde çeşitli diğer bileşenleri barındırabilir. **“Property Editor”** penceresi üzerinden; Group Box'ın adı, fontu, boyutu, başlık metni ve çerçeve stili gibi özellikler ayarlanabilir.



Şekil 2.17. Group Box ile nesneleri gruplama

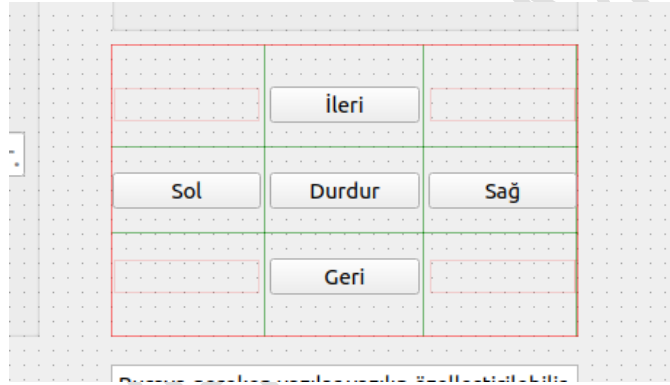
Text Edit: Text Edit, kullanıcıların birden fazla satır içeren metinleri girmesine, düzenlemesine ve görüntülemesine olanak tanıyan bir arayüz bileşenidir. Genellikle kullanıcı girdisi gerektiren formlarda, metin belgelerinde ve not alma uygulamalarında kullanılır.

“Property Editor” penceresi üzerinden; Text Edit'in adı, fontu, boyutu, başlangıç metni, metin hizalaması, metin biçimlendirme seçenekleri ve kaydırma çubukları gibi özellikler ayarlanabilir.



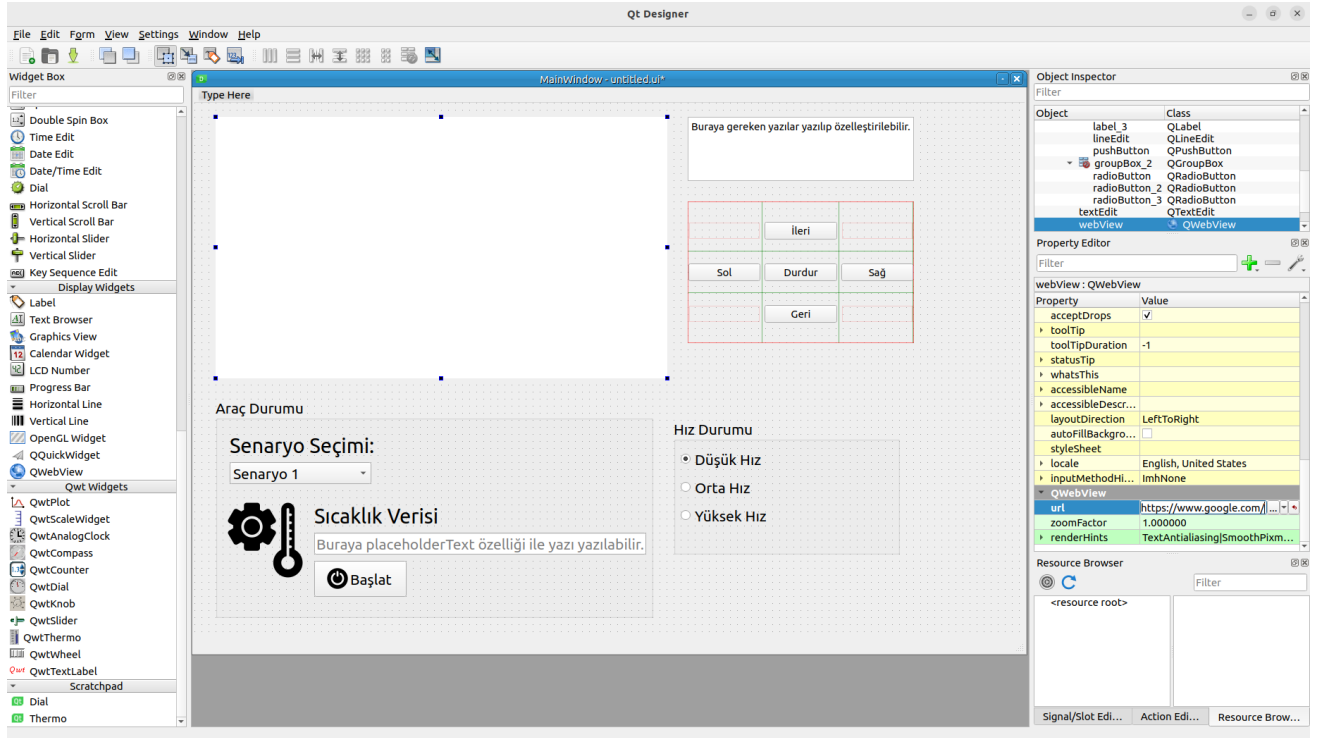
Şekil 2.18. Text Box nesnesi

Layouts: Layouts, kullanıcı arayüzünde bileşenlerin düzenlenmesi ve hizalanması için kullanılan yönetim araçlarıdır. Layouts, bileşenlerin belirli bir düzen içinde yerleştirilmesini sağlar, böylece arayüzün estetik ve işlevsel olması garanti edilir. "**Property Editor**" penceresi üzerinden; Layout'un türü (dikey, yatay, ızgara vb.), boşluklar, kenar boşlukları, hizalama ve içerik boyutu gibi özellikler ayarlanabilir. Şekil 2.19. da örnek olarak Layout nesnesine 5 adet Push Button eklenecektir. Push Button ekleme işlemi için 5 adet buton Layouts nesnesinin içine imleç yardımı ile sürüklenenecektir.



Şekil 2.19. Layout nesnesi

QWebView: QWebView, bir uygulama içinde web içeriği görüntülemeye olanak tanıyan bir arayüz bileşenidir. Bu bileşen; web sayfalarını render edebilir, JavaScript çalıştırabilir ve kullanıcı etkileşimlerini yönetebilir. "**Property Editor**" penceresi üzerinden; QWebView'in adı, boyutu, başlangıç URL'si gibi özellikler ayarlanabilir. Sonraki başlıklarda QWebView nesnesinin kod üzerindeki açıklaması yapılacaktır. Başlangıçta örnek olması için "**Property Editor**" penceresinde bulunan "**QWebView**" bölümündeki "**url**" özelliğine "**https://www.google.com**" bağlantısı eklenir.

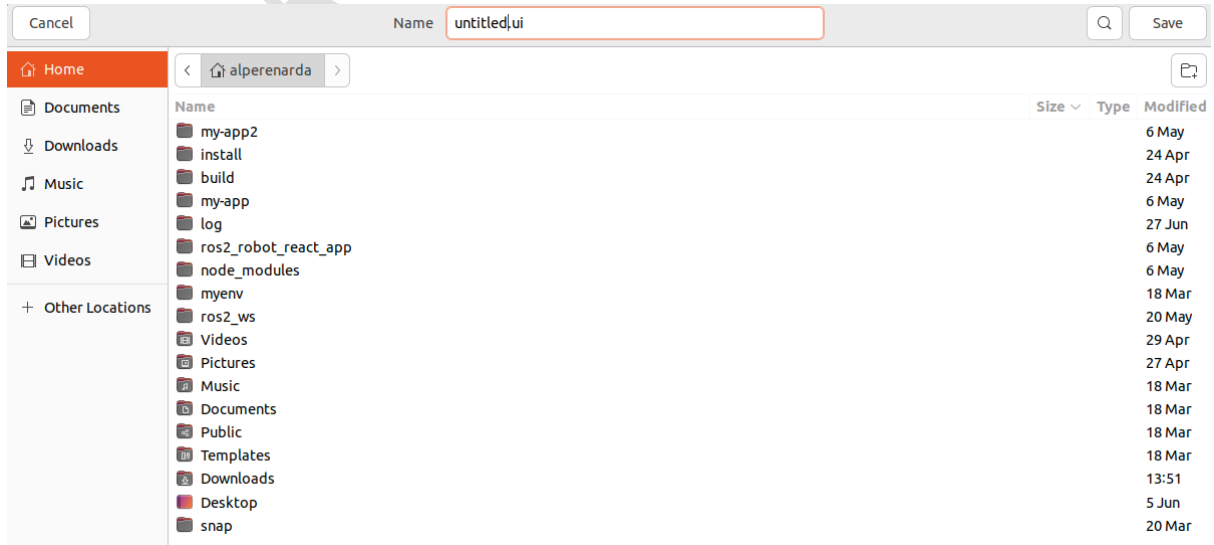


Şekil 2.20. QWebView ve örnek arayüzün son görüntüsü

Oluşturulan örnek arayüz önceki başlıkta oluşturulan “**ros_gui_pkg**” adlı ROS 2 paketine kaydedilecektir. “**ros_gui_pkg**” paketine kaydedildiğinde, Visual Studio Code üzerinden “**Python**” aracılığıyla sonradan özelleştirilebilir ve ekleme yapılabilir.

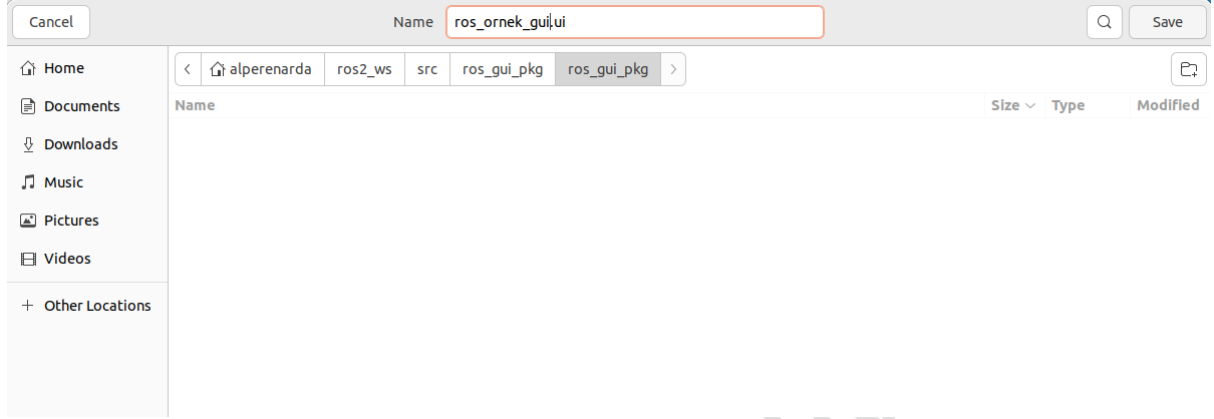
Qt 5 Designer Programından Oluşturulan .ui Uzantılı Arayüzün ROS 2 Paketine Kaydedilmesi ve Dosya Dönüşümü

1. ADIM: Arayüzdeki temel tasarım oluşturulduktan sonra Qt 5 Designer programında iken “**Ctrl + S**” tuş kombinasyonu kullanılır. Ardından Şekil 3.1. deki pencere ekrana gelir.



Şekil 3.1. Ctrl+S kullandıktan sonra ekrana gelen pencere

2. ADIM: Yapılması kritik olan bu adımda, **“.ui uzantılı dosya”**, önceki başlıkta oluşturulan **“ros_gui_pkg”** adlı ROS 2 paketine kaydedilecektir. Bunun için *Şekil 3.1.* deki pencerede **“ros2_ws/src/ros_gui_pkg/ros_gui_pkg”** yoluna gelinir. Tercihe bağlı olarak projenin adı üst çubuktan değiştirilebilir. Bu projeye örnek olması için **“ros_ornek_gui”** adı verilmiştir. Ardından sağ üstteki **“Save”** butonuna tıklanır.



