Politechnika Świętokrzyska
w Kielcach
Wydział Elektrotechniki, Automatyki i Informatyki

Rzeczywistość Wirtualna i Rozszerzona

Instrukcja laboratoryjna 2

Kontrolery ruchu

Przygotował:
mgr inż. Michał Wójcik

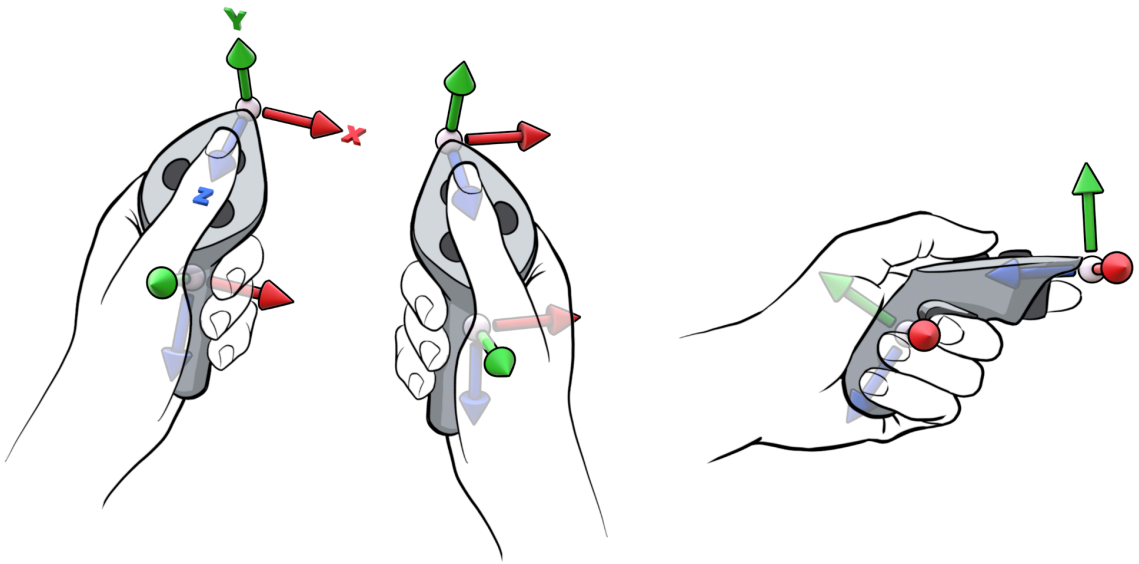
Kielce, 2025

1. Kontroler ruchu VR

Poza wyświetlaczem, podstawą interakcji ze światem wirtualnym są kontrolery. Podobnie jak w przypadku kontrolerów gier znanych z konsol i komputerów osobistych, są to urządzenia wyposażone w zestaw przycisków i czujników pozwalających wykonywać określone akcje. Ze względu na historię rozwoju technologii oraz potrzeby i ograniczenia techniczne producentów konkretnego sprzętu, kontrolery VR mają różne kształty i elementy, które standard OpenXR stara się zunifikować.

1.1. Punkty odniesienia - “Grip” i “Aim”

Pierwszą, oczywistą różnicą pomiędzy kontrolerami od różnych producentów jest ich kształt. Większość współczesnych kontrolerów podlega podobnej konwencji “różdżki” lub “pistoletu”: obłego, podłużnego obiektu, łatwo układającego się w garści, ze spustem w miejscu palca wskazującego i panelem przycisków na wyciągnięcie kciuka. Dłoń nadal układa się różnie w zależności od konstrukcji, dlatego potrzebne było opracowanie dwóch punktów odniesienia: w miejscu chwytu (grip) i w miejscu i kierunku wskazującym (aim).



Źrodło: https://openxr-tutorial.com/windows/d3d11/4-actions.html

Punkt odniesienia “Grip” określa punkt, w którym dłoń chwyta kontroler jak rękojeść miecza - współrzędna Z skierowana jest w dół rękojeści, a współrzędna Y w stronę użytkownika. Z kolei punkt odniesienia “Aim” można wyobrazić sobie jako koniec lufy krótkiego pistoletu - jest on umieszczony zazwyczaj na najdalszym końcu panelu kontrolera, nad palcem wskazującym, a współrzędna Z skierowana jest w kierunku kciuka swobodnie leżącego na panelu - tak, jakby kciuk wskazywał kierunek celowania. Współrzędna Y jest skierowana pionowo w górę, gdy użytkownik przyjmuje pozycję jakby próbował celować z pistoletu.

