Laboppgave

May 21, 2018

1 Introduksjon

Denne oppgaven er ment å gi en introduksjon til Robot Operating System (ROS), samt bruk av ROS med industriroboter.

1.1 KUKA KR-16

Dere skal bruke to KR-16 roboter fra KUKA. Disse robotene kan ha en nyttelast på 16 kg, og er markedsførst som fleksible roboter for bruk til oppgaver som sveising, lakkering, pakking og montering.

En av robotene er montert på en traverskran. Dette øker arbeidsområdet betraktelig, og gjør at man f.eks kan gripe rundt et objekt fra flere forskjellige vinkler uten å måtte flytte selve objektet.

I likhet med robotene i "Den andre labboppgaven" har robotene 6 ledd, hvor 3 av disse er samlet til en håndledd. Mer innformasjon om matematisk modellering og oppbygning av en robotarm finner dere i "Den andre labboppgaven".

Kontrollskapene for robotene ble designet i 2005, og har ganske begrenset regnekraft. Vi ønsker derfor å bruke et ekstern system for å planlegge og kontrollere robotbevegelsene. I "Den andre labboppgaven" vil dere får mulighet til å programmere direkte på kontrollskapet fra ABB, ved hjelp av en teach pendant. I denne oppgaven vil dere bruke ROS for å styre robotene fra en ekstern PC som kjører Linux.

sett inn navn på ABB labben

sett inn navn på ABB labben

sett inn navn på ABB labben

Har vi et ord for teach pendant på norsk?

1.1.1 Tekniske data

Axis	Range	Speed
A1	± 185°	156°/s
A2	+35° /- 155°	156°/s
A3	+154° /- 130°	156°/s
A4	± 350°	330°/s
A5	± 130°	330°/s
A6	± 350°	615°/s

Table 1: Teknisk data KR-16[1]

	X-axis	Y-axis	Z-axis
Max stroke	1500mm	4800mm	1500mm
Max velocity	0.5 m/s	0.83m/s	0.56m/s
Max acceleration	1 m/s^2	1m/s^2	1m/s^2
Calibration point offsets	1517.5mm	3700mm	-200mm
Repeatability		0.4mm	

Table 2: Teknisk data traverskran[2]

