6. pielikums

Eksperimentu stenda datorprogrammas TCP servera kods

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace UR10
{
   public interface ITcpServerOut
   {
      void CommandFinished();
   }

   public interface ITcpServer
   {
      void StartTcpServer();
      void StopTcpServer();
      void SendMoveP(float[] position, float speed);
      void SendDigitalOutOn();
      void SendDigitalOutOff();
      void SendAnalogOutOn(float voltage);
      void SendAnalogOutOff();
      void SendStopL();
      void WaitForTcpClient();
      void SendMoveL(float[] position, float speed);
      void StorePositionPoint(float[] position);
      void ExecuteStoredMoveP();
      void ExecuteStoredMoveL();
      void SetPatternSpeed(float speed);
   }

   class TcpServer : ITcpServer
   {
      bool stopServer = false;
      TcpListener server = new TcpListener(IPAddress.Any, 888);
      NetworkStream stream;
      ITcpServerOut serverUser;
      TcpClient client;

      public TcpServer(ITcpServerOut serverUser)
      {
         this.serverUser = serverUser;
      }

      public void StartTcpServer()
      {
         server.Start();

         while(!stopServer)
         {
            client = server.AcceptTcpClient();
            stream = client.GetStream();

            while (client.Connected)
            {
               byte[] arrayBytesRequest = new byte[client.Available];
               int nRead = stream.Read(arrayBytesRequest, 0, arrayBytesRequest.Length);
```

```
            if (nRead > 0)
            {
                string message = Encoding.ASCII.GetString(arrayBytesRequest);

                if(message == "command_finished")
                {
                    serverUser.CommandFinished();
                }
            }
            else
            {
                if (client.Available == 0)
                {
                    stream.Close();
                }
            }

            Thread.Sleep(1);
        }

        Thread.Sleep(1);
    }
}

public void WaitForTcpClient()
{
    while(client == null)
    {
        Thread.Sleep(1000);
    }
}

public void StopTcpServer()
{
    stopServer = false;
    server.Stop();
}

public void SendMoveP(float[] position, float speed)
{
    byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.MoveP},{string.Join(",", position)},{speed})\n");
    stream.Write(data, 0, data.Length);
}

public void SendMoveL(float[] position, float speed)
{
    byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.MoveL},{string.Join(",", position)},{speed})\n");
    stream.Write(data, 0, data.Length);
}

public void SendDigitalOutOn()
{
    byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.DigitalOut_0},1)\n");
    stream.Write(data, 0, data.Length);
}

public void SendDigitalOutOff()
{
    byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.DigitalOut_0},0)\n");
    stream.Write(data, 0, data.Length);
}

public void SendAnalogOutOn(float voltage)
{
    byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.AnalogOut_0},{voltage * 0.1})\n");
    stream.Write(data, 0, data.Length);
}

public void SendAnalogOutOff()
```

```
        {
            byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.AnalogOut_0},0)\n");
            stream.Write(data, 0, data.Length);
        }

        public void SendStopL()
        {
            byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.StopL})\n");
            stream.Write(data, 0, data.Length);
        }

        public void StorePositionPoint(float[] position)
        {
            byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.StorePositionPoint},{string.Join(",", position)})\n");
            stream.Write(data, 0, data.Length);
        }

        public void ExecuteStoredMoveP()
        {
            byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.ExecuteStoredMoveP})\n");
            stream.Write(data, 0, data.Length);
        }

        public void ExecuteStoredMoveL()
        {
            byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.ExecuteStoredMoveL})\n");
            stream.Write(data, 0, data.Length);
        }

        public void SetPatternSpeed(float speed)
        {
            byte[] data = Encoding.ASCII.GetBytes($"({(float)CommandType.SetPatternSpeed},{speed})\n");
            stream.Write(data, 0, data.Length);
        }

        private enum CommandType
        {
            MoveP,
            DigitalOut_0,
            AnalogOut_0,
            StopL,
            MoveL,
            StorePositionPoint,
            ExecuteStoredMoveP,
            SetPatternSpeed,
            ExecuteStoredMoveL
        }
    }
}
```