Open Robot Control and Simulation Library

– ORCS –

ORCS Direct Control Protocol (ODCP)

Controlling Agents with Direct UDP Commands

Version 1.0
(September 2008)

Christian Rempis
Verena Thomas

Contents

1 ORCS Direct Control Protocol

The ORCS direct control protocol (ODCP) is a low-weight UDP based network interface. For each simulated agent an ODCP server can be started to provide a UDP socket which allows the direct control of the agent by external clients.

The current ODCP 1.0 protocol version is simple to implement as it supports only four commands:

- Connect to the ODCP server of an agent. This notifies the simulator that a client is currently using the interface.
- Disconnect from the ODCP server. Notifies the simulator that a client is not interested in using the interface any more.
- Send motor settings. These settings are applied to the simulator during the next simulation step and kept until another command or controller changes these motor settings.
- Request motor sensor data. The simulator sends a UDP message containing all angle sensors of the motors.

In difference to the ORCS OSP 1.1 protocol, the ODCP protocol is not synchronized. Motor settings are applied when a UDP packet is received, independently of the current simulation step. Therefore motor settings can be applied very irregularly, messages can get lost or be received too late. There is also no protection against multiple controllers, so an agent might be controlled by two UDP clients which leads to unpredictable behavior.

The advantage of the protocol is its simplicity and its ability to be used in parallel to a real-time simulation loop or a ODCP based execution loop. It can be used to induce disturbances to robot parts or to temporarily control agents not otherwise controlled by an ODP 1.1 client.

2 Basic Structure of the Client-Server System

Each agent in an ORCS simulator can provide its own ODCP 1.0 interface. Hereby a UDP server is started that belongs to the controlled agent. It handles all direct control commands from external clients. The port of a server has to be known to an external client. ODCP clients can send a connection message to notify the simulator that a client is using the specific ODCP interface. After using the interface a client can send a termination message, which notifies the simulator that this client is done using the interface. Both messages are optional and help the simulator to react on external clients. Independently of the connection messages any client can send motor control or sensor request messages to the ODCP server to control the agent or to get its motor angles.

3 Data Format

Messages of the ODCP protocol use a single byte as header to identify the command. The remaining part of a datagram packet is dependent on the command type. These commands are explained later in section 4.

As part of the messages, data is submitted. The protocol makes use of integers and single bytes.

Integers are sent as 4 byte numbers in little-endian format. Thus only 32 bit integers are supported. If the OSP client internally works with larger integers, then the remaining bytes have to be truncated to the 32 most significant bits.

```
(char) (value >> 0)
(char) (value >> 8)
(char) (value >> 16)
(char) (value >> 24))
```

Bytes are simply sent without modifications.

4 Protocol Command Reference

4.1 Connecting to an ODCP Server

An ODCP client can (optionally) send an initial connection request datagram to an ODCP server of a simulated agent. This information can be used by modules in the simulator to keep track of external controller clients or to verify that a certain client required for an experiment actually is available.

Connection Request:

```
Client <: [UDP_INIT_COMMUNICATION (byte)]
Server >: [UDP_INIT_COMMUNICATION_ACK (byte),
           15 (int),
           1 (int)]
```

The server reply sends, additionally to the acknowledge header byte, two numbers used for compatibility reasons. In the next version these numbers might be removed.

4.2 Disconnecting from a ODCP Server

To notify the ORCS simulator that a client is not using an ODCP server any more, it can send a disconnection message. A module in the simulator can use this information for instance to take over control itself to ensure that an agent is doing something in the absence of an external controller.

Disconnection Request:

```
Client <: [UDP_END_COMMUNICATION (byte)]
Server >: [UDP_END_COMMUNICATION_ACK (byte)]
```

4.3 Setting the Motor Angles of an Agent

An ODCP client can control an agent by sending the desired motor angles of that agent. Hereby always all motor angles have to be sent at once, as there is no separate identification of single motors. Also the order of the motor settings is dependent on the order of the motors in the `SimObjectGroup` representing the agent in the ORCS simulator.

Motor angles are specified by integers in a range of [0, 1023]. The null position of the motors is set with a 512.

In addition to the motor angles the client has to sent also the desired maximal torque that can be used by the motor to reach the desired motor angle. This maximal torque has to be specified for each motor separately, so each motor can work with a different maximal torque.

Torque settings are are specified by integers in a range of [0, 1023].

Controlling Motors:

```
Client <: [UDP_COMMAND_SET_MOTORS (byte),
           <motorAngle1> (int),
           <motorAngle2> (int),
           ...,
           <motorAngle_n> (int),
           <maxTorque1> (int),
           <maxTorque2> (int),
           ...,
           <maxTorque_n> (int)]
```

The client does not get a reply from the ODCP server, so there is no guarantee that the UDP packet was received or the motor settings have been successfully applied.

4.4 Requesting the Motor Angles of a Agent

An ODCP client can request the current motor angle settings. This is useful to read out the pose of a simulated agent to record movement sequences as a sequence of pose snapshots (called key frames). Motor positions are sent in the same format as in the motor setting command.

Controlling Motors:

```
Client <: [UDP_COMMAND_GET_MOTORS (byte)]

Server >: [UDP_COMMAND_MOTOR_SETTINGS (byte),
           <motorAngle1> (int),
           <motorAngle2> (int),
           ...,
           <motorAngle_n> (int)]
```

5 Appendix: Protocol Command and Reply Identifiers

All identifiers are unsigned chars (this is important when using JAVA because bytes are ranging from -128 to +128 instead of from 0 to 255 there).

Simulation Step Completed Notification:

```
UDP_INIT_COMMUNICATION = 5;
UDP_INIT_COMMUNICATION_ACK = 6;

UDP_END_COMMUNICATION = 7;
UDP_END_COMMUNICATION_ACK = 8;

UDP_COMMAND_SET_MOTORS = 100;
UDP_COMMAND_GET_MOTORS = 150;
UDP_COMMAND_MOTOR_SETTINGS = 151;
```