Unitree D1 - Interfejs usług ramienia mechanicznego (D1 Mechanical Arm Services Interface) Instrukcja obsługi

1. Wprowadzenie

Usługa sterowania ramieniem D1 uruchamia się po włączeniu zasilania serwomechanizmów (przegubów) ramienia. Sterowanie oraz odczyt stanu realizowane są przez wysyłanie/odbieranie komunikatów (JSON jako łańcuch znaków) na tematach DDS. Interfejs umożliwia sterowanie kątami przegubów, przełączanie stanów zablokowany/odciążony (lock/free), kontrolę zasilania silników oraz odczyt danych zwrotnych.

2. Zakres funkcji

Funkcja	Opis
Sterowanie kątem pojedynczego przegubu	Ustawienie docelowego kąta silnika wybranego przegubu.
Sterowanie kątami wszystkich przegubów	Jednoczesne ustawienie kątów wszystkich przegubów (0-6).
Enable/odciążenie pojedynczego silnika przegubu	Stan przegubu: zablokowany (enable) / odciążony (release). W stanie odciążenia możliwe „drag-teaching” z odczytem kątów.
Enable/odciążenie wszystkich silników	Ustawienie stanu lock/free dla wszystkich przegubów; w stanie odciążenia możliwe „drag-teaching”.
Przełącznik zasilania silników	Włącza/wyłącza zasilanie silników (może służyć jako awaryjne zatrzymanie).
Powrót pozycji ramienia do zera	Z dowolnej pozycji powrót do pozycji zerowej.
Sprzężenie zwrotne kątów przegubów	Bieżące (real-time) kąty przegubów.
Sprzężenie zwrotne stanu ramienia	Stan enable/odciążenia, stan zasilania, stan błędu.
Potwierdzenie odbioru komendy	Informacja, że komenda została poprawnie odebrana i zparsowana.
Potwierdzenie wykonania komendy	Informacja, że komenda została wykonana (sukces/porażka).

3. Parametry i specyfikacja (D1-550)

3.1 Parametry konstrukcyjne

Parametr	Wartość
Model	D1-550
Masa	3152 g
Stopnie swobody	6 (oś) + 1 (chwytak)
Długość ramienia	550 mm (bez szczęk), 670 mm (ze szczękami)
Udźwig znamionowy	500 g (wliczając masę chwytaka)
Promień roboczy	550 mm
Typ silników	Serwa magistralowe (bus servo)

3.2 Moment znamionowy silników przegubów

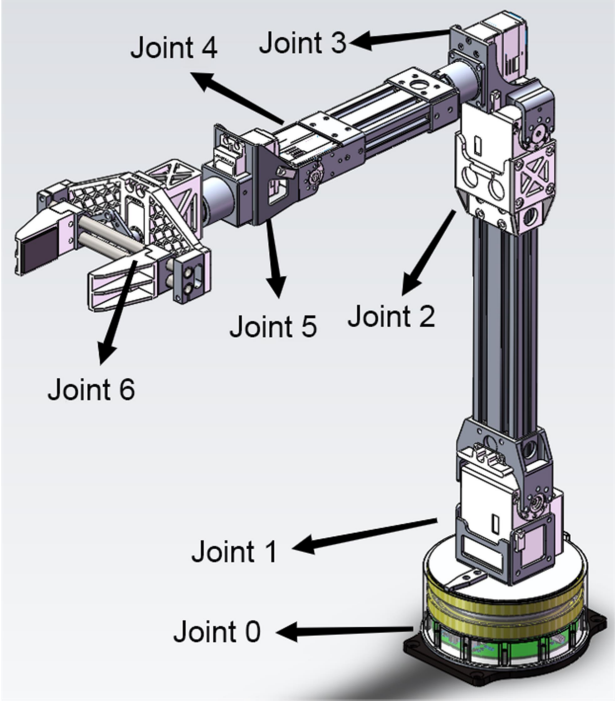
Przegub	Moment
J0	3.3 Nm
J1	3.3 Nm
J2	1.7 Nm
J3	1.7 Nm
J4	1.7 Nm
J5	1.7 Nm
J6 (chwytak)	1.7 Nm

