4.3.1 Einleitung

Das Franka Control Interface (FCI) ermöglicht eine schnelle und direkte Low-Level-Verbindung mit dem Franka Emika Robotern und dessen Greifern. Das FCI liefert den aktuellen Status des Roboters und ermöglicht seine direkte Steuerung mit einem externen, über Ethernet angeschlossenen Workstation-PC. Durch die Verwendung von *libfranka*, einer Open-Source-C++-Schnittstelle, können Echtzeit-Steuerwerte mit 1 kHz über verschiedene Schnittstellen gesendet werden. Darüber hinaus verbindet *franka_ros* die Franka Emika Forschungsroboter mit dem gesamten ROS-Ökosystem. Es integriert *libfranka* in ROS Control.
Da die direkte Steuerung der Franka Roboter über das FCI recht komplex ist und die Nutzung für typische Robotik Anwenderaufgaben (z.B. Pick-Place Aufgaben) einen erheblichen Mehraufwand erfordert, wurde im Rahmen des Kompetenzzentrums das RF Robot Control Framework entwickelt. Dabei handelt es sich um ein Programmier-Framework, welches *franka_ros* und *ros_control* integriert und eine grafische Benutzeroberfläche bereitstellt, um zusätzliche nützliche Funktionen anzubieten, wie z. B. das Einlernen von Aufgabenpunkten und Gelenkkonfigurationen, einfaches Starten von Programmen, Zurücksetzen von Fehlern und viele weitere. Die Komponenten sind, wie bei ROS üblich, für die Ausführung auf mehreren Maschinen ausgelegt. Bei diesem Setup wird erwartet, dass *franka_ros* und *ros_control* auf einem Steuerungs-PC mit Echtzeit-Kernel laufen, der direkt mit dem Franka-Master-Controller verbunden ist. Das Framework ist so konzipiert, dass es die Entwicklung entweder direkt auf dem Steuerrechner oder auf separaten Entwicklerrechnern erlaubt, die über Ethernet mit dem Steuerrechner verbunden sind und keine installierte Echtzeit haben müssen.

Das Framework unterstützt derzeit drei Arten von Bewegungen, die als Klassen definiert sind
- MoveCartesian (RFLIN): Erzeugt lineare Bewegungen von einem kartesischen Punkt zu einem anderen.
- MoveJoint (RFPTP): Erzeugt Bewegungen direkt von einem Punkt zu einem anderen im Joint-Raum
- MoveToContact (RFMTC): Erzeugt lineare Bewegungen in eine bestimmte Richtung, bis eine entgegengesetzte Kraft überschritten wird.

Innerhalb dieser Klassen gibt es eine Reihe von Funktionen, die mit der gewünschten Bewegungsart ausgeführt werden können. So kann sich der Roboter z.B. zu einem gewünschten Punkt bewegen, oder mit einer Relativbewegung in eine bestimmte Richtung von einem Punkt aus. Mehr über die Klassen und Funktionen finden Sie in der Doxygen-Dokumentation.
Dieses Framework unterstützt derzeit den Franka Emika Panda Roboter, eine Simulation mit dem CoppeliaSim Simulator und bietet begrenzte Unterstützung für Universal Robots. Beachten Sie jedoch, dass sich die folgende Dokumentation auf die Verwendung mit einem Franka Emika-Roboter konzentriert.

4.3.2 Übersicht

Das Framework wird in einem Github Repository bereitgestellt und ist unter folgendem Link erreichbar:
https://gitlab.com/roboterfabrik1/rf_robot_framework
Um die Installation zu erleichtern wird das Framework in einem Docker Container bereitgestellt. Dabei handelt es sich um eine isolierte Umgebung, die eine Anwendung und alle ihre Abhängigkeiten enthält, die für deren Ausführung benötigt werden. Container sind somit ähnlich wie virtuelle Maschinen, aber im Gegensatz zu virtuellen Maschinen teilen sie sich den Kernel des Host-Betriebssystems, was sie effizienter und ressourcenschonender macht.

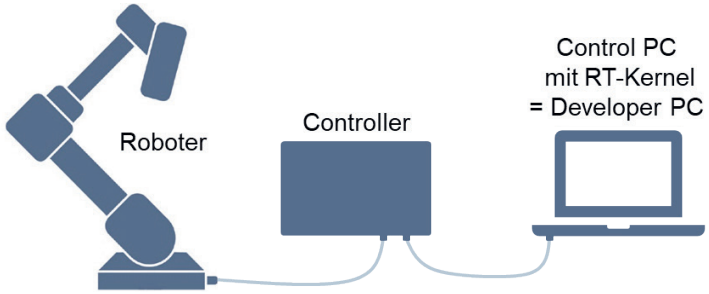
Das Repository beinhaltet:
- ein kompiliertes Docker Image mit passenden Abhängigkeiten für Franka Roboter mit der Desk Version 4.2.1. Es beinhaltet: Ubuntu 20.04, ROS Noetic, libfranka Version: 0.9.0 (Version vom 28.03.2022), franka_ros Version: 0.9.0 (Version vom 24.08.2022), Simulationsumgebung: CoppeliaSim Education Version 4.3.0 Rev 12, Universal Robot ROS Driver Version von https://github.com/UniversalRobots/Universal_Robots_ROS_Driver (Version vom 02.09.2022)
- das dazugehörige Dockerfile um das Image selbst zu kompilieren (→ so können eigene Programme hinzugefügt werden oder falls notwendig die Abhängigkeiten für neue/alte Franka Desk-Versionen angepasst werden)
- Skripte um das Framework zu installieren inklusive: aller benötigten Abhängigkeiten, Docker, VS Code mit Erweiterungen

Verbindungsmöglichkeiten mit dem Franka Roboter

Leider ist für die Nutzung der FCI Schnittstelle und damit auch für die Nutzung des RF Robot Frameworks die Installation eines Real-Time Kernels nötig. Die Installation eines Real-Time Kernels ist recht aufwendig und dauert vergleichsweise lang (je nach PC ca. 1-2 Stunden, siehe auch Kapitel „Installation Real-Time Kernel“). Deshalb wurde das RF Framework so konzipiert, dass es mit verschiedene Verbindungsmöglichkeiten mit dem Franka Roboter zur Verfügung stellt.

Folgende Verbindungsmöglichkeiten gibt es:

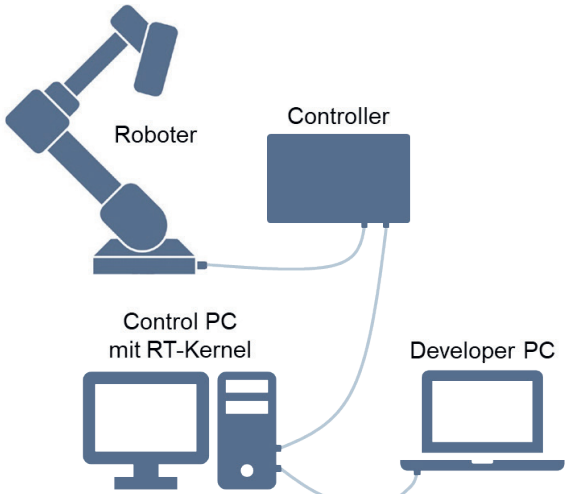
Setup 1



Bei diesem Setup wird nur ein einzelner PC benötigt, welcher gleichzeitig zur Programmierung (= Developer PC) und Ansteuerung (= Control PC) des Franka Roboters genutzt wird. Der PC, welcher einen installierten Real-Time Kernel benötigt, ist direkt mit dem Mastercontroller des Franka Roboters verbunden.

Vorteile:
- Setup einfacher
- Nur ein PC benötigt
- Control PC benötigt nur einen Netzwerkanschluss

Setup 2



Beim zweiten Setup wird ein Control PC mit installiertem Real-Time Kernel direkt mit dem Mastercontroller des Franka Roboters verbunden (via Ethernet). Der Programmcode und das Starten von Programmen erfolgt jedoch auf einem (oder mehreren) Developer PCs, welche wiederum mit dem Control PC verbunden sind (via Ethernet).ad asd d

Vorteile:
- Developer PCs benötigen keinen Realtime Kernel → einfache Installation
- Einfacher Wechsel des Developer PCs möglich
- Anschluss mehrerer Developer PCs möglich und somit auch gut parallel von mehreren Gruppen nutzbar

Hinweise zur Auswahl des Control PCs

- Die benötigten Linux Realtime Kernel unterstützen keine NVIDIA Grafikkarten. Installieren Sie also bitte den Realtime Kernel nicht auf einem System mit NVIDIA Grafikkarte. Die Installation könnte das System komplett unbrauchbar machen und eine Wiederherstellung ist recht aufwendig.
- Auf dem PC muss das Betriebssystem Ubuntu laufen, optimalerweise in der Version 20.04. Es existiert zwar schon die Version 22.04, allerdings sind hier erst wenige Packages getestet und optimiert worden, so dass wir insbesondere im Hinblick auf die zukünftige Nutzung mit ROS Ubuntu 20.04 empfehlen. Ubuntu 18.04 sollte auch noch funktionieren, wurde von uns aber nicht getestet.
- Generell ist eine gute Netzwerkverbindung zwischen PC und Roboter nötig. Deshalb ist eine gute Netzwerkkarte von Vorteil. Lange Ethernetkabel oder Verbindungen über Switches o.ä. sollten auch vermieden werden.
- Falls ein zusätzlicher Developer PC genutzt werden soll, ist es von Vorteil, wenn der Control PC mehrere Netzwerkanschlüsse hat (aber nicht zwingend erforderlich). Diese können bei normalen Desktop PCs einfach und kostengünstig nachgerüstet werden. Alternativ kann auch ein USB-zu-Ethernet- Adapter genutzt werden.

Voraussetzung - Real-Time Kernel Installation

Für die Nutzung von FCI wird ein Real-Time Kernel benötigt damit die versendeten Daten und Befehle zum Roboter priorisiert behandelt werden. Dadurch ist eine hohe Updaterate des Roboters möglich. Ein Real-Time Kernel wird nur auf dem Control PC benötigt. Die Installation auf einem Developer PC ist nicht notwendig.
Die Installation und Konfiguration des Realtime Kernel ist recht aufwendig und wird etwas Zeit in Anspruch nehmen. Die Installation erfolgt durch Eingabe der nachfolgenden Befehle in ein Terminal.
Die Installationsanleitung ist größtenteils von hier übernommen: https://frankaemika.github.io/docs/

Achtung: Nvidia Treiber werden von dem Real-Time Kernel nicht unterstützt.!
→ Nicht auf einem Rechner mit Nvidia Grafikkarte installieren!

Verwendung des Terminal unter Ubuntu

In Linux bzw. Ubuntu kann das Terminal für praktisch alles genutzt werden. Wir werden damit die erforderlichen Pakete installieren, später Programmcode kompilieren und ausführen.

Kurze Übersicht der Befehle im Terminal:
- Terminal öffnen: Im Hauptmenü nach „terminal“ suchen oder **Strg + Alt + T**
- Im Terminal ausgeführten Befehl/Programm abbrechen: **Strg + C**
- Copy/Kopieren: **Strg + Shift + C**
- Paste/Einfügen: **Strg + Shift + V**
- In bestimmtes Verzeichnis wechseln: **cd** z. B. cd /directory/src/
- In Verzeichnis darüber wechseln: **cd ..** (zwei Punkte)
- Befehl ausführen: **Enter**
- Bei erstmaliger Eingabe von **sudo** (ähnlich zu „als Administrator ausführen“ in Windows) muss das Benutzerpasswort eingeben werden
- Bei (Teil-)Eingabe eines Befehls und dann Drücken von **Tab** wird dieser automatisch vervollständigt

Installationsanleitung Real-Time Kernel

- Öffne ein Terminal: Über die Suche „terminal“ oder per Tastenkombination **Strg + Alt + T**
- Danach folgende Befehle (graue Kästen) markieren, kopieren (**Strg + C**), im Terminal-Fenster einfügen (**Strg + Shift + V**) und mit Enter ausführen:
- Zuerst installieren wir ein paar Pakete, die zur Installation des Kernels benötigt werden:

```
sudo apt-get install build-essential bc curl ca-certificates gnupg2 libssl-dev
lsb-release libelf-dev bison flex dwarves zstd libncurses-dev
```

- Im nächsten Schritt müssen die zu installierenden Dateien heruntergeladen werden. Dazu erstellen wir uns zunächst einen Ordner im Download- Verzeichnis und wechseln in das neue Verzeichnis:

```
mkdir ~/Downloads/kernel_files
cd ~/Downloads/kernel_files/
```

- Jetzt müssen wir uns für eine Kernel Version entscheiden. Um die aktuelle genutzte Kernel Version herauszufinden kann man folgenden Befehl nutzen:

```
uname -r
```

- Es wird empfohlen eine Version zu nutzen, die der aktuellen Kernel Version am nächsten ist. Eine Liste aller verfügbaren Realtime Kernel Patches ist unter folgendem Link zu finden: https://www.kernel.org/pub/linux/kernel/projects/rt/
- Es wird empfohlen eine Version zu nutzen, die der aktuellen Kernel Version am nächsten ist. Eine Liste aller verfügbaren Realtime Kernel Patches ist unter folgendem Link zu finden: https://www.kernel.org/pub/linux/kernel/projects/rt/ sdfsd

Beispiel:

```
uname -r
```

liefert: 5.15.0-46-generic → Auf https://www.kernel.org/pub/linux/kernel/projects/rt/ finden wir einen Realtime Kernel Patch zu der Version 5.15.55 und wählen deshalb diese Version.

