Interakcja człowiek–system w XR

WiRR · Wykład 05 · Automatyka i Robotyka · 3 ECTS · 2026/2027 · v1.3.8

dr inż. Mateusz Pomianek

Katedra Informatyki i Automatyki, Politechnika Rzeszowska

27.10.2026

Notatki prowadzącego

Wyobraźcie sobie, że stoję przed wirtualnym pulpitem robota. Chcę uruchomić jedną funkcję, ale przyciski leżą blisko siebie, a kontroler lekko drży. System rozpoznał ruch ręki. Czy to już znaczy, że powinien wykonać polecenie? Oczywiście nie. Dzisiaj będziemy rozdzielać trzy rzeczy: to, na co użytkownik wskazuje, co świadomie wybiera i co ostatecznie uruchamia. Przyjrzymy się kontrolerom, dłoniom, spojrzeniu i głosowi. Zobaczymy też, jak mierzyć skuteczność interakcji i jak nie dopuścić do przypadkowej aktywacji. W XR dobry interfejs ma nie tylko reagować, ale reagować na właściwą intencję.
Urządzenie obserwuje położenie głowy, kontrolera lub dłoni, próbuje rozpoznać zamiar użytkownika i podejmuje decyzję. W drugiej części wykładu pokażemy także, jak takie decyzje mierzyć i sprawdzać przed wykonaniem polecenia.

Cel: wybrać obiekt bez przypadkowego polecenia

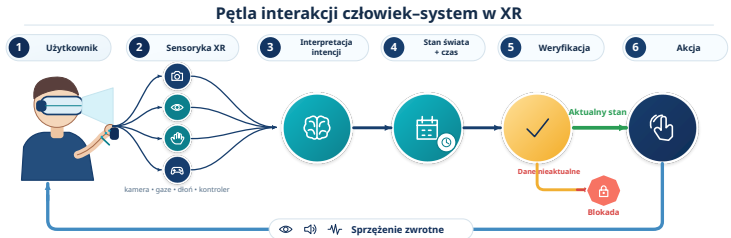


- **Dobieramy** modalność do zasięgu i rozmiaru celu
- **Oddzielamy** wskazanie, wybór oraz wykonanie akcji
- **Oceniamy** szybkość, błędy i wysiłek w jednym zadaniu

Notatki prowadzącego

Na ilustracji mamy kilka obiektów, a zadanie jest proste: znaleźć właściwy, wybrać go i przenieść. Spróbujmy od razu ustalić, co oznacza sukces. Nie wystarczy, że uda się dotknąć jakiegoś elementu. Chcemy trafić w zamierzony cel, nie uruchomić sąsiada i otrzymać czytelne potwierdzenie. Odległość, rozmiar przycisku i sposób wskazywania wpływają na trudność. Kontroler może dawać dokładne potwierdzenie, dłoń większą swobodę, a spojrzenie może ułatwić dostęp do odległych elementów. Nie porównamy ich jednak uczciwie bez tego samego zadania i tych samych kryteriów błędu.
Wysłanie polecenia do robota, usunięcie danych czy uruchomienie mechanizmu to już inna kategoria ryzyka. Interfejs, który jest szybki tylko dlatego, że często się myli, nie jest naprawdę sprawny.

Interakcja jako pętla sprzężenia zwrotnego



Bezpieczeństwo interakcji: wskazanie obiektu nie jest równoznaczne z wykonaniem polecenia. Utrata celu lub zbyt mała pewność musi umożliwiać anulowanie.

Notatki prowadzącego

Spójrzcie na pętlę: użytkownik wykonuje ruch, aplikacja rozpoznaje jego znaczenie i zwraca informację o stanie. Dopiero wtedy użytkownik może zdecydować, czy kontynuować. To sprzężenie zwrotne podobne do tego, które znamy z automatyki. Wyobraźcie sobie windę: naciskam przycisk, widzę podświetlenie i wiem, że żądanie zostało przyjęte. W XR również potrzebuję natychmiastowej informacji, że cel został wskazany. Ale podświetlenie nie oznacza jeszcze wykonania czynności. Jeśli pomylimy te etapy, użytkownik może uruchomić działanie, zanim zdąży sprawdzić, czy wybrał właściwy obiekt.
Najlepiej odróżnić trzy komunikaty: „widzę, na co wskazujesz”, „wybrałeś ten obiekt” i „polecenie zostało wykonane”. Możemy używać podświetlenia, dźwięku czy wibracji, ale najważniejsza jest spójność znaczenia.

Hover, select i activate oznaczają różne zdarzenia



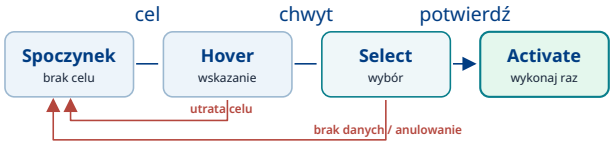
Osobne zdarzenia umożliwiają potwierdzenie i anulowanie.

Podświetlenie celu nie powinno samo uruchamiać kosztownej operacji

Notatki prowadzącego

Posłużmy się latarką. Najpierw kieruję na nią promień kontrolera. Latarka się podświetla, bo pojawiło się zdarzenie hover. Następnie chwytam ją przyciskiem, czyli przechodzę do select. Dopiero kolejne, odrębne polecenie włącza światło: to activate. W XR Interaction Toolkit takie rozdzielenie pozwala niezależnie obsługiwać podświetlenie, trzymanie obiektu i jego użycie. Warto to zachować nawet wtedy, gdy prototyp kusi uproszczeniem. Przy wirtualnej latarce pomyłka jest niegroźna. Gdyby podobny interfejs uruchamiał fizyczny mechanizm, przypadkowa aktywacja mogłaby mieć znacznie poważniejsze skutki.
Ta kolejność jest istotna, bo pozwala uniknąć sytuacji, w której samo spojrzenie albo podniesienie latarki od razu ją uruchamia.
Uwaga na jeszcze jedną pułapkę: activate jako zdarzenie interakcji nie oznacza tego samego co ustawienie GameObject na aktywny przez SetActive.