3.3 Zakresy ruchu przegubów

Element	Zakres
J0	±135°
J1	±90°
J2	±90°
J3	±135°
J4	±90°
J5	±135°
Skok chwytaka	0-65 mm

3.4 Parametry elektryczne i komunikacja

Parametr	Wartość
Wymagania zasilania	24 V / 10 A (zakres 15-48 V)
Moc	240 W
Kontroler	Zintegrowany (4 x Cortex-A55 1.8 GHz)
Interfejsy komunikacyjne	RJ45 (Ethernet 100 Mbps) + USB Type-C (debug port szeregowy)
Metoda sterowania	Subskrypcja DDS (DDS Subscription)
Częstotliwość cyklu sterowania	10 Hz



4. Numeracja osi/przegubów

Ramię składa się z 6 osi podstawowych + chwytak. Numeracja przegubów rozpoczyna się od podstawy jako 0, a kończy na chwytaku jako 6.

5. Protokół komunikacji (JSON jako string)

Wszystkie dane przesyłane są jako łańcuch znaków w formacie JSON. Każdy komunikat zawiera pola:
- seq: numer sekwencji (unikalny identyfikator wiadomości)
- address: kod adresu instrukcji
- funcode: kod funkcji instrukcji
- data: zawartość danych (może zawierać zagnieżdżony obiekt JSON)

Numer seq dla komend generowany jest po stronie wywołującej. Wyjątkiem są komunikaty aktywne (feedback) wysyłane przez ramię, dla których seq ma stałą wartość 10. Odpowiedzi mogą zwracać ten sam seq, aby skojarzyć odpowiedź z komendą.

6. Komendy i komunikaty zwrotne

Funkcja	Kierunek	seq	address	funcode	Dane (data) / uwagi
Kąt pojedynczego przegubu	Komenda	gen.	1	1	id, angle, delay_ms (delay_ms = 0)
Kąty wszystkich przegubów	Komenda	gen.	1	2	mode, angle0..angle6 (mode: 0=10Hz smoothing; 1=trajectory smoothing)
Enable/odciążenie przegubu	Komenda	gen.	1	4	id, mode (0=release; 1=enable)
Enable/odciążenie wszystkich	Komenda	gen.	1	5	mode (0=release; 1=enable)
Zasilanie silników	Komenda	gen.	1	6	power (0=OFF; 1=ON)
Powrót do zera	Komenda	gen.	1	7	brak data
Feedback: kąty	Feedback	10	2	1	angle0..angle6; 10 Hz
Feedback: status ramienia	Feedback	10	2	3	enable_status, power_status, error_status (1=OK; 0=OFF/błąd)
Feedback: status silników	Feedback	10	2	4	motor0_status..motor6_status (1=OK; 0=fault)
Ack odbioru (recv)	Odpowiedź	gen.	3	1	recv_status (1=OK; 0=błąd formatu/zakresu)
Ack wykonania (exec)	Odpowiedź	gen.	3	2	exec_status (1=sukces; 0=porażka)

6.1 Przykładowe komunikaty JSON

```
Pojedynczy przegub (ustaw przegub 5 na 60 stopni):
{"seq":4,"address":1,"funcode":1,"data":{"id":5,"angle":60,"delay_ms":0}}

Wszystkie przeguby (przykład: {0,-60,60,0,30,0,0}):
{"seq":4,"address":1,"funcode":2,"data":{"mode":1,"angle0":0,"angle1":-
60,"angle2":60,"angle3":0,"angle4":30,"angle5":0,"angle6":0}}

Odciążenie wszystkich przegubów:
{"seq":4,"address":1,"funcode":5,"data":{"mode":0}}

Powrót do zera:
{"seq":4,"address":1,"funcode":7}
```