- Jetzt können die Dateien mit curl heruntergeladen werden

```
curl -SLO https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.15.55.tar.xz
curl -SLO https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.15.55.tar.sign
curl -SLO https://www.kernel.org/pub/linux/kernel/projects/rt/5.15/older/patch-5.15.55-rt48.patch.xz
curl -SLO https://www.kernel.org/pub/linux/kernel/projects/rt/5.15/older/patch-5.15.55-rt48.patch.sign
```

- Für eine andere Version müssen in den vorherigen vier Befehlen einfach nur die rot markierten Nummern ausgetauscht werden.
- Die heruntergeladenen Dateien liegen als tar-Datei vor. Wir entpacken diese mit:

```
xz -d *.xz
tar xf linux-*.tar
```

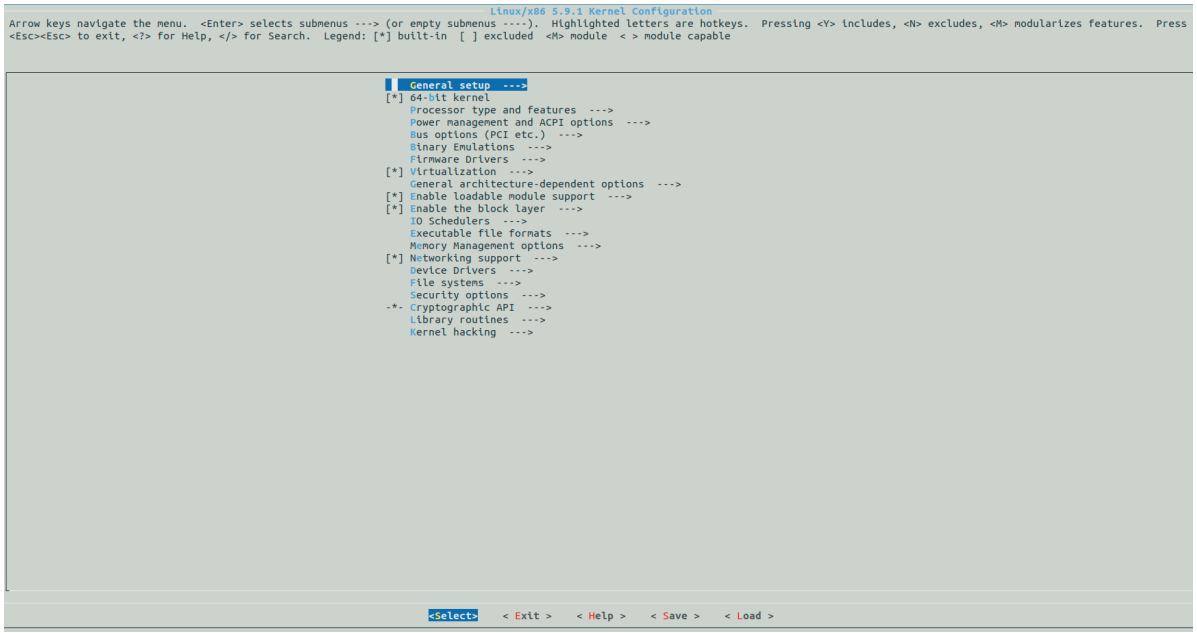
- Und wechseln dann in das entpackte Verzeichnis:

```
cd linux-*/
```

- Danach muss der heruntergeladene Kernel konfiguriert werden:

```
patch -p1 < ../patch-*.patch
cp -v /boot/config-$(uname -r) .config
make olddefconfig
make menuconfig
```

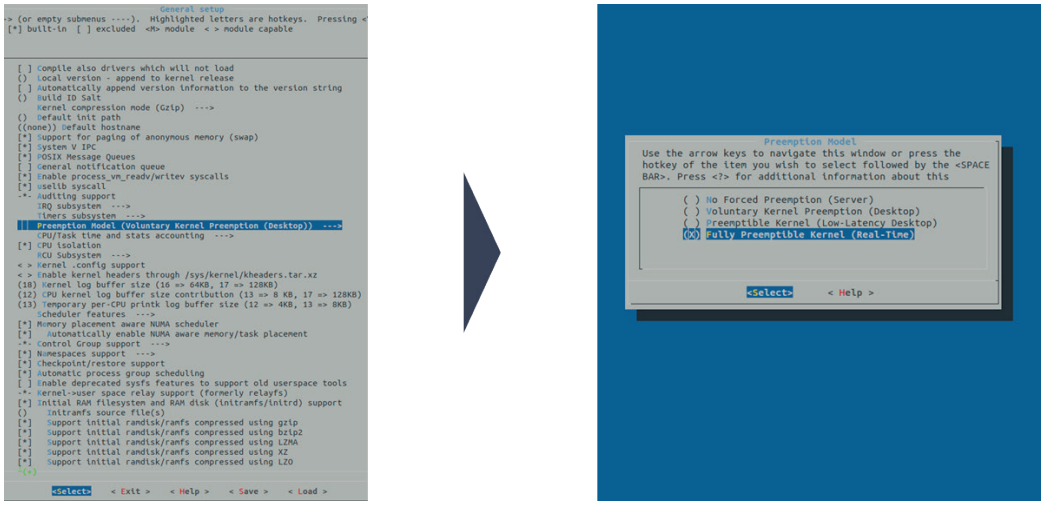
Mit dem letzten Befehl öffnet sich ein grafisches Menü, in dem zwei Einstellungen vorgenommen werden müssen. Navigiert wird mit den Pfeiltasten. Bestätigt mit **Enter** und zurück mit **Esc + Esc** (analog zum Doppelklick mit der Maus).



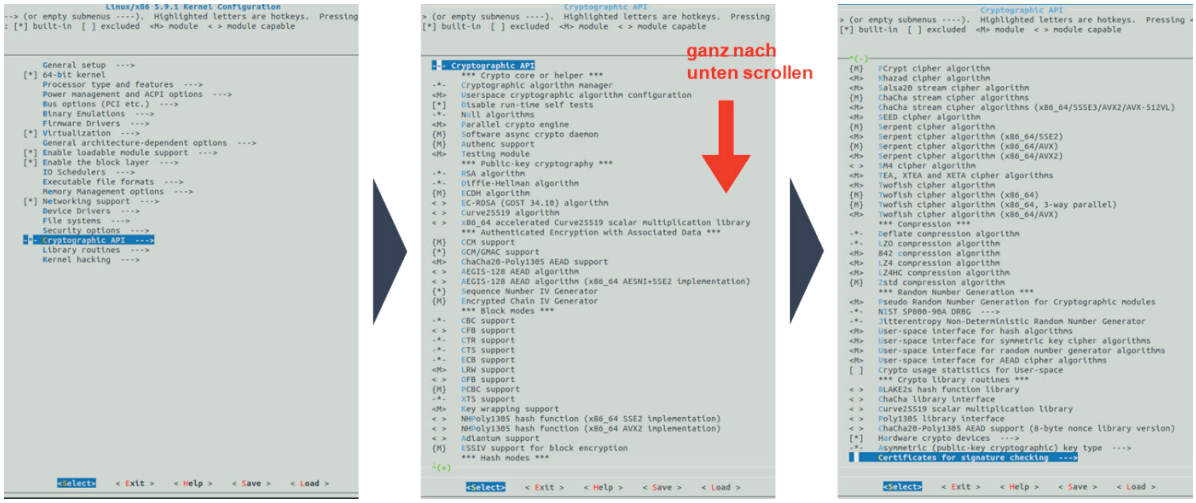
Die folgenden Einstellungen müssen angepasst werden:

- General setup → Preemption Model

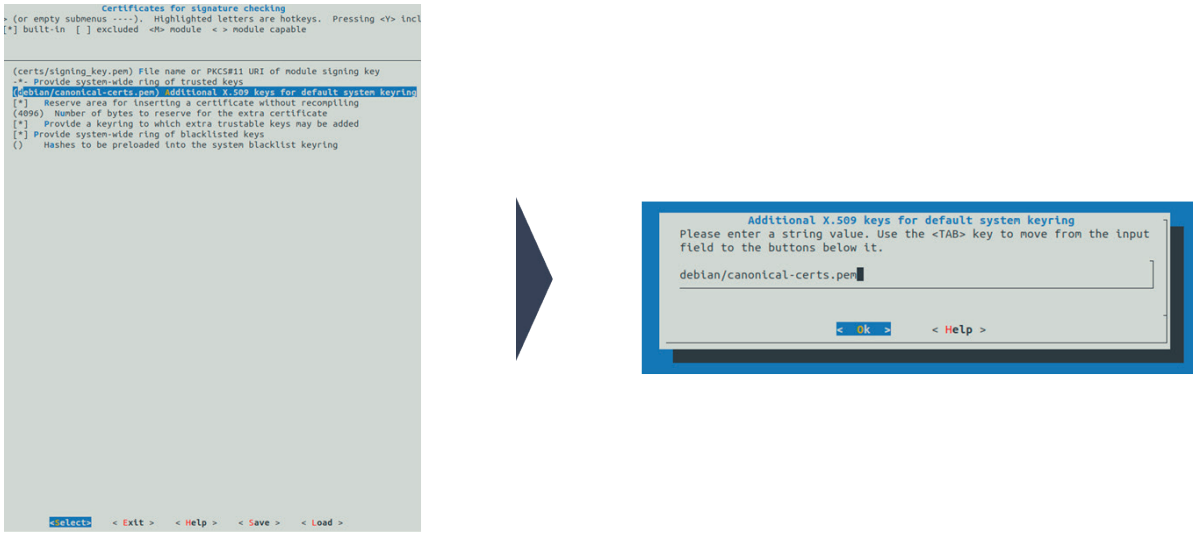
Hier auswählen: Fully Preemptible Kernel (Realtime)



- Cryptographic API → Certificates for signature checking



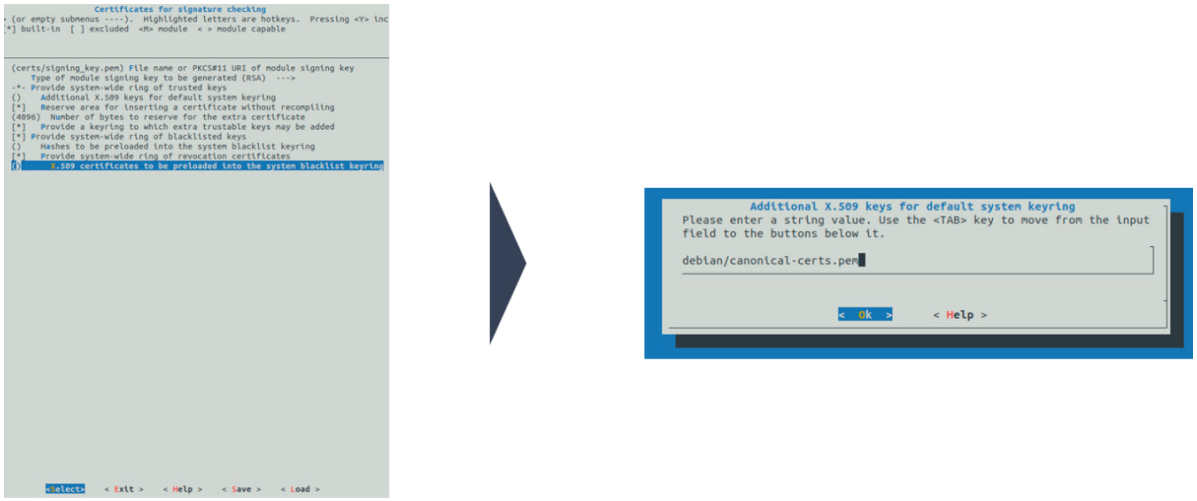
- Cryptographic API → Certificates for signature checking (ganz nach unten scrollen) → Provide system-wide ring of trusted keys → Additional X.509 keys for default system keyring



Wenn der Punkt ausgewählt wird, erscheint eine Texteingabezeile. Der dort stehende Text muss gelöscht werden. Danach bestätigen.