Interakcja jako automat stanów: wskaż, wybierz, wykonaj, anuluj



Wskazanie nie może samo wykonać polecenia

Aplikacja dopuszcza aktywację dopiero po kontroli stanu oraz aktualności pozy

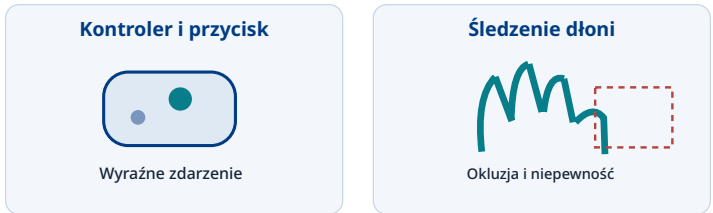
- `HoverEnter`: wskaż; `SelectEnter`: wybierz.
- `Activate`: zweryfikuj warunki, wykonaj **raz**.
- `HoverExit` lub brak aktualnych danych: odrzuć operację.

Logika wykonania: sprawdź `CanExecute()`, ustaw `consumed = true` **przed** wywołaniem `onAccepted`. Po odrzuceniu wymagaj nowego wyboru. Pełna klasa C# z obsługą zdarzeń

Notatki prowadzącego

Tutaj mamy automat stanów i opis jego implementacji. Pełny kod C# jest dołączony do materiałów. Najpierw dopuszczamy wskazanie celu, później wybór, a na końcu świadomą aktywację. Zwróćcie uwagę na warunek CanExecute. Samo odebranie zdarzenia Activate nie wystarczy, jeśli cel przestał być poprawny albo straciliśmy aktualne dane śledzenia. W kodzie po przyjęciu akcji ustawiamy consumed przed wywołaniem zdarzenia, żeby w ramach jednego wyboru nie uruchomić działania drugi raz. Po utracie celu odrzucamy próbę. To podstawowy wzorzec sterowania interfejsem, a nie gotowy moduł bezpieczeństwa robota. Jego rolą jest utrzymanie przewidywalnej sekwencji działań.
Warto zadać sobie jeszcze jedno pytanie: co stanie się, jeśli dwa zdarzenia nadejdą niemal równocześnie? Gdy użytkownik zwalnia chwyt dokładnie w chwili aktywacji, program musi mieć określoną kolejność rozstrzygnięcia.

Kontrolery a śledzenie dłoni

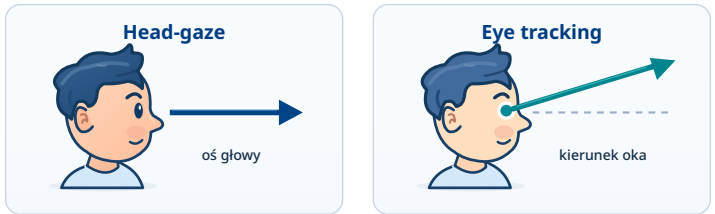


Kontrolery	Śledzenie dłoni
Przyciski zapewniają wyraźne potwierdzenie	Nie wymaga trzymania urządzenia
Dobre dla precyzyjnych poleceń	Wrażliwe na przesłonięcia i jakość obrazu
Zajmują dłonie	Rozpoznanie gestu ma niepewność

Notatki prowadzącego

Które sterowanie wybrałbym do precyzyjnego panelu, a które do swobodnego chwytania przedmiotów? Kontroler daje fizyczny przycisk, wyraźny moment potwierdzenia i często stabilniejszą obsługę małych celów. Śledzenie dłoni uwalnia użytkownika od dodatkowego urządzenia, ale kamera może zgubić palce przy zasłonięciu, a gest może być niejednoznaczny. Dlatego nie powiedziałbym, że jedno rozwiązanie jest zawsze bardziej naturalne. Dla każdej czynności trzeba sprawdzić dokładność, zmęczenie i koszt błędu. Jeżeli użyjemy gestu do działania o dużych konsekwencjach, przyda się osobny, jednoznaczny sposób zatwierdzenia. System nadal musi poprawnie śledzić pozę kontrolera, ale samo zdarzenie przycisku jest dobrze zdefiniowane. Metodę dobieramy do zadania, a nie tylko do tego, która wygląda atrakcyjniej na demonstracji.

Head-gaze, a eye tracking

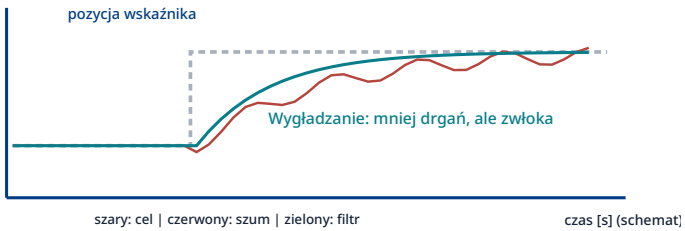


Head-gaze wyznaczamy z orientacji głowy. **Eye tracking** estymuje kierunek spojrzenia oczu. Użytkownik może poruszać oczami bez obrotu głowy, więc tych pomiarów nie należy utożsamiać.

Notatki prowadzącego

Mogę spojrzeć oczami w lewo, nie obracając głowy. Mogę też obrócić głowę w stronę obiektu, ale patrzeć na coś innego. Właśnie dlatego head-gaze i eye tracking nie są zamiennikami. Pierwszy wyznacza kierunek wskazania z orientacji głowy, drugi estymuje kierunek wzroku na podstawie pomiaru oczu. Eye tracking wymaga odpowiedniego sensora i kalibracji, a oba sygnały mają własną niepewność. Jeśli budujemy interfejs gaze, musimy też rozdzielić samo patrzenie od zamiaru wykonania polecenia. Przecież mogę przez chwilę obserwować przycisk tylko dlatego, że czytam umieszczony na nim napis.
Przy analizie uwagi wzrokowej albo wyborze drobnego elementu potrzebujemy rozumieć ograniczenia dokładniejszego pomiaru. Nie wolno w raporcie napisać „użytkownik patrzył na ten punkt”, jeśli w rzeczywistości zarejestrowaliśmy tylko kierunek ustawienia headsetu.

Drżenie wskaźnika: stabilność kontra opóźnienie



Ruch dłoni i szum śledzenia powodują drgania promienia. Wygładzanie zmniejsza jitter, ale zwiększa opóźnienie reakcji.

```
// Wygładzanie pozycji; alpha zależy od dt i tau
float alpha = 1f - Mathf.Exp(-Time.deltaTime / tau);
cursor = Vector3.Lerp(cursor, target, alpha);
```


Notatki prowadzącego

Spróbujcie utrzymać rękę zupełnie nieruchomo. To trudniejsze, niż się wydaje, a pomiar położenia dodatkowo zawiera szum. Na slajdzie widać, jak taki sygnał wpływa na ruch wskaźnika. Filtr wygładzający może ograniczyć drżenie, ale powoduje również pewne opóźnienie. W przykładzie Unity parametr tau decyduje o szybkości reakcji: większe wygładzanie oznacza zwykle większe pozostawanie kursora za szybkim ruchem. Dlatego strojenie filtra nie polega wyłącznie na szukaniu najbardziej gładkiej linii. Musimy sprawdzić, czy użytkownik trafia do celu i jak szybko może zmienić wskazanie. Stabilny obraz nie zawsze oznacza lepsze sterowanie.
W praktyce możemy zastosować filtrację, histerezę zmiany celu albo krótki warunek stabilności przed wyborem. Wszystkie te rozwiązania trzeba zweryfikować w zadaniu, bo zbyt agresywne ustawienia potrafią utrudnić szybkie przełączanie celów.