7. Tematy DDS (interfejsy zewnętrzne)

Usługa udostępnia dwa główne tematy:
- rt/arm_Command - wysyłanie komend sterujących do usługi
- rt/arm_Feedback - odbiór danych/feedbacku publikowanego przez usługę

7.1 Typ wiadomości (ArmString)

```
module unitree_arm {
  module msg {
    module dds_ {
      struct ArmString_ {
        string data_;
      };
    };
  };
```

```
};
```

8. SDK i przykładowe programy

Środowisko uruchomieniowe SDK oraz przykłady wymagają unitree_sdk2. Przed rozpoczęciem prac rozwojowych dla D1 należy wdrożyć unitree_sdk2 na systemie (obecnie wspierany jest Ubuntu).

Pobranie SDK (zgodnie ze źródłem):

https://unitree-firmware.oss-cn-hangzhou.aliyuncs.com/tool/d1_sdk.zip

Po pobraniu: utwórz folder roboczy, użyj cmake do wygenerowania projektu, a następnie make do kompilacji.

Przykładowe programy:

- arm_zero_control.cpp - przykład zerowania (powrót do pozycji 0).
- get_arm_joint_angle.cpp - odczyt kątów przegubów.
- joint_angle_control.cpp - sterowanie kątem pojedynczego przegubu.
- joint_enable_control.cpp - sterowanie stanem enable/odciążenie.
- multiple_joint_angle_control - sterowanie wieloma przegubami jednocześnie.

Pliki w katalogu msg stanowią definicje interfejsu SDK. W projekcie docelowym należy je dodać do własnego projektu lub skopiować do /usr/local/include, dodać ścieżkę w include_directories oraz podlinkować biblioteki przez link_libraries.

9. Pliki do pobrania: wersje, modele i różnice

Pobranie (zgodnie ze źródłem):

Wersja z chwytakiem (gripper): https://oss-global-cdn.unitree.com/static/90b2525be5d84531ab9814f48b2f86a7.zip

Wersja bez chwytaka (gripless): https://oss-global-cdn.unitree.com/static/02c95ece8e354143b874a3c963241467.zip

Modele: STEP uproszczony, D1-550 STEP, D1-550 URDF, pliki open-source chwytaka D1-550 (nazwy wg źródła).

Różnice wersji:

Wersja	Różnice / zastosowanie
Gripper version	Chwytak końcowy może być otwierany i zamykany przez interfejs slider; przydatne do zadań chwytania w symulacji.
Gripless version	Chwytak ma strukturę stałą i jest zamocowany do osi końcowej; brak możliwości otwierania/zamykania; do zadań symulacji samej pozy/pozycji końcówki (end pose).

10. Aktualizacja IAP (upgrade systemu ramienia)

Po podłączeniu ramienia do komputera, w przeglądarce otwórz adres IP ramienia z portem :8080 i przejdź do interfejsu aktualizacji IAP. Przeciągnij pakiet aktualizacji dostarczony przez producenta i kliknij Upload, aby rozpocząć aktualizację.

Domyślny adres IP: 192.168.123.100. Jeżeli adres IP został zmieniony, w źródle nadal podano domyślnie 192.168.123.100.

11. Przykłady użycia (opis + wskazówki)

Poniżej zestawiono przykłady z dokumentacji źródłowej. Fragmenty kodu C++ pozostają bez zmian merytorycznych (to są przykłady użycia unitree_sdk2); opis został przetłumaczony i uporządkowany.

11.1 Sterowanie kątem pojedynczego przegubu

Wyślij na temat rt/arm_Command komendę z funcode=1. Przykład ustawia przegub 5 na 60 stopni (delay_ms=0).

Najważniejsze elementy: TOPIC = rt/arm_Command, JSON w msg.data_().

