

STRESZCZENIE

Tworzony przez nas projekt jest systemem umożliwiającym użytkownikom zdalną obecność. Składa się on z jasno odseparowanych części: sprzętu i oprogramowania. Na sprzęt składa się: autorska serwogłówica z przyczepionymi do niej dwiema niewielkimi kamerami oraz hełmu rzeczywistości wirtualnej. W skład oprogramowania wchodzą: aplikacja z graficznym interfejsem użytkownika, moduł synchronizujący orientację sprzętu oraz zestaw bibliotek mikrokontrolera.

Użytkownik po założeniu hełmu widzi przed oczami stereoskopowy obraz pochodzący z obu kamer. Obrót głowy (zmiana orientacji) hełmu powoduje zmianę orientacji serwogłówicy w płaszczyźnie pionowej oraz poziomej i w efekcie zmianę widzianego przez niego obrazu. Użytkownik dopasowuje parametry obrazu wyświetlanego na ekranie hełmu z poziomu kontrolek aplikacji.

Nad projektem pracował trzyosobowy zespół:

- Jędrzej Domański – jego zadaniem było stworzenie trzonu oprogramowania – aplikacji GUI pobierającej obraz z kamer, odpowiednio przetwarzającej oraz wyświetlającej go na ekranie hełmu rzeczywistości wirtualnej,
- Krzysztof Mróz – jego zadaniem było zaprojektowanie i zbudowanie serwogłówicy oraz zaimplementowanie potrzebnych bibliotek oraz API komunikacyjne,
- Maciej Rabęda – jego zadaniem było stworzenie oprogramowania odpowiedzialnego za synchronizację sprzętu: połączenie z hełmem i serwogłówicą, pobieranie orientacji hełmu rzeczywistości wirtualnej oraz zadawanie odczytanych kątów na serwogłówicę poprzez ustalone API.

Oprogramowanie zostało wykonane w języku C#. Aplikacja GUI stworzona została we framework'u Windows Presentation Foundation. Biblioteki mikrokontrolera napisano w języku C++.

Dokument ten stanowi opis prac przeprowadzonych nad stworzeniem tego systemu.

Autorzy poszczególnych rozdziałów:

- Jędrzej Domański: 2.1, 2.3, 3.1, 3.6.1, 3.6.7, 4.1, 4.2, 4.6 – 4.8, 4.10, 5.2, 5.3,
- Krzysztof Mróz: 2.2.1, 2.2.3 – 2.2.9, 2.3, 3.1, 3.6.2 – 3.6.5, 4.3, 4.4, 4.9,
- Maciej Rabęda: 1.1–1.5, 2.2.2, 2.3, 3.1–3.5, 3.6.6 4.5, 5.1, 6,
- Podrozdział 2.3 został wykonany wspólnie przez wszystkich członków zespołu.

ABSTRACT

The main purpose of our project is telepresence. It consists of two clearly separated parts: hardware and software. Our hardware comprises two small cameras attached to a servo head of our own design and a virtual reality helmet (HMD). Our software consists of a graphics user interface (GUI) application, a module responsible for synchronizing our hardware's orientation and microcontroller's internal libraries.

Upon mounting the helmet, user sees stereoscopic image originating from both cameras. By moving his head (therefore changing helmet's orientation), similar movement is imitated by the servo head vertically and horizontally, thus changing cameras' focus. The user can adjust the displayed image's parameters by using GUI application's controls.

The development team consisted of three members:

- Jędrzej Domański – his task was to create the core of our product – GUI application, which collects the image from the camera, processes it respectively and finally sends and displays it on HMD's screen,
- Krzysztof Mróz – his task was to design and construct the servo head as well as implement required libraries and communicational API,
- Maciej Rabęda – his task was to create the software responsible for synchronizing the hardware: connection between the servo head and the HMD, getting the HMD's orientation and setting the read angles to the servo head through the API agreed beforehand.

Our software has been developed in the C# programming language. The GUI application was constructed using Windows Presentation Foundation. Microcontroller's libraries were written in C++ programming language.

This document shows the full description of work underwent to develop the system.

Section's authors:

- Jędrzej Domański: 2.1, 2.3, 3.1, 3.6.1, 3.6.7, 4.1, 4.2, 4.6 – 4.8, 4.10, 5.2, 5.3,
- Krzysztof Mróz: 2.2.1, 2.2.3 – 2.2.9, 2.3, 3.1, 3.6.2 – 3.6.5, 4.3, 4.4, 4.9,
- Maciej Rabęda: 1.1–1.5, 2.2.2, 2.3, 3.1–3.5, 3.6.6 4.5, 5.1, 6,
- Subsection 2.3 is a joint work of all of the team's members.

SPIS TREŚĆ

1.	WSTĘP.....	9
1.1.	Od instynktu do idei.....	9
1.2.	Narodziny wirtualnej rzeczywistości.....	9
1.3.	Technologii pierwsze kroki.....	10
1.4.	A jak to wygląda dzisiaj?.....	11
1.5.	Nasza praca wśród innych.....	13
2.	WPROWADZENIE.....	15
2.1.	Koncepcja teleobecności.....	15
2.2.	Opis narzędzi.....	16
2.2.1.	Język programowania C# oraz platforma .NET.....	16
2.2.2.	Visual Studio.....	16
2.2.3.	KiCad.....	17
2.2.4.	Circuit Simulator Applet.....	17
2.2.5.	Atmel Studio 6.2.....	18
2.2.6.	In Circuit Debugger – Avr Dragon.....	18
2.2.7.	Arduino Mega oraz Arduino IDE.....	18
2.2.8.	Terminal.....	18
2.2.9.	Oscyloskop Hantek 6022BE wraz z oprogramowaniem PC.....	18
2.3.	Proces tworzenia.....	19
2.3.1.	Opis zespołu.....	19
2.3.2.	Podział prac.....	20
2.3.3.	Komunikacja.....	20
2.3.4.	Współdzielenie zasobów.....	20
2.3.5.	Harmonogram.....	21
3.	FORMALNY OPIS ZADANIA.....	22
3.1.	Słowniczek.....	22
3.2.	Analiza ryzyka.....	23
3.3.	Wymagania funkcjonalne oraz niefunkcjonalne.....	23
3.4.	Przypadki użycia.....	23
3.5.	Diagram klas.....	28
3.6.	Zasada działania.....	30
3.6.1.	Część graficzna.....	30
3.6.2.	Główica.....	31
3.6.3.	Kontroler głowicy.....	31
3.6.4.	Zestaw bibliotek mikrokontrolera.....	32
3.6.5.	Biblioteka komunikacyjna PC – układ wykonawczy.....	32
3.6.6.	Moduł integracyjny.....	33
3.6.7.	Interfejs.....	33