Głos i haptyka

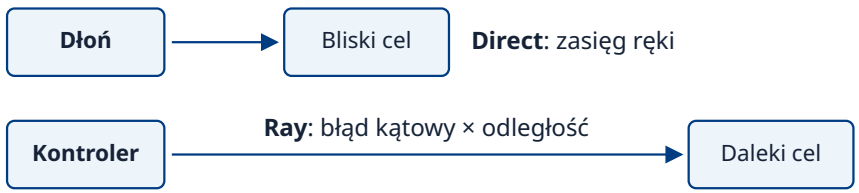


Głos: wygodny bez użycia rąk, lecz zależny od szumu i jakości rozpoznania
Haptyka: potwierdza stan przez wibrację, ale nie dowodzi wykonania komendy

Notatki prowadzącego

Załóżmy, że operator ma zajęte obie ręce i chce zmienić widok danych. Polecenie głosowe może być wtedy znakomitym uzupełnieniem. Ale w hałaśliwej hali rozpoznanie mowy może zawieść, a podobnie brzmiące komendy mogą być mylone. Haptyka działa inaczej: wibracja kontrolera może dać odczuwalne potwierdzenie, że wybraliśmy element lub zmieniliśmy tryb. Nie należy jednak utożsamiać wibracji z wykonaniem zadania przez urządzenie zewnętrzne. Jeśli polecenie dotyczy robota, informacja o faktycznym wykonaniu musi pochodzić z odpowiedniego stanu systemu. Głos przekazuje zamiar, a haptyka może potwierdzać interakcję, lecz żadne z nich samo nie gwarantuje wyniku.
Krótka wibracja kontrolera może potwierdzić wybór, granicę ruchu czy przyjęcie polecenia. Nie zastępuje jednak zawsze komunikatu wizualnego lub dźwiękowego, zwłaszcza jeśli użytkownik nie trzyma kontrolera.

Direct interaction i ray interaction

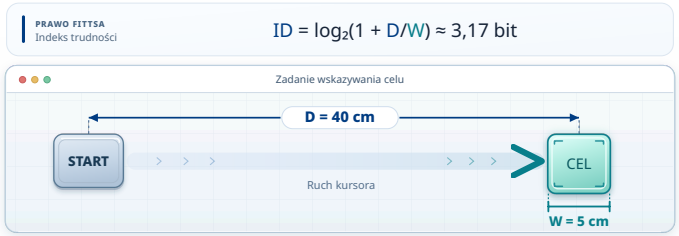


Interakcja bezpośrednia sprawdza się w pobliżu obiektu, promieniowa umożliwia wskazywanie odległych celów

Notatki prowadzącego

Mamy przed sobą dwa obiekty. Pierwszy leży na wirtualnym stole, tuż przy dłoni. Drugi znajduje się kilka metrów dalej. Do pierwszego mogę sięgnąć bezpośrednio, a do drugiego wygodniej skierować promień kontrolera. W Unity odpowiadają temu interakcje direct i ray. To nie tylko kwestia wygody. Przy dużej odległości niewielki błąd orientacji promienia może zamienić się w spore przesunięcie punktu trafienia. Z drugiej strony sięganie ręką do każdego elementu męczy i ogranicza zasięg. Dobrze zaprojektowany interfejs nie musi zamykać użytkownika w jednej metodzie. Może dobierać technikę do odległości i rodzaju zadania. Promień rozwiązuje problem zasięgu, ale wprowadza inny: nawet niewielkie odchylenie kierunku daje coraz większe przesunięcie na odległym celu.

Prawo Fittsa



Dla wskazywania celu czas ruchu można modelować zależnością:

$$MT = a + b \log_2(1 + D/W)$$

gdzie D - odległość, W - szerokość celu w kierunku ruchu, a, b - są parametrami modelu (wyznaczamy je empirycznie dla konkretnej techniki wskazywania)

Notatki prowadzącego

Dlaczego większy przycisk zwykle wybiera się łatwiej niż mały? Prawo Fittsa opisuje to ilościowo. Wzór łączy odległość do celu, jego szerokość w kierunku ruchu i przeciętny czas wskazania. Jeśli zwiększamy D albo zmniejszamy W, indeks trudności rośnie. Ale zwróćcie uwagę na współczynniki a i b. Nie odczytamy ich z samego wzoru, ponieważ zależą od techniki wskazywania i warunków doświadczenia. Mysz, promień kontrolera i gaze nie muszą mieć takich samych parametrów. Na tym etapie możemy porównywać geometryczną trudność zadań, ale jeszcze nie przewidzimy dokładnego czasu ich wykonania.
Najpierw pilnujemy jednostek: nie możemy podstawiać odległości w metrach i szerokości w pikselach bez odpowiedniego przeliczenia. Sam wzór nie powie nam, ile sekund potrzebuje konkretny student korzystający z danego zestawu XR.

Trudność celu: obliczenie bez dopasowania czasu

Dla $D = 40$ cm i $W = 5$ cm:

$$ID = \log_2(1 + 40/5) \approx 3,17 \text{ bit}$$

Dla $W = 2,5$ cm: $ID \approx 4,09$ bit

Bez wyznaczenia współczynników a i b nie znamy czasu MT.

Nie przenosimy ich automatycznie między myszą, promieniem i spojrzeniem.

Notatki prowadzącego

Policzmy przykład ze slajdu. Odległość do celu wynosi czterdzieści centymetrów. Przy szerokości pięciu centymetrów indeks trudności to około 3,17 bita. Zmniejszamy szerokość do dwóch i pół centymetra i otrzymujemy około 4,09 bita. Zauważcie, że nie zmieniliśmy sposobu sterowania ani odległości. Sam mniejszy cel zwiększył trudność zadania. Nie powiedziałbym jednak, o ile sekund wydłuży się wybór. Do tego trzeba zebrać pomiary i wyznaczyć współczynniki modelu. Liczby, które widzicie, są wynikiem podstawienia do wzoru, nie wynikiem badania użytkowników.
Do takiej prognozy potrzebujemy jeszcze empirycznie dopasowanych współczynników równania czasu. Ważniejsze, żebyście umieli powiedzieć, co ten wynik rzeczywiście oznacza, a czego na jego podstawie jeszcze nie wiemy.