Kod (oryginał):

```
#include <unitree/robot/channel/channel_publisher.hpp>
#include <unitree/common/time/time_tool.hpp>
#include "msg/ArmString_.hpp"

#define TOPIC "rt/arm_Command"

using namespace unitree::robot;
using namespace unitree::common;

int main()
{
    ChannelFactory::Instance()->Init(0);
    ChannelPublisher<unitree_arm::msg::dds_::ArmString_> publisher(TOPIC);
    publisher.InitChannel();

    unitree_arm::msg::dds_::ArmString_ msg{};
    msg.data_() =
"{\"seq\":4,\"address\":1,\"funcode\":1,\"data\":{\"id\":5,\"angle\":60,\"delay_ms\":0}}";
    publisher.Write(msg);

    return 0;
}
```

11.2 Sterowanie wieloma przegubami

Wyślij komendę funcode=2 z polami mode oraz angle0..angle6. Przykład używa mode=1.

Kod (oryginał):

```
#include <unitree/robot/channel/channel_publisher.hpp>
```

```
#include <unitree/common/time/time_tool.hpp>
#include "msg/ArmString_.hpp"

#define TOPIC "rt/arm_Command"

using namespace unitree::robot;
using namespace unitree::common;

int main()
{
    ChannelFactory::Instance()->Init(0);
    ChannelPublisher<unitree_arm::msg::dds_::ArmString_> publisher(TOPIC);
    publisher.InitChannel();

    unitree_arm::msg::dds_::ArmString_ msg{};
    msg.data_() =
"{\"seq\":4,\"address\":1,\"funcode\":2,\"data\":{\"mode\":1,\"angle0\":0,\"angle1\":-
60,\"angle2\":60,\"angle3\":0,\"angle4\":30,\"angle5\":0,\"angle6\":0}}";
    publisher.Write(msg);

    return 0;
}
```

11.3 Enable/odciążenie (unload force) silników

Komenda funcode=5 steruje odciążeniem wszystkich przegubów. W źródle podano także zakres trybu 0..80000 (0 - pełne odciążenie, 80000 - pełna blokada). W tabeli protokołu wcześniej pojawia się tryb 0/1; to wygląda na rozbieżność dokumentacji - praktycznie należy kierować się implementacją/SDK i testami na sprzęcie.

Kod (oryginał):

```
#include <unitree/robot/channel/channel_publisher.hpp>
#include <unitree/common/time/time_tool.hpp>
#include "msg/ArmString_.hpp"

#define TOPIC "rt/arm_Command"

using namespace unitree::robot;
using namespace unitree::common;

int main()
{
    ChannelFactory::Instance()->Init(0);
    ChannelPublisher<unitree_arm::msg::dds_::ArmString_> publisher(TOPIC);
    publisher.InitChannel();

    unitree_arm::msg::dds_::ArmString_ msg{};
    msg.data_() = "{\"seq\":4,\"address\":1,\"funcode\":5,\"data\":{\"mode\":0}}";
    publisher.Write(msg);

    return 0;
```

```
}
```

11.4 Powrót do zera

Komenda funcode=7 powoduje powrót ramienia do pozycji zerowej.

Kod (oryginał):

```
#include <unitree/robot/channel/channel_publisher.hpp>
#include <unitree/common/time/time_tool.hpp>
#include "msg/ArmString_.hpp"

#define TOPIC "rt/arm_Command"

using namespace unitree::robot;
using namespace unitree::common;

int main()
{
    ChannelFactory::Instance()->Init(0);
    ChannelPublisher<unitree_arm::msg::dds_::ArmString_> publisher(TOPIC);
    publisher.InitChannel();

    unitree_arm::msg::dds_::ArmString_ msg{};
    msg.data_() = "{\"seq\":4,\"address\":1,\"funcode\":7}";
    publisher.Write(msg);

    return 0;
}
```

11.5 Odczyt kątów przegubów w czasie rzeczywistym

Feedback ma stały seq=10 i jest publikowany z częstotliwością 10 Hz. Dane rozróżnia się przez address + funcode. Wymagane jest nasłuchiwanie (callback) na odpowiednich tematach. Uwaga: w przykładzie źródłowym TOPIC1 ma wartość "arm_Feedback" (bez prefiksu rt/); w praktyce należy sprawdzić rzeczywistą nazwę tematu w systemie ramienia.