4. REALIZACJA PROJEKTU.....	35
4.1. Kamera.....	35
4.1.1. Wstęp.....	35
4.1.2. Wybór – PlayStation Eye.....	35
4.1.3. Parametry techniczne.....	36
4.1.4. CL Eye Platform Driver.....	37
4.1.5. Przystosowanie kamer.....	40
4.2. Przesyłanie obrazu do Oculus Rifta.....	44
4.2.1. Wstęp.....	44
4.2.2. Dlaczego obraz wymaga przetworzenia?.....	44
4.2.3. Dystorsja beczkowa.....	46
4.3. Główica.....	47
4.3.1. Silniki Krokowe.....	47
4.3.2. Przekładnie.....	49
4.3.3. Napęd osi poziomej.....	49
4.3.4. Napęd osi pionowej.....	50
4.3.5. Montaż kamer wideo.....	50
4.4. Kontroler.....	51
4.4.1. Układ zasilania.....	51
4.4.2. Mostek sterowniczy.....	52
4.4.3. Część logiczna.....	54
4.4.4. Komunikacja kontrolera z PC.....	54
4.4.5. Separacja układów o różnych napięciach zasilania.....	58
4.4.6. Projekt płytka drukowana.....	58
4.5. Moduł integracyjny.....	59
4.5.1. Opis ogólny.....	59
4.5.2. Zasada działania po stronie Oculus Rifta.....	59
4.5.3. Co uzyskujemy, czyli quaternion w praktyce.....	60
4.5.4. Teoria a praktyka – Oculus SDK i SharpOVR.....	63
4.5.5. Omówienie modułu.....	64
4.6. Interfejs i okna – ogólny zarys.....	65
4.6.1. Wybór środowiska.....	65
4.6.2. Zarys interfejsu.....	66
4.6.3. Wielowątkowość.....	67
4.7. OculusWindow.....	67
4.7.1. Direct HMD Access czy Extended Desktop to the HMD?.....	68
4.7.2. Zawartość oraz położenie okna – OculusWindow.xaml.....	68
4.7.3. Kontroler okna OculusWindow – OculusWindow.xaml.cs.....	70
4.8. ControlsWindow.....	72
4.8.1. Zawartość oraz położenie okna – ControlsWindow.xaml.....	72

4.8.2. Inicjalizacja okna oraz wątków.....	74
4.8.3. Komunikacja pomiędzy wątkami.....	75
4.8.4. Connection, Controls and Presets.....	77
4.8.5. Image Options.....	79
4.8.6. Lens Correction.....	83
4.8.7. Camera Sensor Parameters.....	85
4.8.8. Ekran Statusu.....	85
4.9. Biblioteki mikrokontrolera.....	86
4.9.1. Technologia oraz filozofia wykonania.....	86
4.9.2. Struktura bibliotek.....	86
4.9.3. Biblioteka nagłówkowa Standarts.h.....	89
4.9.4. Klasa SerialDriver.....	90
4.9.5. Klasa SerialCommunication.....	92
4.9.6. Klasa TimerDriver.....	93
4.9.7. Klasa TimerEventDispatcher oraz struktura TIMER_EVENT.....	95
4.9.8. Struktura GLOBAL_SERVICES_TABLE.....	96
4.9.9. Klasa MotorDriver oraz struktura MOTOR_STATES.....	96
4.9.10. Plik AtmegaRom.c.....	98
4.9.11. API komunikacyjne na komputer PC.....	98
4.9.12. Struktura bibliotek.....	98
4.9.13. API komunikacyjne – klasa ArCameraHeadApi oraz struktura Connection.....	98
4.9.14. Klasa DummyConnection.....	100
4.9.15. Klasa ComConnection.....	100
4.9.16. Plik Exceptions.cs.....	100
4.9.17. Klasa ArCameraHeadData.....	100
4.9.18. Korekcja pozycji zerowej oraz ręczna kontrola głowicy.....	101
4.10. Wymagania programowe.....	101
5. TESTOWANIE.....	102
5.1. Założenia.....	102
5.2. Ustalenie wymagań sprzętowych.....	102
5.2.1. Platforma 1 – laptop HP EliteBook 8540w.....	102
5.2.2. Platforma 2 – netbook Acer AO722.....	103
5.2.3. Platforma 3 – Laptop Acer Aspire 5740G.....	103
5.2.4. Platforma 4a – podkręcony komputer stacjonarny z dedykowanym układem graficznym.....	104
5.2.5. Platforma 4b – komputer stacjonarny ze zintegrowanym układem graficznym.....	104
5.3. Wymagania sprzętowe.....	105
6. PODSUMOWANIE.....	106
6.1. Wnioski.....	106

6.2. Rozwój.....	106
7. WYKAZ LITERATURY.....	107
8. WYKAZ TABEL.....	109
9. WYKAZ RYSUNKÓW.....	110
10. ZAŁĄCZNIKI.....	112

1. WSTĘP

1.1. Od instynktu do idei

Jak daleko sięga ludzka pamięć, człowiek tworzył sztukę jako środek ekspresji, sposób dokumentowania przebytych doświadczeń, czy ilustrowania idei. Śledząc jej rozwój, od malowideł tworzonych na ścianach jaskiń, poprzez rzeźbę, średniowieczne i nowożytne malarstwo, na współczesnej fotografii i kinematografii skończywszy, można zauważyc pewną ciekawą prawidłowość. Rozwój ten wydaje się być ukierunkowany przede wszystkim na znalezienie metody wyrazu, która pozwoliłaby na oddziaływanie na maksymalną liczbę zmysłów potencjalnego odbiorcy – wpływać nie tylko na słuch i wzrok, ale również na węch i dotyk. Jego najciekawszy etap przypada na czasy mniej lub bardziej nam współczesne – dwudziesty oraz dwudziesty pierwszy wiek, kiedy to gwałtowny rozwój technologii przyczynił się do narodzin nietuzinkowych pomysłów, które zrewolucjonizowały zarówno przemysł filmowy, jak i wiele innych, niejednokrotnie kluczowych, aspektów życia człowieka. Jednym z takich pomysłów jest właśnie wirtualna rzeczywistość.

1.2. Narodziny wirtualnej rzeczywistości

Za ojca rzeczywistości wirtualnej uznaje się amerykańskiego kinematografa Mortona Leonarda Heiliga (1926–1997)[1]. Pod wpływem ówczesnej literatury doszedł on do wniosku, że “bez aktywnego udziału widza nie ma przekazu świadomości, nie ma sztuki”[2], w efekcie czego w 1957 roku skonstruował urządzenie nazwane przez niego “Sensorama Simulator”. Wynalazek stanowił urzeczywistnienie, a nawet przeskoczenie pomysłu “Feelies” zawartego w powieści science fiction Aldousa Huxleya pt. “Nowy wspaniały świat” (ang. Brave new world) – filmów, które poruszały, poza wzrokiem i słuchem, również zmysł dotyku. Konstrukcja Heiliga składała się pierwotnie z fotela dla jednej osoby oraz zestawu uchwytów i otworów, przez które mogła patrzeć. Dodatkowo, przy otworach znajdowały się przewody powietrzne oraz doprowadzające zapach. Całość otoczona była swoistym baldachimem, aby zminimalizować wpływ bodźców zewnętrznych na odczucia widza. Na potrzeby kręcenia filmów, Heilig zbudował również specjalny zestaw dwóch równoległych kamer 35 mm, który mógł być obsługiwany przez jednego kamerzystę.