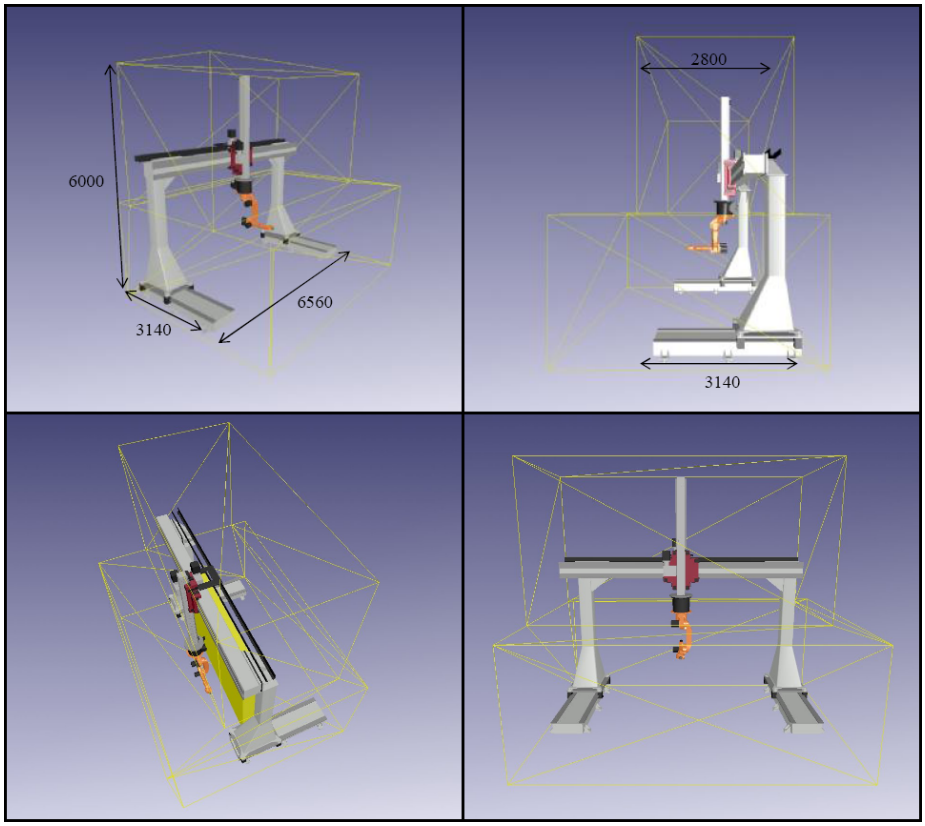

Figure 1: Dismensions for the gantry. Yellow lines marks available space[2]

1.2 Robot Operating System

ROS er et åpen-kildekode rammeverk for roboter. Det startet som et prosjekt i robotikkmiljøet ved Stanford University, og ble videreutviklet av Willow Garage fra 2009 - 2013. Pr juli 2016 har ROS støtte for over 100 roboter, og er tatt i bruk over hele verden[3].

De mest brukte programmeringsspråkene i ROS er C++ og Python, men det er også støtte for LISP, Java og LUA.

Noder i ROS er programmer som er laget for gjøre en spesifikk oppgave. Dette gjør at et system vanligvis består av flere noder, som samarbeider om å fullføre en kompleks oppgave. Figure 2 er et eksempel på et enkelt system for å lese leddvinkler fra en robot og lage en kinematisk modell av denne. Systemet består av flere noder, som er markert som elipser. *kuka_hardware_interface* står for kommunikasjonen mellom ROS og en fysiske roboten. *robot_state_publisher* regner ut alle transformasjonene som er nødvendig for å vise tilstanden til roboten.

Kommunikasjon mellom nodene skjer via spesialiserte meldinger over kommunikasjonskanaler kalt "topics". Disse bruker er satt opp slik at alle noder kan sende meldinger inn på en *topic*, eller abonnere på å få meldinger som er sendt til en *topic*. I figur 2 er hver *topic* markert i en firkant, og pilene viser hvilken vei meldingene går. Piler som peker fra en *node* til en *topic* marker at noden sender meldinger, mens en pil andre veien marker at en *node* lytter på en *topic*.

Unified Robot Description Format (URDF) er et XML format brukt for å beskrive roboter. Ved hjelp av elementer som lenker (links) og ledd (joints) kan man bygge komplekse modeller. En lenke kan være representert som en enkel geometrisk figur, eller bestå av mer komplekse 3D modeller. Dere kan finne URDF fila som beskriver en KR-16 robot på `https://github.com/itk-thrivaldi/kuka_experimental/blob/master/kuka_kr16_support/urdf/kr16_2.urdf`

Xacro er et XML makro språk som gjør det lettere å jobbe med store URDF modeller. I tilegg til en rekke innebygde funksjoner, har Xacro støtte for å inkludere andre filer. Dette gjør det enkelt å sette sammen komplekse modeller fra mindre filer. I denne labben skal dere blant annet bruke `https://github.com/itk-thrivaldi/thrivaldi_common/blob/master/robotlab_support/urdf/robotlab_macro.xacro`, som setter sammen flere mindre modeller til å bli robotene slik de står i labben.

rviz er et visualiseringsverktøy for ROS, som dere skal bruke for å se en 3D modell av labben, og teste bevegelsene før dere kjører de på robotene.

MoveIt! er et Scan'n'Plan rammeverk. Bruksområdet er typisk å scanne et arbeidsområde, detektere objekter for å så planlegge manipulering av objektene.