Kod (oryginał):

```
#include <unitree/robot/channel/channel_subscriber.hpp>
#include <unitree/common/time/time_tool.hpp>
#include "msg/PubServoInfo_.hpp"
#include "msg/ArmString_.hpp"

#define TOPIC "current_servo_angle"
#define TOPIC1 "arm_Feedback"

using namespace unitree::robot;
using namespace unitree::common;

void Handler(const void* msg)
{
```

```
    const unitree_arm::msg::dds_::PubServoInfo_* pm = (const
unitree_arm::msg::dds_::PubServoInfo_*)msg;

    std::cout << "servo0_data:" << pm->servo0_data_() << ", servo1_data:" << pm->servo1_data_() <<
", servo2_data:" << pm->servo2_data_()<< ", servo3_data:" << pm->servo3_data_()<< ", servo4_data:"
<< pm->servo4_data_()<< ", servo5_data:" << pm->servo5_data_()<< ", servo6_data:" << pm-
>servo6_data_() << std::endl;
}

void Handler1(const void* msg)
{
    const unitree_arm::msg::dds_::ArmString_* pm = (const unitree_arm::msg::dds_::ArmString_*)msg;

    std::cout << "armFeedback_data:" << pm->data_() << std::endl;
}

int main()
{
    ChannelFactory::Instance()->Init(0);
    ChannelSubscriber<unitree_arm::msg::dds_::PubServoInfo_> subscriber(TOPIC);
    subscriber.InitChannel(Handler);

    ChannelSubscriber<unitree_arm::msg::dds_::ArmString_> subscriber1(TOPIC1);
    subscriber1.InitChannel(Handler1);

    while (true)
    {
        sleep(10);
    }

    return 0;
}
```

12. D1-T - konfiguracja i użycie (teleoperacja dwóch ramion)

D1-T wykorzystuje router/switch do komunikacji pomiędzy dwoma ramionami. Dane przegubów jednego ramienia są przekazywane do drugiego ramienia do wykonania. Domyślny adres IP ramienia D1 to 192.168.123.100. W konfiguracji D1-T adres drugiego ramienia należy zmienić na inny z tej samej podsieci, np. 192.168.123.99 (lub zgodnie z własną polityką adresacji).

12.1 Konfiguracja routera

W panelu administracyjnym routera ustaw segment sieci WAN/LAN na 192.168.123.xx (zgodnie z przydziałem adresów). Po zapisaniu zmian podłącz PC do routera i sprawdź komunikację poleceniem: ping 192.168.123.100.

12.2 Zmiana adresu IP ramienia

Po potwierdzeniu łączności PC <-> ramię, zaloguj się po SSH do ramienia (hasło wg źródła: 123):

```
ssh ubuntu@192.168.123.100
```

Następnie edytuj konfigurację sieci w /etc/network i ustaw adres IP (np. 192.168.123.99). Po zapisaniu zmian zrestartuj system ramienia.

12.3 Kontrola wielu ramion w jednej sieci

Scenariusze:
- wiele ramion podłączonych bezpośrednio do jednego komputera (wiele kart sieciowych)
- komputer i ramiona podłączone do jednego routera (jedna karta sieciowa)

Metoda 1 - bindowanie po NIC (wiele kart sieciowych)

Jeżeli komputer ma wiele interfejsów sieciowych (NIC), każde ramię można powiązać z konkretną kartą. Wtedy w inicjalizacji ChannelFactory należy wskazać nazwę interfejsu, np.:

```
ChannelFactory::Instance()->Init(0, "eth0");
```

Metoda 2 - unikalne tematy DDS (jedna karta sieciowa, wspólny router)

Jeżeli wszystkie ramiona są w tej samej sieci i używają tych samych tematów DDS, należy nadać każdemu ramieniu unikalne nazwy tematów (np. przez dodanie sufiksu). Wymaga to modyfikacji plików drivera w systemie ramienia.