Rys. 1.1. Plakat przedstawiający Sensoramę[3]

Sensorama dostarczała odbiorcy zupełnie nowe doznania – w przypadku najpopularniejszego filmu "Motorcycle" osoba mogła "przejechać się" motocyklem po ulicach Brooklynu mając przed sobą trójwymiarowy obraz, czując wiatr we włosach, drgania siedzeń i uchwytów, a nawet mogąc odczuć zapach miasta. Na swój sposób stanowiła ona wynalazek wyprzedzający tamte czasy. Mimo że był to wynalazek całkowicie nowatorski, nie zyskał dostatecznego uznania społeczeństwa. Zbyt wysoki poziom skomplikowania maszyny oraz problemy ze znalezieniem inwestorów spowodowały, że projekt Sensoramy pozostał na etapie prototypu.

1.3. Technologii pierwsze kroki

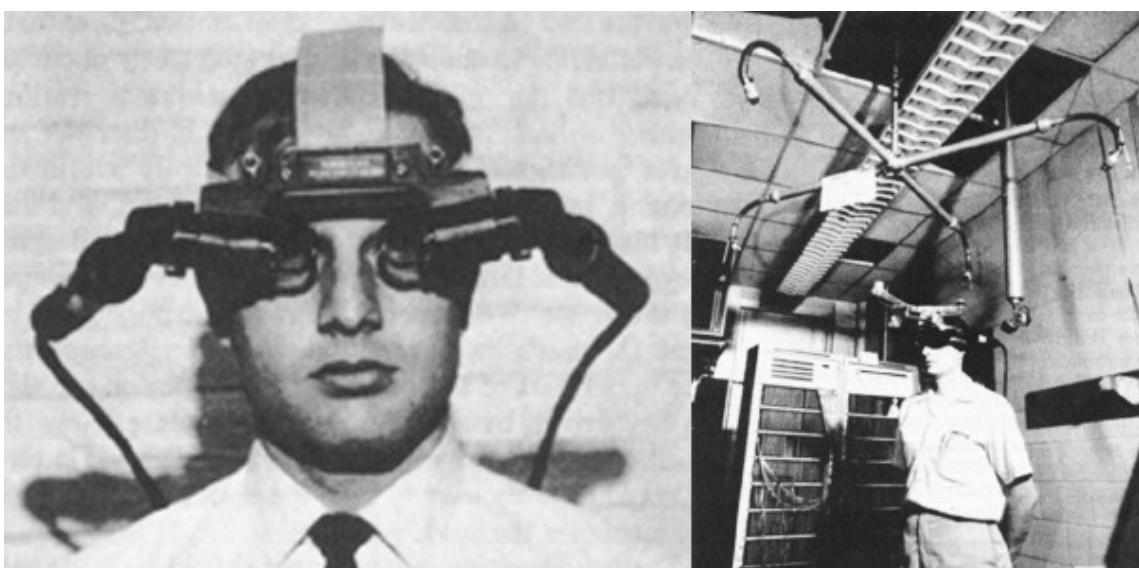
Sensorama nie adresowała jednak całości idei Heiliga – widz wciąż był tylko pasywnym odbiorcą, nie mógł wykonywać żadnych akcji mających jakikolwiek wpływ na jego doznania. Amerykanin pracował równolegle nad drugim projektem zwanym "Telesphere Mask" – pierwszym na świecie HMD (ang. Head-Mounted Display – hełm wideo), który zapewniał szerokokątny, stereoskopowy obraz 3D oraz prawdziwy dźwięk stereo. Konstrukcja ta była o wiele bardziej kompaktowa od swojego poprzednika, ale wciąż stanowiła tylko mały telewizor przymocowany do głowy widza.

O krok naprzód poszła firma Philco Corporation tworząc w roku 1961 HMD łączący pojedynczy wyświetlacz CRT oraz magnetyczny system śledzenia ruchów[4]. Hełm w zestawie ze zdalną kamerą reagującą na ruch głowy użytkownika miał być wykorzystywany w

niebezpiecznych lub niedostępnych dla człowieka miejscach. Z podobnej konstrukcji korzystała firma Bell Helicopter Company. W tym przypadku kamera termowizyjna zamocowana na serwomechanizmie u spodu helikoptera wspomagała lądowania w warunkach nocnych.

Urządzenie, które w całości spełniało zamysł Heiliga, zostało stworzone nie wcześniej niż w 1968 roku przez Ivana Sutherlanda. Wyróżniało się ono od swoich poprzedników dwiema ważnymi cechami: po pierwsze – system śledzenia ruchów hełmu rejestrował zarówno obrót głowy jak i jej przemieszczenie; po drugie – obraz wyświetlany na ekranie HMD był w całości generowany przez komputer w czasie rzeczywistym. Konstrukcja ta nie była jednak pozbawiona wad. Ze względu na swoją dużą masę i zasadę działania, hełm wymagał podczepiania do sufitu, przez co jego mobilność i obszar zastosowań były bardzo ograniczone.

Mimo wszystko to właśnie urządzenie Sutherlanda stanowiło pierwsze pełnoprawne HMD łączące koncepty rzeczywistości wirtualnej (VR) z rzeczywistością rozszerzoną (AR, z ang. augmented reality).



Rys. 1.2. Konstrukcja Sutherlanda nosiła nazwę "Miecz Damoklesa", prawdopodobnie ze względu na swój nietypowy wygląd.[5]

1.4. A jak to wygląda dzisiaj?

Rzeczywistość wirtualna współcześnie jest bardzo powszechnym, dobrze znanim konceptem, z którym człowiek zdążył się zaznajomić oraz nauczył się czerpać wymierne korzyści. Wśród dziedzin ludzkiego życia, które uległy w ostatnim czasie unowocześnieniu są przede wszystkim:

- medycyna i nauka – wirtualna rzeczywistość wspomaga pracę lekarzy poprzez ułatwienie dostępu do ważnych danych medycznych oraz wprowadza zupełnie nowy poziom w ich wizualizacji. Za przykład posłużyć może system, który nanosi zdjęcia pacjenta wykonane podczas rezonansu magnetycznego na jego naturalny "obraz", dzięki czemu chirurdzy mogą sprawniej przeprowadzać operacje. Ponadto hełmy wirtualnej rzeczywistości umożliwiają naukowcom oraz badaczom stereoskopowe widzenie podczas tworzenia modeli i schematów w programach typu CAD;

