LABORATORIUM 6
Bliźniak cyfrowy z ROS 2

ROS 2 • topiki • serwisy • mapowanie stanu • buforowanie • synchronizacja • latency

Problem inżynierski

Jak zsynchronizować model wirtualny w Unity z rzeczywistym systemem robotycznym poprzez ROS 2? Jak zmapować stan robota (pozycja, orientacja, stan chwytaczy) na model wirtualny w czasie rzeczywistym, zmniejszając opóźnienie i artefakty drgania?

Parametry laboratorium

Parametr	Wartość
Tryb pracy	pary; jeden robot/symulator na parę, jeden komputer Unity
Rezultat	bliźniak cyfrowy synchronizowany z ROS 2, pomiar latency, scenariusz operacyjny
Oddanie	commit + evidence/lab06.md; bez tradycyjnego sprawozdania
Wersja	1.0 • 25 października 2026 r.

System oceniania: Checkpointy i oceny

Ocena wynika z osiągnięcia kolejnych checkpointów. Każdy checkpoint weryfikuje prowadzący w ciągu maksymalnie 90 sekund (demonstracja ze złączem ROS 2)..

Ocena	Główne wymagania
3.0	ROS 2 audyt → mapowanie stanu → synchronizacja → pomiar latency
3.5	Buforowanie stanów, interpolacja, redukcja artefaktów
4.0	Scenariusz operacyjny (montaż/diagnostyka), pomiar dokładności pozycji
4.5	Porównanie metod synchronizacji, analiza stabilności
5.0	Predykcja stanu, adaptive buffering, automatyzacja, eksploracja granic ROS 2

Walidacja: Numer indeksu i wariant diagnostyczny

Każdy student wpisuje numer indeksu. Wariant błędu wynika ze wzoru:

$$\text{wariantDiagnostyczny} = (\text{numer_indeksu_1} + \text{numer_indeksu_2}) \bmod 3$$

Wynik: wariant A, B lub C (każda para ma inny błąd do naprawienia)

Wymagania wstępne

Przed zajęciami

Zaliczone Laboratorium 5 albo działająca scena XR z modelami zoptymalizowanymi

ROS 2 zainstalowany na maszynie (Humble albo Jazzy) z przykładowymi pakietami

Unity z pakietem ROS 2 For Unity (ros2-for-unity lub analogiczny)

Dostęp do robota/symulatora Gazebo z uruchamiającym się w ROS 2

Sieć LAN między maszyną ROS 2 a komputerem Unity (localhost albo 192.168.x.x)

Brak czerwonych błędów w Console projektu Unity

Konfiguracja ROS 2 i topiki

Topik	Typ	Rola
/robot/state	JointState	Pozycje i prędkości przegubów
/robot/base/pose	PoseStamped	Pozycja i orientacja bazy robota
/robot/gripper/state	Float64	Stan chwytaków (0–1, zamknięty/otwarty)
/cmd/joint_goals	JointTrajectory (serwis)	Polecenia dla robota
/robot/diagnostics	Diagnostics	Stany błędów i ostrzeżeń

UWAGA: Rzeczywiste topiki zależą od wdrożenia robota. Dostosuj nazwy do faktycznych topików dostępnych w ROS 2.

Bezpieczeństwo i bezpieczna komunikacja z robotem

Robot fizyczny: zawsze działaj pod nadzorem prowadzącego. Poprzez ROS 2 można wysłać polecenia ruchu - nigdy nie rób tego bez uprzedzenia inspektora.

Symulator (Gazebo): bezpieczny dla eksperymentów. Jednak sprawdź, czy komunikacja jest oddzielona od rzeczywistego robota.

Sieć ROS 2: domyślnie bez autentykacji - komunikacja między Unity i ROS 2 węzłami jest otwarta. W laboratorium to norma, ale nigdy nie uruchamiaj na publicznych sieciach bez zabezpieczenia.

Watchdog: jeśli Unity przestanie wysyłać wiadomości, robot powinien przejść w bezpieczny stan (prędkości = 0). Sprawdź timeout timeout w wdrożeniu.

Logi: zapisz wszystkie wysyłane polecenia - będą potrzebne do debugowania i audytu.

Podstawy teoretyczne

4.1. ROS 2 i architektura

ROS 2 (Robot Operating System) to framework do komunikacji między węzłami. Węzeł = proces, który wysyła i odbiera wiadomości. Topik = kanał komunikacji (publish/subscribe). Serwis = synchroniczny request/response. Unity jako węzeł ROS 2 może subskrybować topiki robota i wysyłać komendy.