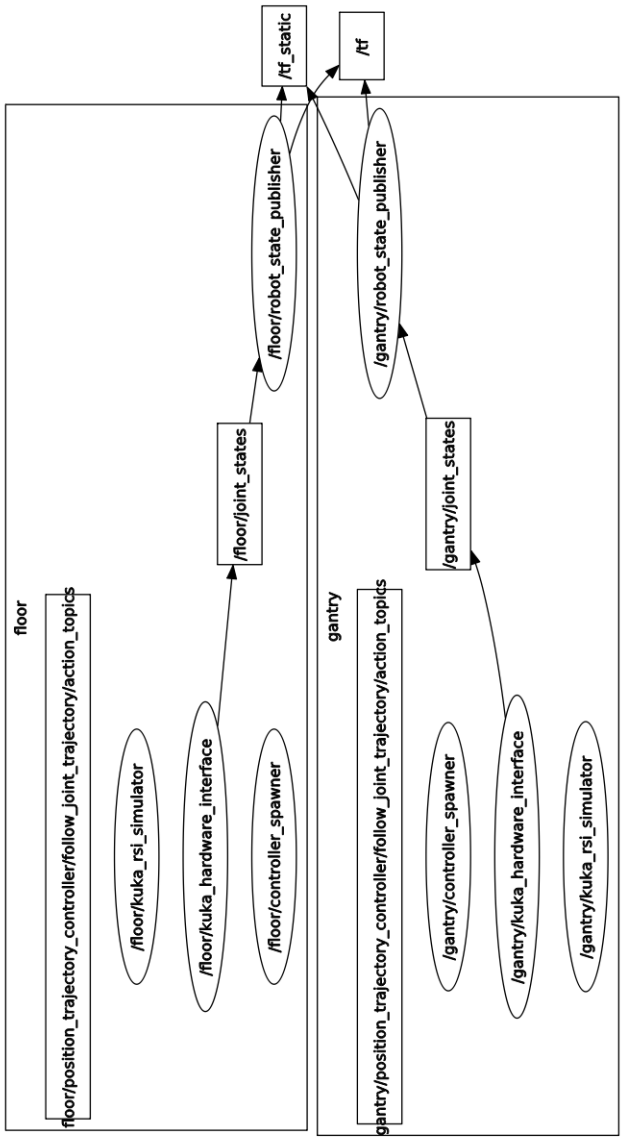

Figure 2: Example of nodes and topics in ROS

Vi skal ikke bruke MoveIt! for å kontrollere robotene, men dere kan kjøre MoveIt! med en simulert versjon av labben om dere ønsker det.

2 Utførelse på laben

Oppgaven delvis basert på ROS veiledningen[1], og dere vil kunne finne utdypende informasjon i den. Om dere ønsker så kan mesteparten av denne laboppgaven gjøres på en virtuell Ubuntu installasjon som dere kan kjøre på en egen PC. Innformasjon om oppsett og installasjon av denne finnes i ROS Industrial (ROS-I) innføringen[2]

Dere skal bruke en PC som kjører Linux (Ubuntu 16.04), og det meste dere skal gjøre kan og/eller må gjøres via en terminal. Denne finner dere i menyen på venstre side av skjermen.

Noen nyttige kommandoer

cd (change directory) Flytter dere til en annen katalog

ls (list) Lister alle filer og kataloger

pwd (print work directory) Gir dere stien til den katalogen dere er i nå

mkdir (make directory) Lager en katalog

rm (remove) Sletter en fil eller katalog

Noen nyttige ROS spesifikke kommandoer

roscd Som `cd` men tar inn ROS pakkenavn som argument

catkin_make Kompilerer ROS pakkene som er i et arbeidsområde

roslaunch Starter et ROS script

2.1 Sett opp arbeidsområde

For å holde orden på script og programmer som ikke er en del av den offisielle ROS distribusjon bruker vi Catkin. Catkin samler disse i et arbeidsområde (workspace) som inneholder kildekode og har støttefunksjoner for å kompilere kildekoden til kjørebare programmer. Vi skal først fjerne eventuelle rester etter forrige gruppe, før vi lager setter opp en nytt arbeidsområde som skal brukes i resten av labben.