Szerokość nominalna i efektywna to różne dane



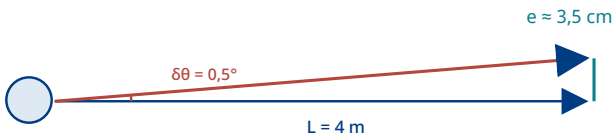
Szerokość nominalna `W`: geometria celu w kierunku ruchu. **Szerokość efektywna `W_e`:** rozrzut punktów zakończenia wielu prób.

Dla określonego protokołu $W_e = 4,133\,SD_x$, gdzie SD_x jest odchyleniem standardowym punktów końcowych wzdłuż osi ruchu. Promień i gaze analizujemy oddzielnie.

Notatki prowadzącego

Na projekcie interfejsu mamy przycisk o szerokości, powiedzmy, pięciu centymetrów. To jego szerokość nominalna. Ale użytkownicy nie trafiają zawsze dokładnie w środek. Z punktów zakończenia wielu ruchów możemy oszacować szerokość efektywną, czyli miarę uwzględniającą rzeczywistą zmienność wskazań. Jedna ze stosowanych metod mnoży odchylenie standardowe położeń w osi ruchu przez 4,133. Nie uzyskamy tego wyniku z jednego kliknięcia ani z samego rozmiaru przycisku. Trzeba też dokładnie ustalić, jak definiujemy trafienie i oś ruchu. Wskazania dłonią, promieniem i spojrzeniem analizujmy oddzielnie.
W ten sposób możemy oddzielić rozmiar, który zaprojektowaliśmy, od precyzji, którą faktycznie uzyskali użytkownicy.

Błąd kątowy staje się błędem bocznym



e = L · tan(δθ) • model geometryczny

$$e = L\tan(\delta\theta)$$

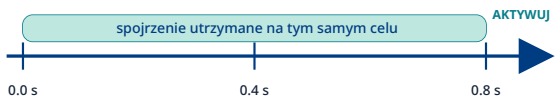
Dla $L = 4$ m i $\delta\theta = 0,5^\circ$: $e \approx 3,5$ cm

Przy odległym małym celu można zwiększyć obszar wyboru lub dodać wspomaganie wskazywania i osobne potwierdzenie

Notatki prowadzącego

Teraz popatrzmy na promień kontrolera z boku. Jego niewielkie odchylenie może być prawie niezauważalne blisko dłoni, lecz po kilku metrach trafia już w inne miejsce. Dla kąta pół stopnia i odległości czterech metrów błąd boczny wynosi około 3,5 centymetra. To czysta geometria: odległość mnożymy przez tangens błędu kątowego. Jeśli przycisk jest węższy niż powstałe przesunięcie, trafianie w niego będzie trudne. Możemy zwiększyć obszar wyboru albo wspomagać wskazywanie, ale nadal potrzebujemy czytelnego potwierdzenia. Nie można całkowicie zastąpić decyzji użytkownika automatycznym przyciąganiem kursora. Wniosek nie brzmi: „ten kontroler ma błąd jednego stopnia”, bo tego tutaj nie mierzyliśmy. Wniosek brzmi: projektując odległy interfejs, musimy uwzględnić, jak geometrycznie wzmacnia się nawet niewielka niepewność kierunku.

Dwell: aktywacja przez spojrzenie



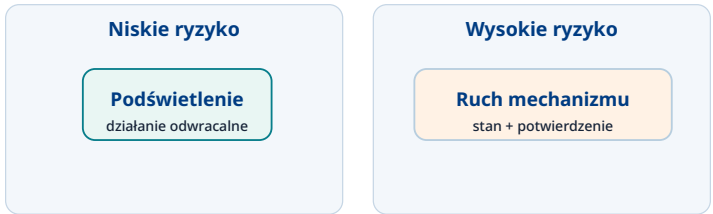
Dwell: aktywacja po utrzymaniu spojrzenia na **jednym celu**. Przykładowy próg `0,8 s` nie jest normą: krótszy sprzyja niezamierzonym aktywacjom, dłuższy wydłuża wybór.

Po `GazeExit` licznik się zeruje; kolejne `GazeEnter` może ponownie rozpocząć pomiar. Jeden ciągły okres spojrzenia powoduje **co najwyżej jedną aktywację**.

Notatki prowadzącego

Co jeśli użytkownik nie może naciskać przycisków? Możemy wykorzystać dwell, czyli utrzymanie spojrzenia na jednym celu przez określony czas. W naszym przykładzie to 0,8 sekundy, ale nie jest to uniwersalnie zalecany próg. Przy zbyt krótkim czasie pojawiają się przypadkowe uruchomienia, a przy długim użytkownik musi niepotrzebnie czekać. Kluczowe są dwie reguły: po opuszczeniu celu licznik się zeruje, a w jednym nieprzerwanym spojrzeniu wykonujemy co najwyżej jedną akcję. Dzięki temu samo dalsze patrzenie na przycisk nie uruchamia go ponownie. Wybór progu trzeba sprawdzić z użytkownikami wykonującymi rzeczywiste zadanie. Przecież patrzymy na pozycje także po to, żeby je przeczytać, a niekoniecznie uruchomić. Zbyt krótki próg prowadzi do przypadkowych aktywacji, a zbyt długi spowalnia pracę i męczy.

Potwierdzenie i anulowanie zależą od ryzyka działania



Ryzyko wyznacza warunki, nie samą modalność

Niski koszt błędu: podświetlenie lub przestawienie koloru **Wysoki koszt błędu:** ruch robota, usunięcie danych, odblokowanie mechanizmu

W drugim przypadku wymagamy świadomego potwierdzenia, możliwości anulowania i sprawdzenia aktualnego stanu.