Şekil 3.2. ROS 2 paketine gelindikten sonra .ui uzantılı dosyanın kaydedilmesi

3. ADIM: **“.ui”** uzantılı dosyayı Python dilinde geliştirmek için, diğer bir deyişle oluşturulan **“.ui”** uzantılı projeyi **“.py”** uzantılı dosyaya dönüştürmek için terminal (veya Terminator uygulaması) açılır. Aşağıdaki komut satırı terminale yazılır:

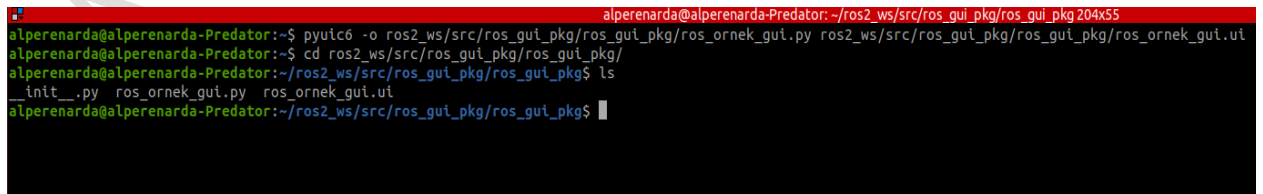
```
“pyuic6 -o ros2_ws/src/ros_gui_pkg/ros_gui_pkg/ros_ornek_gui.py  
ros2_ws/src/ros_gui_pkg/ros_gui_pkg/ros_ornek_gui.ui”
```

Bu komutun, **“.py”** ve **“.ui”** uzantılı dosyaların yolları verilerek yazıldığına dikkat edilmesi gerekir. Bu komut daha ayrıntılı incelenirse üç kısma ayrıldığı görülebilir:

“pyuic6 -o”, çıktı dosyasının adını belirtir ve PyQt 6 ile dosya oluşturmayı sağlar. Bir önceki sürüm olan PyQt 5 kullanılmak istenirse **“pyuic5 -o”** komut parçası kullanılabilir.

“ros2_ws/src/ros_gui_pkg/ros_gui_pkg/ros_ornek_gui.py”, oluşacak dosyayı belirtir.

“ros2_ws/src/ros_gui_pkg/ros_gui_pkg/ros_ornek_gui.ui”, var olan arayüz dosyasını belirtir. Bu dosya için, önceki adımda hangi yola kaydedildiyse o yolun yazılması gerektiğine dikkat edilmelidir.

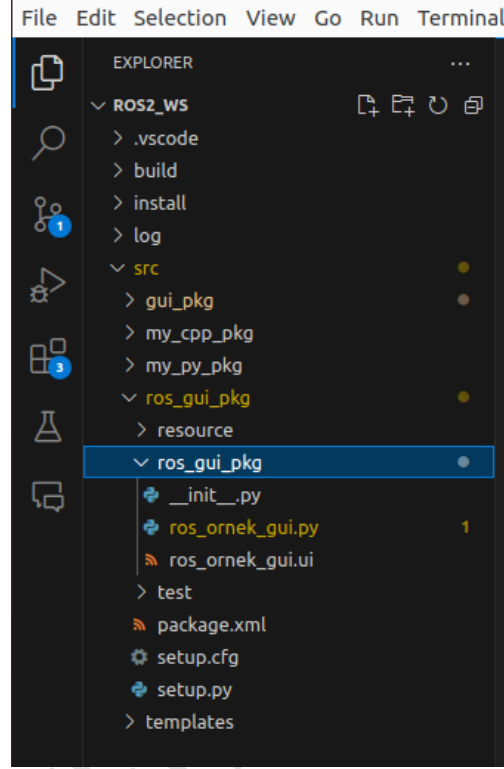


Şekil 3.3. .ui uzantılı dosyadan .py uzantılı dosyaya dönüşüm

Sonuç olarak **“ros_gui_pkg”** paketinde **“__init__.py”**, **“ros_ornek_gui.ui”**, **“ros_ornek_gui.py”** üç adet temel dosya oluşturulmuştur. **“__init__.py”** dosyası, bir dizinin

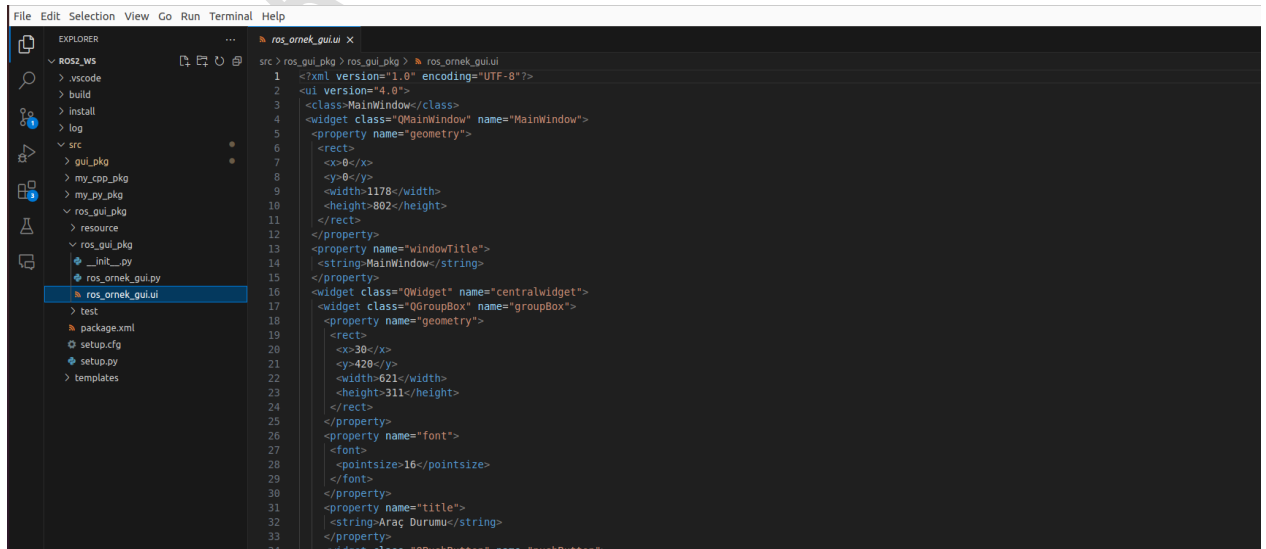
Python paketi olarak tanınmasını sağlar ve bu dizindeki modüllerin içe aktarılabilir hale gelmesini mümkün kılar.

"*ros_ornek_gui.ui*" dosyası, Qt Designer ile oluşturulan arayüz tanımlamalarını içerirken, "*ros_ornek_gui.py*" dosyası bu arayüz tanımlamalarını Python kodu olarak kullanmak için dönüştürülmüş halidir.

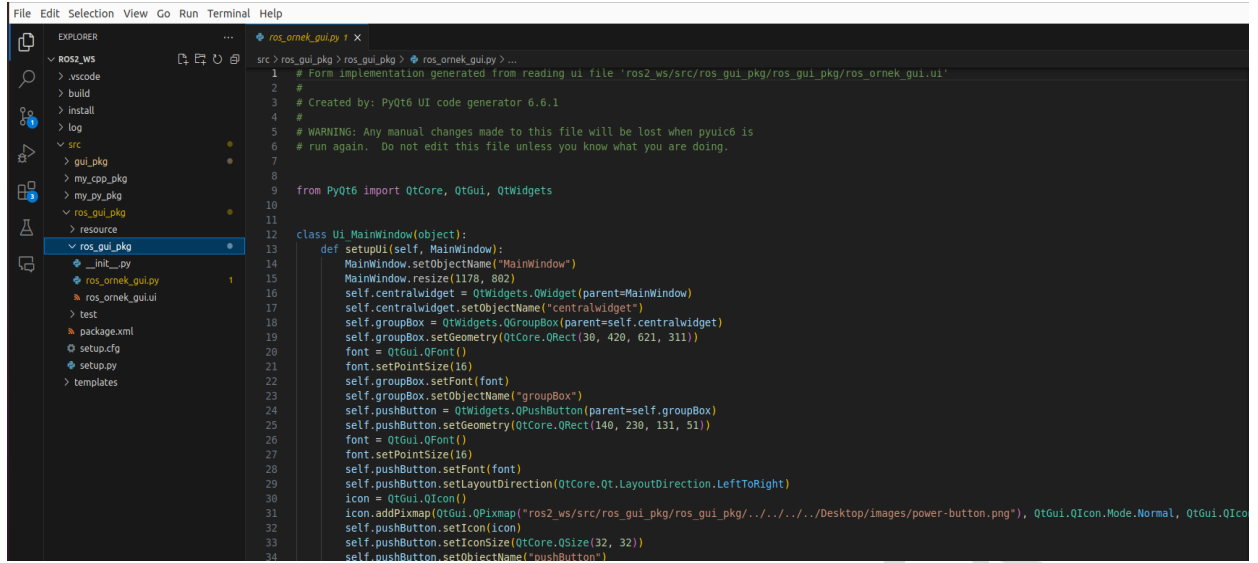


Şekil 3.4. *ros_gui_pkg* paketinin Visual Studio Code içinde görünümü

"*ros_ornek_gui.py*" dosyası, "*ros_ornek_gui.ui*" dosyasından aldığı veriyi otomatik olarak dönüştürür ve programcıya hazır kod verir.

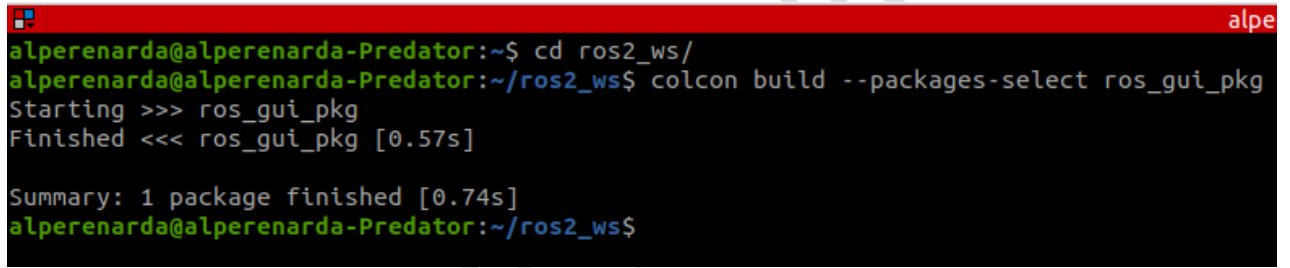


Şekil 3.5. *ros_ornek_gui.ui*



Şekil 3.6. *ros_ornek_gui.py*

4. ADIM: Bu işlemleri kaydetmek için “*ros2_ws*” yoluna gelinir ve “*colcon build*” komutu terminalde çalıştırılır.



Şekil 3.7. *colcon build* komutunun çalıştırılıp işlemlerin kaydedilmesi

Dönüştürülen .py Uzantılı Dosyanın Özelleştirilmesi, Debug Edilmesi ve Yeni Bir Fonksiyonel .py Uzantılı Dosyanın Oluşturulması