- służby mundurowe – nowe technologie pozwalają na skuteczniejsze działania wojska, policji czy straży pożarnej poprzez dostarczanie danych taktycznych w czasie rzeczywistym oraz możliwość tworzenia coraz wierniejszych, responsywnych oraz bliższych rzeczywistości symulacji. Dodatkowo zdalna obecność wykorzystywana jest w systemach pozwalających zastąpić człowieka w miejscach wysoce dla niego niebezpiecznych – chociażby w urządzeniach do rozbrajania bomb;
- edukacja – hełmy wideo znajdują w tym obszarze zastosowanie jako element systemów symulacji sytuacji zbyt niebezpiecznych lub nadmiernie kosztownych, aby przeprowadzać je w rzeczywistości, a także czynią naukę bardziej efektywną, wpływając na więcej zmysłów;
- rozrywka – coraz lepsze i wydajniejsze hełmy rzeczywistości wirtualnej wykorzystywane są również w celach rekreacyjnych. Wykorzystanie trzech wymiarów oraz umieszczenie człowieka w środku akcji zupełnie zmienia oblicze filmów, gier wideo i innych;
- motoryzacja – rozszerzona rzeczywistość wpływa również na komfort i bezpieczeństwo na drogach. Jako przykłady można podać szybę samochodową, na którą naniesiony jest obraz z kamery na podczerwień, wspomagający jazdę w nocy, czy kask motocyklowy wzbogacony o miniaturowy ekran, do którego funkcji zaliczyć można między innymi widok z kamery zamontowanej z jego tyłu, nawigację gps, kontrolę nad telefonem oraz odtwarzaczem multimedialnym i inne,
- inspekcja przemysłowa – umożliwienie wirtualnej szkoleń pracowników w zakresie inspekcji kadłubów okrętów czy budynków,
- wirtualna turystyka – hełmy rzeczywistości wirtualnej wykorzystywane są współcześnie również w systemach pozwalających użytkownikom na zdalne zwiedzanie obiektów turystycznych,
- prototypowanie wirtualne – wprowadzenie wirtualnej rzeczywistości pozwala uniknąć ogromnych kosztów, które wiążą się z konwencjonalnym tworzeniem prototypów nie nakładając na nie fizycznych ograniczeń.

W dziedzinie hełmów rzeczywistości wirtualnej, na przestrzeni ostatnich paru lat, można było zaobserwować znaczący rozwój. Na rynku pojawiło się wiele modeli stworzonych przez różnych producentów – do głównych wyróżników tych urządzeń można było zaliczyć między innymi rozdzielcość ekranu, konfigurowalność, mobilność, wygodę użycia czy chociażby integracją z urządzeniem audio, jednak żaden z nich nie odniósł komercyjnego sukcesu. Najważniejszymi przyczynami ich niskiej popularności poza ograniczeniami technicznym, często wąskim zakresem zastosowań konkretnego modelu, niezadowalającym poziomem wsparcia technicznego, czy trudnym procesem tworzenia oprogramowania wykorzystującego choć w połowie możliwości sprzętu jest przede wszystkim wysoka cena. Nadieżą na wyjście urządzeń typu HMD ze statusu niszowych był właśnie projekt Oculus Rift firmy Oculus VR. Głównym celem konstruktorów było stworzenie hełmu, który byłby wydajniejszy od konkurencyjnych rozwiązań, przy zachowaniu przystępności cenowej dla typowego, pojedynczego gracza. Zamierzone cele zostały przez producenta osiągnięte: urządzenie sprawia wrażenie solidnego,

osiada dobrej jakości ekran, szybko reaguje na akcje ze strony użytkownika, a jego cena nie przekracza na obecną chwilę \$300–350 (wersja Development Kit 2), co jest bardzo dobrym wynikiem na tle konkurencji. Mimo tego, że projekt znajduje się wciąż w fazie rozwojowej, to już można mówić o komercyjnym sukcesie – wersja dla deweloperów sprzedała się bardzo dobrze, a branża czeka z niecierpliwością na w pełni funkcjonalną wersję konsumencką.



Rys. 1.3. Oculus Rift DK1 rozebrany na czynniki pierwsze [6]

1.5. Nasza praca wśród innych

Obecnie istnieje co najmniej kilka konstrukcji bardzo zbliżonych do projektu opisywanego w tej pracy – nie powinno to jednak stanowić zaskoczenia, gdyż koncept takiego systemu został już opracowany w latach 60-tych dwudziestego wieku. Jednym z nich jest samodzielnie wykonany przez Davida Schneidera zestaw DIY (ang. Do-It-Yourself – “zrób to sam”) umożliwiający stereoskopowy podgląd lotu modelu samolotu. Choć sam autor przyznał, że o wiele lepsze rezultaty można byłoby osiągnąć wykorzystując w tym celu hełm Oculus Rift, to nie można odmówić mu pomysłowości i sukcesu w minimalizacji kosztów – całkowity koszt jego projektu wynosił mniej niż \$250[7].

Drugą, bardziej efektywną konstrukcją tego typu jest praca wykonana przez Satoshiego Narai w ramach badań na Uniwersytecie Elektro–Komunikacji w Tokio. Połączył on hełm video oraz własny układ elektroniczny z kamerą zamocowaną na serwogłówce. Film zamieszczony na stronie internetowej^[8] nie przedstawia jednak tego zestawu w dobrym świetle – jego działanie obarczone jest sporymi opóźnieniami, a rozdzielcość ekranu urządzenia pozostawia wiele do życzenia.

W przedstawionych realiach tworzony przez nas projekt odnajduje się bardzo dobrze. Ze względu na wykorzystanie Oculus Rifta możemy liczyć zarówno na mniejsze latencje, jak i wyższą rozdzielcość ekranu. W połączeniu z budową serwogłówki i szybkością jej działania powinno to mieć bezpośredni wpływ na osiągnięcie zadowalającego “user-experience”, w

efekcie czego mamy szansę na popularyzację tego typu rozwiązań nie tylko w skali własnego, wąskiego, akademickiego otoczenia, ale również w szerokiej skali zastosowań o zakresie globalnym.

2. WPROWADZENIE

2.1. Koncepcja teleobecności

Idąc za Wikipedią, „Teleobecność (ang. telepresence) to zbiór rozwiązań technicznych pozwalających sprawić wrażenie, że osoba faktycznie przebywająca w określonym miejscu jest postrzegana [przyp. autora: postrzega], że jest obecna gdzie indziej”[9]. Na ową iluzję zazwyczaj składają się obraz, dźwięk oraz wszelkiego rodzaju inne rozwiązania techniczne określone ogółem jako „manipulacje”. Wymienia się wśród nich, zależnie od zapotrzebowania użytkownika, szeroką gamę urządzeń od prostych wysięgników, na skomplikowanych sztucznych rękach kończąc. Od wirtualnej obecności przede wszystkim odróżnia ją obcowanie w świecie realnym, a nie środowisku stworzonym sztucznie.

Najczęściej teleobecność znajduje zastosowanie w telekonferencjach – szczególnie w firmach posiadających fizycznie oddalone, blisko współpracujące ze sobą jednostki, pozwalając na zarówno oszczędność czasową, jak i finansową poprzez zminimalizowanie potrzeby transportu pracowników. Rozwiązania te bazują przede wszystkim na przesyłaniu wysokiej jakości wideo i dźwięku w czasie rzeczywistym, stwarzając wrażenie współobecności osób z różnych lokalizacji (Rys. 2.1.). Ważny jest tutaj odpowiedni dobór sprzętu, aby np. wielkość osoby na ekranie odpowiadała jej fizycznym wymiarom, wideo i audio były skompresowane w co najwyżej niewielkim stopniu, itp.