Notatki prowadzącego

Nie każda pomyłka kosztuje tyle samo. Jeśli przypadkowo zmienię kolor wirtualnego obiektu, zwykle łatwo to odwrócić. Zupełnie inaczej, gdy polecenie ma poruszyć robotem albo usunąć ważne dane. Wtedy sam hover czy przypadkowy gest nie wystarczy. Potrzebujemy świadomej aktywacji, aktualnego stanu systemu, uprawnień i możliwości anulowania. Zwróćcie uwagę, że dodatkowy krok nie musi oznaczać źle zaprojektowanego interfejsu. Czasem to właśnie on zapobiega kosztownej pomyłce. Projektujemy więc poziom potwierdzenia proporcjonalnie do skutków błędu, a nie według zasady, że każda czynność ma mieć jak najmniej kliknięć. Nie oznacza to jednak, że każde kliknięcie w XR powinno otwierać kolejne okno z pytaniem „czy na pewno?”. Dla drobnych, odwracalnych działań często wystarczy wyraźna informacja zwrotna i opcja cofnięcia.

5 minut przerwy

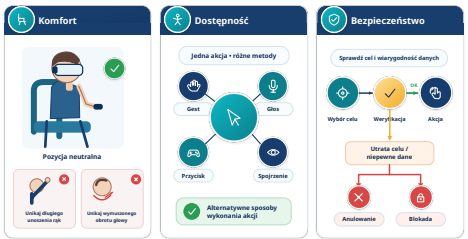
Pytania do samodzielnego powtórzenia

1. Jak odróżnić hover, select i activate przy sterowaniu kontrolerem lub gestem?
2. Jak rozmiar i odległość celu wpływają na trudność w prawie Fittsa?
3. Czym różnią się head-gaze i eye tracking oraz jak uniknąć Midas Touch?
4. Kiedy wiek danych lub brak potwierdzenia powinien zablokować aktywację?

Notatki prowadzącego

Zróbmy pięć minut przerwy. Do tej pory przeszliśmy od wskazania i wyboru obiektu do świadomej aktywacji. Widzieliśmy też, jak błąd kąta i rozmiar celu wpływają na wskazywanie. Po przerwie przejdziemy do ergonomii, dostępności i przykładu implementacji w Unity, a potem do sposobu porównywania metod interakcji.
Do tej pory patrzyliśmy głównie na to, w jaki sposób system rozpoznaje wskazanie i podejmuje decyzję o aktywacji. Po przerwie zajmiemy się szerszym pytaniem: czy taka interakcja jest wygodna, dostępna i bezpieczna, kiedy człowiek rzeczywiście używa systemu, a nie tylko wykonuje pojedynczą demonstrację.

Komfort, dostępność i bezpieczeństwo

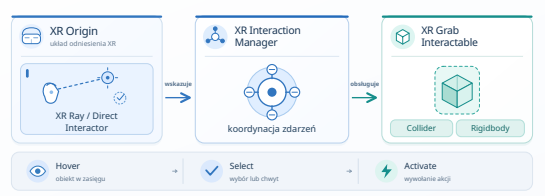


- **Komfort**: ograniczaj długie unoszenie ramion i wymuszony obrót głowy
- **Dostępność**: zapewnij alternatywę dla gestów i poleceń głosowych
- **Bezpieczeństwo**: wymuś anulowanie przy utracie celu lub danych

Notatki prowadzącego

System może poprawnie rozpoznawać gest, a mimo to być niewygodny. Wyobraźcie sobie panel umieszczony wysoko, który przez całą zmianę trzeba obsługiwać uniesioną dłonią. Dokładność śledzenia niczego tu nie naprawi, jeśli użytkownik szybko się zmęczy. Podobnie interfejs wymagający precyzyjnego gestu może być niedostępny dla części osób. Dlatego przy projektowaniu sprawdzam komfort pozycji ciała, możliwość wykonania zadania inną metodą oraz reakcję na utratę danych. Jeśli system nie jest pewny celu, powinien pozwolić przerwać działanie. To wspólne wymagania użyteczności i bezpieczeństwa, nie trzy niezależne dodatki.
Nie możemy zakładać, że każda osoba widzi drobne elementy, słyszy potwierdzenia albo może wykonać precyzyjny gest. A jeśli mamy do czynienia z aplikacją pracującą blisko fizycznych przeszkód, czy interfejs nie odciąga uwagi od otoczenia?

Unity: komponenty XRI i anulowanie dwell



Komponenty: `XR Origin`, `XR Interaction Manager`, `XR Ray/Direct Interactor` i `XR Grab Interactable` z `Collider` oraz `Rigidbody`.

Zdarzenia: `Hover`, `Select` i `Activate` podłącz do metod bramki. `GazeExit()` zeruje `gazing`, `activated` i `gazeStart`.

Ograniczenie: DwellSelector symuluje sygnał gaze, **nie mierzy spojrzenia oczu**.

Notatki prowadzącego

Przejdźmy teraz do Unity. XR Origin określa odniesienie dla urządzeń użytkownika, a XR Interaction Manager zarządza współpracą interaktorów i obiektów. W scenie mamy przykładowy XR Ray Interactor oraz obiekt z XR Grab Interactable. Zdarzenia Hover, Select i Activate łączymy z logiką aplikacji, ale nie traktujemy ich jako tego samego sygnału. Zwróćcie też uwagę na metodę GazeExit w dołączonym przykładzie C#. Kiedy wskaźnik opuszcza cel, zerujemy stan dwell i kolejne spojrzenie zacznie nową próbę. Warto zaznaczyć, że pokazany skrypt nie śledzi oczu. Otrzymuje tylko zasymulowane zdarzenia wejścia i wyjścia celu.
Po opuszczeniu celu kasujemy zgromadzony czas, a przy wejściu na kolejny cel rozpoczynamy nowy pomiar. Przy testowaniu sprawdzilibyśmy co najmniej trzy przypadki: ciągłe patrzenie, przerwanie przed progiem i szybkie przejście z jednego obiektu na drugi.

Test A/B: jedna zmiana, ten sam cel

A: wskazywanie bezpośrednie. **B:** wskazywanie promieniem. Identyczne cele, odległości i reguły potwierdzenia.

```
float elapsedSeconds = Time.unscaledTime - trialStart;
Debug.Log($"Condition={condition}, Time_s={elapsedSeconds:F3}");
```

Dodatkowo zapisuj nietrafione wybory, anulowania i kolejność warunków. W kolejnych osobach równoważ kolejność A/B, aby ograniczyć efekt uczenia się.