1. ADIM: PyQt, eski sürümlerde bulunan “*QWebView*” kütüphanesi nedeniyle kütüphane hatası verir. Bu hatanın çözülmesi için “*ros_ornek_gui.py*” dosyasının en altındaki satırda bulunan “*from QtWebKitWidgets.QWebView import QWebView*” satırı silinir ve yeni kütüphane terminalden yüklenir. Pip yardımı ile aşağıdaki kod satırı terminalde çalıştırılır:

“pip install PyQt6-WebEngine”


```
sumeyra@sumeyra:~$ pip install PyQt6-WebEngine
Defaulting to user installation because normal site-packages is not writeable
Collecting PyQt6-WebEngine
  Downloading PyQt6_WebEngine-6.7.0-cp38-abi3-manylinux_2_28_x86_64.whl (268 kB)
    268.6/268.6 KB 1.3 MB/s eta 0:00:00
Requirement already satisfied: PyQt6>=6.2.0 in ~/.local/lib/python3.10/site-packages (from PyQt6-WebEngine) (6.7.0)
Collecting PyQt6-WebEngine-Qt6<6.8.0,>=6.7.0
  Downloading PyQt6_WebEngine_Qt6-6.7.2-py3-none-manylinux_2_28_x86_64.whl (26.1 MB)
    26.1/26.1 MB 1.3 MB/s eta 0:00:00
Requirement already satisfied: PyQt6-sip<14,>=13.6 in ~/.local/lib/python3.10/site-packages (from PyQt6-WebEngine) (13.6)
Requirement already satisfied: PyQt6-Qt6<6.8.0,>=6.7.0 in ~/.local/lib/python3.10/site-packages (from PyQt6>=6.2.0) (6.7.2)
Collecting PyQt6-WebEngineSubwheel-Qt6==6.7.2
  Downloading PyQt6_WebEngineSubwheel_Qt6-6.7.2-py3-none-manylinux_2_28_x86_64.whl (76.1 MB)
    76.1/76.1 MB 2.0 MB/s eta 0:00:00
Installing collected packages: PyQt6-WebEngineSubwheel-Qt6, PyQt6-WebEngine-Qt6, PyQt6-WebEngine
Successfully installed PyQt6-WebEngine-6.7.0 PyQt6-WebEngine-Qt6-6.7.2 PyQt6-WebEngineSubwheel-Qt6-6.7.2
sumeyra@sumeyra:~$
```

Şekil 4.1. Terminalden pip ile PyQt6-WebEngine kurulumu

“from PyQt6.QtWebEngineWidgets import QWebEngineView” kütüphane satırı
“ros_ornek_gui.py” dosyasına eklenir.

2. ADIM: Önceki adımda “QWebView” kütüphanesi yerine “QWebEngineView”
kütüphanesi kurulduğu için Şekil 4.2. deki kod parçası “ros_ornek_gui.py” dosyasından
kaldırılır.

```
self.pushButton_5.setObjectName("pushButton_5")
self.gridLayout.addWidget(self.pushButton_5, 1, 1, 1, 1)
self.pushButton_8 = QtWidgets.QPushButton(parent=self.gridLayoutWidget)
self.pushButton_8.setObjectName("pushButton_8")
self.gridLayout.addWidget(self.pushButton_8, 1, 0, 1, 1)
self.webView = QWebView(parent=self.centralwidget)
self.webView.setGeometry(QtCore.QRect(30, 20, 641, 371))
self.webView.setUrl(QtCore.QUrl("https://www.google.com.tr/"))
self.webView.setObjectName("webView")
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(parent=MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 1178, 22))
self.menubar.setObjectName("menubar")
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(parent=MainWindow)
```

Şekil 4.2. QWebView kütüphanesinin kodlarının kaldırılması

Bu işlemin ardından aşağıdaki kod parçası “ros_ornek_gui.py” dosyasına eklenir ve Şekil
4.3. deki kod ortaya çıkar:

```
self.webView = QWebEngineView(parent=self.centralwidget)

self.webView.setGeometry(QtCore.QRect(30, 49, 960, 731))

self.webView.setUrl(QtCore.QUrl("http://www.google.com"))

self.webView.setObjectName("webView")
```

İlerleyen adımlarda “self.webView.setUrl(QtCore.QUrl(''))” satırı kullanılarak “webView”
nesnesi URL’ye göre özelleştirilecektir. Dosyayı kaydetmek için dosyanın üzerindeyken
“Ctrl + S” tuş kombinasyonu kullanılır.

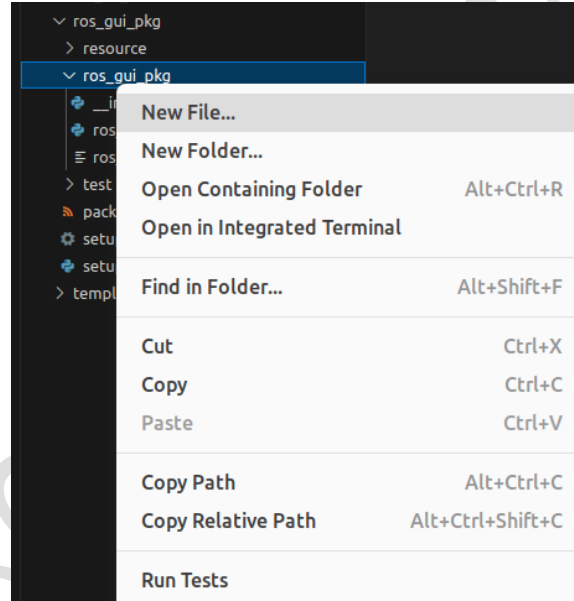
```

self.pushButton_8.setObjectName("pushButton_8")
self.gridLayout.addWidget(self.pushButton_8, 1, 0, 1, 1)
self.webView = QWebEngineView(parent=self.centralwidget)
self.webView.setGeometry(QtCore.QRect(30, 20, 641, 371))
self.webView.setUrl(QtCore.QUrl("http://www.google.com"))
self.webView.setObjectName("webView")
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(parent=MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 1178, 22))

```

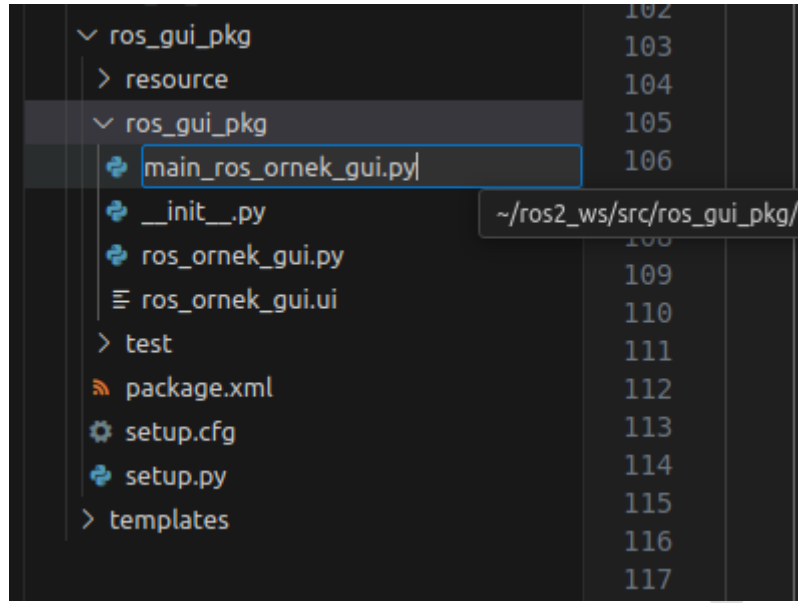
Şekil 4.3. QWebEngineView kütüphanesinin kodlarının eklenmesi

3. ADIM: Genel olarak tasarım amaçlı olarak kullanılan “*ros_ornek_gui.py*” dosyasının diğer amaçlarla kullanılması tercih edilen bir yöntem değildir. Bunun için fonksiyonel işlemlerin ve ROS 2 işlemlerinin gerçekleştirileceği yeni bir .py uzantılı dosya oluşturulacaktır. Visual Studio Code programının sol tarafında bulunan dosyalar sekmesindeki “*ros_ornek_gui.py*” dosyasının bulunduğu konuma gidilir. Bu konuma sağ tıklanır ve ortaya çıkan sağ tıklama menüsünden “*New File*” seçeneği seçilir.



Şekil 4.4. Visual Studio Code üzerinden yeni .py uzantılı dosya oluşturma

Gerekli isim yeni dosyaya verilir. Bu dosya için “*main_ros_ornek_gui.py*” ismi verilmiştir. Ardından Enter tuşuna basılır. Burada oluşturulan dosyanın, tasarım amaçlı olan “*ros_ornek_gui.py*” dosyası ile aynı konumda olmasına dikkat edilmelidir.



Şekil 4.5. Visual Studio Code üzerinden oluşturulan yeni .py uzantılı dosyaya isim verme

4. ADIM: Oluşturulan “**main_ros_ornek_gui.py**” dosyasına aşağıda verilen şablon kopyalanıp yapıştırılır. Bu şablon gereksinimlere göre özelleştirilebilir ve değiştirilebilir:

“import sys

from PyQt6.QtWidgets import QMainWindow, QApplication

class MainWindow(QMainWindow):

def __init__(self, parent=None):

super(MainWindow, self).__init__(parent)

self.ui = Ui_MainWindow()

self.ui.setupUi(self)

def main():

app = QApplication(sys.argv)

win = MainWindow()

win.show()

sys.exit(app.exec())

if __name__ == "__main__":

main()”

```
EXPLORER
src
  ros_gui_pkg
    ros_gui_pkg
      __init__.py
      main_ros_ornek_gui.py
      ros_ornek_gui.py
      ros_ornek_gui.ui
    test
  package.xml
  setup.cfg
  setup.py
  templates

src > ros_gui_pkg > ros_gui_pkg > main_ros_ornek_gui.py > ...
1 import sys
2 from PyQt6.QtWidgets import QMainWindow, QApplication
3
4
5
6 class MainWindow(QMainWindow):
7     def __init__(self, parent=None):
8         super(MainWindow, self).__init__(parent)
9         self.ui = Ui_MainWindow()
10        self.ui.setupUi(self)
11
12 def main():
13     app = QApplication(sys.argv)
14     win = MainWindow()
15     win.show()
16     sys.exit(app.exec())
17
18 if __name__ == "__main__":
19     main()
```

Şekil 4.6. *main_ros_ornek_gui.py* dosyasına şablon kodun eklenmesi

Bu kodun 9. satırı olan “*self.ui = Ui_MainWindow()*” satırının hata verdiği görülmektedir. Bu hata, “*ros_ornek_gui.py*” kodundan alınan “*Ui_MainWindow*” sınıfının “*main_ros_ornek_gui.py*” kodu için extend(kalıtım) edilmemesinden kaynaklanmaktadır. Kısacası “*ros_ornek_gui.py*” ve “*main_ros_ornek_gui.py*” dosyaları sınıfsal olarak birbirine bağlı değildir. Bunun çözümü için aşağıdaki “*ros_ornek_gui.py*” dosyasının kütüphane kodu “*import*” edilir ve hata giderilir.

Kütüphanenin, “*from paketin_ismi.dosyanin_ismi import Sinifin_ismi*” şeklinde yazıldığına dikkat edilmelidir.

