ANRO

Laboratorium 2

Grupa pierwsza

Robot nr 2

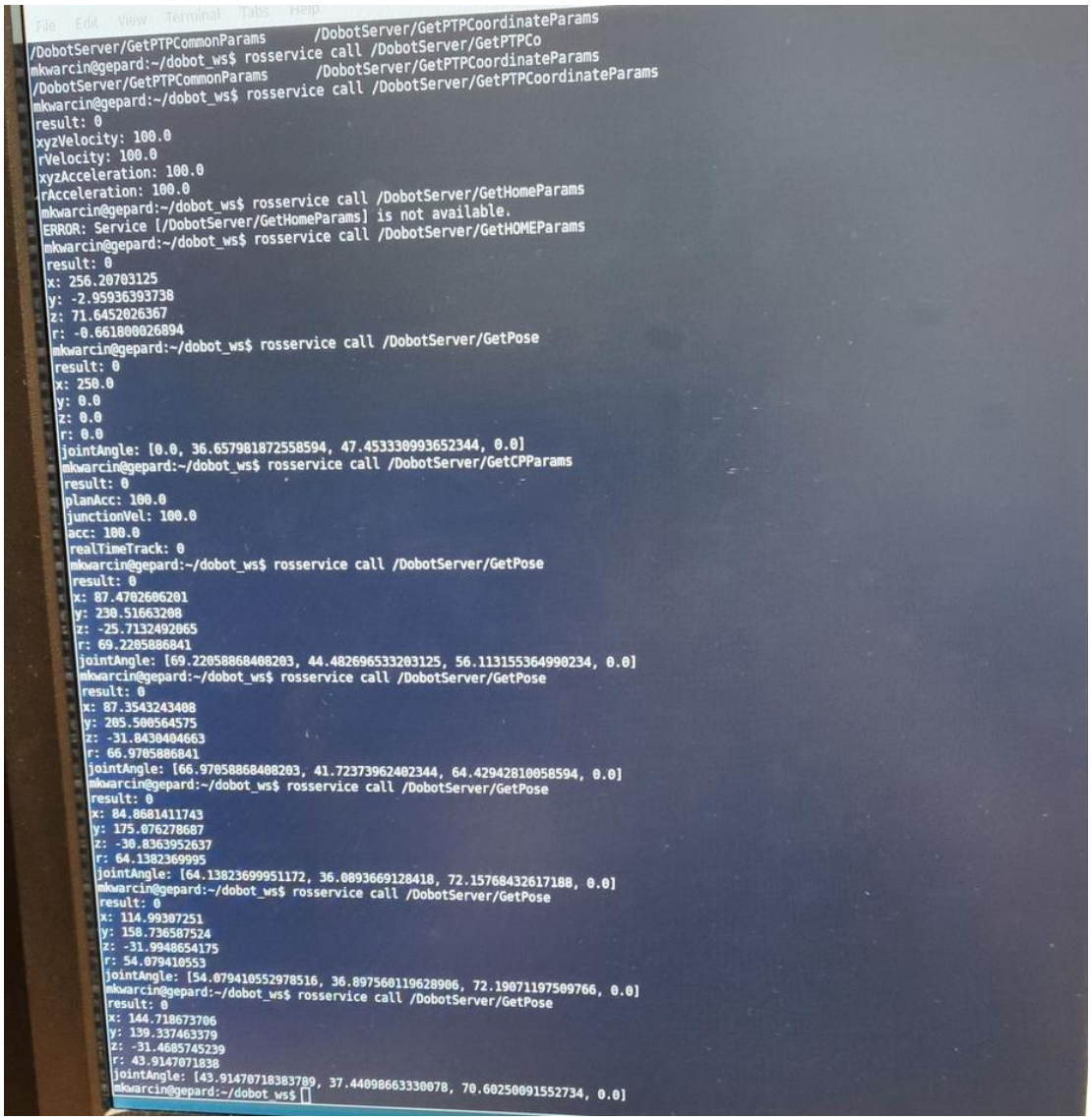
Zespół:

Michał Kwarciński

Kacper Marchlewicz

Zadanie 1 – Komunikacja z robotem przez terminal – zastosowanie poznanych poleceń ROS

Korzystając z serwisu SetHOMECmd wykonaliśmy procedurę bazowania robota. Następnie uruchomiliśmy serwis GetPTPCoordinateParams i poruszaliśmy ramieniem robota w różne strony. Otrzymaliśmy następujące koordynaty:



Zadanie 2 – Pick&Place z wykorzystaniem ROS (napisanie własnego węzła)

Zamontowaliśmy na ramienia chwytak dwupalczasty. Po krótkich eksperymentach zdecydowaliśmy się na ruch PTP Jump. Pozwala on na uniknięcie sytuacji, w której chwytak zahaczał o kostkę w trakcie przemieszczania się. Korzystając ze skryptu DobotClient_PTP.cpp stworzyliśmy kod wykonujący zadanie. Dodaliśmy wykorzystanie serwisów SetEndEffectorGripper w celu kontroli chwytaka i SetWAITCmd aby zapewnić odpowiednią dla robota przerwę w kolejności wykonywania zadań. Do każdego z dodanych serwisów musieliśmy zrobić nowych klientów. Robot zaczyna i kończy pracę z otwartym chwytakiem.

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "dobot/SetCmdTimeout.h"
#include "dobot/SetQueuedCmdClear.h"
#include "dobot/SetQueuedCmdStartExec.h"
#include "dobot/SetQueuedCmdForceStopExec.h"
#include "dobot/GetDeviceVersion.h"

#include "dobot/SetEndEffectorParams.h"
#include "dobot/SetPTPJointParams.h"
#include "dobot/SetPTPCoordinateParams.h"
#include "dobot/SetPTPJumpParams.h"
#include "dobot/SetPTPCommonParams.h"
#include "dobot/SetPTPCmd.h"
#include "dobot/SetEndEffectorGripper.h"
#include "dobot/SetWAITCmd.h"

int counter = 0;

int main(int argc, char **argv)
{
  int wait_time = 500;
  ros::init(argc, argv, "DobotClient");
  ros::NodeHandle n;

  ros::ServiceClient client;

  // SetCmdTimeout
  client = n.serviceClient<dobot::SetCmdTimeout>("/DobotServer/SetCmdTimeout");
  dobot::SetCmdTimeout srv1;
  srv1.request.timeout = 3000;
  if (client.call(srv1) == false) {
    ROS_ERROR("Failed to call SetCmdTimeout. Maybe DobotServer isn't started yet!");
    return -1;
  }

  // Clear the command queue
  client = n.serviceClient<dobot::SetQueuedCmdClear>("/DobotServer/SetQueuedCmdClear");
  dobot::SetQueuedCmdClear srv2;
  client.call(srv2);

  // Start running the command queue
  client = n.serviceClient<dobot::SetQueuedCmdStartExec>("/DobotServer/SetQueuedCmdStartExec");
  dobot::SetQueuedCmdStartExec srv3;
```

```
  client.call(srv3);

  // Get device version information
  client = n.serviceClient<dobot::GetDeviceVersion>("/DobotServer/GetDeviceVersion");
  dobot::GetDeviceVersion srv4;
  client.call(srv4);
  if (srv4.response.result == 0) {
    ROS_INFO("Device version:%d.%d.%d", srv4.response.majorVersion, srv4.response.minorVersion,
srv4.response.revision);
  } else {
    ROS_ERROR("Failed to get device version information!");
  }

  //Set end effector parameters
  ros::ServiceClient GripperClient;
  GripperClient = n.serviceClient<dobot::SetEndEffectorGripper>("/DobotServer/SetEndEffectorGripper");
  // for open
  dobot::SetEndEffectorGripper srvopen;
  srvopen.request.enableCtrl = 1;
  srvopen.request.grip = 0;
  srvopen.request.isQueued = true;

  // for close
  dobot::SetEndEffectorGripper srvclose;
  srvclose.request.enableCtrl = 1;
  srvclose.request.grip = 1;
  srvclose.request.isQueued = true;