1.2. Profile interakcji

OpenXR zakłada kompatybilność z wieloma urządzeniami, które różnią się nie tylko kształtem ale i sposobem interakcji - ilością i rodzajem przycisków, gałek, czujników itd. Aby ujednolicić dostęp do tych danych, OpenXR wprowadziło pojęcie **profili interakcji**.

> *Profil interakcji jest kolekcją interakcji wspieranych przez dane urządzenie i środowisko uruchomieniowe, zdefiniowaną przez tekstowe ścieżki.*[1]

Grupa Khronos definiuje 3 części ścieżek:

- **Ścieżka Profilu:** `"/interaction_profiles/<vendor_name>/<type_name>"` Określa profil powiązany z konkretnym rodzajem urządzenia, od danego dostawcy, np. `"/interaction_profiles/oculus/touch_controller"`
- **Ścieżka Użytkownika:** definiuje konkretne urządzenie, np. lewy kontroler `"/user/hand/left"`
- **Ścieżka Komponentu:** definiuje część urządzenia, np. przycisk, czujnik lub pozycję **grip** oraz **aim**, `"/input/grip/pose"`

Przykładowa ścieżka dla interakcji wciśnięcia "menu" w prawym Prostym Kontrolerze Khronos wygląda zatem następująco:
`"/interaction_profiles/khr/simple_controller/user/hand/right/input/menu/click"`
API OpenXR umożliwia połączenie ścieżek interakcji z **akcjami** w świecie gry/aplikacji, takimi jak zamknięcie/otwarcie menu kontekstowego, wybranie elementu lub zmiana koloru obiektu, za pomocą funkcji **SuggestBindings()**. Na laboratorium nie będziemy używać czystego API dla języka C++, ale warto wiedzieć jak jego architektura wpływa na decyzje twórców wtyczek dla Unity.

2. Kontrolery ruchu w Unity

XR Interaction Toolkit zawiera gotowe komponenty pozwalające na śledzenie ruchu kontrolerów i interakcje używając wbudowanego w silnik systemu **Input System**.

2.1. Unity Input System

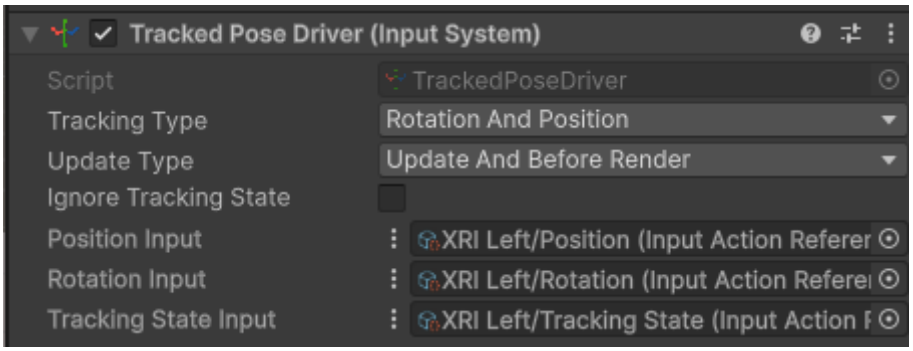
Input System to wprowadzony od wersji 2019.1 silnika Unity nowy system obsługi wejścia z urządzeń takich jak klawiatura, mysz i kontrolery gier. Podobnie jak w przypadku interakcji OpenXR, system bazuje na **akcjach** i **powiązaniach** z konkretnymi elementami urządzeń wejściowych. Dokumentacja pakietu Input System znajduje się pod adresem https://docs.unity3d.com/Packages/com.unity.inputsystem@1.15/manual/index.html.

