ROS Workshop

Ascend NTNU

September 2017

1 Introduksjon

Samarbeid mellom gruppene er svært viktig for at vi skal klare å løse oppdraget. Å designe gode grensesnitt mellom systemene blir derfor svært viktig og det er det denne workshopen handler om! Målet i dag er at hver gruppe skal lage svært forenklede versjoner av systemene til Control, Perception og AI, og få en simulert drone til å fly gjennom fire punkter.

2 Gruppene

Gruppene er satt sammen av medlemmer på tvers av gruppene. Tanken er at hver gruppe skal ha sin egen perception, control og AI gruppe som lager hver sine ROS noder - slik vi gjør det i Ascend.

3 Oppgaven

Utfordringen dere står ovenfor er å få en simulert drone til å fly gjennom 4 forhåndsdefinerte punktet - autonomt. Vedlagt ligger kode som gir et svært enkelt grensesnitt mot den simulerte dronen. Tanken er ikke at dere skal bruke mye tid på logikken til styring av dronen, men heller bruke tiden på å lage gode grensesnitt innad i gruppen. Her kreves det mye samarbeid og det er viktig at gruppen blir enig om et grensesnitt før dere starter å skrive kode!

3.1 Control

Control gruppemedlemmene skal lage en ROS-node som tar inn posisjonsdata og en ønsket posisjon, og skal få dronen til å fly til den posisjonen. Control har også ansvar for takeoff og evt landing. I den vedlagte koden ligger C++ klassen ControlDrone som tar seg av all kommunikasjon med den simulerte dronen. Klassen har 3 funksjoner som kan være nyttig.

Table 1: Nyttige funksjoner for Control i ControlDrone klassen

Navn	**Beskrivelse**
void takeoff()	Får dronen til å ta av til 2 meters høyde
void land()	Lander på nåværende posisjon
void setTarget(x, y, z)	Setter en posisjon som dronen flyr til

Husk at dronen trenger litt tid til å utføre kommandoene!

3.2 Perception

Perception sin oppgave er å hente ut posisjonsdataen fra dronen og gjøre den tilgjengelig for de andre. Dronen er simulert med GPS, så dronen har sitt eget posisjonsestimat vi kan bruke når vi simulerer! I den vedlagte koden ligger C++ klassen PerceptionDrone som tar seg av henting av data fra dronen. Klassen har egentlig kun en nyttig funksjon (NB: Bruker C++11 STL std::array)

Table 2: Nyttige funksjoner for Perception i PerceptionDrone klassen

Navn	**Beskrivelse**
std::array getPosition()	Returnerer float array med x, y og z posisjon

Oppgaven til Perception blir da å hente ut posisjonsestimatet, og sende posisjonen ut med en fast frekvens.

3.3 AI

AI gruppen skal bestemme hva målet til dronen skal være. Ettersom dronen skal fly i gjennom 4 punkter er det AI sin oppgave å holde styr på hvilket punkt dronen skal prøve å komme seg til. Det innebærer også å finne ut når dronen er nærme nok punktet til at vi kan si at punktet er nådd. Husk at posisjonsdataen ikke er perfekt, og dronen klarer heller aldri å holde posisjonen helt perfekt pga simulert sensorstøy. Dere bestemmer selv hvilke punkter dronen skal fly igjennom!

4 Oppsett av ROS workspace for oppgaven

For å ikke bruke for mye tid på unødvendige ting, er hele den vedlagte mappen en ROS pakke som kan kopieres direkte inn i deres catkin workspace. Hele

byggesystemet er satt opp allerede, og det ligger 3 kildekodefiler i src mappen, en for hver undergruppe. Hver av de 3 kildekodefilene er satt opp til å bygges som separate ROS noder. Å lage en ny ROS pakke kan i starten være litt komplisert, men dette er noe strengt tatt ikke alle trenger å forholde seg til.

Ettersom flere skal jobbe i samme ROS pakke, anbefales det å utnytte det dere lærte på git kurs. Alle undergruppe burde derfor lage en egen branch og utvikle fra den. Da unngår dere masse trøbbel ved kompilering!

5 Simulatoren

Simulatoren ligger fritt tilgjengelig på nett, men den krever en del oppsett og en del prosessorkraft. En PC kommer til å bli satt opp med simulatoren for de som ønsker å teste underveis!