  // Wait canfigurator
  ros::ServiceClient waiterclient;
  waiterclient = n.serviceClient<dobot::SetWAITCmd>("/DobotServer/SetWAITCmd");
  dobot::SetWAITCmd srvWait;
  srvWait.request.timeout = wait_time;
  srvWait.request.isQueued = true;
  waiterclient.call(srvWait);

  // Set Gripper
  client = n.serviceClient<dobot::SetQueuedCmdStartExec>("/DobotServer/SetQueuedCmdStartExec");
  // Set PTP joint parameters
  do {
    client = n.serviceClient<dobot::SetPTPJointParams>("/DobotServer/SetPTPJointParams");
    dobot::SetPTPJointParams srv;

    for (int i = 0; i < 4; i++) {
      srv.request.velocity.push_back(100);
    }
    for (int i = 0; i < 4; i++) {
      srv.request.acceleration.push_back(100);
    }
    client.call(srv);
  } while (0);

  // Set PTP coordinate parameters
  do {
    client = n.serviceClient<dobot::SetPTPCoordinateParams>("/DobotServer/SetPTPCoordinateParams");
```

```
    dobot::SetPTPCoordinateParams srv;

    srv.request.xyzVelocity = 100;
    srv.request.xyzAcceleration = 100;
    srv.request.rVelocity = 100;
    srv.request.rAcceleration = 100;
    client.call(srv);
} while (0);

// Set PTP jump parameters
do {
    client = n.serviceClient<dobot::SetPTPJumpParams>("/DobotServer/SetPTPJumpParams");
    dobot::SetPTPJumpParams srv;

    srv.request.jumpHeight = 50;
    srv.request.zLimit = 200;
    client.call(srv);
} while (0);

// Set PTP common parameters
do {
    client = n.serviceClient<dobot::SetPTPCommonParams>("/DobotServer/SetPTPCommonParams");
    dobot::SetPTPCommonParams srv;

    srv.request.velocityRatio = 50;
    srv.request.accelerationRatio = 50;
    client.call(srv);
} while (0);

client = n.serviceClient<dobot::SetPTPCmd>("/DobotServer/SetPTPCmd");
dobot::SetPTPCmd srv;

while (true)
{
    GripperClient.call(srvopen);

    srv.request.ptpMode = 0;
    srv.request.x = 200;
    srv.request.y = 55;
    srv.request.z = -20;
    srv.request.r = 15;
    client.call(srv);

    waiterclient.call(srvWait);

    GripperClient.call(srvclose);

    waiterclient.call(srvWait);

    srv.request.x = 210;
    srv.request.y = -75;
    srv.request.z = -20;
    client.call(srv);

    GripperClient.call(srvopen);

    waiterclient.call(srvWait);
```

```
      srvopen.request.enableCtrl = 0;
      GripperClient.call(srvopen);
      break;