---

[1] tłumaczenie, **4 Interactions - OpenXR Tutorial Documentation,** rozdział 4.3.1 https://openxr-tutorial.com/linux/vulkan/4-actions.html#interaction-profiles-and-bindings [sprawdzano 11.10.2025]

2.2. Tracked Pose Driver

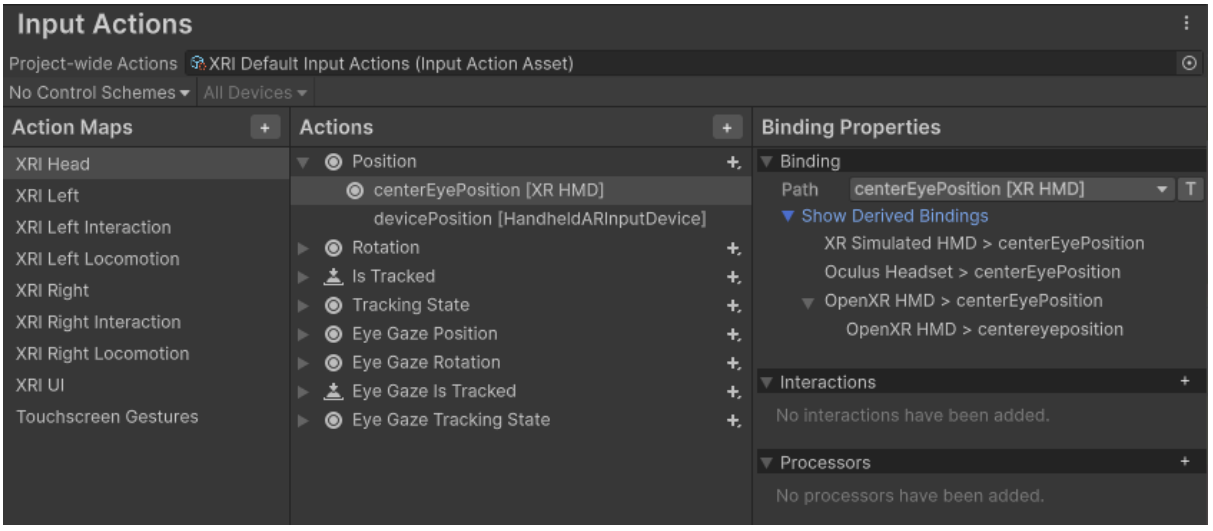
Tracked Pose Driver to element XR Interaction Toolkit obsługujący przeniesienie ruchu urządzenia do sceny 3D - zarówno samego headsetu jak i kontrolerów lub innych urządzeń śledzonych. Poniżej zrzut ekranu komponentu ustawionego dla lewego kontrolera VR:



Przypięty do GameObjectu, komponent potrzebuje podania trzech odwołań do akcji jako właściwości:

- pozycji urządzenia
- rotacji urządzenia
- stanu śledzenia urządzenia

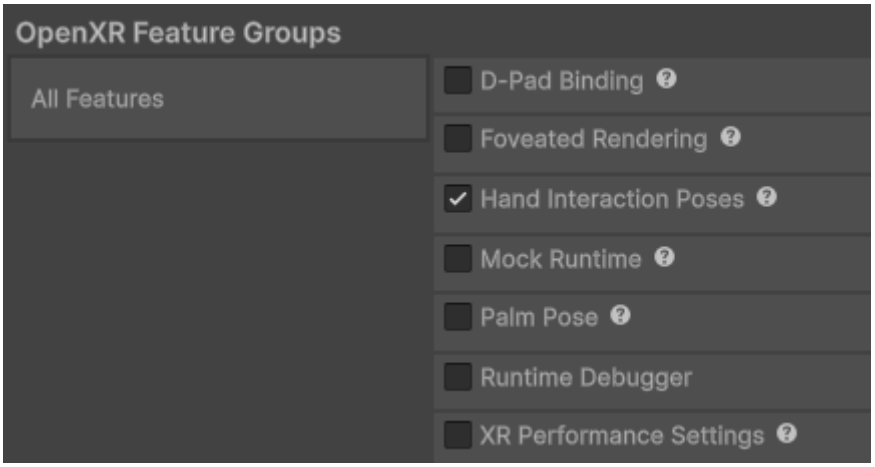
XR Interaction Toolkit udostępnia zestaw domyślnych akcji w pakiecie Starter Assets - **XRI Default Input Actions.inputactions**. Zestaw ten został załadowany na laboratorium 1. podczas ustawiania profili interakcji.



Domyślny zestaw akcji wejścia zawiera mapy akcji dla różnych scenariuszy użycia - śledzenia głowy, kontrolerów, interakcji związanych z kontrolerami, interfejsem użytkownika czy ekranami dotykowymi. Można zaobserwować, że w przypadku lewego kontrolera ścieżka referencji np. pozycji zgadza się ze schematem z powyższego zrzutu ekranu: **XRI Left/Position**.