“*from ros_gui_pkg.ros_ornek_gui import Ui_MainWindow*”

```
EXPLORER
src
  ros_gui_pkg
    ros_gui_pkg
      __init__.py
      guiros.py
      main_guiros.py
      main_rovergui_2_0.py
      README.txt
      rovergui.ui
      rovergui_2_0.py
    resource
    test
  package.xml
  README.md

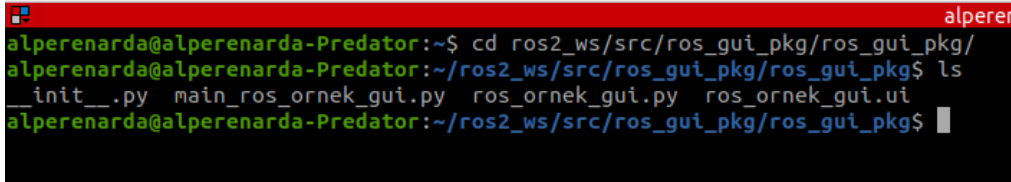
src > ros_gui_pkg > ros_gui_pkg > main_ros_ornek_gui.py > ...
1 import sys
2 from PyQt6.QtWidgets import QMainWindow, QApplication
3
4 from ros_gui_pkg.ros_ornek_gui import Ui_MainWindow
5
6 class MainWindow(QMainWindow):
7     def __init__(self, parent=None):
8         super(MainWindow, self).__init__(parent)
9         self.ui = Ui_MainWindow()
10        self.ui.setupUi(self)
11
12 def main():
13     app = QApplication(sys.argv)
14     win = MainWindow()
15     win.show()
16     sys.exit(app.exec())
17
18 if __name__ == "__main__":
19     main()
```

Şekil 4.7. *ros_ornek_gui.py* dosyasındaki *Ui_Main_Window* sınıfının yeni dosyaya import edilmesi

Oluşturulan .py Uzantılı Dosyaların Çalıştırılması

1. **ADIM:** Bilgisayardaki terminal (veya Terminator uygulaması) açılır. Oluşturulan dosyaları “*çalıştırılabilir (executable)*” yapmak için bu dosyaların bulunduğu yola terminal üzerinden gidilir:

“*cd ros2_ws/src/ros_gui_pkg/ros_gui_pkg/*”



```
alperenarda@alperenarda-Predator:~$ cd ros2_ws/src/ros_gui_pkg/ros_gui_pkg/
alperenarda@alperenarda-Predator:~/ros2_ws/src/ros_gui_pkg/ros_gui_pkg$ ls
__init__.py  main_ros_ornek_gui.py  ros_ornek_gui.py  ros_ornek_gui.ui
alperenarda@alperenarda-Predator:~/ros2_ws/src/ros_gui_pkg/ros_gui_pkg$
```

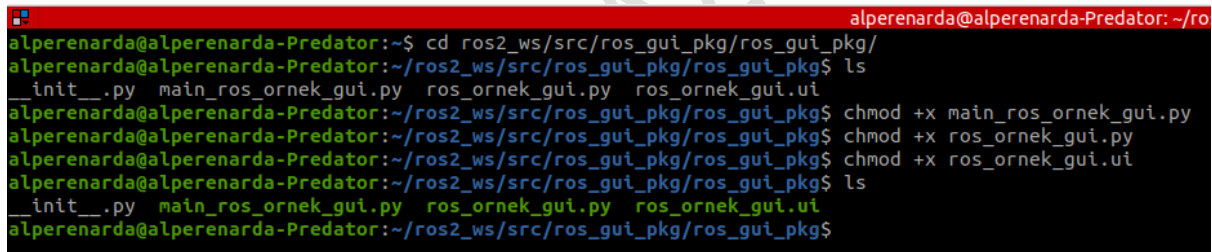
Şekil 5.1. Henüz çalıştırılabilir olmayan beyaz yazı tipine sahip arayüz dosyaları

2. **ADIM:** Bu dosyaların her birine bulunduğu dizinde iken aşağıdaki komut uygulanır:

“*chmod +x main_ros_ornek_gui.py*”

“*chmod +x ros_ornek_gui.py*”

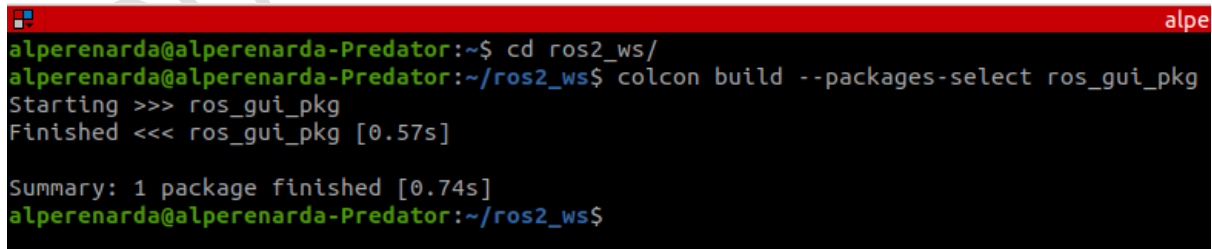
“*chmod +x ros_ornek_gui.ui*”



```
alperenarda@alperenarda-Predator:~$ cd ros2_ws/src/ros_gui_pkg/ros_gui_pkg/
alperenarda@alperenarda-Predator:~/ros2_ws/src/ros_gui_pkg/ros_gui_pkg$ ls
__init__.py  main_ros_ornek_gui.py  ros_ornek_gui.py  ros_ornek_gui.ui
alperenarda@alperenarda-Predator:~/ros2_ws/src/ros_gui_pkg/ros_gui_pkg$ chmod +x main_ros_ornek_gui.py
alperenarda@alperenarda-Predator:~/ros2_ws/src/ros_gui_pkg/ros_gui_pkg$ chmod +x ros_ornek_gui.py
alperenarda@alperenarda-Predator:~/ros2_ws/src/ros_gui_pkg/ros_gui_pkg$ chmod +x ros_ornek_gui.ui
alperenarda@alperenarda-Predator:~/ros2_ws/src/ros_gui_pkg/ros_gui_pkg$ ls
__init__.py  main_ros_ornek_gui.py  ros_ornek_gui.py  ros_ornek_gui.ui
alperenarda@alperenarda-Predator:~/ros2_ws/src/ros_gui_pkg/ros_gui_pkg$
```

Şekil 5.2. Artık çalıştırılabilir olan yeşil yazı tipine sahip arayüz dosyaları

3. **ADIM:** “*ls*” terminal komutu çalıştırıldığında artık bu dosyaların çalıştırılabilir(executable) olduğu görülür. Bu işlemi kaydetmek için “*ros2_ws*” yoluna gidilir ve “*colcon build*” komutu terminalde çalıştırılır.

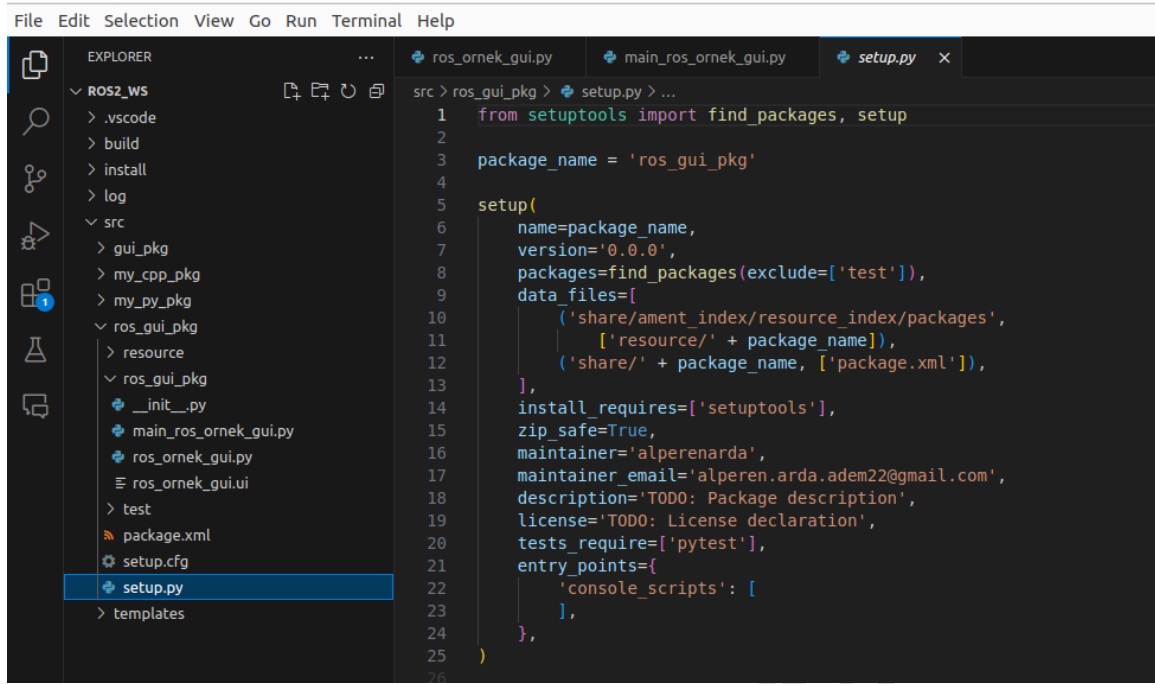


```
alperenarda@alperenarda-Predator:~$ cd ros2_ws/
alperenarda@alperenarda-Predator:~/ros2_ws$ colcon build --packages-select ros_gui_pkg
Starting >>> ros_gui_pkg
Finished <<< ros_gui_pkg [0.57s]

Summary: 1 package finished [0.74s]
alperenarda@alperenarda-Predator:~/ros2_ws$
```

Şekil 5.3. colcon build komutunun çalıştırılıp işlemlerin kaydedilmesi

4. **ADIM:** Visual Studio Code programının sol tarafında bulunan dosyalar sekmesindeki “*setup.py*” dosyasına tıklanır. Tıklandıktan sonra ekrana Şekil 5.4. teki “*setup.py*” kodu gelir. “*setup.py*” dosyası, Python projelerinin paketlenmesi, bağımlılıklarının yönetilmesi ve dağıtılması için kullanılır.



Şekil 5.4. setup.py dosyası

Oluşturduğumuz dosyaların isimleri, “*dosya_adi = paket_adi.fonksiyon_dosyasinin_adi*” formatında “*setup.py*” dosyasındaki “*console_scripts*” bölümüne yazılır.

"ros_ornek_gui = ros_gui_pkg.main_ros_ornek_gui:main",

Bu komut, ros_ornek_gui.py dosyası için tanımlandı.

"main_ros_ornek_gui = ros_gui_pkg.main_ros_ornek_gui:main",

Bu komut, main_ros_ornek_gui.py dosyası için tanımlandı.