Optional: Je nach Kernelversion gibt es hier noch eine zusätzliche Einstellung:

- Cryptographic API → Certificates for signature checking (**ganz nach unten scrollen**) → Provide system-wide ring of revocation certificates → X.509 certificates to be preloaded into the system blacklist keyring



Wenn der Punkt ausgewählt wird, erscheint eine Texteingabezeile. Der dort stehende Text muss **gelöscht** werden. Danach bestätigen.

- Wurden alle Optionen gesetzt, muss das Menü mit doppelten Drücken von **Esc** beendet werden. Die Konfiguration muss anschließend gespeichert werden:

- Im Anschluss muss der Kernel kompiliert werden. Der Prozess kann eine längere Zeit dauern (**bis zu ca. einer Stunde!**). Solange im Terminal eine Ausgabe durchläuft ist alles in Ordnung. Es muss dann lediglich gewartet werden.
- ACHTUNG: Der Rechner darf dabei nicht in den Energiesparmodus wechseln und sollte auch nicht den Bildschirm ausschalten. Deshalb am besten vorher folgende Einstellungen in Ubuntu vornehmen:
- Settings → Power → Power Saving → Blank Screen: Never
- Settings → Power → Suspend & Power Button → Automatic Suspend: Off
- Kompilierung des Kernels:

```
make -j20
```

- Nach dem Kompilieren wird der kompilierte Kernel installiert:

```
sudo make modules_install -j20
sudo make install -j20
```

- Die Einstellungen des Bootloaders müssen danach geupdated werden, damit der neue Kernel beim Neustart auch angezeigt wird:

```
sudo update-grub
```

- Jetzt muss das System neu gestartet werden.

```
sudo reboot
```

- Nach dem Neustart sollte im Bootmenü (GRUB) unter „Advanced Options for Ubuntu“ die installierte Kernelversion auswählbar sein. In unserem Beispiel wäre das: Ubuntu, mit Linux 5.15.55-rt48
- Nach Auswahl wird der Kernel gestartet. Je nach Einstellung und System kann dies aber auch automatisch erfolgen und GRUB wird nicht benötigt.
- Nachdem das System gebootet ist, kann geprüft werden, ob die richtige Kernel Version installiert ist. Dazu im Terminal eingeben:

```
uname –a
```

- Der angezeigte Text sollte den String **PREEMPT RT** und die Versions- Nummer **5.15.55** enthalten.

Beispielausgabe:
Linux irtpc058 5.15.55-rt48 #1 SMP PREEMPT_RT Thu Sep 1 08:07:49 CEST 2022 x86_64 x86_64 x86_64 GNU/Linux

- Falls die Ausgabe **PREEMPT RT** enthält, können die nächste 4 Schritte übersprungen werden

- Falls der neue Realtime Kernel nicht gestartet wird und beim Starten des PCs das Grub Bootmenu nicht erscheint, muss folgender Befehl ausgeführt werden:

```
sudo gedit /etc/default/grub
```

- Im sich öffnenden Fenster müssen die folgenden Variablen geändert/hinzugefügt werden:

```
GRUB_DEFAULT=saved
GRUB_SAVEDEFAULT=true
GRUB_TIMEOUT_STYLE=menu
GRUB_TIMEOUT=10
```

- Anschließend folgenden Befehl ausführen:

```
sudo update-grub
```

- PC neustarten und im GRUB Bootmenu den Kernel ändern wie oben beschrieben.
- Abschließend müssen dem eigenen Benutzer noch die Rechte für die Verwendung der Realtime Berechtigungen gegeben werden:

```
sudo addgroup realtime
sudo usermod -a -G realtime $(whoami)
```

- Und in der Datei **/etc/security/limits.conf** Parameter gesetzt werden:
- Datei im Texteditor öffnen mit:

```
sudo gedit /etc/security/limits.conf
```

- Am Ende der Datei dann Folgendes anfügen:

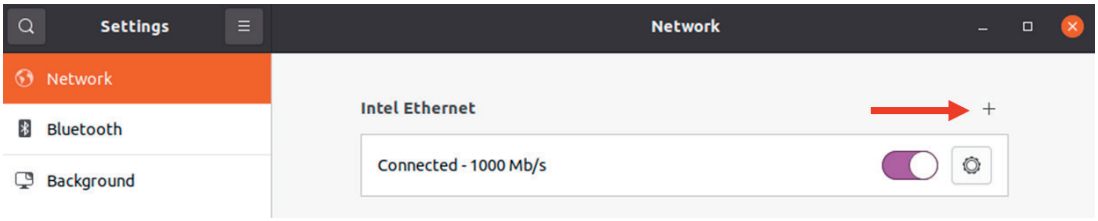
```
@realtime soft rtprio 99
@realtime soft priority 99
@realtime soft memlock 102400
@realtime hard rtprio 99
@realtime hard priority 99
@realtime hard memlock 102400
```

Damit ist die Installation des Real-Time Kernels abgeschlossen.

4.3.3 RF Robot Control Framework - Installation

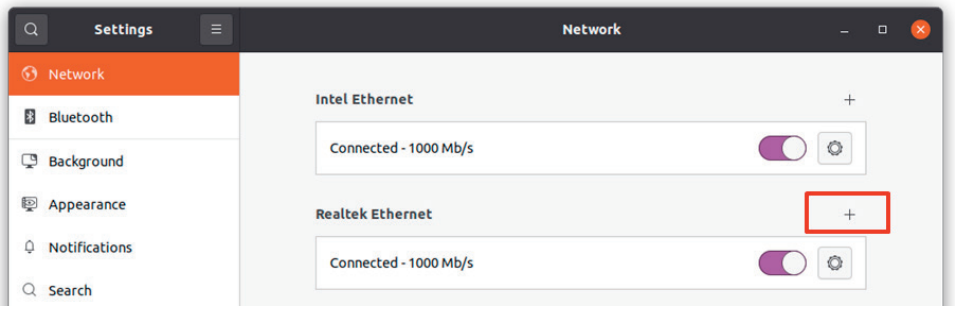
4.3.3.1 Vorbereitung – Einrichten der Netzwerkverbindung

Die Einrichtung der Netzwerkverbindung kann gegebenenfalls übersprungen werden. Es ist jedoch wichtig, dass dem Franka Roboter eine statische IP Adresse zugewiesen wird (standardmäßig wird die IP Adresse über DHCP bezogen).
Um die (Erst-)Konfigurationsschritte zu öffnen, muss ein Bediengerät (PC mit Firefox o. Chromium) über ein Ethernet-Kabel an den X5-Anschluss an der Basis des Franka Roboters (nicht am Controller!) angeschlossen werden. Die Ethernet-Verbindung des Bediengeräts muss so konfiguriert sein, dass die IP-Adresse automatisch über DHCP bezogen wird (**Settings → Network**).



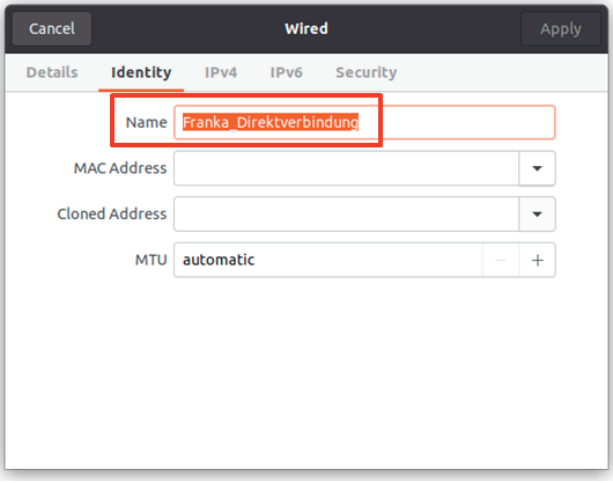
Durch Klicken auf das + wird ein neues Profil angelegt (Standardeinstellungen beibehalten, d. h. **IPv4 Method** auf **Automatic (DHCP)** eingestellt lassen).
Sobald der Roboter eingeschaltet ist, wird dem Schnittstellengerät eine automatische IP-Adresse zugewiesen. Nun kann die URL **robot.franka.de** in die Adresszeile eines Webbrowsers (Firefox o. Chromium) eingeben und mit Enter bestätigt werden. Als nächstes kann die Konfiguration des Roboters vorgenommen werden:

- Falls die Erstkonfiguration noch nicht stattgefunden hat, sollte dafür das Handbuch des Roboters gelesen werden.
- Andernfalls melden wir uns mit **Username** und **Password** an, um die Oberfläche von Desk zu öffnen.
- Nun navigiere zu den **Settings** des Roboters.
- Klicke in der linken Leiste auf **Network**.
- Unter Shop Floor network können die folgenden Einstellungen vorgenommen werden:
  - statische IP-Adresse des Roboters (hier: 192.168.1.11)
  - Netzmaske (hier: 255.255.255.0)
- Klicke zum Abschluss auf Apply
- Schließlich kann der PC über das Ethernet-Kabel mit dem Ethernet-Anschluss des Master Controllers verbunden werden
- Weitere Informationen:
  https://frankaemika.github.io/docs/getting_started.html#control-network-configuration
- Gehe zu **Settings → Network** und klicke auf das **+** rechts von Ethernet, um eine neue Ethernet-Verbindung hinzuzufügen

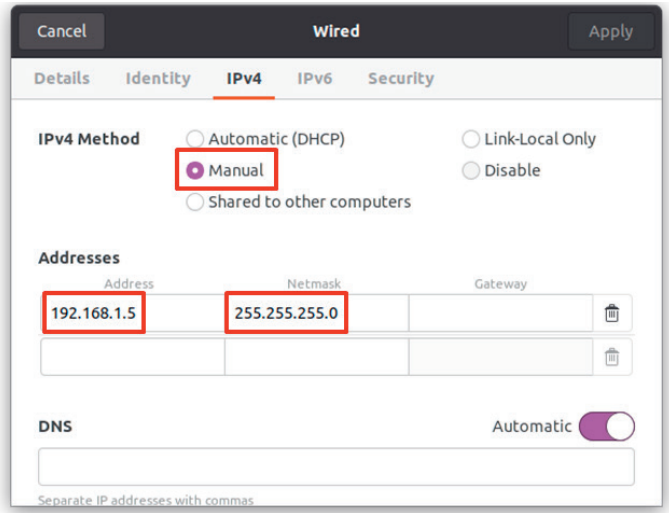


Hinweis: Falls der Control PC über zwei Netzwerkkarten verfügt, muss hier die **Netzwerkkarte** ausgewählt werden, die für die **Direktverbindung zum Roboter** verwendet werden soll.
Die andere **Netzwerkkarte** dient dem Control PC als **Zugang zum Netzwerk** bzw. Internet.

- Unter **Identity** kann der Name der neuen Verbindung festgelegt werden:



- Wechsle zu **IPv4** und schalte die Methode auf **Manual** um:



- Vergib eine statische IPv4 Adresse in der gleichen IP-Range wie Franka:
  - z. B. Franka IP = 192.168.1.11
  - → Rechner IP = 192.168.1.5
- Hinweis: Hier ist mit „Rechner IP“ die IP-Adresse der Direktverbindung zwischen PC und Roboter gemeint. Nicht die IP-Adresse des Rechners im Netzwerk!
- Setze die Netzmaske wie beim Franka Roboter (z. B. 255.255.255.0)
- Teste die Verbindung indem du die IP vom Franka Roboter (hier: 192.168.1.11) in die Adresszeile von einem Webbrowser eingibst

4.3.3.2 Vorbereitung - PC

Öffne ein Terminal: über die Suche „Terminal“, oder per Tastenkombination **Strg + Alt + T**.
Gib nacheinander die folgenden Befehle in das neue Terminal ein:

- Git (Software zur verteilten Versionsverwaltung von Dateien) installieren:

```
sudo apt install git
```

- Erstelle einen neuen Arbeitsbereich:

```
mkdir -p ~/catkin_ws/src
```

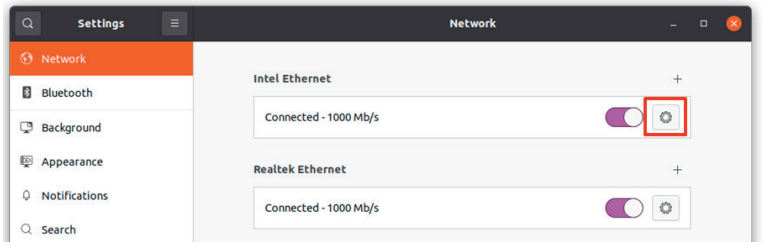
- Wechsle das Verzeichnis:

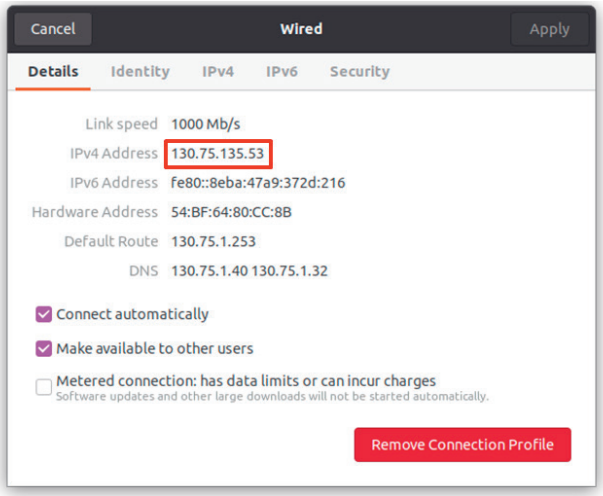
```
cd ~/catkin_ws/src
```

- Klone das Framework Repository:

```
git clone -b main https://gitlab.uni-hannover.de/roboterfabrik/rf_robot_framework.git .
```

- Nun müssen wir die **IP-Adresse des Rechners** herausfinden, mit der sich der Rechner im Netzwerk befindet. In dieser Anleitung besitzt der Control PC **zwei Ethernet-Karten**. Eine stellt die **Direktverbindung zum Roboter** her und die andere Karte dient nur **Verbindung mit dem Netzwerk** über eine statische IP-Adresse.
- Um die **IP-Adresse des Control PC zum Netzwerk** herauszufinden, gehen wir zu den **Settings** und **Network** und klicken auf das Zahnrad:



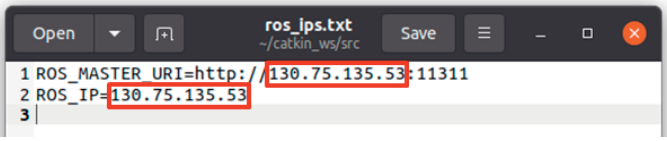


- Ändere die ROS_MASTER_URI und die ROS_IP auf dem Developer PC in der Datei ros_ips.txt:

```
gedit ./ros_ips.txt
```

- Durch den Befehl öffnet sich ein Fenster, in dem folgende Einstellungen gesetzt werden müssen:

```
ROS_MASTER_URI = http://<IP vom Control PC>:11311
ROS_IP = <IP vom Developer PC>
```

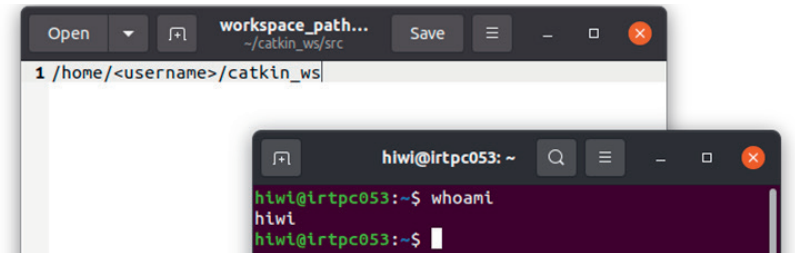


Hinweise:

- Mit „IP vom Control PC“ ist die IP-Adresse gemeint, mit der sich der Computer im Netzwerk befindet. Nicht gemeint ist die IP der „Franka_Direktverbindung“!
- Die dritte Zeile der Datei ros_ips.txt muss vorhanden und leer sein!
- Bei **Setup 2** ist der Control-PC auch der Developer PC, weshalb **beide IPs gleich** sind!

- Ändere den Pfad zum Catkin Workspace in der Datei workspace_path.txt:
  - Es muss nur der Nutzername geändert werden. Dazu am besten folgenden Befehl in einem Terminal ausführen:

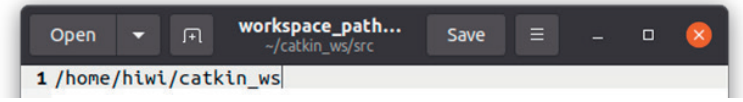
```
whoami
```



In dem dargestellten **Beispiel** liefert der Befehl whoami den Benutzernamen bzw. username **hiwi**.

- Anschließend die Ausgabe kopieren und den Platzhalter <username> in der workspace_path.txt Datei durch den Benutzernamen ersetzen:

```
gedit ./workspace_path.txt
```



Setup 1

- Führe folgende Skripte auf dem Developer PC (=Control PC) aus:

```
./host_preparation.sh
./kernel_preparation.sh
```

Setup 2

- Führe folgendes Skript auf dem Developer PC aus:

```
./host_preparation.sh
```

- Führe folgende Skripte auf dem Control PC aus:

```
./host_preparation.sh
./kernel_preparation.sh
```

Anschließend in beiden Fällen (Setup 1, Setup 2) Neustart des PCs durchführen.

- Jetzt muss nur noch das kompilierte Docker Image heruntergeladen werden:

```
cd ~/catkin_ws/src
./pull_docker_image.sh
```

Nun ist die PC Vorbereitung abgeschlossen.

Anpassung des Docker Containers

Nur wenn kein Zugriff auf das GitLab Docker Repository möglich ist oder das Container Image selbst gebaut werden soll, kann folgender Befehl ausgeführt werden:

```
contbuild
```

Dieser Befehl sollte nach Möglichkeit vermieden werden, es sei denn er ist ausdrücklich gewünscht. Der Download des Docker Images ist unbedingt zu bevorzugen. Wenn das Docker Image via `contbuild` neu erstellt wird, werden auch die neusten Versionen von *libfranka* und *franka_ros* installiert, was möglicherweise zu Inkompatibilitäten führen kann. Eine Möglichkeit für die Installation bestimmter Versionen ist aber als Kommentar im Dockerfile angegeben.

4.3.4 RF Robot Control Framework - Startup

Dieses Kapitel führt durch den Prozess der Inbetriebnahme des Roboters und die Steuerung durch das RF Robot Control Framework.
Das Tutorial geht von Setup 1 aus, d.h., der PC sollte direkt mit dem Roboter Controller verbunden sein und der Echtzeit-Kernel sollte laufen (Control PC = Developer PC).

1. Starte den Franka Controller
2. Öffne einen Browser
3. Gib die IP des Roboters in die Suchleiste des Webbrowsers ein. Dadurch wird das Franka Desk Interface geöffnet. Es wird zum Ent- und Verriegeln der Gelenke und zum Aktivieren des Franka Control Interface (FCI) verwendet.
4. In Desk: Entriegle die Gelenke. Der Roboter wird sich leicht bewegen. Wenn die Gelenke entriegelt sind, wechselt die Farbe der Schnittstelle von gelb zu weiß.
5. Aktiviere das FCI, indem du im Menü auf der rechten Seite auf **Activate FCI** klickst. Das Browserfenster kann anschließend minimiert oder geschlossen werden.

Hinweis: Es empfiehlt sich die Installation des Terminal-Multiplexers **Terminator**, der es ermöglicht, mehrere Terminals innerhalb eines einzigen Fensters zu benutzen. Das Fenster lässt sich mit der Tastenkombination **Strg + Shift + O** horizontal bzw. mit **Strg + Shift + E** vertikal teilen.
Die Installation von Terminator erfolgt mit dem folgenden Befehl in einem Terminal:

```
sudo apt install terminator
```

6. Öffne ein Terminal auf dem Control-PC (nachfolgend **Control-Terminal**)
7. Starte einen Docker Container mit Echtzeitfähigkeit auf dem Control-PC. Führe dazu folgenden Befehl im **Control-Terminal** aus:

```
contkernel
```

8. Starte die Control Node für die Verbindung zu Franka via FCI mit folgendem Befehl in dem Control-Terminal:

```
roslaunch rf_robot_control Panda.launch robot_ip:=<Roboter IP> load_gripper:=true
```

- Stelle sicher, dass die richtige IP-Adresse des Roboters für den Platzhalter <Roboter IP> im obigen Befehl eingegeben wurde. In unserem Beispiel lautet die Roboter IP **192.168.1.11**
- Damit dieser Befehl funktioniert, müssen die Robotergelenke bei der Befehlseingabe entriegelt sein.

Hinweis: Möchte man eine Node im Terminal beenden, funktioniert das über die Tastenkombination **Strg + C**. Das Beenden eines Docker Containers geht über **Strg + D** im Terminal.

9. Öffne ein neues Terminal auf dem Developer-PC (nachfolgend Developer-Terminal). Im Terminator geht das z. B. über **Strg + Shift + E**, indem man das aktuelle Fenster vertikal teilt.
10. Starte hier einen Docker Container auf dem Developer-PC. Führe dazu folgenden Befehl in dem Developer-Terminal aus:

```
cont
```

Dieser Schritt startet den Docker Container und baut den gesamten Arbeitsbereich auf, einschließlich der installierten Pakete.
Achtung: Falls weitere Geräte (z. B. USB-Kamera) genutzt werden sollen, müssen diese mit dem PC verbunden sein, bevor der Container gestartet wird.
11. Um das Projekt zu kompilieren, muss in dem Developer-Terminal das Folgende ausgeführt werden:

```
cd ~/catkin_ws
catkin build
```

Info: Falls Setup 1 verwendet wird (Developer-PC = Control-PC), müssen die Befehle trotzdem in zwei verschiedenen Terminals ausgeführt werden.

12. Jetzt kann mit der Entwicklung begonnen werden und Franka über ein Graphical User Interface (GUI) angesteuert werden. Um das GUI zu starten, führe folgenden Befehl in dem Developer-Terminal aus:

```
rosrun rf_robot_control node
```

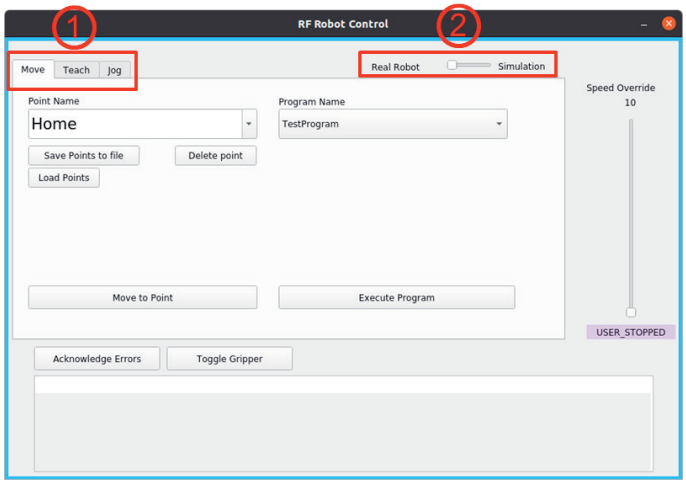
Die Benutzeroberfläche bzw. GUI von RF Robot Control wird geöffnet. Hier können Punkte angelernt bzw. „geteached“ werden und das eigene Programm ausgeführt werden. Falls Änderungen an einem Programm vorgenommen wurden, muss das Projekt neu kompiliert werden. Dazu muss das GUI geschlossen und die Schritte 11 und 12 erneut ausgeführt werden. Sobald wir in VS unseren Code anfangen zu schreiben, ist es komfortabler die Schritte 11 und 12 über ein Terminal im Fenster von VS auszuführen.
Hinweis: Wenn sich das Programm nicht wie gewünscht verhält, ist es wahrscheinlich, dass das Projekt nicht neu kompiliert wurden. Vor dem Kompilieren sollte der Code zunächst gespeichert werden. Die Funktionen des Frameworks werden im nächsten Abschnitt beschrieben.

4.3.5 RF Robot Control Framework - Nutzung

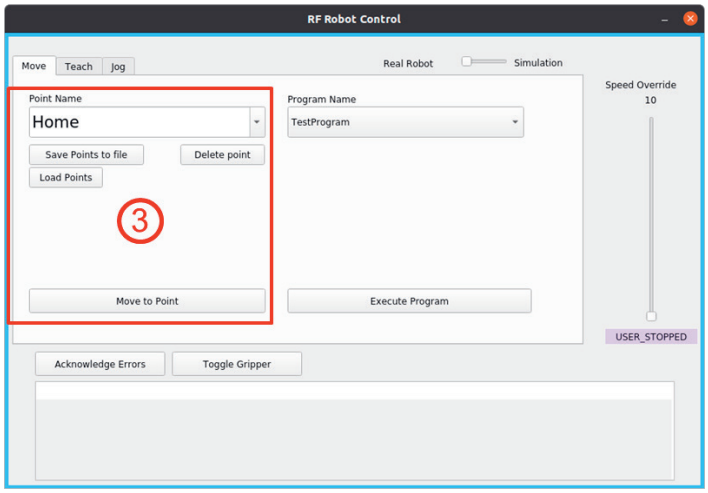
4.3.5.1 Graphical User Interface (GUI)

Im Folgenden werden die Funktionen des GUIs von **RF Robot Control** beschrieben:

1. Registerkarten zum Wechseln zwischen den Dialogfeldern **Move**, **Teach** und **Jog**:
   - **Move** ist der Modus, in dem der Roboter Bewegungen ausführt. Entweder beim Anfahren von zuvor „geteachten“ Punkten oder zum Ausführen eines erstellen Programms
   - **Teach** ist der Modus zum Einspeichern bzw. „Teachen“ von Punkten
   - **Jog** zum Variieren der Gelenkwinkel des 7-achsigen Roboters (Info: funktioniert nur im Modus Simulation! (siehe (2))
2. Schiebeschalter zum Wechseln zwischen den Modi **Real Robot** (blau Fensterumrandung) und **Simulation** (grüne Fensterumrandung). Auf die Simulationsumgebung wird im Kapitel „RF Robot Control Framework – Simulationsumgebung“ noch genauer eingegangen.