4.2. Mapowanie stanu

Robot publikuje JointState (pozycje przegubów qi, prędkości q'i, torki). Unity odczytuje te wartości i transformuje joint_state do transform hierarchii. Każdy przegub mapuje się na jeden Rigidbody albo Hinge Joint.

4.3. Latency i buforowanie

Sieć sieciowa wprowadza opóźnienie (latency). Jeśli robot wysyła stan co 50 ms, a sieć dodaje 100 ms, model będzie zawsze 150 ms spóźniony. Bufor ringowy przechowuje ostatnie N stanów, umożliwiając interpolację między nimi.

4.4. Interpolacja i predykcja

Proste interpolacja liniowa: między stanem t i t+dt oblicz stan pośredni. Predykcja: estymuj następny stan na podstawie prędkości i przyspieszenia.

Plan pracy

Etap	Dowód ukończenia
Miniquiz i kontrola ROS 2	Odpowiedzi + ros2 node list wyświetla węzły
Audyt topików i mapowanie stanu	Wszystkie topiki dostępne, mapowanie rozumiane
Implementacja synchronizacji (bez buforowania)	Model zmienia pozycję razem z robotem, latency zmierzony
Buforowanie i interpolacja	Model jest gładszy, artefakty drgania zredukowane
Scenariusz operacyjny i pomiar dokładności	Zadanie wykonane (montaż/diagnostyka), błędy policzone
Diagnoza błędu synchronizacji	Objaw, hipoteza, test, przyczyna, commit
Oddanie i exit ticket	Evidence/lab06.md + odpowiedź

Checkpoint 3.0 - Minimalna kompetencja

Wymagania do zaliczenia:

✓ Połączenie ROS 2 ↔ Unity działa bez błędów
✓ Audyt: wszystkie topiki robota znalezione i dostępne
✓ Mapowanie stanu: każdy przegub robota ma odpowiadający go w hierarchii modelu
✓ Synchronizacja: model zmienia pozycję, gdy robot się porusza
✓ Latency: zmierzony czas opóźnienia (wiadomość wysłana → model zmienia pozycję)
✓ Pomiar latency: 3 próby, mediana, kontekst sieciowy (localhost / LAN)
✓ Buforowanie: prosta kolejka N ostatnich stanów
✓ Interpolacja: model nie drży, ruch jest płynny
✓ Diagnoza: objaw, dwie hipotezy, test rozstrzygający
✓ Evidence/lab06.md zawiera: numery indeksów, topiki, latency, wariant diagnostyczny

Część A. Kontrola ROS 2 i topików

A.1. Miniquiz (10 minut, indywidualnie)

1. Wyjaśnij różnicę między topikiem a serwisem w ROS 2.
2. Co to jest JointState i które pola zawiera?
3. Czym różni się latency sieciowe od opóźnienia renderowania?
4. Kiedy interpolacja może być gorsza niż brak buforowania?
5. Ile powinno być stanów w buforze ringowym dla 50 ms publikacji + 100 ms latency?

A.2. Inicjalizacja ROS 2 w Unity (10 minut)

1. git switch lab06-start i utwórz gałąź team-<nr>/lab06-<nazwiska>
2. Dodaj pakiet ROS 2 For Unity do projektu (jeśli nie istnieje)
3. Utwórz skrypt ROS2Manager, który inicjalizuje węzeł Unity
4. Uruchom i sprawdź: ros2 node list powinien pokazywać węzeł Unity

A.3. Audyt topików (10 minut)

1. Uruchom robota/symulatora Gazebo
2. Terminal: ros2 topic list - wyświetl wszystkie dostępne topiki
3. Dla każdego topika: ros2 topic echo <nazwa> - sprawdź zawartość
4. Zanotuj: nazwy topików, typy, częstotliwość publikacji (Hz)
5. Mapa topików: którzy topiki są potrzebni dla bliźniaka cyfrowego?

PUNKT KONTROLNY A: ROS 2 węzeł widoczny, wszystkie topiki zlokalizowane, mapping stanów robota → model stworzony.