    }
    return 0;
}
```

Zadanie 3 – Budowa wieży o zadanej wysokości

Zmodyfikowaliśmy kod z zadania drugiego. Dodaliśmy możliwość uruchomienia pliku o zadanym parametrze opisującym wysokość wieży. Zależnie od wartości kończy działanie programu w odpowiednim miejscu.

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "dobot/SetCmdTimeout.h"
#include "dobot/SetQueuedCmdClear.h"
#include "dobot/SetQueuedCmdStartExec.h"
#include "dobot/SetQueuedCmdForceStopExec.h"
#include "dobot/GetDeviceVersion.h"

#include "dobot/SetEndEffectorParams.h"
#include "dobot/SetPTPJointParams.h"
#include "dobot/SetPTPCoordinateParams.h"
#include "dobot/SetPTPJumpParams.h"
#include "dobot/SetPTPCommonParams.h"
#include "dobot/SetPTPCmd.h"
#include "dobot/SetEndEffectorGripper.h"
#include "dobot/SetWAITCmd.h"

int counter = 0;

int main(int argc, char **argv)
{
    int wait_time = 500;
    ros::init(argc, argv, "DobotClient");
    ros::NodeHandle n = ros::NodeHandle("~");

    int parametr = 0;
    n.getParam("height", parametr);
    std::cout << "Tower height: " << parametr << std::endl;
    ros::ServiceClient client;

    // SetCmdTimeout
    client = n.serviceClient<dobot::SetCmdTimeout>("/DobotServer/SetCmdTimeout");
    dobot::SetCmdTimeout srv1;
    srv1.request.timeout = 3000;
    if (client.call(srv1) == false) {
        ROS_ERROR("Failed to call SetCmdTimeout. Maybe DobotServer isn't started yet!");
        return -1;
    }

    // Clear the command queue
    client = n.serviceClient<dobot::SetQueuedCmdClear>("/DobotServer/SetQueuedCmdClear");
    dobot::SetQueuedCmdClear srv2;
```

```
  client.call(srv2);

  // Start running the command queue
  client = n.serviceClient<dobot::SetQueuedCmdStartExec>("/DobotServer/SetQueuedCmdStartExec");
  dobot::SetQueuedCmdStartExec srv3;
  client.call(srv3);

  // Get device version information
  client = n.serviceClient<dobot::GetDeviceVersion>("/DobotServer/GetDeviceVersion");
  dobot::GetDeviceVersion srv4;
  client.call(srv4);
  if (srv4.response.result == 0) {
    ROS_INFO("Device version:%d.%d.%d", srv4.response.majorVersion, srv4.response.minorVersion,
srv4.response.revision);
  } else {
    ROS_ERROR("Failed to get device version information!");
  }

  //Set end effector parameters
  ros::ServiceClient GripperClient;
  GripperClient = n.serviceClient<dobot::SetEndEffectorGripper>("/DobotServer/SetEndEffectorGripper");
  // for open
  dobot::SetEndEffectorGripper srvopen;
  srvopen.request.enableCtrl = 1;
  srvopen.request.grip = 0;
  srvopen.request.isQueued = true;

  // for close
  dobot::SetEndEffectorGripper srvclose;
  srvclose.request.enableCtrl = 1;
  srvclose.request.grip = 1;
  srvclose.request.isQueued = true;



  // Wait canfigurator
  ros::ServiceClient waiterclient;
  waiterclient = n.serviceClient<dobot::SetWAITCmd>("/DobotServer/SetWAITCmd");
  dobot::SetWAITCmd srvWait;
  srvWait.request.timeout = wait_time;
  srvWait.request.isQueued = true;
  waiterclient.call(srvWait);

  // Set Gripper
  client = n.serviceClient<dobot::SetQueuedCmdStartExec>("/DobotServer/SetQueuedCmdStartExec");
  // Set PTP joint parameters
  do {
    client = n.serviceClient<dobot::SetPTPJointParams>("/DobotServer/SetPTPJointParams");
    dobot::SetPTPJointParams srv;

    for (int i = 0; i < 4; i++) {
      srv.request.velocity.push_back(100);
    }
    for (int i = 0; i < 4; i++) {
      srv.request.acceleration.push_back(100);
    }
    client.call(srv);
```

```
} while (0);

// Set PTP coordinate parameters
do {
   client = n.serviceClient<dobot::SetPTPCoordinateParams>("/DobotServer/SetPTPCoordinateParams");
   dobot::SetPTPCoordinateParams srv;

   srv.request.xyzVelocity = 100;
   srv.request.xyzAcceleration = 100;
   srv.request.rVelocity = 100;
   srv.request.rAcceleration = 100;
   client.call(srv);
} while (0);

// Set PTP jump parameters
do {
   client = n.serviceClient<dobot::SetPTPJumpParams>("/DobotServer/SetPTPJumpParams");
   dobot::SetPTPJumpParams srv;

   srv.request.jumpHeight = 50;
   srv.request.zLimit = 200;
   client.call(srv);
} while (0);

// Set PTP common parameters
do {
   client = n.serviceClient<dobot::SetPTPCommonParams>("/DobotServer/SetPTPCommonParams");
   dobot::SetPTPCommonParams srv;

   srv.request.velocityRatio = 50;
   srv.request.accelerationRatio = 50;
   client.call(srv);
} while (0);

client = n.serviceClient<dobot::SetPTPCmd>("/DobotServer/SetPTPCmd");
dobot::SetPTPCmd srv;

while (true)
{
   GripperClient.call(srvopen);

   srv.request.ptpMode = 0;
   srv.request.x = 200;
   srv.request.y = 55;
   srv.request.z = 20;
   srv.request.r = 15;
   client.call(srv);

   waiterclient.call(srvWait);

   GripperClient.call(srvclose);

   waiterclient.call(srvWait);

   srv.request.x = 210;
   srv.request.y = -75;
   srv.request.z = -20;
```

```
client.call(srv);

waiterclient.call(srvWait);

GripperClient.call(srvopen);
if(parametr == 1)
{
   srv.request.x = 200;
   srv.request.y = 55;
   srv.request.z = 100;
   client.call(srv);

   srvopen.request.enableCtrl = 0;
   GripperClient.call(srvopen);
   break;
}
srv.request.x = 200;
srv.request.y = 55;
srv.request.z = 0;
client.call(srv);

waiterclient.call(srvWait);

GripperClient.call(srvclose);

waiterclient.call(srvWait);

srv.request.x = 210;
srv.request.y = -75;
srv.request.z = 0;
client.call(srv);

waiterclient.call(srvWait);

GripperClient.call(srvopen);

if(parametr == 2)
{
   srv.request.x = 200;
   srv.request.y = 55;
   srv.request.z = 100;
   client.call(srv);

   srvopen.request.enableCtrl = 0;
   GripperClient.call(srvopen);
   break;
}
srv.request.x = 200;
srv.request.y = 55;
srv.request.z = -20;
client.call(srv);

waiterclient.call(srvWait);

GripperClient.call(srvclose);

waiterclient.call(srvWait);
```

```
      srv.request.x = 210;
      srv.request.y = -75;
      srv.request.z = 20;
      client.call(srv);

      GripperClient.call(srvopen);

      waiterclient.call(srvWait);

      if(parametr == 3)
      {
        srv.request.x = 200;
        srv.request.y = 55;
        srv.request.z = 100;
        client.call(srv);

        srvopen.request.enableCtrl = 0;
        GripperClient.call(srvopen);
        break;
      }
    }
    return 0;
}
```