Last inn standard ROS variabler:

```
source /opt/ros/kinetic/setup.bash
```

Slett arbeidsområdet til forrige gruppe hvis det finnes:

```
rm -rf ~/catkin_ws
```

[1] http://wiki.ros.org/ROS/Tutorials

[2] http://ros-industrial.github.io/industrial_training/_source/setup/PC-Setup---ROS-Kinetic.html

Lag en katalog til arbeidsområdet:

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

Bytt katalog til arbeidsområdet:

```
cd ~/catkin_ws
```

Lag nødvendig konfigurasjon for arbeidsområdet:

```
catkin_make
```

Last variabler fra det nyopprettede arbeidsområdet:

```
source devel/setup.bash
```

2.2 Installer ROS pakker for robotlaben

Vi må installere tre pakker for å kunne bruke ROS til å kontrollere robotene. To av disse inneholder modeller og URDF filer som beskriver labben, mens den siste er driveren som vi bruker for å kommunisere med kontrollskapene fra KUKA. Alle pakkene ligger på GitHub[3], som gjør at vi kan bruke gitfor å laste ned pakkene.

burde hatt med en beskrivelse av git

Bytt katalog til kildekode katalogen i arbeidsområdet:

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

Last ned `kuka_experimental` pakken fra GitHub:

```
git clone https://github.com/itk-thrivaldi/kuka_experimental
```

Last ned `thrivaldi_common` pakken fra GitHub:

```
git clone https://github.com/itk-thrivaldi/thrivaldi_common
```

Last ned `kuka_kvp_hw_interface` pakken fra GitHub:

```
git clone https://github.com/itk-thrivaldi/kuka_kvp_hw_interface
```

Bytt katalog til arbeidsområdet:

```
cd ~/catkin_ws
```

Installer evnetuelle manglende ROS pakker:

```
rosdep install --from-path src --ignore-src
```

Kompiler all kildekode i arbeidsområdet:

[3] `https://github.com/itk-thrivaldi`

```
catkin_make
```

For å være sikker på at ROS finner pakkene vi har installert må vi laste inn system-variablene på nytt:

```
source ~/catkin_ws/devel/setup.bash
```

2.3 Lag ROS pakke

For å kunne lage et eget program for å bevege robotene trenger vi først å lage en Catkin pakke.

Bytt katalog til kildekode katalogen i arbeidsområdet:

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

Bruk Catkin for å opprette en ny pakke:

```
catkin_create_pkg ttk4100 rospy std_msgs
```

For at ROS programmene skal finne pakken vår, må vi kompilere den og laste inn **Bytt katalog til arbeidsområdet:**

```
cd ~/catkin_ws
```

Kompiler all kildekode i arbeidsområdet:

```
catkin_make --pkg ttk4100
```

For å være sikker på at ROS finner pakkene vi har opprettet må vi laste inn system-variablene på nytt:

```
source devel/setup.bash
```

Vi kan nå bruke roscd for å bytte katalog til pakken vår:

```
roscd ttk4100
```

Mer info om å lage ROS pakker finnes på `http://wiki.ros.org/ROS/Tutorials/CreatingPackage`

2.4 Visualisering av labben

For å sjekke at URDF filene er korrekt installert, skal vi laste disse til en parametertjener, for å så visualisere modellen i rviz. Siden dette krever at flere ROS noder kjører samtidig, skal vi lage en `launch` fil som starter alle nodene for oss. Denne skal vi plassere i en underkatalog `launch` i pakken vår.

Opprett launch katalogen:

```
mkdir launch
```

Start et enkelt tekstbehandlingsprogram:

```
gedit launch/test.launch
```

Skriv eller kopier koden over til test.launch:

```
1 <?xml version="1.0"?>
2 <launch>
3   <include file="$(find␣robotlab_support)/launch/load_robotlab.
        ↪ launch" />
4   <node name="joint_state_publisher" pkg="joint_state_publisher"
        ↪ type="joint_state_publisher">
5     <param name="use_gui" value="true" />
6   </node>
7   <node name="robot_state_publisher" pkg="robot_state_publisher"
        ↪ type="robot_state_publisher"/>
8   <node name="rviz" pkg="rviz" type="rviz" args="-d␣$(find␣
        ↪ robotlab_support)/config/rviz.rviz" required="true" />
9 </launch>
```

Linje 3 laster URDF beskrivelsen av robotene til paramtertjeneren. Linje 4 og 7 starter støtteprogrammer som generer informasjonen om leddvinkler. Med en virkelig robot ville denne innformasjonen vært hentet fra roboten. Linje 8 starter rviz, som vi bruker for å gi en grafisk representasjon av robotene.