Her iki komutun da aynı fonksiyon dosyasına (*ros_gui_pkg.main_ros_ornek_gui*) bağlı olduğunu belirtmek önemlidir. Yani hem “*ros_ornek_gui*” dosyası hem de “*main_ros_ornek_gui*” dosyası, terminalde çalıştırıldığında aynı dosyadaki main fonksiyonunu çalıştıracaktır.

```

entry_points={
    'console_scripts': [
        "main_ros_ornek_gui = ros_gui_pkg.main_ros_ornek_gui:main",
        "ros_ornek_gui = ros_gui_pkg.main_ros_ornek_gui:main",
    ],
}

```

Şekil 5.5. setup.py dosyasına komutların eklenmesi

Bu eklemeler sayesinde, her iki komut da terminalden çalıştırılabilir ve projedeki ilgili Python fonksiyonları erişilebilir hale gelir. “*setup.py*” dosyasının güncel hali Şekil 5.6. görünümündedir:


```

src > ros_gui_pkg > setup.py > ...
1  from setuptools import find_packages, setup
2
3  package_name = 'ros_gui_pkg'
4
5  setup(
6      name=package_name,
7      version='0.0.0',
8      packages=find_packages(exclude=['test']),
9      data_files=[
10         ('share/ament_index/resource_index/packages',
11          ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='alperenarda',
17     maintainer_email='alperen.arda.adem22@gmail.com',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             "main_ros_ornek_gui = ros_gui_pkg.main_ros_ornek_gui:main",
24             "ros_ornek_gui = ros_gui_pkg.main_ros_ornek_gui:main",
25         ],
26     },
27 )
28

```

Şekil 5.6. setup.py dosyası

5. ADIM: Bu adımların ardından arayüz projesi hazır hale gelmektedir. “*setup.py*” dosyasını kaydetmek için klavye üzerinden “*Ctrl + S*” tuş kombinasyonu kullanılır ve Visual Studio Code programı kapatılır. ROS 2 projesini kaydetmek için terminal açılır, “*ros2_ws*” yoluna gelinir ve “*colcon build*” komutu çalıştırılır.

```

alperenarda@alperenarda-Predator:~$ cd ros2_ws/
alperenarda@alperenarda-Predator:~/ros2_ws$ colcon build --packages-select ros_gui_pkg
Starting >>> ros_gui_pkg
Finished <<< ros_gui_pkg [0.57s]

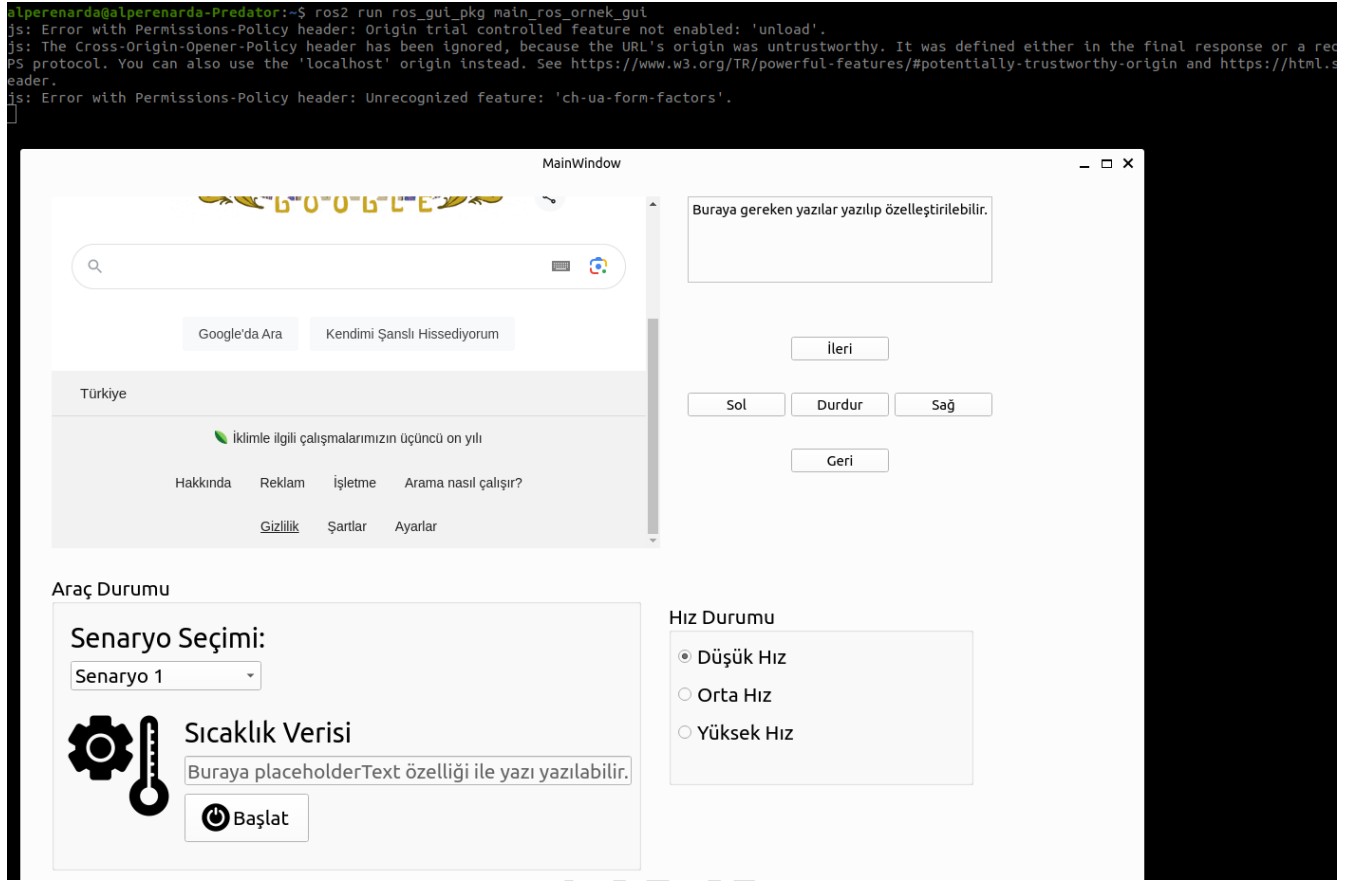
Summary: 1 package finished [0.74s]
alperenarda@alperenarda-Predator:~/ros2_ws$

```

Şekil 5.7. colcon build komutunun çalıştırılıp işlemlerin kaydedilmesi

6. ADIM: Projeyi çalıştırmak için bilgisayardaki terminal (veya Terminator uygulaması) açılır. Aşağıdaki komut çalıştırılır:

“*ros2 run ros_gui_pkg main_ros_ornek_gui*”



Şekil 5.8. Arayüz projesinin çalıştırılması

ROS 2 ile Örnek Veri Yayınlama

Bu adıma kadar, çalışan bir temel arayüz oluşturuldu. Bu başlık altında örnek olması açısından ROS 2 ile veri yayınlama gösterilecektir. Bu örnekte, **“Başlat”** butonuna tıklandığında terminalde(ros topic echo’da) **“Araç Başlatıldı”** veya **“Araç Durduruldu”** mesajının gözükmesi ele alınacaktır.

1. ADIM: Visual Studio Code programı açılır. **“ros_gui_pkg”** paketindeki **“main_ros_ornek_gui.py”** dosyasına tıklanır. **“ros_ornek_gui.py”** dosyasından ziyade **“main_ros_ornek_gui.py”** dosyasının üzerinde işlem yapılma sebebinin **“main_ros_ornek_gui.py”** dosyasının fonksiyonel programlama için oluşturulduğu önceki başlıkta belirtilmişti. Mesaj yayınlanacağı için ve dolayısıyla ROS 2 kullanılacağı için **“rclpy”** kütüphanesi import edilecektir. String tipinde bir mesaj yayınlanacağı için **“std_msgs”** kütüphanesinden **“String”** sınıfı import edilecektir. Son olarak ROS 2 ve PyQt6 bileşenlerinin aynı anda çalışması için QTimer fonksiyonu import edilecektir.

“import rclpy”

“from rclpy.node import Node”

“from std_msgs.msg import String”

“from PyQt6.QtCore import QTimer”


```
src > ros_gui_pkg > ros_gui_pkg > main_ros_ornek_gui.py > ...
1  import sys
2  from PyQt6.QtCore import QTimer
3  from PyQt6.QtWidgets import QMainWindow, QApplication
4
5  from ros_gui_pkg.ros_ornek_gui import Ui_MainWindow
6
7  import rclpy
8  from rclpy.node import Node
9  from std_msgs.msg import String
10
```

Şekil 6.1. ROS 2 iletişimi için rclpy ve String kütüphanelerinin eklenmesi

2. ADIM: Programlama kolaylığı açısından, bu kodda **"sınıf"** yapısı kullanılmaktadır. PyQt ve ROS 2 fonksiyonlarını sınıf içinde kullanabilmek için **"MainWindow"** sınıfı, **"Ui_MainWindow"** ve **"Node"** sınıflarından kalıtım (extend) almaktadır. Aksi takdirde, bu sınıfta hem ROS 2 hem de PyQt özelliklerini kullanmak karmaşık hale gelebilir ve hatalara yol açabilir.

```
class MainWindow(QMainWindow, Ui_MainWindow, Node):
    def __init__(self, parent=None):
        QMainWindow.__init__(self, parent)
        Node.__init__(self, 'main_window_node')
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
```

Şekil 6.2. Ui_MainWindow ve Node sınıflarının extend edilmesi

Burada **"__init__"** fonksiyonunun olduğuna dikkat edilmelidir ve program başlatılırken kullanılacak fonksiyonların **"__init__"** fonksiyonunun içinde çağırılması gerektiğine dikkat edilmelidir. Ek olarak **"Node"** nesnesinin çalışması için **"__init__"** fonksiyonuna **"Node.__init__(self, 'main_window_node')"** satırı eklenmelidir.

3. ADIM: Butona ilk tıklandığında **"Araç Başlatıldı"** mesajını almak için ve ikinci kez tıklandığında **"Araç Durduruldu"** mesajını almak için **"main_ros_ornek_gui.py"** dosyasında bir adet **"publisher"** oluşturulacaktır. **"publisher"**, ROS 2 yazılım platformunda mesaj yayınlamak için kullanılan bir fonksiyondur. ROS 2 ve Python fonksiyonlarının kullanımının bilindiği varsayılarak aşağıdaki fonksiyon **"main_ros_ornek_gui.py"** dosyasında bulunan **"MainWindow"** sınıfının içine yazılır:

```
"self.get_logger().info("ROS initialized.")
```

```
self.connect_ros()
```

```
def connect_ros(self):
```

```
self.start_vehicle = self.create_publisher(String, 'start', 10)"
```

Burada ROS 2 için bir “*connect_ros*” fonksiyonu oluşturulur ve bu fonksiyonunun içinde de “*publisher*” başlatılır. Program başladığında “*connect_ros*” fonksiyonunun da başlatılması için “*__init__*” fonksiyonu içerisine “*self.connect_ros()*” satırı eklenir.

```
11 class MainWindow(QMainWindow, Ui_MainWindow, Node):
12     def __init__(self, parent=None):
13         QMainWindow.__init__(self, parent)
14         Node.__init__(self, 'main_window_node')
15         self.ui = Ui_MainWindow()
16         self.ui.setupUi(self)
17
18         self.get_logger().info("ROS initialized.")
19         self.connect_ros()
20
21     def connect_ros(self):
22         self.start_vehicle = self.create_publisher(String, 'start', 10)
```

Şekil 6.3. Mesaj yollamak için publisher eklenmesi

4. ADIM: Bu adımda, butona tıklama ile araç başlatma veya durdurma işlemi gerçekleştirilecektir. Bunun devamlı değil, sadece butona tıklama ile tetiklenen bir işlem olmasını sağlamak için “*start_vehicle_button*” adlı bir fonksiyon oluşturulur. Ayrıca, bu fonksiyonu çağırmak için “*__init__*” fonksiyonu içinde “*clicked*” sinyali bağlanır. Durum değişikliklerini takip etmek amacıyla “*self.i = 0*” şeklinde bir sayaç tanımlanır. Bu sayaç, her tıklamada değiştirilen duruma göre aracın başlatılması veya durdurulmasını kontrol eder.