6 Ulike typer grensesnitt i ROS

6.1 Messages (.msg)

Meldinger i ROS skal brukes til kontinuerlige strømmer av data. Når en melding blir sendt fra en node er det "fire and forget", altså noden bryr seg egentlig ikke om meldingen kom frem eller ikke. En melding kan gå tapt, så da er det viktig at det kommer ny data rett etterpå. De andre grensesnittene i ROS bygger videre på meldinger for å dekke andre bruksområder. Innebygd i ROS finnes det en haug med forskjellige meldingstyper (altså typen data meldingen inneholder), og det er også mulig å lage egne typer.

6.2 Services (.srv)

Services brukes når en node vil sende en forespørsel til en annen node. Med en service vil den ene noden vente på å få et svar fra en annen node, og konseptet er egentlig helt likt som med webservere der klienter sender en forespørsel til en server og får et svar i respons. Med en service er det forventet å få et svar ganske umiddelbart!

6.3 Actions (.action)

Action bygger videre på services, men er ment å brukes til operasjon som tar lengre tid å utføre. En action består av en forespørsel/kommando, feedback underveis mens systemet utfører kommandoen og til slutt er resultat. F.eks kan en kommando være at dronen skal reise fra A til B, feedback kan være posisjonen underveis og resultatet kan være om dronen nådde frem eller ikke.

7 ROS i C++

7.1 ROS node som ikke gjør noe:

```cpp
#include <ros/ros.h>

int main(int argc, char** argv) {
    ros::init(argc, argv, "node_name");
    ros::NodeHandle n; //Håndterer all kommunikasjon med ROS
    ros::spin(); //Kjører en uendelig loop og henter meldinger
    return 0;
}
```

7.2 ROS node som publiserer enkel data med fast frekvens:

```cpp
#include <ros/ros.h>
#include <std_msgs/Int32.h> //Del av ROS pakken std_msgs

int main(int argc, char** argv) {
    ros::init(argc, argv, "node_name");
    //Håndterer all kommunikasjon med ROS
    ros::NodeHandle n;
    //Setter en "publisher"
    ros::Publisher pub = n.advertise<std_msgs::Int32>("et/topic", 10);
    ros::Rate loop_rate(2); //2 Hz
    //Kjør frem til ROS avslutter
    while(ros::ok()) {
        //Meldingen som skal sendes - klassen er autogenerert av catkin
        std_msgs::Int32 msg;
        //Fyll inn datafeltene til meldingen - her heter variablen data
        msg.data = 5;
        //Publiser meldingen via pub
        pub.publish(msg);
        //La ROS oppdatere status, hente og sende meldinger
        ros::spinOnce();
        //Vent resterende tid for å oppnå 2Hz
        loop_rate.sleep();
    }
    return 0;
}
```

7.3 ROS Service server

```cpp
#include <ros/ros.h>
#include <ros_workshop/AddTwoInts.h>

using Request = ros_workshop::AddTwoInts::Request;
using Response = ros_workshop::AddTwoInts::Response;

//Når en klient kaller
bool callback(Request& req, Response& resp ) {}
    resp.ans = req.a + req.b;
    return true;
}

int main(int argc, char** argv) {
    ros::init(argc, argv, "node_name");
    //Håndterer all kommunikasjon med ROS
    ros::NodeHandle n;
    //Setter opp en service server
    ros::ServiceServer server = n.advertiseService("service_name", callback);
    //Kjører i en evig loop frem til noden avsluttes av ROS
    ros::spin();
}
```

7.4 ROS Service server

```
#include <ros/ros.h>
#include <ros_workshop/AddTwoInts.h>

int main(int argc, char** argv) {
    ros::init(argc, argv, "node_name");
    //Håndterer all kommunikasjon med ROS
    ros::NodeHandle n;
    //Setter opp en service server
    ros::ServiceClient client = n.serviceClient("service_name");

    //Setter opp en request
    ros_workshop::AddTwoInts clientCall;
    clientCall.request.a = 5;
    clientCall.request.b = 10;
    //Kaller servicen
    if(client.call(clientCall)) {
        int value = clientCall.response.ans;
        ROS_INFO("Value is: %i", value);
    } else {
        ROS_WARN("Error!");
    }
}
```