2.2.1. XR Interaction Toolkit a OpenXR

Standard OpenXR jest obsługiwany w Unity w postaci wtyczki dostawcy, konfigurowanej w **Project Settings > XR Plug-In Management > OpenXR**. Nie wszystkie funkcjonalności są zawsze potrzebne, stąd nie wszystkie są też domyślnie włączone - między innymi pozycje Grip oraz Aim nie są wyliczane dla kontrolerów dopóki nie włączy się opcji **Hand Interaction Poses.**



Zgodnie z dokumentacją wtyczki zmienia się też nazewnictwo na potrzeby silnika: pozycja *grip* to *devicePose*, a pozycja *aim* nazywa się *pointer*. Ścieżki dowiązań dla kontrolerów w Input Systemie wyglądają następująco:

```
<HandInteractionPoses>{LeftHand}/devicePose/position
```

W tym wypadku: pozycja “grip” lewej dłoni, gdzie:

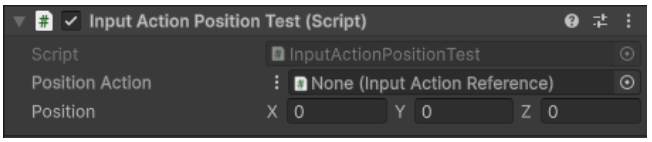
- Layout urządzenia
- Urządzenie
- Akcja
- Składowa akcji

2.3. Wykorzystanie akcji w kodzie komponentu

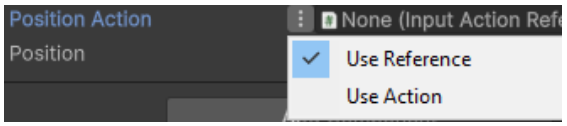
Najprostszym sposobem na wykorzystanie akcji wejścia przy pisaniu własnych komponentów jest zadeklarowanie publicznego członka klasy typu **InputActionProperty**. Członkowie klas deklarowani publicznie pojawiają się w Inspektorze silnika Unity. Następnie, już w silniku, można przypisać własną referencję do akcji po prostu przeciągając ją z widoku plików lub wybierając z menu kontekstowego. Poniższy listing przedstawia przykład pobierania pozycji urządzenia przy każdej klatce animacji i zapisywania jej do członka `position`.

```
using UnityEngine;
using UnityEngine.InputSystem;

public class InputActionPositionTest : MonoBehaviour
{
    /* Używając dekoratora można ujawnić
     * pole *prywatne* w Inspektorze
     [SerializeField]
     private InputActionProperty positionAction;
    */
    public InputActionProperty positionAction;

    public Vector3 position;

    void Update()
    {
        position = positionAction.action.ReadValue<Vector3>();
    }
}
```

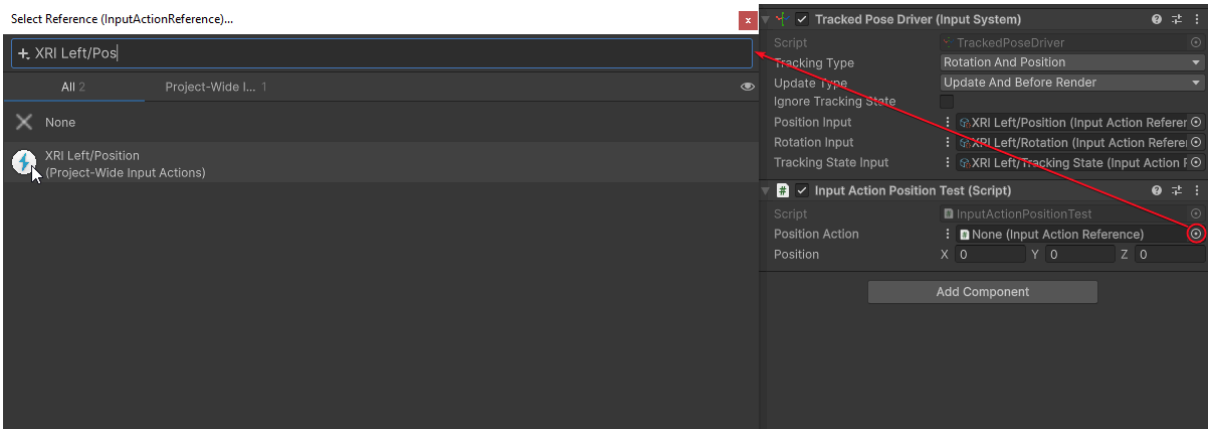
Tak wygląda komponent w inspektorze przypięty do obiektu w grze:



Aby wykorzystać akcję z zestawu gotowych akcji, najlepiej wykorzystać referencję. Wybranie opcji “Use Action” spowoduje utworzenie nowej akcji, do której trzeba będzie samemu dołączyć odpowiednie dowiązanie.



Referencję można przeciągnąć ręcznie z przeglądarki plików lub użyć menu kontekstowego przy polu:



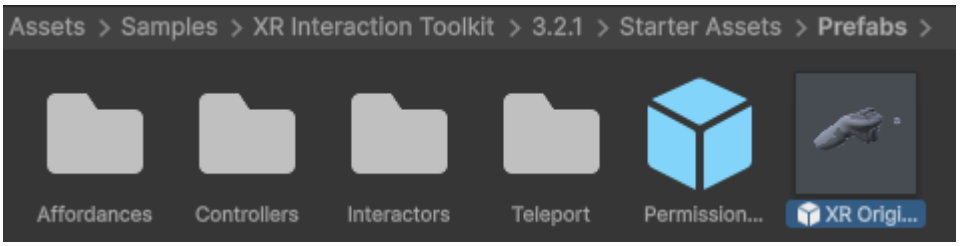
Reagować na akcje można na kilka sposobów:

- Sprawdzać ich wartość - jak w listingu powyżej, za pomocą metody **ReadValue**
- Reagować na wydarzenia - każda akcja posiada funkcje zwrotne **started**, **performed** oraz **cancelled**, do których można subskrybować własne metody:

```
public InputActionProperty clickAction;
// Metoda Start wykonuje się tylko w pierwszej klatce gry
void Start()
{
    clickAction.action.performed += OnClicked;
}
// Obiektu context można użyć np. do pobrania wartości akcji
void OnClicked(InputAction.CallbackContext context)
{
    Debug.Log("Clicked!");
}
```

3. Zadania

1. Korzystając z zasobów udostępnionych w XR Interaction Toolkit, dodaj model kontrolera do każdej z dłoni, **nie używając** kompletnego prefabrykatu XR Origin:



   XR Origin stwórz ręcznie tak, jak w instrukcji 1. Modele kontrolerów są dostępne w folderze Controllers.
   UWAGA! Może okazać się, że modele będą źle obrócone lub wypozycjonowane. Ręcznie dopasuj modele kontrolerów tak, by zgadzały się z rzeczywistym ruchem twoich dłoni [1 pkt]
2. Po wykonaniu zadania 1., spraw by wciśnięcie przycisku na kontrolerze podświetliło ten przycisk na modelu kontrolera na dowolny kolor. [2 pkt]
3. Korzystając z Unity Input Action System napisz skrypt, który po wciśnięciu spustu na kontrolerze wystrzeli z niego kulkę w stronę, w którą celuje kontroler. [4 pkt]
4. Korzystając z Unity Input Action System napisz skrypt, który w inspektorze wyświetli pozycję oraz rotację standardowych pozycji dłoni OpenXR: **Grip** i **Aim**. Czy widać różnicę między nimi? [4 pkt]

4. Źródła

Unity Technologies, *XR Interaction Toolkit Manual*, wersja 3.0 [online]. [n.d.], dostęp: 11.10.2025. Dostępny w Internecie: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html

Unity Technologies, *OpenXR Plugin for Unity*, wersja 1.15 [online]. [n.d.], dostęp: 11.10.2025. Dostępny w Internecie: https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.15/manual/index.html

Unity Technologies, *Input System Manual*, wersja 1.15 [online]. [n.d.], dostęp: 11.10.2025. Dostępny w Internecie: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.15/manual/

OpenXR Tutorial, *Chapter 4: Actions (Linux/Vulkan)* [online]. [n.d.], dostęp: 11.10.2025. Dostępny w Internecie: https://openxr-tutorial.com/linux/vulkan/4-actions.html