Rys. 2.4. Profesjonalna sala do wideokonferencji[9]

Powyższe przeznaczenie odbiega jednak od tworzonego przez nas prototypu – do zastosowań mu bliższych można wymienić między innymi prace w niebezpiecznym bądź niedostępny środowisku, zdalną medycynę, czy edukację. Główna idea pozostaje jednak ta

sama – dzięki teleobecności możemy nie tylko oszczędzać czas i pieniądze, ale również ludzkie zdrowie, a nawet życie.

Godnym uwagi przykładem jest niedawno otwarte Laboratorium Zanurzonej Wizualizacji Przestrzennej na Wydziale Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej (Rys. 2.2). Instalacja typu CAVE (ang. Cave Automatic Virtual Environment), poza umożliwieniem “wejścia” do wirtualnej rzeczywistości bez HMD nawet kilku osobom jednocześnie, pozwala na niezwykle kompleksowe wykorzystanie do teleobecności.



Rys. 2.5. Laboratorium Zanurzonej Wizualizacji Przestrzennej na Wydziale ETI Politechniki Gdańskiej[10]

2.2. Opis narzędzi

2.2.1. Język programowania C# oraz platforma .NET

Język C# jest flagowym i najpopularniejszym językiem programowania stworzonym przez Microsoft. Jego główne cechy to: składnia bazująca na C/C++, zarządzalność oraz pełna obiektowość całego rozwiązania.

Platforma .NET jest prawdopodobnie największym dzisiaj zestawem bibliotek programistycznych, środowiskiem uruchomieniowym oraz centralnym środkiem integracji różnych języków programowania, w tym C#, umożliwiającym stosowanie szerokiej ich gamy w każdym jednym projekcie. Istnieje ponadto otwartoźródłowa reimplementacja tego środowiska o nazwie Mono umożliwiająca uruchamianie aplikacji .NET na innych systemach operacyjnych.

2.2.2. Visual Studio

Microsoft Visual Studio to zintegrowane środowisko programistyczne (IDE) stworzone przez firmę Microsoft. Pozwala na tworzenie aplikacji konsolowych oraz z graficznym interfejsem użytkownika – w tym aplikacje Windows Forms czy Windows Presentation Foundation (WPF) – w językach C#, C++, Visual Basic itd. Do jego najważniejszych zalet można zaliczyć intuicyjną obsługę podstawowych funkcjonalności, łatwość tworzenia, komplikacji i debugowania projektów pisanych we wspieranych językach. Dodatkową zaletą jest darmowa wersja tego środowiska (Express), której licencja pozwala na tworzenie aplikacji komercyjnych. Nie pozwala ona przede wszystkim na instalację żadnej z dodatkowych wtyczek oraz nie zawiera przydatnych usprawnień (np. debugowania rozszerzonego o podgląd pamięci), lecz mimo to jest ona całkowicie wystarczająca do tworzenia względnie złożonych projektów.

2.2.3. KiCad

Oprogramowanie KiCad jest zestawem narzędzi wspomagającym projektowanie układów elektronicznych. W jego skład wchodzą:

- Eschema – aplikacja służąca do projektowania schematów ideowych układów elektronicznych umożliwiający generowanie gotowych do druku kart z tymi schematami oraz eksportowanie list połączeń.
- CvPcb – aplikacja służąca do przypisywania obudów (footprint) do elementów zawartych w liście połączeń. Należy tutaj zauważyć, iż oprogramowanie KiCad traktuje każdy element elektroniczny oddzielnie, a nie jak np. pakiet PADS, który grupuje wszystkie elementy danego typu i wymusza wybór tych samych parametrów (przede wszystkim obudów) dla takiej grupy.
- Pcbnew – aplikacja służąca do tworzenia projektów płyt drukowanych. Umożliwia rozpoczęcie projektowania od zainportowanej liczby połączeń, jednakże pozwala także na całkowite zaprojektowanie układu tylko i wyłącznie na poziomie płytki drukowanej – możliwość ta jest często wykorzystywana przez osoby projektujące nieskomplikowane układy bazujące na mikrokontrolerach, jednak autor nie zastosował takiego rozwiązania.
- GrabView, Bitmap2Component, PCB Calculator – zestaw dodatkowych narzędzi wspierających projektanta, które pozwalają na inspekcję plików gerber, tworzenie obudów na podstawie obrazów rastrowych oraz „szwajcarski scyzoryk” elektronika ułatwiający podstawowe obliczenia takie jak: wymagana szerokość ścieżki, wymagany prześwit między ścieżkami czy np. parametry prostych filtrów.
- Zintegrowany menedżer bibliotek komponentów oraz kształtów obudów.

Pakiet ten jest rozwijany na licencji open source, zaś jego pierwotnym autorem jest Jean-Pierre Charras, badacz w LIS (Laboratoire des Images et des Signaux) i nauczyciel w IUT de Saint Martin d'Hères (Francja) w obszarze inżynierii elektrycznej i przetwarzania obrazu[11].

2.2.4.Circuit Simulator Applet

Stworzony i rozwijany przez Paula Falstada applet Javy będący uproszczonym symulatorem układów elektronicznych ze wsparciem zarówno dla elementów cyfrowych, jak i analogowych. W przeciwieństwie do profesjonalnych aplikacji tego typu symulacja jest ciągła, a użytkownik widzi zmiany działania układu dokładnie w momencie jego edycji. Nie jest to program zalecany, ani też użyty przez autorów tego projektu jako główny symulator układów elektronicznych, ale doskonale sprawdził się w roli narzędzia do szybkiego testowania rozwiązań i odrzucania błędnych założeń.

2.2.5.Atmel Studio 6.2

Zintegrowane środowisko programistyczne stworzone i rozwijane przez Atmel Corporation służące do tworzenia oprogramowania mikrokontrolerów tego samego producenta ,a bazujące na Visual Studio. Pakiet zawiera gotowy zestaw narzędzi niezbędnych do komplikacji oprogramowania na platformy AVR oraz ARM, pełne wsparcie dla szerokiej gamy programatorów oraz, co najważniejsze współpracuje z urządzeniami odrobaczającymi typu in system debugger. Języki programowania wspierane przez tą aplikację to: C, C++ oraz Assembly.

2.2.6.In Circuit Debugger – Avr Dragon

Produkt firmy Atmel przeznaczony, przede wszystkim do współpracy z układami scalonymi tego producenta. Wspiera programowanie poprzez interfejsy ISP, JTAG oraz równoległy wysokonapięciowy. Umożliwia odrobaczanie oprogramowania układów scalonych osadzonych już w docelowym układzie za pomocą technik takich jak punkty wstrzymania oraz podgląd pamięci.

2.2.7.Arduino Mega oraz Arduino IDE

Płytką prototypową Arduino Mega bazująca na układzie Atmega2560 jest popularnym zestawem uruchomieniowym, który we współpracy z oprogramowaniem Arduino pozwala na szybkie tworzenie i testowanie oprogramowania w języku C na mikrokontrolery. W czasie powstawania projektu zestaw ten został wykorzystany do szybkiego prototypowania i testowania założeń dotyczących oprogramowania, które potem zostały stworzone od podstaw bez wykorzystania bibliotek Arduino.