Część B. Mapowanie stanu

B.1. Subskrypcja topiku JointState (15 minut)

```
// Skrypt ROS 2 dla stanu przegubów
using ROS2;
public class RobotStateListener : MonoBehaviour {
    int pairIndexSum = 0; // WPISZ: suma numerów indeksów
    ISubscription<JointState> jointSub;
        void Start() {
        var node = ROS2.ROS2.CreateROS2Node("unity_listener");
        jointSub = node.CreateSubscription<JointState>(
            "/robot/state",
```

```
            OnJointStateReceived
        );
    }
        void OnJointStateReceived(JointState msg) {
        // msg.name[] = nazwy przegubów
        // msg.position[] = aktualne pozycje
        // msg.velocity[] = aktualne prędkości
            for (int i = 0; i < msg.name.Length; i++) {
            Debug.Log($"Joint {msg.name[i]}: pos={msg.position[i]:F2}");
        }
    }
}
```

1. Dodaj skrypt do obiektu w scenie
2. Wpisz sumę numerów indeksów
3. Uruchom: Console powinno wyświetlać pozycje przegubów co kilkadziesiąt ms

B.2. Aplikacja stanów do modelu (15 minut)

1. Dla każdego przegubu robota (rotacyjny, liniowy): Hinge Joint albo Slider Joint w Unity
2. W metodzie OnJointStateReceived: ustaw pozycję każdego Jointa
3. Test: poruszeń robotem w Gazebo - model w Unity powinien się poruszać
4. Opóźnienie będzie widoczne, ale model powinien dokładnie śledzić robot

PUNKT KONTROLNY B: Model dokładnie śledzi pozycję przegubów robota. Opóźnienie (latency) jest widoczne.

Część C. Pomiar latency i buforowanie

C.1. Pomiar latency (15 minut)

```
// Pomiar latency
using ROS2;
using System;
public class Latency
Measurement : MonoBehaviour {
    int pairIndexSum = 0; // WPISZ
    List<double> latencies = new List<double>();
        void OnJointStateReceived(JointState msg) {
        // msg.header.stamp zawiera czas wysłania (ROS time)
        long rosTimeUs = msg.header.stamp.sec * 1_000_000L +
                    msg.header.stamp.nanosec / 1_000L;
        long currentTimeUs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds() * 1_000L;
            long latencyUs = currentTimeUs - rosTimeUs;
        double latencyMs = latencyUs / 1000.0;
            latencies.Add(latencyMs);
            if (latencies.Count % 100 == 0) {
            double medianMs = latencies[latencies.Count / 2];
            Debug.Log($"LATENCY_MEASUREMENT median={medianMs:F2}ms seed={pairIndexSum}");
}
    }
}
```

1. Wpisz sumę numerów indeksów
2. Uruchom robot i Unity przez ~30 sekund
3. Zanotuj 3 mediany latency (każda z ~100 próbek)
4. Oblicz średnią medianę
5. Kontekst: localhost lub adres IP?

C.2. Buforowanie ringowe (20 minut)

```
// Bufor ringowy stanów
public class RobotStateBuffer {
   int bufferSize = 5; // Przechowuj 5 ostatnich stanów
   Queue<(double time, JointState state)> stateBuffer =
       new Queue<(double, JointState)>(5);
      public void AddState(JointState state) {
      stateBuffer.Enqueue((Time.time, state));
      if (stateBuffer.Count > bufferSize) {
         stateBuffer.Dequeue();
      }
   }
      public JointState InterpolateState(double time) {
      // Znajdź dwa stany wokół 'time'
      // Interpoluj liniowo między nimi
      var (t1, state1) = GetClosestStateBefore(time);
      var (t2, state2) = GetClosestStateAfter(time);
            float alpha = (float)((time - t1) / (t2 - t1));
      return LerpJointState(state1, state2, alpha);
   }
}
```

1. Dodaj bufor do RobotStateListener
2. Zamiast natychmiast aplikować state, przechowuj w buforze
3. Update(): interpoluj stan z buforu
4. Efekt: model powinien być znacznie gładszy, artefakty drgania zredukowane

PUNKT KONTROLNY C: Latency zmierzony (3 próby, mediana), buforowanie działające, model płynny bez drżeń.

Część D. Scenariusz operacyjny i pomiar dokładności

D.1. Scenariusz - montaż części (20 minut)

1. Robot wykonuje sekwencję ruchów: np. podchodzi do obiektu, chwyta, podnosi, przenosi, odkłada
2. Symulacja: część wirtualna jest montowana razem z ruchem robota
3. Obserwacja: czy model wizualnie odzwierciedla ruchy robota?
4. Pomiar: czy pozycja chwytu modelu zgadza się z rzeczywistą pozycją?

D.2. Pomiar dokładności pozycji (20 minut)

1. Zdefiniuj 5 testowych pozycji robota (np. różne konfiguracje przegubów)
2. Dla każdej pozycji: zatrzymaj robot, czekaj na ustabilizowanie (2s)
3. Porównaj: Expected position (z robota) vs Observed position (z modelu w Unity)
4. Błąd = |Expected - Observed| (w cm)
5. Zanotuj 5 błędów

// Pomiar dokładności| Pozycja | Expected [cm] | Observed [cm] | Błąd [cm]
	---------	---------------	---------------	----------
1	___	___	___
2	___	___	___
3	___	___	___
4	___	___	___
5	___	___	___
Średni błąd: ___ cm

PUNKT KONTROLNY D: Scenariusz wykonany, 5 pomiarów dokładności, średni błąd obliczony.