“*self.i = 0*”

```
self.ui.pushButton.clicked.connect(self.start_vehicle_button)
```

```
def connect_ros(self):
```

```
self.start_vehicle = self.create_publisher(String, 'start', 10)
```

```
def start_vehicle_button(self):
```

```
msg = String()
```

```
if self.i % 2 == 0:
```

```
    msg.data = "Araç Başlatıldı."
```

```
    self.start_vehicle.publish(msg)
```

```
else:
```

```
    msg.data = "Araç Durduruldu."
```

```
    self.start_vehicle.publish(msg)
```

```
self.i += 1
```

```

11 class MainWindow(QMainWindow, Ui_MainWindow, Node):
12     def __init__(self, parent=None):
13         QMainWindow.__init__(self, parent)
14         Node.__init__(self, 'main_window_node')
15         self.ui = Ui_MainWindow()
16         self.ui.setupUi(self)
17
18         self.get_logger().info("ROS initialized.")
19         self.connect_ros()
20         self.i = 0
21         self.ui.pushButton.clicked.connect(self.start_vehicle_button)
22
23     def connect_ros(self):
24         self.start_vehicle = self.create_publisher(String, 'start', 10)
25
26     def start_vehicle_button(self):
27         msg = String()
28         if self.i % 2 == 0:
29             msg.data = "Araç Başlatıldı."
30             self.start_vehicle.publish(msg)
31         else:
32             msg.data = "Araç Durduruldu."
33             self.start_vehicle.publish(msg)
34         self.i += 1

```

Şekil 6.4. start_vehicle_button fonksiyonunun eklenmesi ve clicked yöntemi ile bağlanması

5. ADIM: Çalıştırma fonksiyonu olan “**main**” fonksiyonunda ROS 2 “**Node**” sınıfını başlatmak için aşağıdaki değişiklikler yapılır:

“def main():

 app = QApplication(sys.argv)

 rclpy.init(args=None)

 win = MainWindow()

 win.show()

 def spin_ros():

 try:

 rclpy.spin_once(win, timeout_sec=0.01)

 except rclpy.exceptions.ROSInterruptException:

 pass

 timer = QTimer()

 timer.timeout.connect(spin_ros)

 timer.start(10)

app.exec()

rclpy.shutdown()”

“*QTimer*”; ROS 2 ve PyQt6'nın birlikte sorunsuz çalışabilmesi için, belirli aralıklarla “*rclpy.spin_once*” fonksiyonunu çağırarak ROS 2 düğümlerinin yanıt vermesini sağlar.

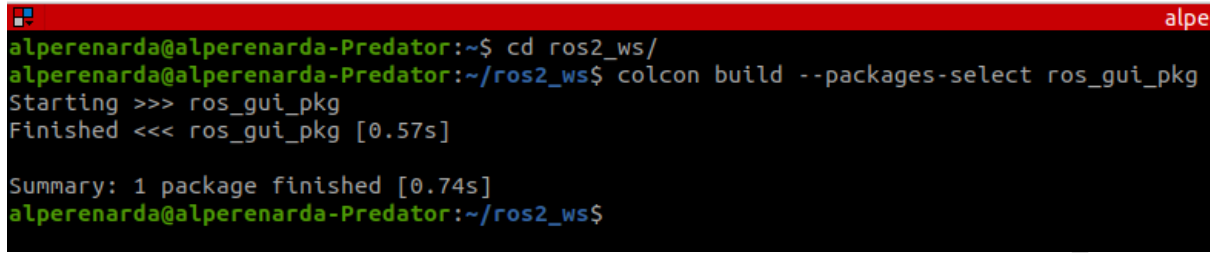
```
37 def main():
38     app = QApplication(sys.argv)
39     rclpy.init(args=None)
40     win = MainWindow()
41     win.show()
42
43     def spin_ros():
44         try:
45             rclpy.spin_once(win, timeout_sec=0.01)
46         except rclpy.exceptions.ROSInterruptException:
47             pass
48
49     timer = QTimer()
50     timer.timeout.connect(spin_ros)
51     timer.start(10)
52     app.exec()
53     rclpy.shutdown()
54
55 if __name__ == "__main__":
56     main()
57
```

Şekil 6.5. main fonksiyonu

```
7 import rclpy
8 from rclpy.node import Node
9 from std_msgs.msg import String
10
11 class MainWindow(QMainWindow, Ui_MainWindow, Node):
12     def __init__(self, parent=None):
13         QMainWindow.__init__(self, parent)
14         Node.__init__(self, 'main_window_node')
15         self.ui = Ui_MainWindow()
16         self.ui.setupUi(self)
17
18         self.get_logger().info("ROS initialized.")
19         self.connect_ros()
20         self.i = 0
21         self.ui.pushButton.clicked.connect(self.start_vehicle_button)
22
23     def connect_ros(self):
24         self.start_vehicle = self.create_publisher(String, 'start', 10)
25
26     def start_vehicle_button(self):
27         msg = String()
28         if self.i % 2 == 0:
29             msg.data = "Araç Başlatıldı."
30             self.start_vehicle.publish(msg)
31         else:
32             msg.data = "Araç Durduruldu."
33             self.start_vehicle.publish(msg)
34         self.i += 1
35
36
37 def main():
38     app = QApplication(sys.argv)
39     rclpy.init(args=None)
40     win = MainWindow()
41     win.show()
42
43     def spin_ros():
44         try:
45             rclpy.spin_once(win, timeout_sec=0.01)
46         except rclpy.exceptions.ROSInterruptException:
47             pass
48
49     timer = QTimer()
50     timer.timeout.connect(spin_ros)
51     timer.start(10)
52     app.exec()
53     rclpy.shutdown()
```

Şekil 6.6. main_ros_ornek_gui.py

6. ADIM: Bu deęişiklikleri yaptıktan sonra “**Ctrl + S**” tuş kombinasyonu kullanılır ve dosya kaydedilir. Visual Studio Code programı kapatılır. Terminal (veya Terminator uygulaması) açılır ve “**ros2_ws**” yoluna gelindikten sonra “**colcon build**” komutu çalıştırılır.



```
alperenarda@alperenarda-Predator:~$ cd ros2_ws/
alperenarda@alperenarda-Predator:~/ros2_ws$ colcon build --packages-select ros_gui_pkg
Starting >>> ros_gui_pkg
Finished <<< ros_gui_pkg [0.57s]

Summary: 1 package finished [0.74s]
alperenarda@alperenarda-Predator:~/ros2_ws$
```

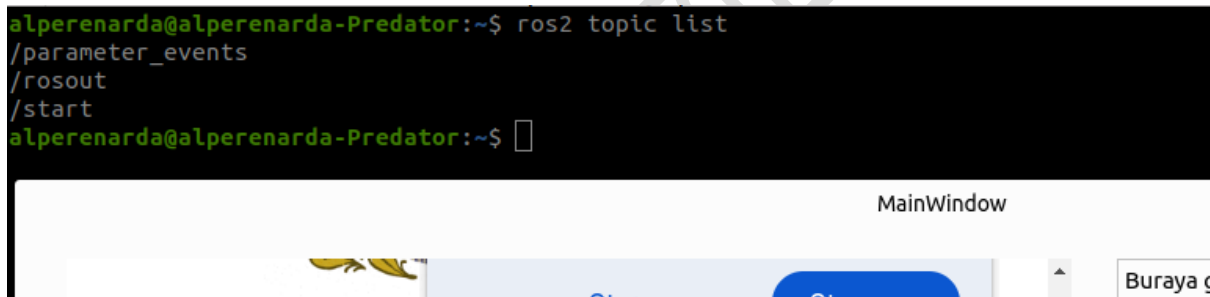
Şekil 6.7. colcon build komutunun çalıştırılıp işlemlerin kaydedilmesi

7. ADIM: Projeyi çalıştırmak için bilgisayardaki terminal (veya Terminator uygulaması) açılır. Aşağıdaki komut çalıştırılır:

“ros2 run ros_gui_pkg main_ros_ornek_gui”

8. ADIM: Projeyi çalıştırdıktan sonra sahip olduğumuz publisher’ın yayınladığı topic’i görebilmek için terminalde (veya Terminator uygulamasında) aşağıdaki komut çalıştırılır:

“ros2 topic list”



```
alperenarda@alperenarda-Predator:~$ ros2 topic list
/parameter_events
/rosout
/start
alperenarda@alperenarda-Predator:~$
```

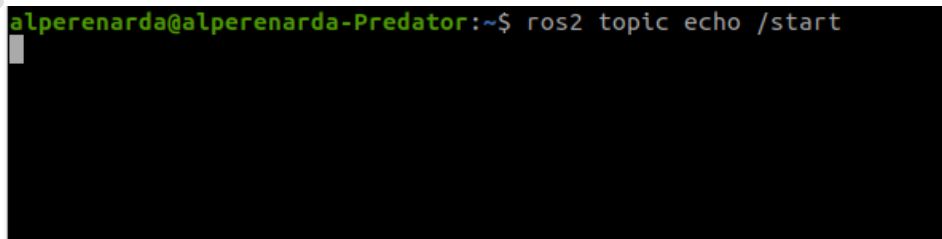
The screenshot also shows a GUI window titled 'MainWindow' with a button labeled 'Buraya ç'.

Şekil 6.8. topic listesinde /start topic’inin görünümü

Şekil 6.8. de görüldüğü üzere arayüz çalıştırılınca “**/start**” topic’i terminalde görünmektedir. Bu topic Başlat butonuna basıldığında gerekli veriyi yayınlayacaktır.

9. ADIM: Oluşturulan publisher tarafından yayınlanan veriyi görmek için terminalde (veya Terminator uygulamasında) aşağıdaki komut çalıştırılır. Bu komutta, daha önceden belirlediğimiz “**/start**” topic’inin isminin kullanıldığına dikkat edilmelidir:

“ros2 topic echo /start”



```
alperenarda@alperenarda-Predator:~$ ros2 topic echo /start
```

Şekil 6.9. /start topic’inin görünümü

Şekil 6.9. da görüldüğü üzere şu anda “publisher” bir şey yayınlamıyor. Başlat tuşuna bir kez tıkladığımızda Şekil 6.10. daki mesaj karşımıza çıkmaktadır.

```
alperenarda@alperenarda-Predator:~$ ros2 topic echo /start
data: Araç Başlatıldı.
---
```

Şekil 6.10. Başlat tuşuna basıldığında publisher tarafından yayınlanan mesaj

```
alperenarda@alperenarda-Predator:~$ ros2 topic echo /start
data: Araç Başlatıldı.
---
data: Araç Durduruldu.
---
```

Şekil 6.11. Başlat tuşuna ikinci kez basıldığında publisher tarafından yayınlanan mesaj

Bu örnekte, ROS 2 ve PyQt6 kullanarak bir arayüzde buton tıklama ile araç başlatma ve durdurma işlemleri başarıyla gerçekleştirildi. Uygulamanın sonunda, terminalde yayınlanan mesajlar üzerinden bu işlemlerin doğru bir şekilde çalıştığı gözlemlenmektedir.

Haritalandırma, Resim Ekleme ve Diğer Özelleştirmeler

Bu başlık altında “*QWebView*” ile harita eklemenin nasıl yapılacağı ve oluşturduğumuz “*ros_gui_pkg*” paketinin başka bilgisayarlara taşınacağı zaman resimlerin nasıl ekleneceği gösterilecektir.