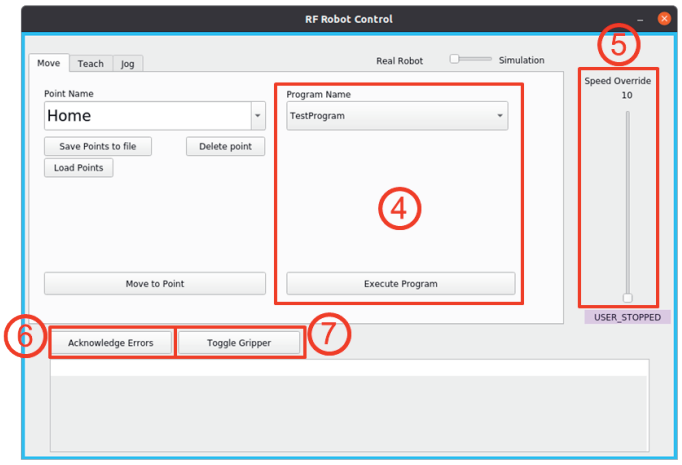


3. Dialog der gespeicherten Punkte:
   - **Point Name**: Dropdown-Liste der zuvor „geteachten“ Punkte
   - Button **Save Points to file** zum Speichern der Punkte in einer Datei (Ort: rf_robot_control/config/points.yaml)
   - Button **Load Points** zum Öffnen von gespeicherten Punkten (yaml-Datei)
   - Button **Delete point** um einzelne Punkte aus der Dropdown-Liste zu löschen
   - Button **Move to Point** um Roboter zu aus Dropdown-Liste ausgewählten, zuvor eingespeicherten Punkt zu bewegen

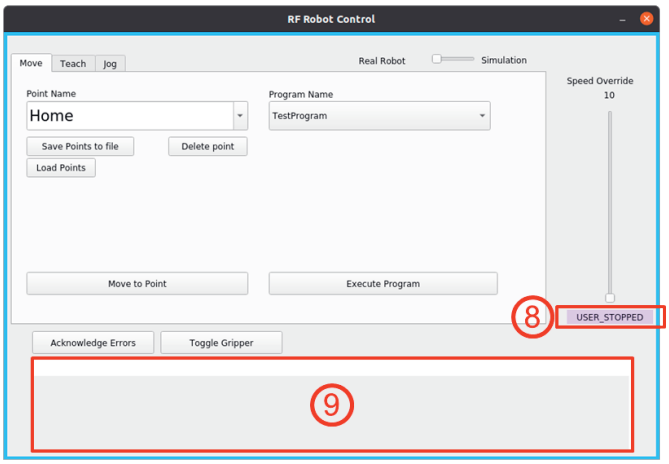


Hinweis: Die Punkte werden standardmäßig in der Datei points.yaml gespeichert. Es ist nicht notwendig nach dem „Teachen“ und Löschen von Punkten, die Punkte in der Datei points.yaml abzuspeichern. Das geschieht bereits automatisch. Ebenso wird die Datei points.yaml geladen, sobald das GUI geöffnet wird.

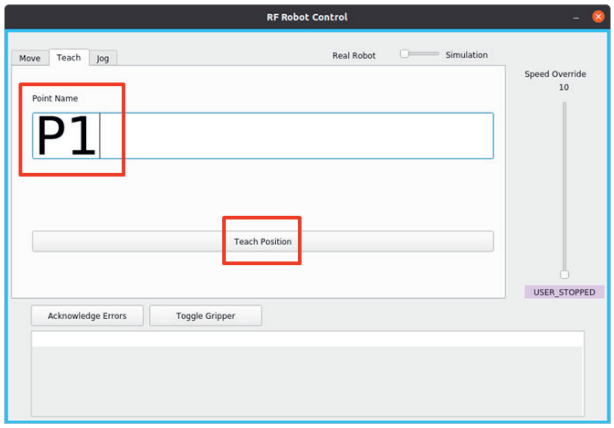
4. Dialog zum Aufrufen und Ausführen des Programms:
   - **Program Name**: Dropdown-Liste zum Auswählen des Programms, das ausgeführt werden soll (Standardprogramm: TestProgram)
   - Button **Execute Program** zum Ausführen des zuvor ausgewählten Programms
5. Schieberegler **Speed Override** zur Geschwindigkeitsübersteuerung des Roboters in Prozent
6. Button **Acknowledge Errors** zum Quittieren von Fehlermeldungen in der Anzeige
7. Button **Toggle Gripper** zum Öffnen und Schließen des Greifers (engl. gripper)



8. Statusanzeige des Roboters (versch. Modi):
   - **USER_STOPPED**: User Stop Button ist gedrückt, Roboter bewegt sich nicht
   - **IDLE**: User Stop Button ist gelöst, Roboter ist untätig (engl. idle)
   - **MOVE**: User Stop Button ist gelöst, Roboter bewegt sich
   - **GUIDING**: User Stop Button ist gedrückt, Roboter wird manuell geführt
   - **REFLEX**: Fehler bei Programmausführung
9. Anzeige von Informationen, Fehler- und Warnmeldungen:



Registerkarte Teach zum Einspeichern von Punkten:

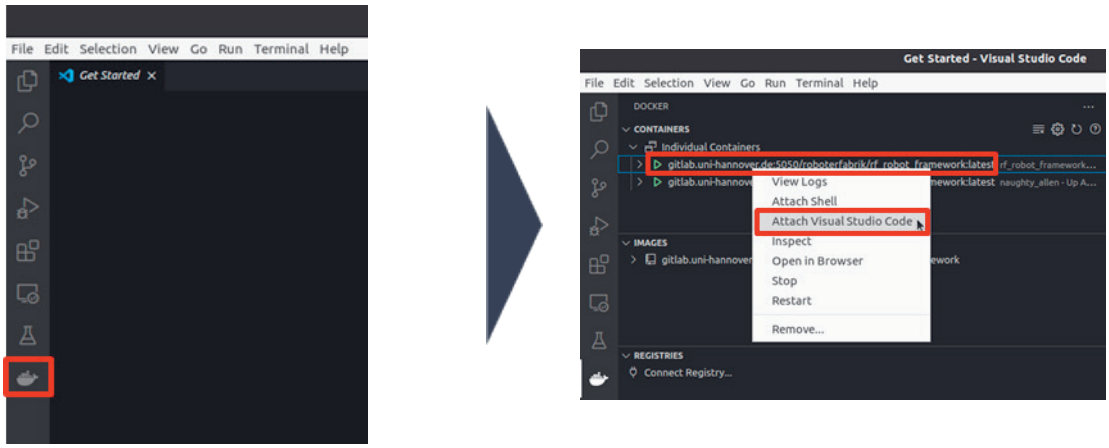


Nachdem der **User Stop** des Roboters gelöst wurde, kann der Roboter bei gleichzeitiger Betätigung des **Guiding Tasters** und **Zustimmungsschalters** manuell bewegt werden, um eine neuen Punkt zu „teachen".
Dazu bewegt man den Roboter zunächst zu dem gewünschten Punkt. Dann kann man dem Punkt unter **Point Name** einen Namen vergeben (z. B. P1) und mit einem Klick auf den Button **Teach Position** abspeichern.

4.3.5.2 Programmierung mit Visual Studio Code

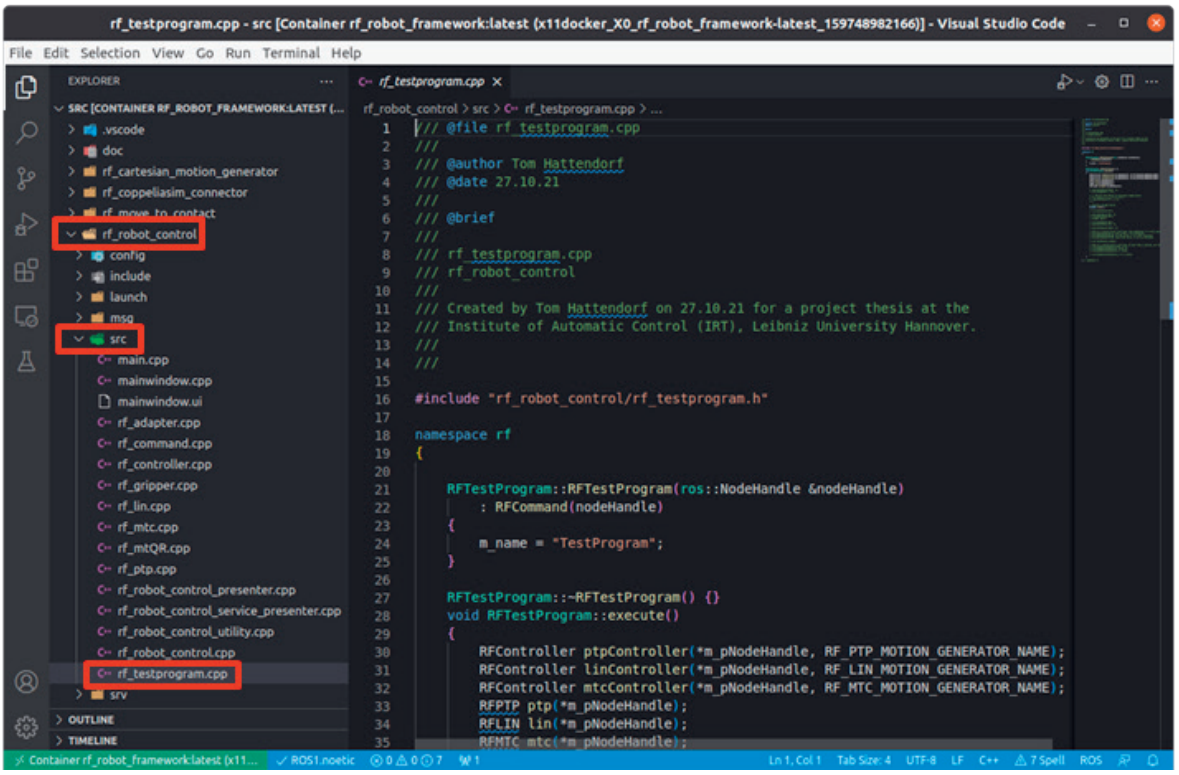
Im Folgenden wird beispielhaft anhand eines Testprogramms beschrieben wie man eigene Programme mit dem RF Robot Control Framework mithilfe des Editors Visual Studio Code schreibt. Dafür gibt es im Repository schon das Beispielprogramm „TestProgram".
Um den Code eines Programms zu editieren bzw. um ein eigenes Programm zu schreiben, ist es am einfachsten den Arbeitsbereich des Docker Containers in Visual Studio Code zu laden. Dazu sind die folgenden Schritte nötig.

- Öffne **VS Code** und klicke mit der linken Maustaste im Menü auf der linken Seite auf die Docker Erweiterung. Sobald der PC-Container gestartet ist, ist er auf der linken oberen Seite unter der Kategorie **Containers** zu sehen.
- Klicke mit der rechten Maustaste auf den ersten Eintrag **gitlab.uni-hannover.de:5050/roboterfabrik/rf_robot_framework:latest** und dann auf **Attach Visual Studio Code**.

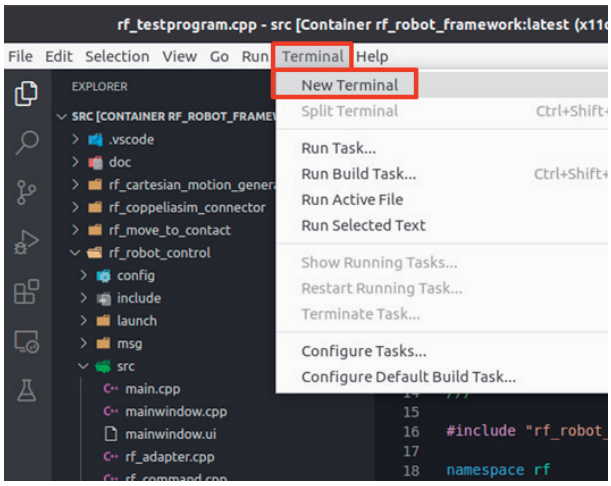


Es öffnet sich ein neues Fenster von VS Code mit dem Arbeitsverzeichnis des Ordners catkin_ws.
Das ursprüngliche VS Code Fenster kann nun geschlossen werden.