Część E. Zadanie diagnostyczne

Wyznaczenie wariantu:

wariant = (numer_indeksu_1 + numer_indeksu_2) mod 3

Wariant	Objaw	Hipotezy
A	Model nie zmienia pozycji mimo że robot się porusza	Brak subskrypcji topiku | topik ma inną nazwę | konwersja jednostek robi się czegoś źle
B	Model drzy (szybkie oscylacje) zamiast płynnie się ruszać	Brak buforowania | rozmiar buforu za mały | interpolacja taka jak Update(), nie FixedUpdate()
C	Błąd pozycji rośnie w czasie (drift), mimo że robot jest w tym samym miejscu	Błąd akumulacyjny w transformacjach | drift IMU robota | błąd zaokrąglenia w konwersji

1. Opisz objaw mierzalnie (np. "Model zmiania pozycję z opóźnieniem 500ms")
2. Sformułuj dwie hipotezy z różnymi przewidywaniami
3. Wykonaj test: sprawdź log, zmień parametr
4. Popraw jedną rzecz; powtórz scenariusz
5. git commit -m "fix(lab06): <przyczyna>"

PUNKT KONTROLNY E: Diagnoza zawiera objaw, dwie hipotezy, test, przyczynę, wynik. Commit opisuje przyczynę.

Checkpoint 3.5 - Zadowalająca kompetencja

Wymagania dodatkowe (oprócz Checkpointu 3.0):

✓ Subskrypcja dodatkowych topików: /robot/diagnostics, /robot/gripper/state
✓ Mapowanie stanu chwytu: model pokazuje otwarte/zamknięte chwytaki
✓ Obsługa błędów ROS 2: jeśli topik nie odpowiada, model przechodzi w "idle" (nie zmienia pozycji)
✓ Stabilność: bufor >= 5 stanów, latency < 200ms (lub wskazanie, jeśli > 200ms)
✓ Raport: wszystkie topiki zmapowane, żaden nie jest ignorowany

Weryfikacja (prowadzący, max 90s):

✓ Checkpoint 3.0: wszystkie kryteria

✓ Dodatkowe topiki: diagnostyka i chwytak

✓ Obsługa błędów: model safe state

✓ Stabilność: latency < 200ms

Zaliczenie: 2/4 dodatkowe kryteria (+ Checkpoint 3.0)

Checkpoint 4.0

Wymagania dodatkowe (oprócz Checkpointów 3.0 + 3.5):

✓ Pełny scenariusz operacyjny (montaż, diagnostyka, konserwacja) przeprowadzony bez błędów
✓ Pomiar dokładności: 5 pozycji, średni błąd < 5cm
✓ Analiza stabilności: latency, rozmiar buforu, wpływ na gładkość ruchu
✓ Porównanie: synchronizacja bez buforu vs z buforem (tabela wyników)
✓ Wniosek: czy bliźniak cyfrowy jest wystarczająco dokładny dla docelowego zastosowania?

Weryfikacja (prowadzący, max 90s):

✓ Checkpointy 3.0 + 3.5: wszystkie kryteria

✓ Scenariusz: bez błędów, pozycje prawidłowe

✓ Pomiar dokładności: 5 wartości, średnia

✓ Porównanie bufor/bez bufora

Zaliczenie: 2/4 dodatkowe kryteria (+ poprzednie). Ocena finalna: 4.0

Checkpoint 4.5 - Wysoka kompetencja

Wymagania dodatkowe (oprócz Checkpointów 3.0 + 4.0):

✓ Porównanie metod synchronizacji: Bufor ringowy vs Predykcja (Linear + IMU-based)
✓ Dla każdej metody: pomiary latency-corrected error, gładkość (variance FPS)
✓ Analiza wpływu rozmiaru buforu (3 vs 5 vs 10 stanów)
✓ Zaproponowana optymalizacja: która metoda dla jakich przypadków użycia?
✓ Wnioski: Trade-off dokładność ↔ latency ↔ złożoność

Weryfikacja (prowadzący, max 90s):

✓ Checkpointy 3.0–4.0: wszystkie kryteria

✓ Porównanie metod: tabela wyników

✓ Analiza buforu: 3 rozmiary

✓ Propozycja + rekomendacje

Zaliczenie: 2/4 dodatkowe kryteria (+ poprzednie). Ocena finalna: 4.5

Checkpoint 5.0

Wymagania dodatkowe (oprócz Checkpointów 3.0 + 4.0 + 4.5):