2.2.8.Terminal

W początkowych fazach projektu, podczas tworzenia sterownika układu USART mikrokontrolera, wykorzystano aplikację Terminal autorstwa Br@y++ w wersji 1.93b. Jest to terminal dla interfejsów szeregowych typu COM działający pod systemem operacyjnym Windows.

2.2.9. Osciłkop Hantek 6022BE wraz z oprogramowaniem PC

Klasyczne metody testowania i znajdowania błędów w oprogramowaniu opierają się na przerywaniu pracy programu lub gromadzeniu obszernych dzienników zdarzeń. W związku z tym są one bezużyteczne przy sprawdzaniu zależności czasowych w przypadku mikrokontrolerów. Osciłkop Hantek 6022BE został zastosowany do bezinwazyjnego sprawdzania tych parametrów projektowanego układu. Rzeczone urządzenie pomiarowe to niedroga przystawka USB wykorzystująca komputer do komunikacji z człowiekiem oraz jako źródło zasilania charakteryzowana przez następujące parametry: próbkowanie 48 MHz, ośmiobitowy przetwornik analogowo–cyfrowy, impedancja wejściowa $1 \text{ M}\Omega$, 25 pF oraz maksymalne napięcie na zaciskach wejściowych równe 35 V.

2.3. Proces tworzenia

W tworzenie projektu został zaangażowany zespół złożony z trzech osób studiujących na różnych kierunkach oraz podlegających różnym katedrom. Sztywny podział zadań na moduły gwarantował praktycznie niezależną pracę każdego członka zespołu w zakresie modułu, za który odpowiadał. Niezbędne było natomiast scalenie komponentów w jedną całość.

2.3.1. Opis zespołu

Jędrzej Domański – Informatyka (KISI)

Umiejętności:

- dobra znajomość C++/C#,
- dobra znajomość zagadnień dotyczących multimedialnych, wideo, grafiki komputerowej, przetwarzania obrazów, postprodukcji audio/video, czy kompresji multimedialnych,
- duże doświadczenie związane z testowaniem oraz profilowaniem wydajności sprzętu oraz aplikacji,
- umiejętność pracy w zespole wyniesiona z pracy zawodowej.

Krzysztof Mróz – Elektronika i Telekomunikacja (KSE)

Umiejętności:

- akademicka znajomość zagadnień z dziedziny projektowania układów elektronicznych,
- empirycznie zdobyte doświadczenie w dziedzinie tworzenia prostych układów elektromechanicznych,
- dobra znajomość języków programowania zdobyta zarówno zawodowo – przez tworzenie stron internetowych oraz na stanowisku programisty BIOSu, jak i w ramach samokształcenia,
- empiryczne umiejętności wytwarzania prostych układów elektronicznych.

Maciej Rabęda – Informatyka (KIO)

Umiejętności:

- dobra znajomość C/C++/C#/Java,

- doświadczenie zawodowe w zakresie tworzenia i utrzymywania oprogramowania (głównie C i asembler),
- doświadczenie w pracy ze sprzętem wyniesione z pracy zawodowej,
- umiejętność pracy w zespole wyniesiona z pracy zawodowej.

2.3.2. Podział prac

Jędrzej Domański

- Wybór wydajnej oraz odpornej na czynniki zewnętrzne metody przechwytywania, przekształcania oraz wyświetlania obrazu na ekranie hełmu wirtualnego Oculus Rift,
- Stworzenie aplikacji GUI przenoszącej obraz z kamery na ekran hełmu wirtualnego wraz z implementacją niezbędnych rozwiązań zgodnych z obraną metodą,
- Projekt i implementacja interfejsu użytkownika umożliwiającego kontrolę nad wszystkimi niezbędnymi zmiennymi oraz modyfikatorami (w tym przekształcenia obrazu) oraz spinającego cały projekt,
- Dokumentacja.

Krzysztof Mróz

- Zaprojektowanie i wykonanie serwogłówicy z zaczepionymi do niej kamerami,
- Zaprojektowanie i wykonanie układu kontrolera głowicy,
- Stworzenie bibliotek / oprogramowania serwogłówicy,
- Zaprojektowanie i zaimplementowanie API do komunikacji z serwogłówicą,
- Dokumentacja.

Maciej Rabeda

- Stworzenie mechanizmu synchronizującego orientację hełmu z orientacją serwogłówicy,
- Zaimplementowanie mechanizmu kalibracji wektora początkowego orientacji hełmu,
- Dokumentacja.

2.3.3. Komunikacja

Ze względu na fakt, iż każdy członek zespołu ma stosunkowo mocno obciążony harmonogram, komunikacja odbywała się przeważnie drogą elektroniczną (komunikatory, e-mail) oraz telefoniczną. Spotkania twarzą w twarz między informatykami odbywały się dosyć często, natomiast między całym zespołem tylko w razie potrzeby – na przykład w chwili synchronizacji części informatycznej (oprogramowania) z elektroniczną (sprzętową, serwogłówica).

2.3.4. Współdzielenie zasobów

Rozłączny i wyizolowany charakter pracy pozwalał na współdzielenie kodu bez zdalnego repozytorium. Podjęto decyzję, o zastosowaniu prostego dysku sieciowego portalu

Dropbox. W przypadku dokumentacji – treść pracy tworzona była w dokumencie współdzielonym trzymanym na Google Drive, a formatowanie i przygotowanie tekstu do druku następowało już w programie Microsoft Word.

2.3.5. Harmonogram

Poniżej zamieszczono ramowy harmonogram prac uzgodniony w fazie przygotowań do projektu. Oprócz poniższych terminów przewidziano również spotkania odbywające się raz na trzy tygodnie, na których zespół miał korygować ewentualne błędy założeń początkowych oraz prezentować postępy na froncie robót.

- 1 VII 2014 – 31 VIII 2014: Studia zagadnienia i analiza dostępnych źródeł
- 1 IX 2014 – 15 IX 2014: Uzgodnienie założeń początkowych oraz szczegółów odnośnie projektowanych API.
- 16 IX 2014 – 15 X 2014: Stworzenie pierwszych prototypów udowadniających realizowalność idei, tzn.: biblioteki komunikacyjnej Oculus Rift, biblioteki pobierającej obraz z kamer wideo oraz oprogramowania kontrolera głowicy wraz z API komunikacyjnym
- 16 X 2014 – 1 XI 2014: Stworzenie sprawnego prototypu układu elektromechanicznego głowicy kamerowej, dopracowanie bibliotek wideo oraz bibliotek Oculus Rift.
- 2 XI 2014 – 15 XI 2014: Ukończenie tworzenia sprawnego prototypu głowicy zdolnego do komunikacji z komputerem oraz poruszania kamerami zgodnie z danymi sterującymi zadanymi przez API komunikacyjne.
- 16 XI 2014 – 1 XII 2014: Integracja systemu oraz pomiary parametrów stworzonej całości.