- Nun können wir anfangen unseren Code in VS Code zu schreiben. Dazu navigieren wir durch die Verzeichnisstruktur zu der Datei rf_testprogram.cpp.
- Die Datei enthält den C++-Code des Programms „TestProgram", welches über das GUI RF Robot Control ausgeführt werden kann.

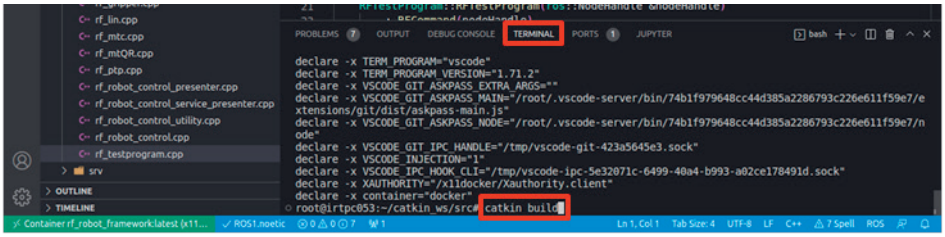


- Sobald wir in VS Code arbeiten, bietet es sich an, in einem einzigen Fenster zu arbeiten. Dort können wir den Code kompilieren und das GUI öffnen bzw. den Code ausführen.
- Dazu können wir in VS Code ein neues Terminal öffnen (Terminal → New Terminal).



- Sobald der Code in der Datei rf_testprogram.cpp zu Ende editiert und gespeichert (**Strg + S**) wurde, kann der Code in dem neuen Terminal mit dem folgenden Befehl kompiliert werden:

```
Catkin build
```



- Nach dem Kompilieren des Codes, kann das GUI über den folgenden Befehl geöffnet werden

```
rosrun rf_robot_control node
```

- Über das GUI lässt sich schließlich der Code ausführen und die Bewegung des Roboters kann gestartet werden.

4.3.5.3 Doxygen Dokumentation

Für Informationen zu den verschiedenen Klassen und Befehlen des Frameworks kann die **Doxygen Dokumentation** im Webbrowser gelesen werden:

1. Öffne ein neues Terminal mit **Strg + Alt + T** und gibt den folgenden Befehl ein, um in das Verzeichnis der Dokumentation zu wechseln und um das zip-Archiv der Dokumentation zu extrahieren:

```
cd ~/catkin_ws/src/doc && unzip doxygen_output.zip
```

2. Gib in ein Terminal den nächsten Befehl ein, um die Doxygen Dokumentation z.B. im Firefox Browser zu öffnen (alternativ kann man die Datei über den Dateimanager öffnen):

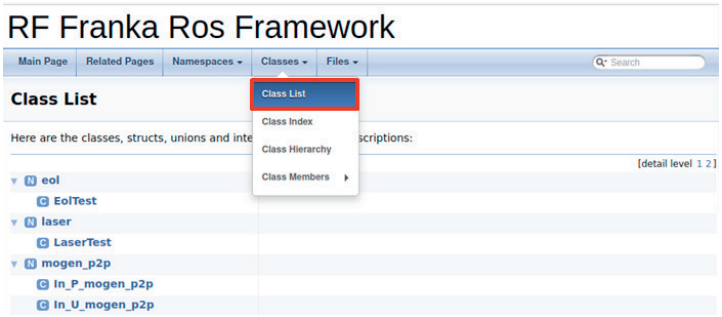
```
firefox ~/catkin_ws/src/doc/doxygen_output/html/index.html
```

3. In dem sich öffnenden Fenster kann die Dokumentation nun gelesen werden.
   Aktuell unterstützt das Framework drei verschiedene Bewegungsarten (Klassen):
   - MoveCartesian (**RFLIN**): erzeugt lineare Bewegungen von einem kartesischen Punkt zu einem anderen
   - MoveJoint (**RFPTP**): erzeugt Bewegungen direkt von einem Punkt zu einem anderen im Gelenkraum (engl. joint space)
   - MoveToContact (**RFMTC**): ein linearer Punkt bewegt sich in eine bestimmte Richtung bis eine entgegengesetzte Kraft überschritten wird

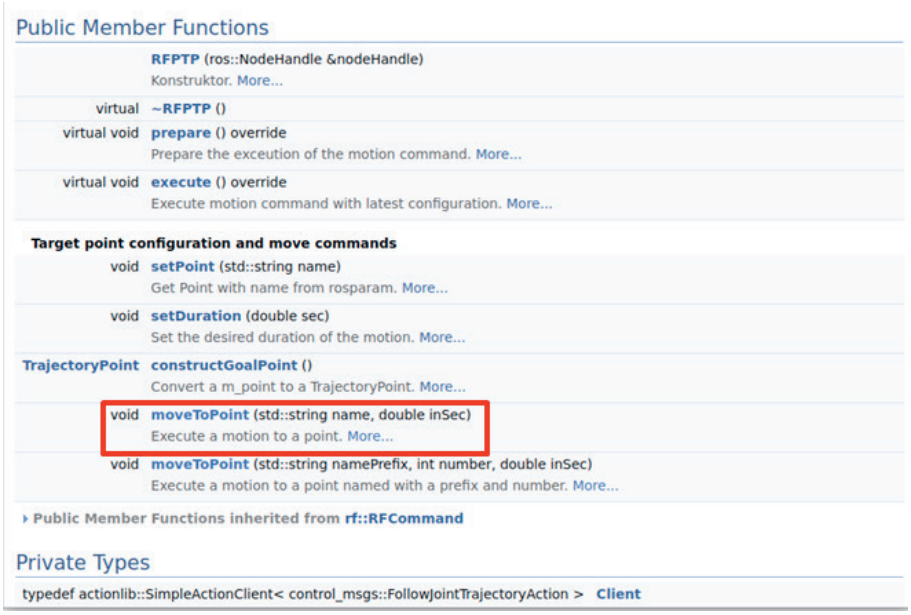
Neben den Klassen der Bewegungsarten (**RFLIN**, **RFPTP**, **RFMTC**) ist die Klasse des Greifers (**RFGRIPPER**) von besonderer Bedeutung bei der Programmierung von Roboteranwendungen.

In der Doxygen Dokumentation des Frameworks können die verschiedenen **Klassen** genauer studiert werden. Dort kann man sich die möglichen **Befehle** samt der erforderlichen **Parametrisierung** anschauen. Im Folgenden wird am **Beispiel** der Klasse **RFPTP** gezeigt, wie man durch die Doxygen Dokumentation navigiert, um die **Befehle** und **Parameter** genauer zu verstehen:

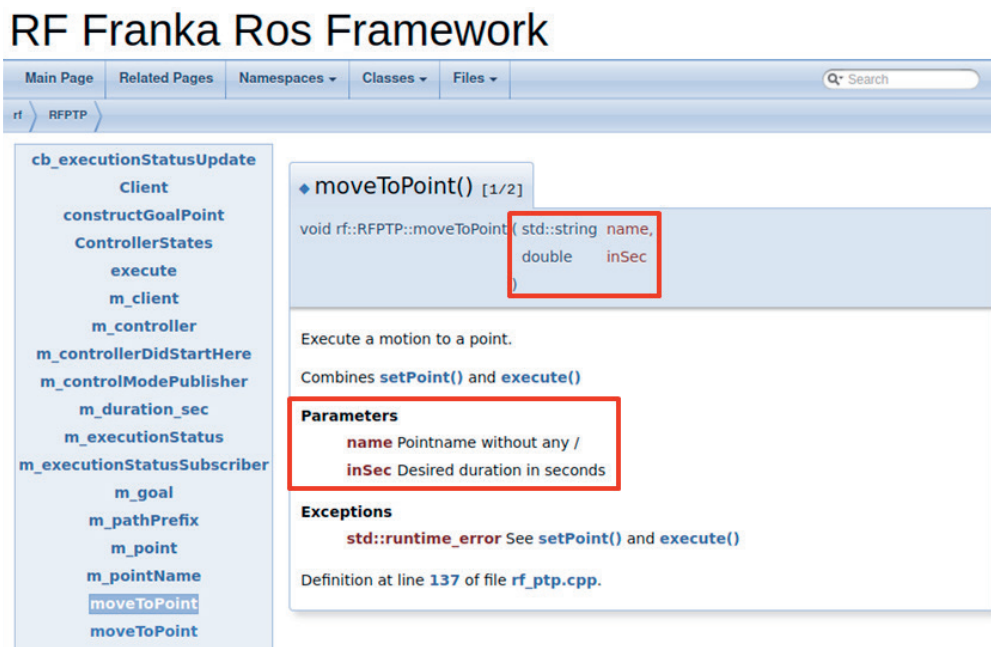
1. Auf der Startseite bzw. **Main Page** der Doxygen Dokumentation kann man in der Navigationsleiste die Maus auf „**Classes**“ bewegen und auf „**Class List**“ klicken:



2. Auf der Seite zu den verschiedenen Klassen (Class List) kann man zu RFPTP navigieren, um nähere Informationen darüber zu erhalten, welche Befehle die Klasse bereitstellt:



4. Nun kann man genauere Informationen zum Befehl moveToPoint() erhalten:
   - Parameter: Reihenfolge + Datentypen
   - Kurzbeschreibung der Parameter

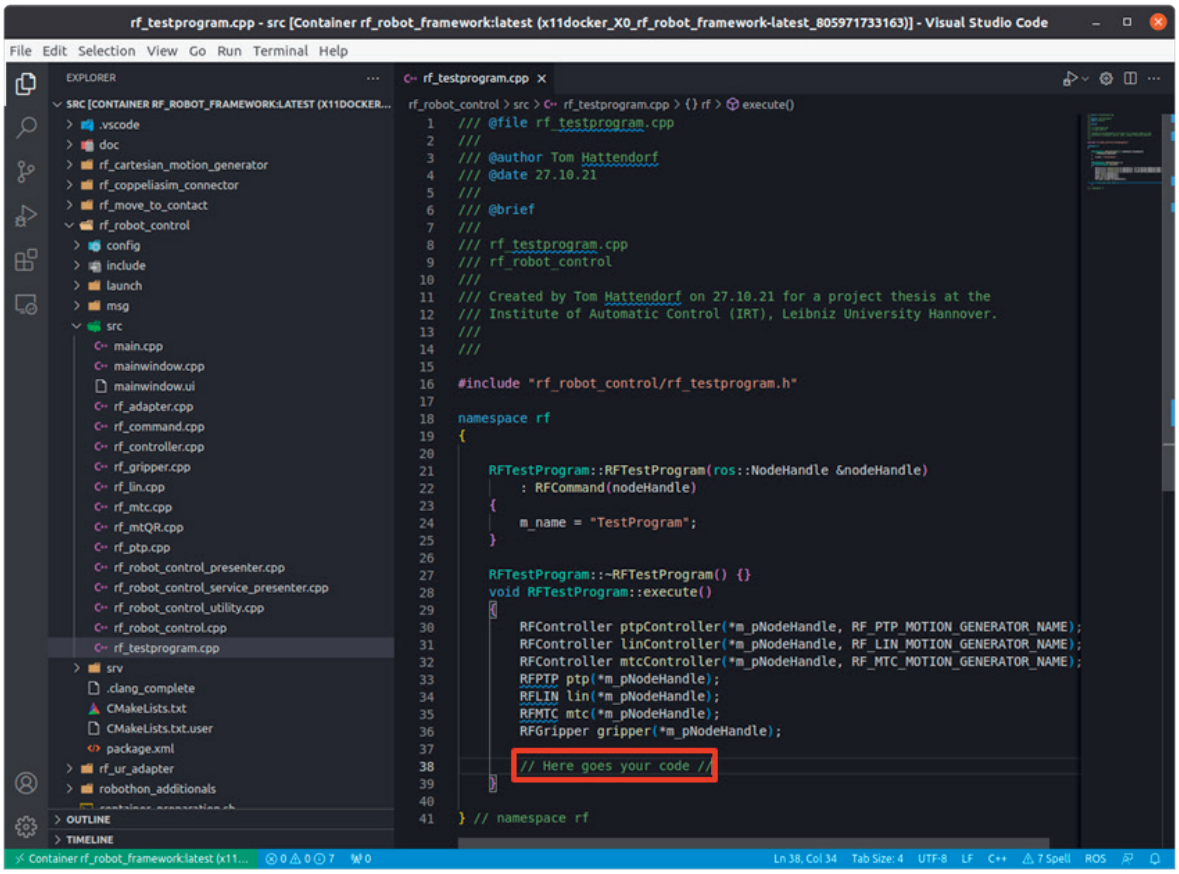


Weitere wichtige Klassen bzw. Befehle sind:

- **RFGRIPPER**
  - gripper.open()
  - gripper.close()
  - gripper.grasp()
- **RFLIN**
  - lin.setSpeed()
  - lin.moveRelativeToPoint()
- **RFPTP**
  - ptp.moveToPoint()
- **RFMTC**
  - mtc.moveToContact()

4.3.5.4 Programmierbeispiel

Als nächsten thematisieren wir anhand eines **Beispiels** die Programmierung mit Hilfe des Frameworks. Dazu arbeiten wir in **Visual Studio Code** in der Datei rf_testprogram.cpp. Das Programm lädt bereits alle notwendigen Controller für die drei Bewegungsarten sowie für den Gripper. Die Stelle an welcher der eigene Code eingefügt werden sollte, ist im folgenden Bild in Rot markiert. Der Rest der Datei kann zunächst unverändert bleiben.