1. ADIM: Visual Studio Code programı açılır. Sol tarafta bulunan dosyalar sekmesindeki “*ros_ornek_gui.py*” dosyasına tıklanır. Şekil 7.2 deki kod ekrana gelir.

2. ADIM: 120. satırda yer alan “*QWebView*” nesnesine atanan URL, başlangıçta “*https://www.google.com*” olarak belirlenmişti. LIDAR sensöründen alınan verilerin yerel ağ üzerinden yayımlandığını varsayarak bu URL’yi dinamik olarak bulunduğunuz ağa uygun şekilde değiştirmek daha uygun olacaktır. Bu doğrultuda, Flask ile “*http://127.0.0.1:5000/map*” adresinden yayın yapan bir web uygulamasını kullanarak, “*QWebView’e*” bu yeni linki atayabilirsiniz. Bu adres, “*http://127.0.0.1:5000/map*”, mevcut ağ ve Flask uygulamasına özeldir; adres farklılık gösterebilir.

```
117 self.gridLayout.addWidget(self.pushButton_8, 1, 0, 1, 1)
118 self.webView = QWebView(parent=self.centralwidget)
119 self.webView.setGeometry(QtCore.QRect(30, 20, 641, 371))
120 self.webView.setUrl(QtCore.QUrl["http://127.0.0.1:5000/map"])
121 self.webView.setObjectName("webView")
122 MainWindow.setCentralWidget(self.centralwidget)
```

Şekil 7.1. webView nesnesinin URL'sini değiştirme

```

1  # Form implementation generated from reading ui file 'ros2_ws/src/ros_gui_pkg/ros_gui_pkg/ros_ornek_gui.ui'
2  #
3  # Created by: PyQt6 UI code generator 6.6.1
4  #
5  # WARNING: Any manual changes made to this file will be lost when pyuic6 is
6  # run again. Do not edit this file unless you know what you are doing.
7
8
9  from PyQt6 import QtCore, QtGui, QtWidgets
10 from PyQt6.QtWebEngineWidgets import QWebEngineView
11
12
13 class Ui_MainWindow(object):
14     def setupUi(self, MainWindow):
15         MainWindow.setObjectName("MainWindow")
16         MainWindow.resize(1178, 802)
17         self.centralwidget = QtWidgets.QWidget(parent=MainWindow)
18         self.centralwidget.setObjectName("centralwidget")
19         self.groupBox = QtWidgets.QGroupBox(parent=self.centralwidget)
20         self.groupBox.setGeometry(QtCore.QRect(30, 420, 621, 311))
21         font = QtGui.QFont()
22         font.setPointSize(16)
23         self.groupBox.setFont(font)
24         self.groupBox.setObjectName("groupBox")
25         self.pushButton = QtWidgets.QPushButton(parent=self.groupBox)
26         self.pushButton.setGeometry(QtCore.QRect(140, 230, 131, 51))
27         font = QtGui.QFont()
28         font.setPointSize(16)
29         self.pushButton.setFont(font)
30         self.pushButton.setLayoutDirection(QtCore.Qt.LayoutDirection.LeftToRight)
31         icon = QtGui.QIcon()
32         icon.addPixmap(QtGui.QPixmap("ros2_ws/src/ros_gui_pkg/ros_gui_pkg/../../Desktop/images/power-button.png"),
33         self.pushButton.setIcon(icon)
34         self.pushButton.setIconSize(QtCore.QSize(32, 32))
35         self.pushButton.setObjectName("pushButton")
36         self.label = QtWidgets.QLabel(parent=self.groupBox)
37         self.label.setGeometry(QtCore.QRect(140, 140, 181, 51))
38         font = QtGui.QFont()
39         font.setPointSize(22)
40         self.label.setFont(font)
41         self.label.setObjectName("label")
42         self.lineEdit = QtWidgets.QLineEdit(parent=self.groupBox)
43         self.lineEdit.setGeometry(QtCore.QRect(140, 190, 471, 31))

```

Şekil 7.2. *ros_ornek_gui.py*

3. ADIM: Bu değişiklikleri yaptıktan sonra **“Ctrl + S”** tuş kombinasyonu kullanılır ve dosya kaydedilir. Visual Studio Code programı kapatılır. Terminal (veya Terminator uygulaması) açılır ve **“ros2_ws”** yoluna gelindikten sonra **“colcon build”** komutu çalıştırılır.

```

alperenarda@alperenarda-Predator:~$ cd ros2_ws/
alperenarda@alperenarda-Predator:~/ros2_ws$ colcon build --packages-select ros_gui_pkg
Starting >>> ros_gui_pkg
Finished <<< ros_gui_pkg [0.57s]

Summary: 1 package finished [0.74s]
alperenarda@alperenarda-Predator:~/ros2_ws$

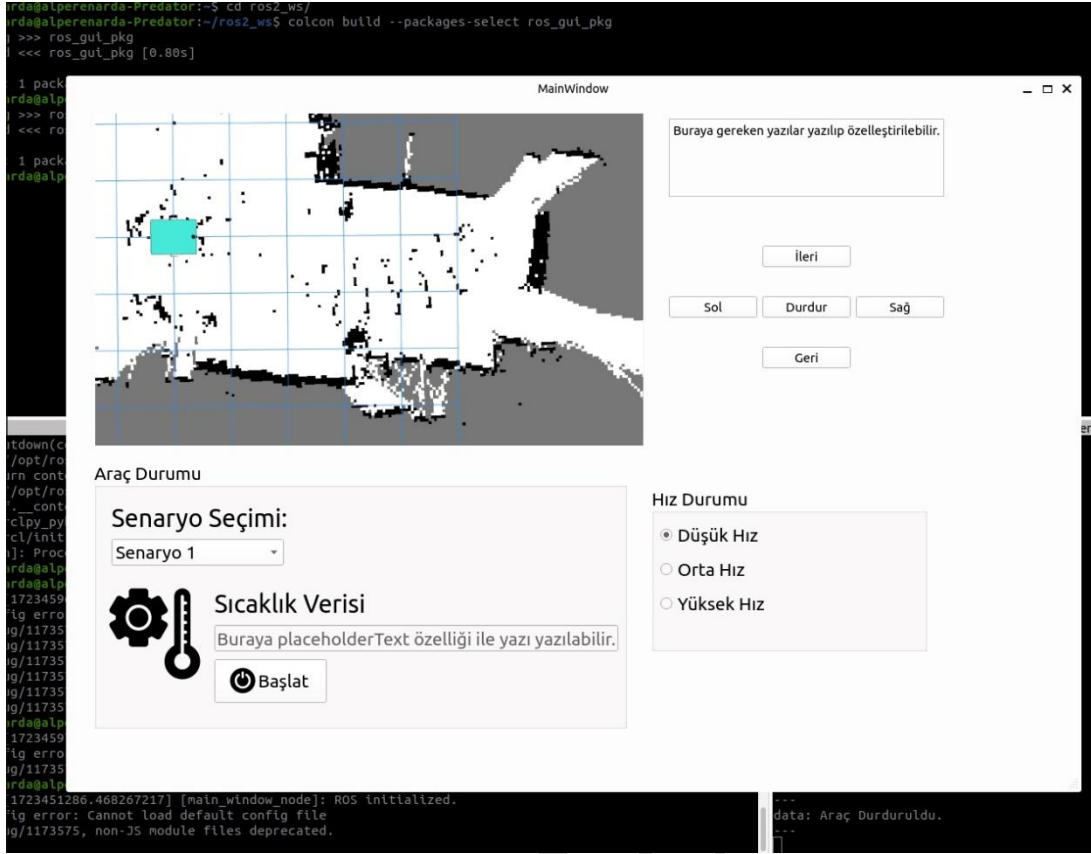
```

Şekil 7.3. *colcon build* komutunun çalıştırılıp işlemlerin kaydedilmesi

4. ADIM: Projeyi çalıştırmak için bilgisayardaki terminal (veya Terminator uygulaması) açılır. Aşağıdaki komut çalıştırılır:

“ros2 run ros_gui_pkg main_ros_ornek_gui”

Şayet LIDAR sensörü açıksa ve Robot ile stabil bir bağlantı sağlanırsa Şekil 7.4. teki harita ekrana gelecektir.



Şekil 7.4. QWebView ile haritalandırma

Sonuç olarak, yapılan adımlar sonunda LIDAR sensöründen alınan veriler, QWebView bileşeni aracılığıyla yerel ağda yayınlanan harita üzerinde başarıyla görüntülenebilir.

5. ADIM: Önceki başlıklarda resim eklenmişti fakat bu resimler projenin paylaşıldığı diğer bilgisayarlar da gözükmeyebilir. Bunun çözümü için “*ros_ornek_gui.py*” ve “*main_ros_ornek_gui.py*” dosyalarına “*os*” kütüphanesi import edilir ve resimlerin yolunu belirlemeye yarayan “*current_dir*” değişkeni eklenir:

“import os”

“current_dir = os.path.dirname(os.path.abspath(__file__))”

Bu satır “*kalıcı*” resimlerin hangi yolda olması gerektiğini terminale yazar.

```

9  from PyQt6 import QtCore, QtGui, QtWidgets
10 from PyQt6.QtWebEngineWidgets import QWebEngineView
11
12 import os
13 current_dir = os.path.dirname(os.path.abspath(__file__))
14

```

Şekil 7.5. os kütüphanesinin import edilmesi ve current_dir değişkeni (ros_ornek_gui.py)


```

1  import sys
2  from PyQt6.QtCore import QTimer
3  from PyQt6.QtWidgets import QMainWindow, QApplication
4
5  import os
6  current_dir = os.path.dirname(os.path.abspath(__file__))
7
8  from ros_gui_pkg.ros_ornek_gui import Ui_MainWindow
9

```

Şekil 7.6. os kütüphanesinin import edilmesi ve current_dir değişkeni (main_ros_ornek_gui.py)

“main_ros_ornek_gui.py” dosyasındaki “main()” fonksiyonuna ise “current_dir” değişkeninin terminale bastırılması için “print(current_dir)” satırı eklenir:

```

40  def main():
41      app = QApplication(sys.argv)
42      rclpy.init(args=None)
43      win = MainWindow()
44      win.show()
45      print(current_dir)
46
47  def spin_ros():

```

Şekil 7.7. print(current_dir)

6. ADIM: Bu değişiklikleri yaptıktan sonra “Ctrl + S” tuş kombinasyonu kullanılır ve dosya kaydedilir. Visual Studio Code programı kapatılır. Terminal (veya Terminator uygulaması) açılır ve “ros2_ws” yoluna gelindikten sonra “colcon build” komutu çalıştırılır.

```

alperenarda@alperenarda-Predator:~$ cd ros2_ws/
alperenarda@alperenarda-Predator:~/ros2_ws$ colcon build --packages-select ros_gui_pkg
Starting >>> ros_gui_pkg
Finished <<< ros_gui_pkg [0.57s]

Summary: 1 package finished [0.74s]
alperenarda@alperenarda-Predator:~/ros2_ws$

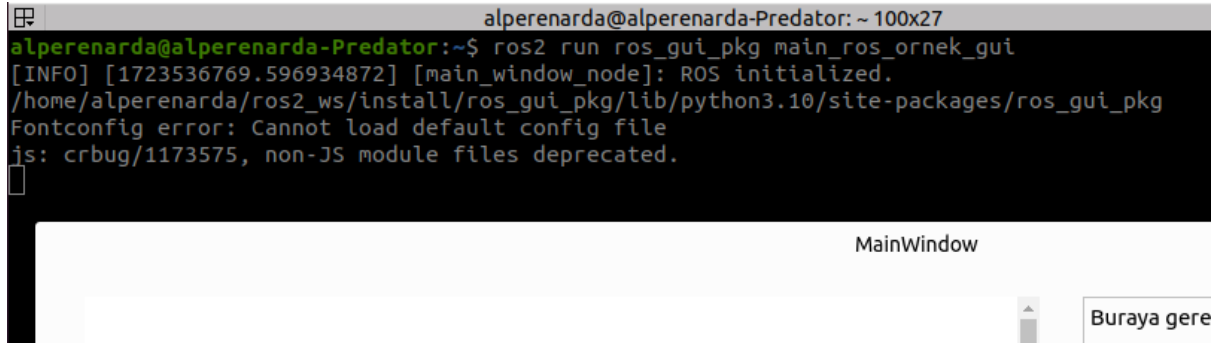
```

Şekil 7.8. colcon build komutunun çalıştırılıp işlemlerin kaydedilmesi

7. ADIM: Projeyi çalıştırmak için bilgisayardaki terminal (veya Terminator uygulaması) açılır. Aşağıdaki komut çalıştırılır:

“ros2 run ros_gui_pkg main_ros_ornek_gui”

Arayüz başlatıldığında terminalde “current_dir” satırı karşımıza çıkar.