3. FORMALNY OPIS ZADANIA

3.1. Słowniczek

- HMD – Head Mounted Display
- VR – Virtual Reality
- AR – Augmented Reality
- THD – Through–Hole Device
- THT – Through–Hole Technology
- UUID – Universaly Unique Identifier
- WPF – Windows Presentation Foundation
- IDE – Integrated Development Environment
- CAVE – Cave Automatic Virtual Environment
- DK1 – Development Kit v.1
- DK2 – Development Kit v.2
- px – piksel
- VGA – Video Graphics Array
- QVGA – Quarter Video Graphics Array
- PAL – Phase Alternating Line
- NTSC – National Television System Committee
- FOV – Field of View
- FPS – Frames per Second
- kl./s – klatki na sekundę
- {m:n, l:k, ..., x:y} – notacja użyta do opisu układów przekładni. Liczby oddzielone dwukropkiem oznaczają współczynnik przełożenia momentu siły między dwoma bezpośrednio na siebie oddziałującymi elementami – np. zębatkami, zaś każdy przecinek oznacza, iż fragment przekładni charakteryzowany przez kolejny współczynnik jest bezpośrednio sprzężony z poprzednim – np. za pomocą wspólnej osi. Przekładnia zdefiniowania przez taki zestaw elementów ma następujący współczynnik przełożenia:

$K = \frac{m}{n} \frac{l}{k} \dots \frac{x}{y}$. Oznacza to, że szybkość kątowa na wyjściu takiej przekładni jest K-krotnie większa niż na jego wejściu.

- F_CPU – częstotliwość zegara taktującego procesor
- 0xnn, gdzie $n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ – zapis liczby w formacie szesnastkowym
- „*” (gwiazdka) – oznacza dowolny ciąg znaków

3.2. Analiza ryzyka

Ze względu na interdyscyplinarność, modułowość i stopień złożoności naszego projektu zidentyfikowaliśmy następujące potencjalne zagrożenia dla jego realizacji w ramach ustalonego harmonogramu:

- Problemy z wydajnością tworzonego rozwiązania,
- Problemy związane z brakiem fizycznego dostępu zespołu do hełmu wirtualnego,
- Problemy związane z wykorzystywanyimi subkomponentami i bibliotekami,
- Problemy związane z błędnym projektem układu elektronicznego,
- Niedostateczna wiedza w zakresie szeroko pojętej informatyki,
- Zakres pracy wykracza poza obecne umiejętności zespołu,
- Niedostateczną ilością czasu biorąc pod uwagę napięty harmonogram poszczególnych członków zespołu.

3.3. Wymagania funkcjonalne oraz niefunkcjonalne

Wymagania funkcjonalne:

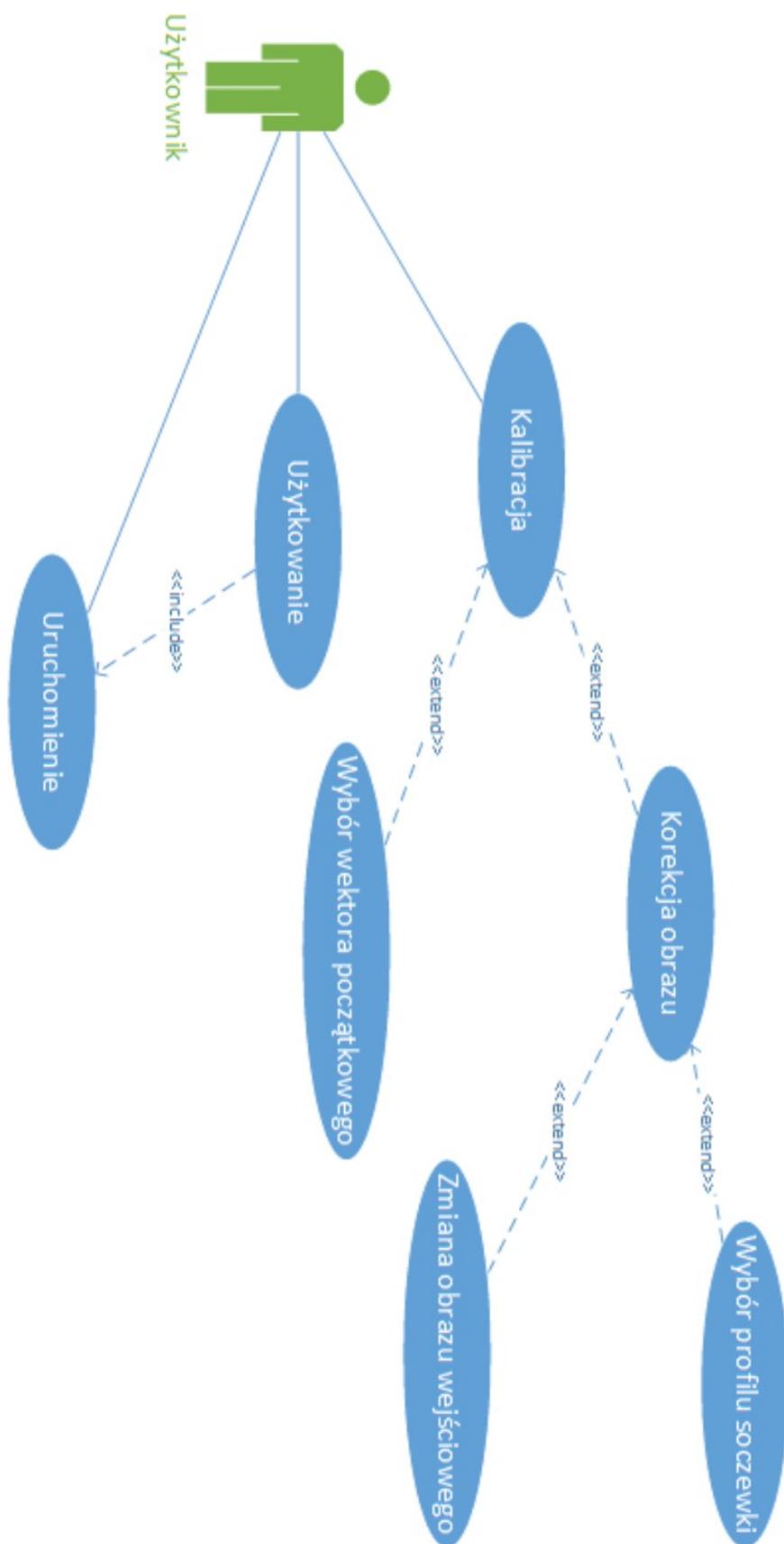
- Stworzenie mechanicznie i elektronicznie sprawnej serwogłówicy,
- Przekazywanie obrazu otrzymanego z kamery na ekran HMD,
- Przygotowanie mechanizmów synchronizacji ruchów HMD z ruchami serwogłówicy,
- Możliwość kalibracji wektora początkowego HMD,
- Możliwość modyfikacji parametrów obrazu wyświetlanego na ekranie HMD,
- Możliwość wyboru predefiniowanych ustawień obrazu w zależności od stosowanej w HMD soczewki.

Wymagania niefunkcjonalne:

- Aplikacja napisana w języku C#,
- System powinien działać pod systemami od Windows 7 w górę (zarówno 32-bit, jak i 64-bit),
- System powinien działać poprawnie, jego szybkość działania nie jest krytyczna, aczkolwiek pożądana jest akceptowalna wydajność układu,
- Termin zdania projektu do 9 grudnia 2014.