Beim Programmieren stehen zwei wichtige **Hilfsmittel** zur Verfügung:

- Doxygen Dokumentation
- Auto-Vervollständigung (IntelliSense in Visual Studio Code)

IntelliSense kann in VS Code mit der Taste **Tab** aufgerufen werden. Dadurch wird intelligente Autovervollständigung und Hilfe bei Parametern ausgelöst.

4.3.5.4.1 Beispielaufgabe

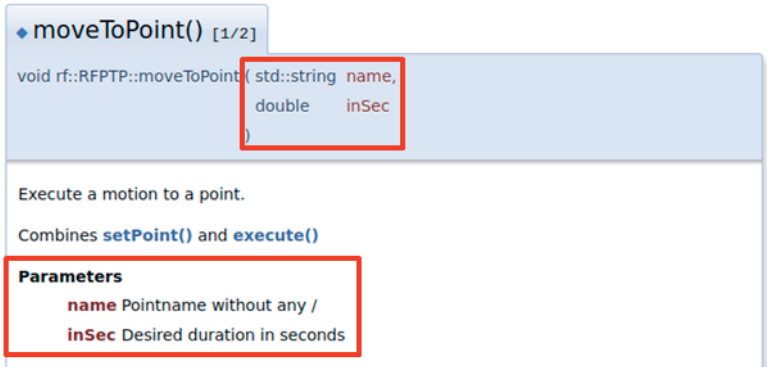
Schreibe ein einfaches Programm zur Durchführung der folgenden Teilaufgaben.
Wir gehen davon aus, dass die Punkte P1, P2 und Home bereits in einem vorherigen Schritt über das GUI RF Robot Control eingespeichert bzw. „geteached“ wurden.

Teilaufgaben:

1. Bewege den Roboter mit einer Dauer von 2 Sekunden mit der Bewegungsart PTP (PointToPoint) zu Punkt P1 und öffne den Greifer.
2. Bewege den Roboter linear von P1 mit einer maximalen Geschwindigkeit von 0,1 m/s um 5 cm nach unten.
3. Nimm einen Würfel mit einer Kantenlänge von 1 cm auf.
4. Bewege den Roboter linear mit der Standardgeschwindigkeit zu Punkt P2.
5. Lege das Objekt an Punkt P2 ab.
6. Bewege den Roboter mit PTP zurück zum Punkt Home.

4.3.5.4.2 Lösung Beispielaufgabe

Teilaufgabe 1: Bewege den Roboter mit einer Dauer von 2 Sekunden mit der Bewegungsart PTP (Point-ToPoint) zu Punkt P1 und öffne den Greifer.

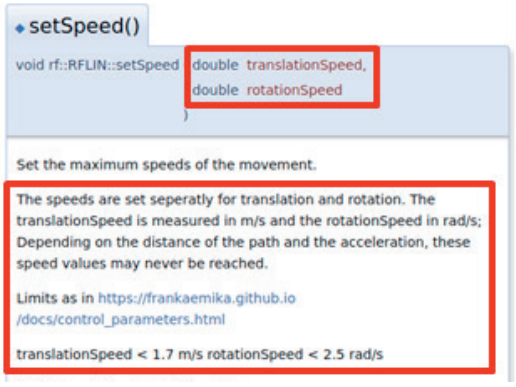


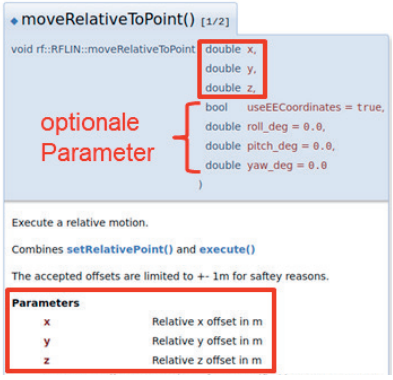

```
27    RFTestProgram::~RFTestProgram() {}
28    void RFTestProgram::execute()
29    {
30        RFController ptpController(*m_pNodeHandle, RF_PTP_MOTION_GENERATOR_NAME);
31        RFController linController(*m_pNodeHandle, RF_LIN_MOTION_GENERATOR_NAME);
32        RFController mtcController(*m_pNodeHandle, RF_MTC_MOTION_GENERATOR_NAME);
33        RFPTP ptp(*m_pNodeHandle);
34        RFLIN lin(*m_pNodeHandle);
35        RFMTC mtc(*m_pNodeHandle);
36        RFGripper gripper(*m_pNodeHandle);
37
38        // Task 1:
39        // PTP motion is in general faster than a linear motion
40        ptp.moveToPoint("P1", 2);
41        gripper.open();
42    }
43
44    } // namespace rf
```

Hinweise:

- Vor dem Kompilieren speichern!
- Am Ende eines Befehls Semikolon nicht vergessen!
- Code ein- und auskommentieren funktioniert mit **Strg + Shift + 7**

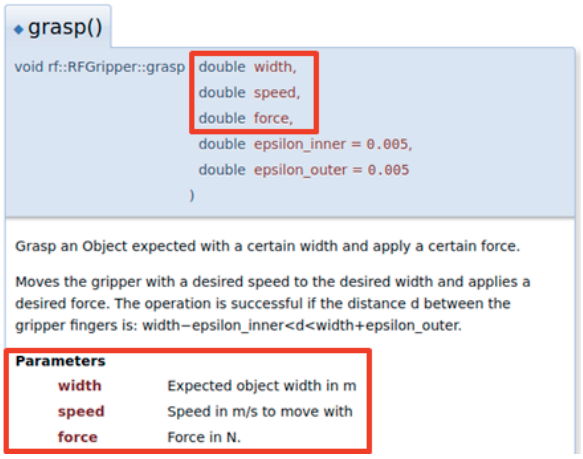
Teilaufgabe 2: Bewege den Roboter linear von P1 mit einer maximalen Geschwindigkeit von 0,1 m/s um 5 cm nach unten.






```
38        // Task 1:
39        // PTP motion is in general faster than a linear motion
40        ptp.moveToPoint("P1", 2);
41        gripper.open();
42
43        // Task 2:
44        // Set the desired speed for translation. Rotation set as default value.
45        // Execute a relative motion.
46        lin.setSpeed(0.1, 0.5);
47        lin.moveRelativeToPoint(0, 0, 0.05);
48    }
```

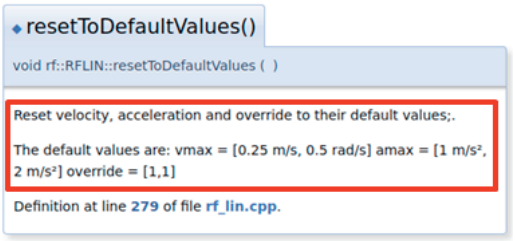
Teilaufgabe 3: Nimm einen Würfel mit einer Kantenlänge von 1 cm auf.

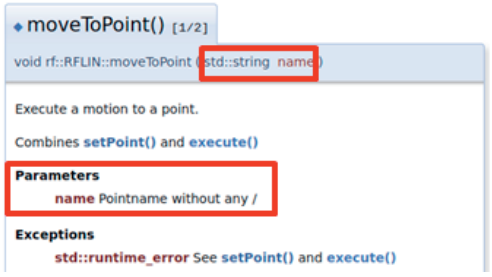



```
48
49      // Task 3:
50      // Grasp the object.
51      // gripper.close() could also be possible, but we know the exact object size.
52      gripper.grasp(0.01, 0.05, 20);
53  }
```

Hinweis: Das Programm bricht ab, wenn sich der Würfel nicht an dem Punkt befindet, wo er aufgenommen bzw. gegriffen werden soll.

Teilaufgabe 4: Bewege den Roboter linear mit der Standardgeschwindigkeit zu Punkt P2.

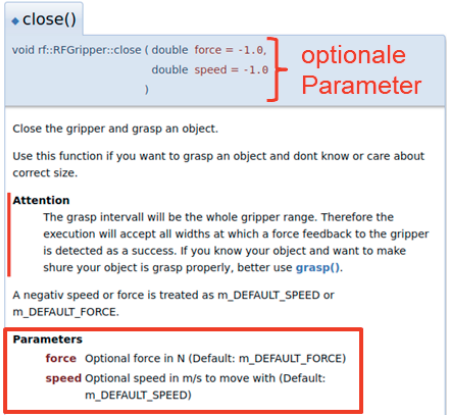





```
53
54      // Task 4:
55      // The speed from Task 2 is still set. Therefore, we have to reset it.
56      lin.resetToDefaultValues();
57      lin.moveToPoint("P2");
58  }
```

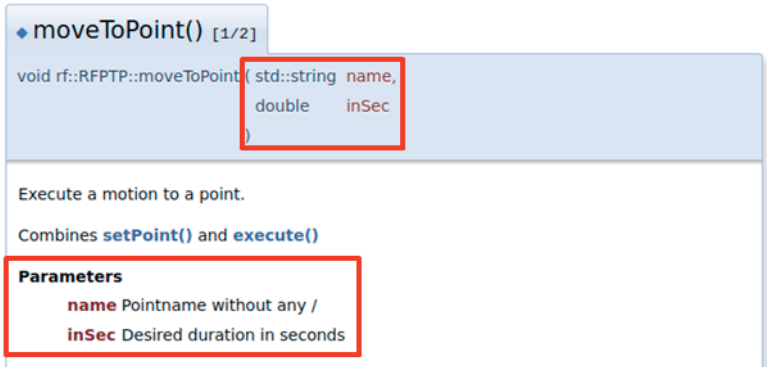
Hinweis: Das Programm bricht ab, wenn sich der Würfel nicht an dem Punkt befindet, wo er aufgenommen bzw. gegriffen werden soll.

Teilaufgabe 5: Lege das Objekt an Punkt P2 ab.



```
58
59      // Task 5:
60      // Simply open the gripper with default speed.
61      gripper.open();
62  }
```

Teilaufgabe 6: Bewege den Roboter mit PTP zurück zum Punkt Home.




```
62
63      // Task 6:
64      ptp.moveToPoint("Home", 2);
65  }
```

4.3.5.4.3 Exception Handling

Nachdem wir unser erstes Programm erstellt haben, können wir uns mit der Behandlung von Ausnahmen (engl. **exceptions handling**) auseinandersetzen. Bei der Arbeit mit echten Robotern können **unvorhersehbare Dinge** passieren, daher lösen viele Befehle, wenn sie nicht erfolgreich sind, Ausnahmen aus.

Beispiel: Nehmen wir an, du willst einen Stein vom Tisch greifen und ihn auf einen anderen legen. Dafür würdest du den Befehl `gripper.grasp()` verwenden. Einer der Parameter ist die erwartete Objektbreite. Unter normalen Umständen kann das Objekt problemlos gegriffen werden. Was passiert jedoch, wenn man vergisst, einen Stein an der gewünschten Stelle zu platzieren?
In diesem Fall wird von `gripper.grasp()` eine **Ausnahme geworfen**, weil der Greifer weit mehr als die eingestellte Objektbreite geschlossen wird. Das ist eigentlich eine gute Sache, weil etwas schiefgelaufen ist. Diese Ausnahme (exception) kann mit einem **try-catch-Block** behandelt werden.

In der Programmiersprache C++ sind **try-catch-Blöcke** eine Art und Weise Ausnahmen zu behandeln. Alles, was schief gehen kann, wird im try-catch-Block stehen. Wenn etwas eine Ausnahme auslöst, kann der **catch-Block** das Ausnahmeobjekt abfangen und entsprechend darauf reagieren. Andernfalls wird der Code im catch-Block niemals ausgeführt werden. Die meisten Ausnahmen im Framework sind std::runtime_error.
Wenn irgendwo eine Ausnahme ausgelöst wird, geht das Programm auf dem Aufrufstapel (engl. **call stack**) nach oben, um einen geeigneten catch-Block zu finden. Wenn kein Block gefunden wird, wird das Programm beendet. Mit anderen Worten: Wenn man eine Ausnahme nicht abfängt, wird das Programm nicht fortgesetzt. Aus Sicherheitsgründen ist das eigentlich eine gute Sache.
Nun schauen wir uns ein **Beispiel** an: Der nachfolgende Codeschnipsel ist die Grundroutine zum Greifen eines Steines.