Notatki prowadzącego

Jeśli chcemy ocenić dwie techniki, nie możemy dać jednej dużych przycisków, a drugiej malutkich. W naszym porównaniu A to interakcja bezpośrednia, a B to wskazywanie promieniem. Zachowujemy te same cele, odległości i zasady potwierdzenia. Mierzymy czas od wspólnie zdefiniowanego początku do zakończenia próby. W kodzie zapisujemy sekundy, ale osobno trzeba rejestrować nietrafione wybory i anulowania. Pamiętajmy też o efekcie uczenia się: jeśli wszyscy zaczną od A, ta metoda może wyglądać gorzej tylko dlatego, że była pierwsza. Dlatego równoważymy kolejność i opisujemy warunki testu.
Najgorsze, co możemy zrobić, to dać użytkownikom dwa zupełnie różne zadania i przypisać różnicę wyników metodzie interakcji. Trzeba też pamiętać o uczeniu się: druga próba często wychodzi lepiej tylko dlatego, że uczestnik zna już zadanie.

Jak oceniać interakcję: skuteczność, czas i koszt błędu



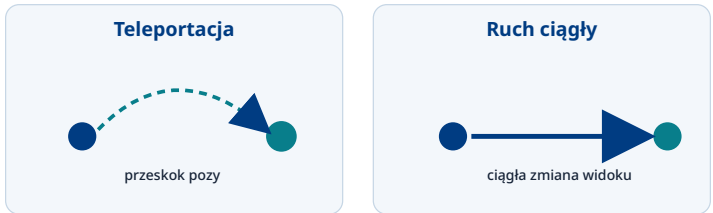
Protokół A/B: te same cele + równoważona kolejność

- **Skuteczność:** odsetek poprawnie zakończonych prób
- **Czas:** od wskazania celu do potwierdzonego wykonania; jawnie opisz początek i koniec
- **Błąd:** przypadkowe aktywacje, korekty, anulowania i konsekwencje pomyłek

Notatki prowadzącego

Wyobraźcie sobie dwa wyniki. W pierwszej metodzie średni wybór zajmuje sekundę, ale użytkownicy często uruchamiają zły przycisk. W drugiej trwa nieco dłużej, lecz błędów jest mniej. Która jest lepsza? Bez znajomości konsekwencji błędu nie ma jednej odpowiedzi. Dlatego podczas oceny interakcji mierzymy co najmniej skuteczność, czas i rodzaj pomyłek. Ważne jest też, czy liczymy czas do samego wskazania, czy do potwierdzonego wykonania. W raporcie zapisujemy liczbę prób, kolejność metod i warunki śledzenia. Nie pokazujmy modelowych przykładów jako wyników prawdziwego eksperymentu.
Osobno warto rejestrować przypadkowe aktywacje, bo to nie to samo co nietrafienie, po którym użytkownik jeszcze zdąży skorygować ruch. Ważne, żeby wcześniej uzgodnić definicje: czy czas kończy się na pierwszym wskazaniu, czy dopiero po poprawnym wykonaniu zadania?

Lokomocja i konflikt sensoryczny

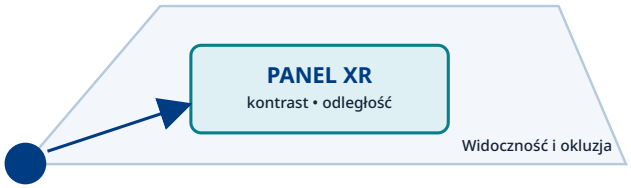


Ruch ciągły: może wywoływać konflikt bodźców wzrokowych i przedsionkowych
Teleportacja: ogranicza ciągły ruch obrazu, ale wymaga podtrzymania orientacji w przestrzeni

Notatki prowadzącego

Zmieniamy temat z wybierania przycisków na przemieszczanie się w VR. Przy ruchu ciągłym możemy obserwować przesuwające się otoczenie, mimo że fizycznie stoimy w miejscu. To może prowadzić do konfliktu między wzrokiem a układem przedsionkowym. Teleportacja usuwa część ciągłego ruchu obrazu, ale w zamian użytkownik nagle zmienia położenie i może stracić orientację. Żadna z tych metod nie jest automatycznie najlepsza. Do szkolenia, eksploracji i precyzyjnej pracy mogą pasować inne rozwiązania. Chcę, żebyśmy oceniali zarówno komfort, jak i to, czy użytkownik nadal rozumie swoje położenie w scenie.
Oczy dostają sygnał ruchu, ale narząd przedsionkowy nie rejestruje odpowiadającego mu przyspieszenia całego ciała. Znaczenie mają także obroty, opóźnienia, sposób wyświetlania, cechy użytkownika i charakter zadania.

HMI przestrzenne

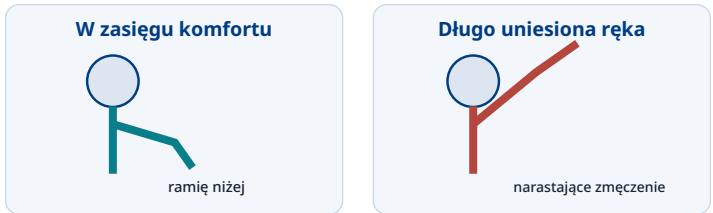


Projektując panel przestrzenny, kontroluj **odległość, kontrast, okluzję** i widoczność informacji krytycznych, nie tylko współrzędne obiektu

Notatki prowadzącego

Klasyczny interfejs ma zwykle stałe miejsce na ekranie. W XR panel może podążać za głową, być przypięty do robota albo pozostawać w określonym miejscu świata. Każde rozwiązanie ma inne konsekwencje. Panel blisko twarzy może zasłaniać otoczenie, a zbyt daleko może stać się nieczytelny. Jeśli przypniemy komunikat do urządzenia, musimy jeszcze pilnować rejestracji i okluzji. Szczególnie ważne są informacje krytyczne: użytkownik powinien móc je odczytać wtedy, gdy są potrzebne. Dlatego przy projektowaniu HMI przestrzennego sprawdzamy odległość, kontrast, wielkość tekstu i położenie względem zadania.
Czy etykieta, która wygląda czytelnie na nieruchomym zrzucie ekranu, pozostaje czytelna, gdy odwrócę głowę? HMI, czyli interfejs człowiek–maszyna, trzeba oceniać w kontekście wykonywanej czynności.

Ergonomia: gorilla arm

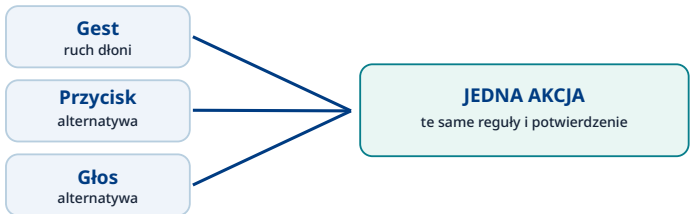


Długie unoszenie ramion może powodować zmęczenie (*gorilla arm*). Powtarzalne czynności projektuj w komfortowym zakresie ruchu.