✓ Predykcja stanu: model ROS 2 kalmanowy lub physics-based
✓ Adaptive buffering: rozmiar buforu zmienia się na podstawie latency w czasie rzeczywistym
✓ Eksploracja granic ROS 2: testowanie przy różnych latency (symulacja opóźnienia sieciowego)
✓ Zaproponowana produkcyjna architektura: jak wdrożyć bliźniaka cyfrowego w systemie robotycznym?
✓ Wnioski: limitacje, przyszłe kierunki, porównanie z innymi metodami (DDS, protobuf)

Weryfikacja (prowadzący, max 90s):

✓ Checkpointy 3.0–4.5: wszystkie kryteria

✓ Predykcja: implementacja + pomiary

✓ Adaptive buffering: działające

✓ Eksploracja granic: dane z różnych latency

Zaliczenie: 2/4 dodatkowe kryteria (+ poprzednie). Ocena finalna: 5.0

Oddanie

```
# Laboratorium 6 - <para / nazwiska>
## Identyfikacja pary
- Numer indeksu student 1: _____
- Numer indeksu student 2: _____
- Suma numerów: _____
- Przypisany wariant diagnostyczny: ___ (A/B/C)
## ROS 2 i topiki
- Topiki użyte: _____
- Częstotliwość publikacji: __ Hz
- Typ łączności: localhost / LAN (IP: _____)
## Pomiar latency| Próba | Mediana [ms] | Min [ms] | Max [ms] |
|---|---|---|---|
| 1 | __ | __ | __ |
| 2 | __ | __ | __
|| 3 | __ | __ | __
|- Średnia mediana: __ ms
## Buforowanie
- Rozmiar buforu: __ stanów
- Typ interpolacji: liniowa / sześcienna
- Płynność: TAK / NIE (artefakty drżenia?)
## Scenariusz operacyjny
- Zadanie: montaż / diagnostyka / konserwacja
- Status: wykonany bez błędów / z błędami
## Pomiar dokładności pozycji| Pozycja | Expected [cm] | Observed [cm] | Błąd [cm] |
|---|---|---|---|
| 1 | __ | __ | __ |
| 2 | __ | __ | __ |
| 3 | __ | __ | __ |
| 4 | __ | __ | __
|| 5 | __ | __ | __ |
- Średni błąd: __ cm
## Diagnoza (wariant ___)
- Objaw: ___
- Hipoteza 1: ___
- Hipoteza 2: ___
- Test rozstrzygający: ___
- Przyczyna: ___
```

- Wynik przed / po: ___ / ___

Wnioski

1. ___
2. ___
3. ___

![Bliźniak cyfrowy w akcji](img/lab06-digital-twin.png)

Każda osoba w parze musi mieć co najmniej jeden merytoryczny commit.

Lista końcowa studenta - przed oddaniem

☐ Węzeł ROS 2 w Unity zainicjalizowany i widoczny w ros2 node list.
☐ Wszystkie topiki robota znalezione i dostępne.
☐ Model zmienia pozycję, gdy robot się porusza.
☐ Latency zmierzony (3 próby, mediana).
☐ Buforowanie ringowe implementacyjne.
☐ Model jest płynny, bez artefaktów drżenia.
☐ Scenariusz operacyjny przeprowadzony bez błędów.
☐ Pomiar dokładności: 5 pozycji, średni błąd < 5cm.
☐ Diagnoza zawiera objaw, dwie hipotezy, test, przyczynę, wynik.
☐ Evidence/lab06.md i zrzut znajdują się w repozytorium.
☐ Każda osoba w parze wykonała co najmniej jeden merytoryczny commit.

Pytania kontrolne

1. Wyjaśnij różnicę między topikiem a serwisem w ROS 2.
2. Czym jest JointState i które pola zawiera?
3. Jakie informacje zawiera header.stamp w wiadomości ROS 2?
4. Dlaczego bufor ringowy zmniejsza artefakty drgania?
5. Jaki powinien być rozmiar buforu przy latency 100ms i publikacji 50ms?
6. Czym różni się interpolacja liniowa od predykcji?
7. Czy bliźniak cyfrowy musi być realtime? Dlaczego?

Exit ticket (odpowiedź indywidualnie)

Jedno zdanie: Które ograniczenie ROS 2 (latency, topiki, skalowanie) będzie najtrudniejsze do przezwyciężenia w Twoim przyszłym projekcie robotycznym i jak byś to chciał rozwiązać?