Pozostałe uwagi i problemy

Zabolała nas nieznajomość gitlaba. Poprosiliśmy kolegę o pomoc, co okazało się totalną katastrofą. Wystąpił błąd, przez który został usunięty cały folder dobot. Poskutkowało to godzinnym opóźnieniem, gdyż musieliśmy od początku rozpocząć konfigurację folderu dobot. To wydarzenie dało nam świetną okazję do nauki gitlaba w praktyce.

Link do filmu:

https://1drv.ms/u/s!AkMYzt8ywVnohkg4K6ZJ-urjB9v6?e=V4zblE

Pytania Kontrolne

Czy ROS jest językiem programowania?

Nie, jest on platformą programistyczną, zbiorem pakietów i narzędzi do tworzenia oprogramowania robotów.

Wymień podstawowe elementy struktury ROS.

Węzeł – fragment kodu korzystający z ROS

Temat – mechanizm przesyłania wiadomości pomiędzy węzłami

Usługa – mechanizm komunikacji między węzłami w której jeden węzeł wysyła zlecenie do drugiego i otrzymuje odpowiedź zwrotną

Akcja - kombinacja wielu usług, składa się z celu, sprzężenia i rezultatu, może być wykonana wielokrotnie

Przy komunikacji klient-serwer, ile może być serwerów dla tego samego serwisu?

Serwer może być tylko jeden, lecz może posiadać wiele klientów.

Jaka jest podstawowa różnica pomiędzy akcją a serwisem (usługą)?

Akcja może przychodzić wielokrotnie, wykorzystuje sprzężenie zwrotne. Usługa natomiast wysyła raz zlecenie i oczekuje odpowiedzi.