Notatki prowadzącego

Spróbujcie przez chwilę utrzymać rękę przed sobą, jakbyście obsługiwali wiszący w powietrzu panel. Na początku nic trudnego, ale przy wielu powtórzeniach pojawia się zmęczenie barku i ramienia. W literaturze projektowej zjawisko to nazywa się gorilla arm.
Krótka demonstracja nie pokaże nam skali problemu. Jeśli operator ma wykonywać daną czynność przez dłuższy czas, interfejs powinien pozwalać na wygodniejszy zakres ruchu albo alternatywne sterowanie. Ergonomię sprawdzamy podczas zadania o realistycznej długości, nie tylko na kilkusekundowej prezentacji możliwości sensora.
Najczęściej używane działania dobrze jest umieszczać tam, gdzie nie wymagają dużych, powtarzalnych ruchów. Czasami zwykły przycisk kontrolera okazuje się wygodniejszy i bardziej precyzyjny niż najbardziej efektowna animacja dłoni.

Dostępność poprzez multimodalność



Zapewnij alternatywne sposoby wykonania czynności, jeżeli modalność wymaga sprawności niedostępnej części użytkowników

Notatki prowadzącego

Załóżmy, że aplikację potrafi obsługiwać tylko osoba wykonująca bardzo precyzyjny gest palcami. Dla części użytkowników będzie to bariera, niezależnie od jakości algorytmu. Dlatego możemy dodać kontroler, głos, spojrzenie albo przełącznik. Ale sama obecność kilku sposobów sterowania nie oznacza jeszcze dostępności. Wszystkie potrzebne czynności, łącznie z anulowaniem i potwierdzeniem, muszą być możliwe do wykonania. Warto też sprawdzić, czy użytkownik rozumie informację zwrotną. Dostępność oceniamy więc na poziomie całej ścieżki zadania, a nie jednej atrakcyjnej funkcji.
To, że użytkownik może otworzyć menu głosem, niewiele pomaga, jeśli do jego zamknięcia potrzebny jest niedostępny dla niego gest. Polega na zaprojektowaniu kilku realnych dróg do tego samego rezultatu, bez uzależniania sukcesu od jednej zdolności ruchowej czy sensorycznej.

Dodatkowy kanał musi prowadzić do tej samej czynności

Alternatywa dla gestu może wykorzystywać przycisk, klawiaturę lub przełącznik

Ten sam cel, anulowanie i potwierdzenie muszą pozostać dostępne. Zwiększenie liczby modalności bez sprawdzenia ich zgodności może zwiększać liczbę pomyłek.

Notatki prowadzącego

Wyobraźmy sobie, że ten sam przycisk wirtualny można uruchomić kontrolerem albo głosem. Nie chcemy tworzyć dwóch niezależnych mechanizmów, z których każdy inaczej ocenia warunki wykonania. Lepiej, żeby oba kanały przekazywały zamiar do jednej logiki aplikacji. Wtedy wspólnie sprawdzamy uprawnienia, aktualność stanu, wybór celu i konieczność potwierdzenia. Użytkownik powinien też otrzymać porównywalną informację o wyniku. Jeśli alternatywny kanał omija którąś kontrolę, mamy niespójny interfejs. Multimodalność ma zwiększać dostępność, a nie otwierać dodatkową drogę do przypadkowych poleceń. Warto więc centralnie sprawdzać, czy działanie jest obecnie dozwolone i czy nie zostało już rozpoczęte. Znaczenie ma, czy system jednoznacznie wykonał ją raz i jasno zakomunikował rezultat.

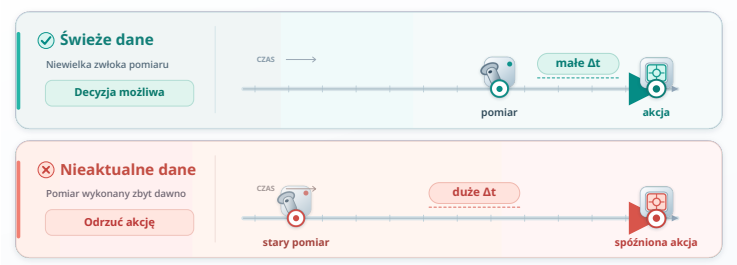
Spojrzenie a prywatność

Dane eye trackingu mogą ujawniać zachowania użytkownika (fenotypowanie cyfrowe). Zbieraj wyłącznie dane niezbędne do realizacji zadania

Notatki prowadzącego

Spojrzenie może służyć do wybierania elementów, ale warto zastanowić się, co jeszcze ujawniają jego pomiary. Szczegółowa historia kierunku wzroku może pokazywać, na czym użytkownik skupiał uwagę, jakie obszary pomijał i jak zmieniało się jego zachowanie. Takie dane wymagają ostrożności. Jeżeli do uruchomienia przycisku wystarczy bieżąca informacja o celu, nie musimy zapisywać pełnego przebiegu patrzenia. Zbierajmy tylko to, co jest rzeczywiście potrzebne, jasno określajmy cel i ograniczajmy dostęp. To, że gogle umożliwiają rejestrację sygnału, nie znaczy, że aplikacja powinna przechowywać go bez końca.
To, że ktoś patrzył na obiekt, nie dowodzi jeszcze, że zamierzał go wybrać, polubił go albo odczuwał konkretną emocję. Dane o spojrzeniu są przesłanką dotyczącą zachowania, a nie bezpośrednim odczytem myśli.

Świeżość danych interakcji

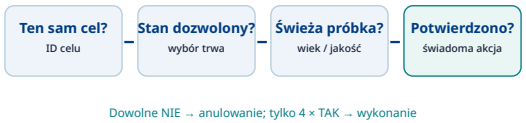


Poprawnie rozpoznany gest może skutkować niebezpieczną komendą przy nieaktualnej pozie człowieka lub robota. Kontroluj czas pomiaru, transformacje i stan komunikacji.