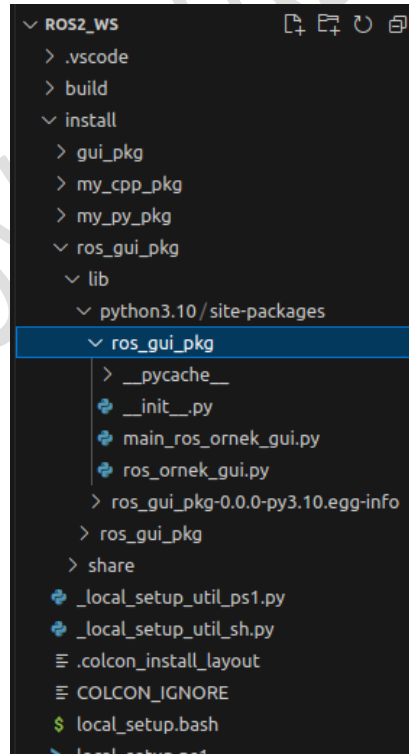
Şekil 7.9. *current_dir* yolunun terminalde görünümü

Sonuç olarak, Resim dosyalarının

"/home/alperenarda/ros2_ws/install/ros_gui_pkg/lib/python3.10/site-packages/ros_gui_pkg" yoluna kaydedilmesi gerekmektedir. Bu yol, uygulamanın çalışması sırasında resim dosyalarına sabit bir konumdan erişebilmesi için gereklidir. Bu nedenle bu dizinde bir ***"images"*** klasörü oluşturulmalı ve tüm resimler buraya kaydedilmelidir.

8. ADIM: Visual Studio Code programı açılır. Visual Studio Code üzerinden

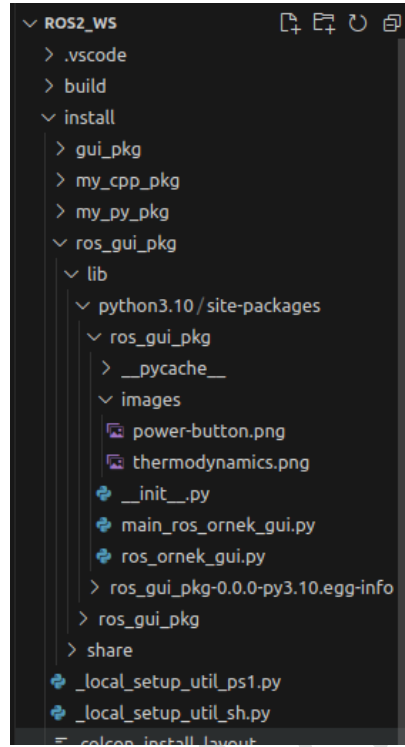
"/home/alperenarda/ros2_ws/install/ros_gui_pkg/lib/python3.10/site-packages/ros_gui_pkg" yoluna gidilir. Bu yola ulaşıldığında, Visual Studio Code'un sol tarafındaki pencerede Şekil 7.10. daki ekran görüntüsü karşımıza çıkar.



Şekil 7.10. *current_dir* yolunun Visual Code Studio üzerinden görünümü

Bu yolun üstünde ***"images"*** klasörü açmak için ***"ros_gui_pkg"*** dosyasının üstüne sağ tıklanır. ***"New Folder"*** seçeneğine tıklanır. İsim olarak ***"images"*** yazılır ve klavyedeki Enter

tuşuna basılır. Eklediğimiz ve bundan sonra ekleyeceğimiz resimler bu yola eklenmelidir. Aksi takdirde proje paylaşılrken resimler gözükmebilir.



Şekil 7.11. Resimlerin `current_dir` yoluna eklenmiş hali

9. ADIM: Resim yollarını kod üzerinden değiştirmek için "`ros_ornek_gui.py`" dosyasına gidilir. Bu dosyada, resim yollarını içeren satırlar bulunur. Örneğin, 32. satırda yer alan:

`icon.addPixmap(QtGui.QPixmap("ros2_ws/src/ros_gui_pkg/ros_gui_pkg/../../Desktop/images/power-button.png"), QtGui.QIcon.Mode.Normal, QtGui.QIcon.State.Off)` satırı, aşağıdaki gibi değiştirilir:

`icon.addPixmap(QtGui.QPixmap(current_dir + "/images/power-button.png"), QtGui.QIcon.Mode.Normal, QtGui.QIcon.State.Off)`

Bu değişiklikle, resim yolu dinamik hale getirilir ve ilgili resmin doğru şekilde yüklenmesi sağlanır. Diğer tüm resimlere de aynı işlem uygulanmalıdır.

```
icon = QtGui.QIcon()
icon.addPixmap(QtGui.QPixmap(current_dir + "/images/power-button.png"), QtGui.QIcon.Mode.Normal, QtGui.QIcon.State.Off)
self.pushButton.setIcon(icon)
self.pushButton.setIconSize(QtCore.QSize(32, 32))
```

Şekil 7.12. `current_dir` değişkenini kullanarak resim yollarını konfigüre etme

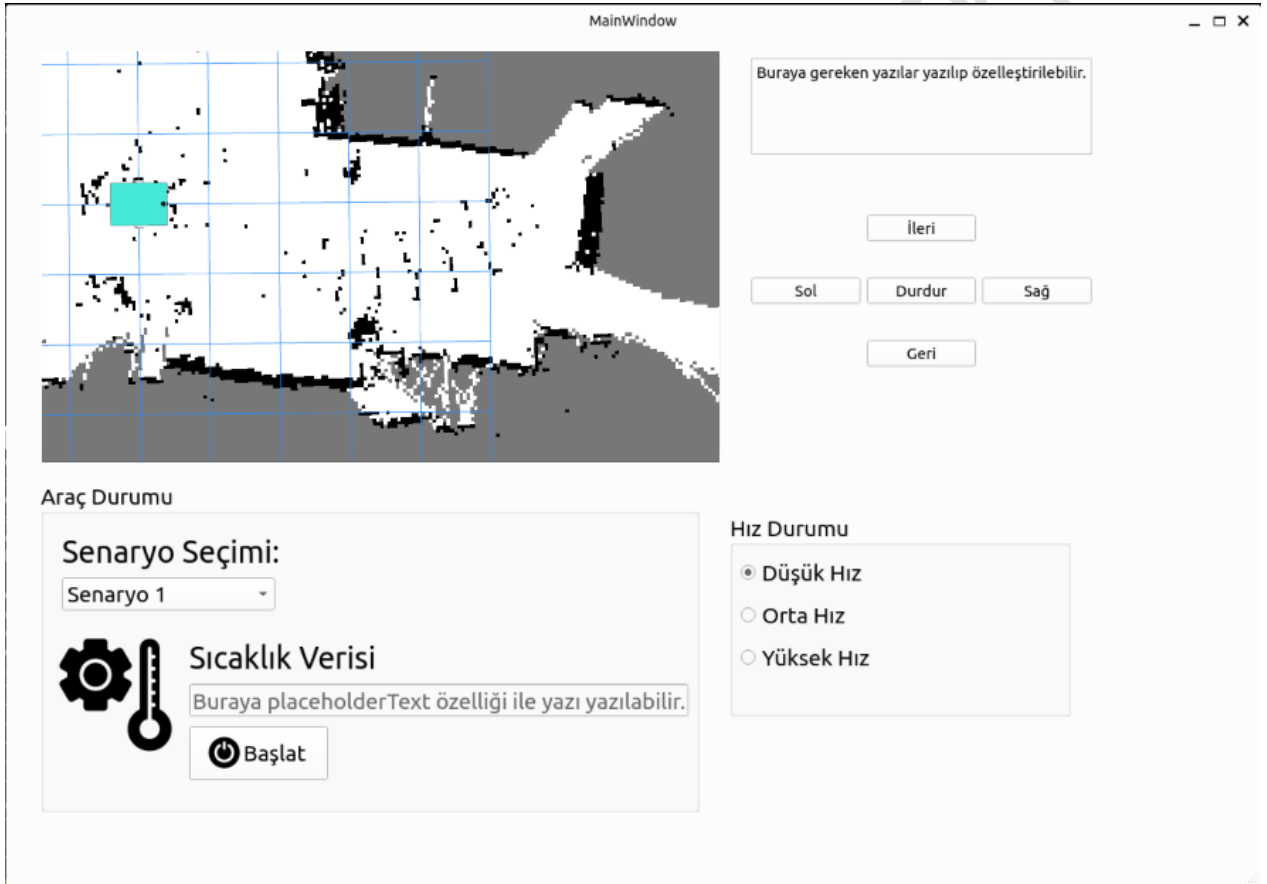
10. ADIM: Bu değişiklikleri yaptıktan sonra **“Ctrl + S”** tuş kombinasyonu kullanılır ve dosya kaydedilir. Visual Studio Code programı kapatılır. Terminal (veya Terminator uygulaması) açılır ve **“ros2_ws”** yoluna gelindikten sonra **“colcon build”** komutu çalıştırılır.

```
alperenarda@alperenarda-Predator:~$ cd ros2_ws/  
alperenarda@alperenarda-Predator:~/ros2_ws$ colcon build --packages-select ros_gui_pkg  
Starting >>> ros_gui_pkg  
Finished <<< ros_gui_pkg [0.57s]  
  
Summary: 1 package finished [0.74s]  
alperenarda@alperenarda-Predator:~/ros2_ws$
```

Şekil 7.13. colcon build komutunun çalıştırılıp işlemlerin kaydedilmesi

11. ADIM: Projeyi çalıştırmak için bilgisayardaki terminal (veya Terminator uygulaması) açılır. Aşağıdaki komut çalıştırılır:

“ros2 run ros_gui_pkg main_ros_ornek_gui”



Şekil 7.14. Bu kaynak için arayüzün son görüntüsü

SONUÇ

Bu projede, PyQt 6 kullanılarak ROS 2 ile entegre bir arayüz geliştirilmiştir. Qt 5 Designer ile oluşturulan arayüz, ROS 2 paketine kaydedilmiş ve Python ile özelleştirilebilir hale getirilmiştir. Sonuç olarak bu entegrasyon, robotik projelerde kullanıcı arayüzü oluşturmayı kolaylaştırıcı niteliktedir.