Codeschnipsel (**ohne** try-catch-Block):

```
38      lin.moveToPoint("Pickup");
39
40      gripper.grasp(0.015, 0.05, 20);
41
42      lin.moveToPoint("Place");
43      gripper.open();
44      lin.moveToPoint("Home");
45  }
```

Befindet sich kein Stein am Punkt Pickup, löst gripper.grasp() eine unabgefangene Ausnahme aus. Der Roboter wird einfach am Punkt Pickup mit geschlossenem Greifer verharren. Die Bewegungen zu den Punkten Place und Home werden nicht ausgeführt.

Codeschnipsel (**mit** try-catch-Block):

```
38      lin.moveToPoint("Pickup");
39
40      try
41      {
42          gripper.grasp(0.015, 0.05, 20);
43      }
44      catch (const std::runtime_error &e)
45      {
46          ROS_ERROR_STREAM("Gripper grasp failed. Aborting and moving back to Home: " << e.what());
47          gripper.open();
48          lin.moveToPoint("Home");
49          throw;
50      }
51
52      lin.moveToPoint("Place");
```

Befindet sich kein Stein am Punkt Pickup, löst `gripper.grasp()` eine Ausnahme aus, die im catch-Block abgefangen wird. Als Reaktion wird der Benutzer darüber informiert, dass der Vorgang fehlgeschlagen ist. Der Roboter öffnet wieder den Greifer und bewegt sich zurück nach Home.

Bitte beachte, dass die Ausnahme am Ende des catch-Blocks erneut ausgelöst wird. Das liegt daran, dass die Funktion `execute()` ihren Zweck nicht erfüllt hat und daher ebenfalls fehlgeschlagen ist.

Allerdings befindet sich der Roboter nun in einem definierten Zustand (Home, Greifer offen).

4.3.6 RF Robot Control Framework – Simulationsumgebung

CoppeliaSim, früher bekannt als V-REP, ist ein **Robotersimulator**, der in Industrie, Bildung und Forschung eingesetzt wird. Innerhalb des RF Robot Control Frameworks kann er genutzt werden um Code zunächst in einer Simulationsumgebung zu testen bevor ein echter Roboter benötigt wird. In der aktuellen Version funktionieren jedoch noch keine Befehle, die Kraft- oder Momentmessung benötigen (wie z. B. MoveToContact) und es gibt noch weitere kleinere Einschränkungen, die am Ende des Kapitels zusammengefasst sind.
Um den Robotersimulator zu starten, muss zunächst der folgende Befehl in ein **Developer Terminal** eingegeben werden, um den Developer Container zu starten:

```
cont
```

Öffne nun VS Code und wähle über die Docker Erweiterung in der linken Leiste den Container **rf_robot_framework:latest** mit einem Rechtklick aus und klicke auf **Attach Visual Studio Code**. Das ursprüngliche VS Code-Fenster kann geschlossen werden.
Nun kann mit der Entwicklung des Codes begonnen werden wie im Kapitel „Programmierung mit Visual Studio Code" beschrieben wurde. Um das Projekt zu kompilieren, kannst du in einem Terminal innerhalb von VS Code (Terminal → New Terminal) den nächsten Befehl eingeben:

```
catkin build
```

Durch einen **Rechtsklick auf das Terminal**, welches du innerhalb von VS Code geöffnet hast, und anschließendem Klick auf **Split Terminal** kannst du ein **weiteres Terminal** innerhalb von VS Code öffnen.

Möchtest du nun **CoppeliaSim starten**, gib den nachstehenden Befehl ein:

```
~/coppelia_sim/coppeliaSim.sh
```

Die Simulationsumgebung ist noch leer bzw. enthält keinen zu simulierenden Roboter.
Für die Verbindung mit dem RF Robot Control Framework ist es notwendig die **BlueZero** Schnittstelle von CoppeliaSim zu nutzen. Dafür kann der **BlueZero Server** wie folgend beschrieben gestartet werden:

- Klicke auf **Modules → Connectivity → B0 remote API server** und bestätige im sich öffnenden Fenster mit **Yes**

Nun können die **Roboter-Modelle** für die Simulation in CoppeliaSim ausgewählt werden.
Hinweis: Dafür bitte nicht die Modelle aus CoppeliaSim direkt auswählen, sondern die Modelle welche im Repository mitgeliefert werden. Ansonsten ist nicht garantiert, dass die Simulation einwandfrei funktioniert.
Es kann zwischen den Modellen des Franka Emika Panda und den Universal Roboter Modellen UR3 und UR5 gewählt werden.
Klicke auf **File → Load model...** und klicke dich zu dem Pfad der Modelle durch:

```
/root/catkin_ws/src/rf_coppeliasim_connector/models
```

Dort kann z.B. der Franka Emika Panda mit Greifern (engl. gripper) ausgewählt werden.
Das Modell des Franka Emika Panda mit Greifern sollte nun erfolgreich in die Simulationsumgebung von CoppeliaSim geladen worden sein.

Wir wechseln wieder zu **VS Code** und öffnen ein **weiteres Terminal** (Rechtsklick → Split Terminal). Wir sollten nun 3 Terminals innerhalb von VS Code offen haben:

1. Terminal zum Kompilieren des Codes via catkin build
2. Terminal zum Starten von CoppeliaSim via ~/coppelia_sim/coppeliaSim.sh
3. Terminal zum Starten des Hardware Interface und Controller Managers.

Das **Hardware Interface** und den **Control Manager** starten wir nun in unserem 3. Terminal mit dem folgenden Befehl (Info: Dabei wird auch der sog. roscore gestartet):

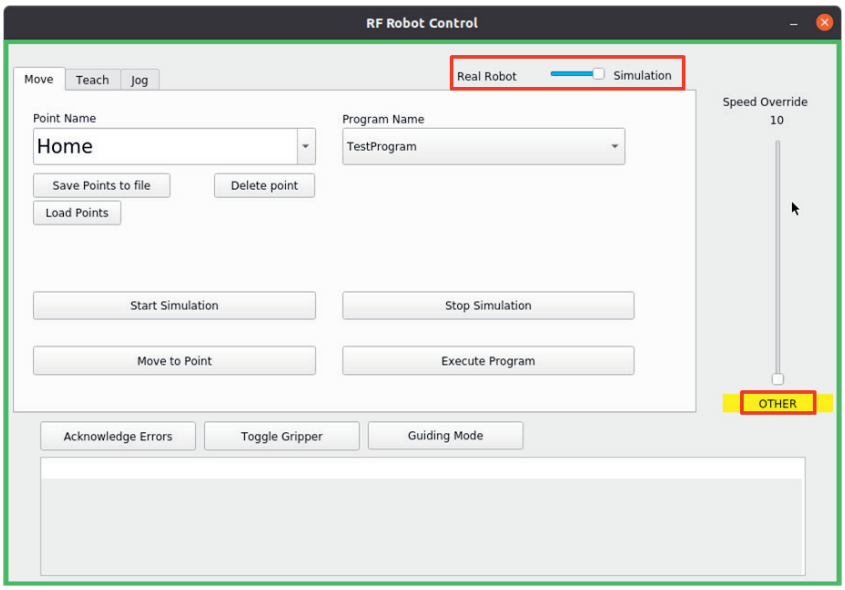
```
roslaunch rf_coppelia_interface panda.launch
```

Hinweis: Sollte zuvor ein Universal Roboter (Modell UR3 oder UR5) ausgewählt worden sein, muss der letzte Teil des Befehls universal.launch lauten.

Um nun das **GUI** von **RF Robot Control** zu öffnen, geben wir in ein 4. Terminal (Rechtsklick → Split Terminal) innerhalb von VS Code diesen Befehl ein:

```
rosrun rf_robot_control node
```

Sobald das GUI von RF Robot Control in der **Statusanzeige** den Modus **OTHER** (gelb) anzeigt, kann die Simulation gesteuert werden. Zunächst muss der Schieberegler von **Real Robot** auf **Simulation** umgestellt werden:

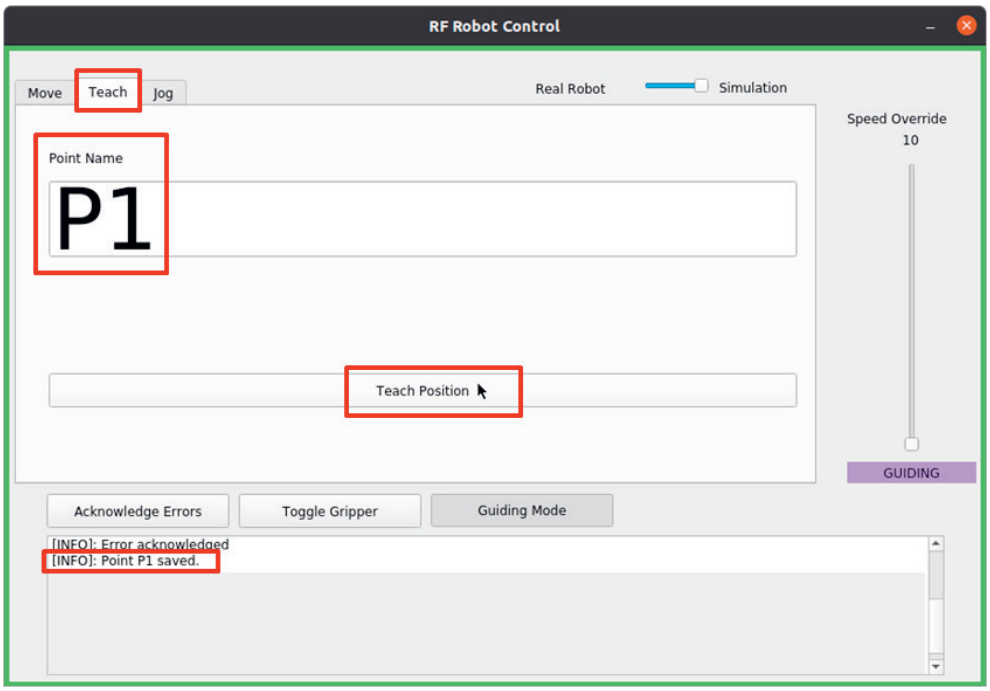


Die Benutzeroberfläche ist analog zum Modus Real Robot aufgebaut. Im Modus Simulation können wir:

- Punkte “teachen“
- den Roboter manuell bewegen bzw. „joggen“
- eingespeicherte Punkte via Move to Point anfahren
- Programme ausführen

- Wollen wir nun **Punkte** virtuell in der Simulationsumgebung „**teachen**“, dann können wir dazu die Registerkarte **Jog** aufrufen und anschließend auf den Button **Guiding Mode** klicken. Die Statusanzeige sollte nun den Modus **GUIDING** (violett) anzeigen.
- Durch Variation der **Schieberegler** für die verschiedenen Gelenke (engl. joints) können wir den Roboter **manuell führen** (engl. to guide). Die Gelenke sind ausgehend von der Roboterbasis bis zum Endeffektor von 1 bis 7 durchnummeriert.

- Das Speichern der Punkte erfolgt genau wie beim echten Roboter. Durch Klicken auf die Registerkarte **Teach** kommen wir in den Dialog zum Einspeichern des zuvor angefahrenen Punktes.
- Nach der Namensvergabe im Feld **Point Name** klicken wir auf den Button **Teach Position**.



In der Anzeige erhalten wir die Information, dass der Punkt P1 erfolgreich eingespeichert wurde. Schließlich können wir weitere Punkte (P2, P3 etc.) durch einen Wechsel zwischen den Registerkarten **Jog** und **Teach** speichern. Sind wir mit dem Einspeichern unserer Punkte fertig, können wir das GUI von RF Robot Control schließen und zu VS Code wechseln.
In VS Code können wir die zuvor gespeicherten Punkte für unseren **Code** verwenden.

Die Simulationsumgebung hat aktuell noch folgende Einschränkungen:

- Der Roboter verfügt über keine Nachgiebigkeit (engl. compliance) in bestimmte Raumrichtungen.
- Funktionen wie MoveToContact oder das Durchführen von Aufgaben mit definierten Kräften fehlen.
- Außerdem ist das Greifen mit einer bestimmten Kraft nicht unterstützt. Jedoch können wir den Greifer öffnen und schließen und so ein Objekt greifen.

Nachdem wir unseren Code geschrieben, gespeichert und kompiliert haben, können wir erneut das **GUI RF Robot Control** öffnen.
Dort wählen wir über den Schieberegler **Simulation** aus und klicken auf den Button **Start Simulation**. Das zuvor geschriebene Programm kann nun durch einen Klick auf den Button **Execute Program** gestartet und ausgeführt werden.
In **CoppeliaSim** kann der Code durch die Bewegungen des Roboters nachvollzogen werden.