Notatki prowadzącego

Załóżmy, że aplikacja rozpoznała dłoń nad przyciskiem, ale zaraz potem straciła śledzenie. Pozycja, którą mamy w pamięci, opisuje już przeszłość. Jeśli mimo to uruchomimy działanie, możemy użyć niewłaściwego celu. W systemie współpracującym z robotem podobny problem dotyczy również wieku danych o jego pozie. Dlatego przed aktywacją sprawdzamy nie tylko wartość sygnału, ale też znacznik czasu i ważność transformacji. Jeśli dane są zbyt stare albo niepewne, nie próbujemy odgadywać zamiaru. Odrzucamy próbę i pozwalamy użytkownikowi wskazać obiekt ponownie. To prostsze niż późniejsza obsługa skutków błędnej komendy.
Nie ma jednego uniwersalnego progu odpowiedniego dla każdej aplikacji: zależy on od szybkości ruchu, opóźnień i skutków pomyłki. Ważna jest sama kolejność: weryfikujemy stan przed aktywacją, nie dopiero po tym, jak użytkownik zauważy, że zadziałał zły przycisk.

Bramka wykonania: czy wolno teraz uruchomić akcję?



Warunki **muszą być spełnione równocześnie**: ten sam cel, dozwolony stan, świeża próbka i odrębna aktywacja. Demonstrator zakłada **jeden interaktor i jeden obiekt**; nie jest bramką bezpieczeństwa robota.

W demonstratorze: `hovered && selected && !consumed && IsFresh()`.

Jeżeli warunek zawiedzie, **odrzuć aktywację**. Po przyjęciu zablokuj ponowne wykonanie aż do nowego wyboru.

Notatki prowadzącego

Wszystkie wcześniejsze zasady spotykają się tutaj, w bramce wykonania. Pytamy, czy nadal wskazujemy ten sam obiekt, czy został wybrany, czy śledzenie jest wiarygodne i czy ostatnia próbka jest dostatecznie świeża. Jeśli choć jeden warunek jest fałszywy, polecenie odrzucamy i anulujemy wybór. Na slajdzie mamy uproszczone połączenie tych warunków przez operator AND. Po przyjęciu akcji ustawiamy blokadę kolejnego wykonania w ramach tej samej selekcji. Nie traktujcie tego jako gotowego sterownika bezpieczeństwa robota. Demonstrator pokazuje jeden interaktor i jeden obiekt, ale dobrze ilustruje, jak oddzielić rozpoznanie gestu od decyzji o jego skutku.
W międzyczasie następuje chwilowa utrata śledzenia, ale do aplikacji dociera opóźnione zdarzenie zatwierdzenia. Jest to wzorzec logiki naszej aplikacji, a nie automatyczna gwarancja bezpieczeństwa dowolnego urządzenia XR.

Podsumowanie

- Modalność wybieramy według zadania i niepewności sygnału.
- Komfort, dostępność i prywatność są wymaganiami projektowymi.
- Polecenie nie jest wiarygodne bez świeżego stanu świata.

Notatki prowadzącego

Na koniec wróćmy do wirtualnego przycisku z początku wykładu. Wskazanie to nie wybór, a wybór to jeszcze nie wykonanie. Teraz wiemy też, dlaczego małe i odległe cele są trudniejsze, czemu wygładzanie wskaźnika może opóźniać reakcję i kiedy spojrzenie wymaga dodatkowego potwierdzenia. Druga część dotyczyła komfortu, dostępności i ochrony danych. Wszystko łączy jedna zasada: aplikacja ma przyjmować tylko właściwie rozpoznaną, aktualną i świadomie zatwierdzoną intencję. W praktyce sprawdzimy to pomiarami skuteczności, czasu, błędów i rzeczywistą obsługą zadania.
Mówiliśmy dziś o geometrii wskazywania, prawie Fittsa, zdarzeniach i stanach interakcji, filtrowaniu sygnału, ergonomii, testowaniu i alternatywnych kanałach sterowania. Użytkownik rozumie, co system właśnie rozpoznał, potrafi skorygować pomyłkę i wie, kiedy polecenie zostało rzeczywiście wykonane.

Studium przypadku: aktywacja po utracie śledzenia

1. `HoverEnter` i `SelectEnter`: użytkownik wskazał i wybrał cel.
2. `ReportTrackingLost()`: ostatnia poza traci ważność, wybór zostaje odrzucony.
3. Spóźnione `Activate()`: **odmowa wykonania**, bez wywołania `onAccepted`.
4. Dopiero nowa, świeża próbka i **nowy wybór** pozwalają ponowić aktywację.

Demonstrujemy ochronę logiki interfejsu, nie funkcję bezpieczeństwa robota.

Notatki prowadzącego

Prześledźmy jeden konkretny przypadek, zamiast ponownie wyliczać pytania egzaminacyjne. Użytkownik kieruje kontroler na przycisk i go wybiera. Następnie system traci dane śledzenia, ale wskutek opóźnienia dociera jeszcze zdarzenie Activate. W naszym demonstratorze ReportTrackingLost unieważnia tracking i wywołuje HoverExit, więc wcześniejszy wybór nie powinien już uprawniać do wykonania akcji. Taką próbę trzeba odrzucić, bez uruchamiania onAccepted. Po odzyskaniu świeżej pozy użytkownik musi ponownie wskazać i wybrać cel. Dopiero wtedy można świadomie aktywować działanie. To ilustracja spójnej sekwencji zdarzeń interfejsu XR, nie zastępstwo niezależnego sterownika bezpieczeństwa fizycznego robota.

Bibliografia

- Unity, XR Interaction Toolkit.
- MacKenzie (1992), *Fitts' Law as a Research and Design Tool in HCI*, doi:10.1207/s15327051hci0701_3.
- LaValle, Virtual Reality: interakcja.

Notatki prowadzącego

Jeśli będziecie rozwijać własny interfejs XR, zacznijcie od dokumentacji XR Interaction Toolkit i sprawdźcie, jakie zdarzenia udostępnia używana wersja pakietu. Artykuł MacKenziego pomaga zrozumieć, jak analizować czas i dokładność wskazywania zgodnie z prawem Fittsa. W podręczniku LaValle'a znajdziecie szersze omówienie interakcji w VR. Pamiętajcie, że dokumentacja API, model matematyczny i wyniki badania użytkowników odpowiadają na różne pytania. Dopiero połączenie tych źródeł pozwala rozsądnie zaprojektować, zaimplementować i ocenić system. Na tym zamykamy temat interakcji człowiek–system w XR.
Na zakończenie jedna zasada, którą warto zapamiętać: poprawnie wykryty sygnał to jeszcze niekoniecznie poprawnie wykonana interakcja. O jakości decyduje cały proces, od wskazania celu, przez kontrolę stanu i decyzję systemu, aż po informację zwrotną i możliwość skorygowania błędu.