Zatrzymanie i wyłączenie usług (na ramieniu):

```
sudo systemctl stop marm_controller.service
sudo systemctl stop marm_control.service
sudo systemctl stop marm_communication.service
sudo systemctl stop marm_subscripber.service

sudo systemctl disable marm_controller.service
sudo systemctl disable marm_control.service
sudo systemctl disable marm_communication.service
sudo systemctl disable marm_subscripber.service
```

Edycja tematów w kodzie (przykład):

W pliku marm_code/src/marm_communication_node.cpp zmień np.:

```
#define SubArmCommand_Topic "rt/arm_Command"
# ... na:
#define SubArmCommand_Topic "rt/arm_Command_1"
```

Rekompilacja i restart (na ramieniu):

```
cd marm_code/build/
make clean
make

sudo systemctl enable marm_communication.service
sudo systemctl enable marm_control.service
sudo systemctl enable marm_controller.service
sudo systemctl enable marm_subscripber.service
```

```
sudo reboot
```

Przykład kompletnego zestawu sufiksów tematów (wg źródła):

```
/marm_code/src/marm_communication_node.cpp
#define SubArmCommand_Topic "rt/arm_Command_1"
#define SubServoCurrentAngle_Topic "current_servo_angle_1"
#define PubArmFeedback_Topic "rt/arm_Feedback_1"
#define PubArmZero_Topic "arm_zero_1"
#define PubServoAngle_Topic "set_servo_angle_1"
#define PubServoAngleControl_Topic "set_servo_angle_control_1"
#define PubServoDumping_Topic "set_servo_dumping_1"

/marm_code/src/marm_control_node.cpp
#define PubServoAngle_Topic "set_servo_angle_1"
#define PubArmFeedback_Topic "rt/arm_Feedback_1"
#define SubArmZero_Topic "arm_zero_1"
#define SubServoAngleControl_Topic "set_servo_angle_control_1"
#define SubServoCurrentAngle_Topic "current_servo_angle_1"

/marm_code/src/marm_controller_node.cpp
#define PubArmFeedback_Topic "rt/arm_Feedback_1"
#define PubServoAngle_Topic "current_servo_angle_1"
#define SubServoAngle_Topic "set_servo_angle_1"
#define SubServoDumping_Topic "set_servo_dumping_1"
```

Typowe problemy kompilacji i rozwiązania:

1. Błąd kompilacji - usuń cały folder build i skompiluj ponownie.
2. Błędy czasu plików (clock skew) - ustaw poprawny czas systemowy, np.:

```
sudo date -s yyyy-mm-dd
```

Po nadaniu unikalnych tematów w driverze, w programach sterujących (na PC) ustaw odpowiedni TOPIC, np. rt/arm_Command_1, aby sterować konkretnym ramieniem.

13. Uruchamianie programu ramienia (acquisition vs execution)

Ramię bez urządzenia ręcznego określono jako ramię wykonawcze (execution), a ramię z urządzeniem ręcznym jako ramię pozyskujące dane (acquisition). Dla ramienia wykonawczego można używać programu domyślnego. Dla ramienia pozyskującego należy zatrzymać program domyślny i uruchomić własny program akwizycji.

Zatrzymanie usług (tymczasowo):

```
sudo systemctl stop marm_controller.service
sudo systemctl stop marm_control.service
sudo systemctl stop marm_communication.service
sudo systemctl stop marm_subscripber.service
```

Wyłączenie usług (na stałe):

```
sudo systemctl disable marm_controller.service
sudo systemctl disable marm_control.service
sudo systemctl disable marm_communication.service
sudo systemctl disable marm_subscripber.service
```

Status usług można sprawdzić poleceniem: systemctl status <nazwa_usługi> (w źródle pojawia się literówka „statis”).

14. Uwagi końcowe

W źródle pokazano demonstrację teleoperacji ramienia. Kod źródłowy tej demonstracji nie został udostępniony; w razie potrzeby należy zaimplementować ją samodzielnie.