Etter å ha lagret fila, og lukket gedit kan vi kjøre den:

```
roslaunch ttk4100 test.launch
```

Hvis alt fungerer som det skal ser dere nå en grafisk representasjon av robotene, og en boks hvor dere kan sette verdien på leddene til robotene. Når dere forander på disse skal modellen oppdatere seg fortløpende.

Når dere er klar for å gå videre, kan dere trykke **ctrl+c** i terminalvinduet. Dette avslutter den kjørende prossesen.

2.5 Last ned filer

I tilegg til de tre pakkene som ble lastet ned fra GitHub i steg 2.2 er det gjort klar noen filer som skal brukes i pakken dere opprettet i steg 2.3. Denne inneholder koden som vi skal bruke for å bevege roboten, og inneholder kommentarer som forklarer hva vert steg gjør.

Last ned filer:

Her må dere finne ut hva som er beste måten å distribuere filene på

```
wget https://github.com/ivareri/ttk4100/archive/master.zip
```

Pakk ut de nedlastede filene:

Dette steget må tilpasses slik at de ender opp med script/ katalogen under roten på ttk4100 pakken

```
unzip master.zip
```

Dere har nå fått en ny katalog, `script` som inneholder filene vi skal bruke for å snu esken.

floor.py Inneholder navn på alle ledd, og forhåndsdefinerte posisjoner

gantry.py Samme som floor.py, men for Gantry roboten

__init__.py En tom fil. Den er nødvdening for at Python skal kunne importere andre filer fra katalogen

RobotActionClient.py Inneholder koden som sender posisjonene til ROS

node.py Den filen som dere skal kjøre. Den bruker RobotActionClient til å flytte robotene.

Sett rett flag på `node.py` slik at den er kjørbar:

```
chmod +x scripts/node.py
```

2.6 Simuler programmet

I dette delen trenger dere to terminaler for å kunne simulere labben samtidig som dere kjører programmet.

I terminal 1

```
roslaunch robotlab_support robot_state_visualize_robotlab.launch
```

I terminal 2

```
rosrun ttk4100 node.py
```

Hvis alt fungerer skal dere ha fått opp et rviz vindu med robotlabben. Når dere starter `node.py` skal robotene gå igjennom de samme beveglsene som vi ønsker å ha i labben. For å gjøre koden så enkel som mulig gjør vi ikke noe kollisjonsjekking. I det første steget vil robotene kollidere, men oppstartposisjonen i labben skal sørge for at det ikke skjer når dere kjører koden i lab.

Når programmet har kjørt ferdig, og studassen er fornøyd med bevegelsene kan dere lukke rviz og gå videre til å kjøre koden på robotene.

2.7 Kjør programmet på roboten

Vi må lage en ny launch fil for å starte robotdriveren og rviz. Denne skal være lik den forige dere lagde, bortsett fra at dere skal bytte ut `joint_state_publisher` seksjonen med robotdriveren. Det som skal ut er i den første kodeblokken, og det som skal inn er i den andre.

```
<node name="joint_state_publisher" pkg="joint_state_publisher"
    ↪ type="joint_state_publisher">
  <param name="use_gui" value="true" />
</node>
```

```
<include file="$(find␣robotlab_support)/launch/
    ↪ robot_interface_streaming_robotlab.launch" >
  <arg name="kvp_floor" value="true" />
  <arg name="kvp_gantry" value="true"/>
</include>
```

I terminal 1, bytt ut robot.launch med navnet på filen dere lagde over

```
roslaunch ttk4100 robot.launch
```

I terminal 2

```
rosrun ttk4100 node.py
```

References

[1] *Technical data kr 16*, KUKA Roboter GmbH, Mar. 2004.

[2] P. Liljebäck, T. Mugaas, T. Kavli, A. Ferber, H. Schumann-Olsen, G. Johansen, and A. A. Transeth, *Design document for robotic lab*, version Rev 1.2.2, SINTEF ICT, Jul. 28, 2008.

[3] T. Foote, *Ros community metrics report*, http://download.ros.org/downloads/metrics/metrics-report-2016-07.pdf, Jul. 2016.