3.4. Przypadki użycia

Poniższy diagram przedstawia przypadki użycia naszego systemu:



Rys. 3.6. Diagram przypadków użycia naszego systemu

Tabela 3.1. Przypadek użycia – Użytkowanie

Nazwa	Użytkowanie
ID	UC-1
Opis	Użytkownik po wstępnej konfiguracji sprzętu może korzystać z pełnej funkcjonalności systemu
Warunki początkowe	Brak.
Przebieg	<ol style="list-style-type: none"> 1. Użytkownik podłącza hełm rzeczywistości wirtualnej oraz serwogłówce wraz z kamerami do komputera, 2. Użytkownik upewnia się, że hełm został wykryty przez system oraz jego ekran traktowany jest jako dodatkowy monitor, 3. Użytkownik uruchamia aplikację, 4. Uruchamiają się dwa okna: sterowania oraz z obrazem z kamer, 5. Użytkownik przesuwa okno z obrazem z kamer na ekran Oculusa i maksymalizuje je 6. Po nawiązaniu połączenia (patrz: przypadek użycia „Uruchomienie”) użytkownik obracając głowę, zmienia orientację hełmu rzeczywistości wirtualnej i w efekcie zmienia orientację serwogłówicy i obraz wyświetlany na ekranie hełmu.
Warunki końcowe	Aplikacja jest uruchomiona. Hełm rzeczywistości wirtualnej jest podłączony do komputera i wykryty przez aplikację. Serwogłówca wraz z kamerami jest podłączona do komputera. Serwogłówca reaguje na zmianę orientacji hełmu rzeczywistości wirtualnej. Wraz ze zmianą orientacji serwogłówicy następuje zmiana obrazu wyświetlanego na ekranie hełmu.
Przebiegi alternatywne	<ol style="list-style-type: none"> 3.1. W przypadku braku podłączenia hełmu aplikacja wyświetla stosowny błąd w kontrolce statusów. Aplikacja nie zezwoli na wykonanie jakichkolwiek akcji w ramach przypadków użycia „Uruchomienie” i „Wybór wektora początkowego”. 3.2. W przypadku braku podłączonych kamer, aplikacja wyświetla stosowny błąd w kontrolce statusów. Okno przeznaczone do wyświetlania obrazu z kamer nie prezentuje żadnego obrazu. Aplikacja nie zezwoli na wykonanie jakichkolwiek akcji w ramach przypadków użycia „Kalibracja” i jego rozszerzeń.

Tabela 3.2. Przypadek użycia – Kalibracja

Nazwa	Kalibracja
ID	UC-2
Opis	Użytkownik manipulując kontrolkami znajdującymi się w oknie kontroli może dostosowywać działanie systemu do własnych preferencji.
Warunki początkowe	Aplikacja jest uruchomiona. Hełm rzeczywistości wirtualnej jest podłączony do komputera i wykryty przez aplikację. Serwogłówca wraz z kamerami jest podłączona do komputera.
Przebieg	<ol style="list-style-type: none"> 1. Użytkownik manipuluje kontrolkami znajdującymi się w oknie kontroli.
Warunki końcowe	Obraz wyświetlany na ekranie hełmu lub wektor początkowy hełmu zostały zmodyfikowane.
Przebiegi alternatywne	<ol style="list-style-type: none"> 1.1. Użytkownik może manipulować odpowiednimi kontrolkami w celu skorygowania obrazu wyświetlanego na ekranie hełmu rzeczywistości wirtualnej. Patrz przypadek użycia „Korekcja obrazu”, 1.2. Użytkownik może manipulować odpowiednimi kontrolkami w celu ustawienia obecnej orientacji hełmu rzeczywistości wirtualnej jako orientację początkową. Patrz przypadek użycia „Wybór wektora początkowego”.

Tabela 3.3. Przypadek użycia – Uruchomienie

Nazwa	Uruchomienie
ID	UC-3
Opis	Użytkownik uzyskuje połączenie z serwogłowicą poprzez port szeregowy
Warunki początkowe	Aplikacja jest uruchomiona. Hełm rzeczywistości wirtualnej jest podłączony do komputera i wykryty przez aplikację. Serwogłówica wraz z kamerami jest podłączona do komputera.
Przebieg	<ol style="list-style-type: none"> 1. Użytkownik wybiera jedno połączenie z listy wszystkich dostępnych połączeń COM, 2. Użytkownik naciska przycisk „Connect”, 3. W kontrolce statusów pojawia się informacja o próbie nawiązania połączenia z serwogłówicą, 4. Po krótkiej chwili w kontrolce statusów pojawia się informacja o udanej próbie nawiązania połączenia z serwogłówicą.
Warunki końcowe	Serwogłówica zaczyna reagować na zmianę orientacji serwogłówicy. Kąty orientacji serwogłówicy są wyświetlane w odpowiednich kontrolkach w oknie kontrolek.
Przebiegi alternatywne	4.1. Aplikacja pokazuje informację o napotkanych problemach w przypadku nieudanej próby połączenia. Aplikacja rezygnuje z dalszych prób połączenia poprzez wybrany port.

Tabela 3.4. Przypadek użycia – Korekcja obrazu

Nazwa	Korekcja obrazu
ID	UC-4
Opis	Użytkownik koryguje obraz wyświetlany na hełmie wirtualnym
Warunki początkowe	Aplikacja jest uruchomiona. Kamery zostały wykryte. Obraz widoczny jest w oknie podglądu.
Przebieg	1. Użytkownik zmienia wartości dowolnych dostępnych parametrów korekcji obrazu wyświetlanego na ekranie hełmu wirtualnego w zależności od swoich preferencji
Warunki końcowe	Obraz został zmodyfikowany w zależności od wybranej opcji.
Przebiegi alternatywne	1.1. Użytkownik może wybrać wartość spoza ustalonego zakresu. W tym wypadku ustawiana jest zawsze wartość domyślna.

Tabela 3.5. Przypadek użycia – Wybór profilu soczewki

Nazwa	Wybór profilu soczewki
ID	UC-5
Opis	Wybór i zastosowanie predefiniowanych ustawień przetworzenia obrazu dla hełmu
Warunki początkowe	Aplikacja jest uruchomiona. Kamery zostały wykryte. Obraz z obydwu kamer jest widoczny w oknie podglądu.

Przebieg	<ol style="list-style-type: none"> 1. Użytkownik wybiera jeden z trzech przycisków: "A", "B" lub "C", należących do części interfejsu dotyczącej predefiniowanych ustawień. 2. W zależności od wybranego przycisku ustawiane są różne parametry przetworzenia obrazu nadawanego na ekran hełmu. Ustawienia zdefiniowane są empirycznie i dostosowane dla wymiennych soczewek stosowanych w hełmie
Warunki końcowe	Ustawienie różnych parametrów obrazu w zależności od wybranego przycisku. W przypadku zastosowania odpowiedniej soczewki, obraz wyświetlany na ekranie hełmu jest dla użytkownika akceptowalny.
Przebiegi alternatywne	<ol style="list-style-type: none"> 1.1. Użytkownik może wybrać też przycisk "Default", który spowoduje powrót ustawień do wartości domyślnych.

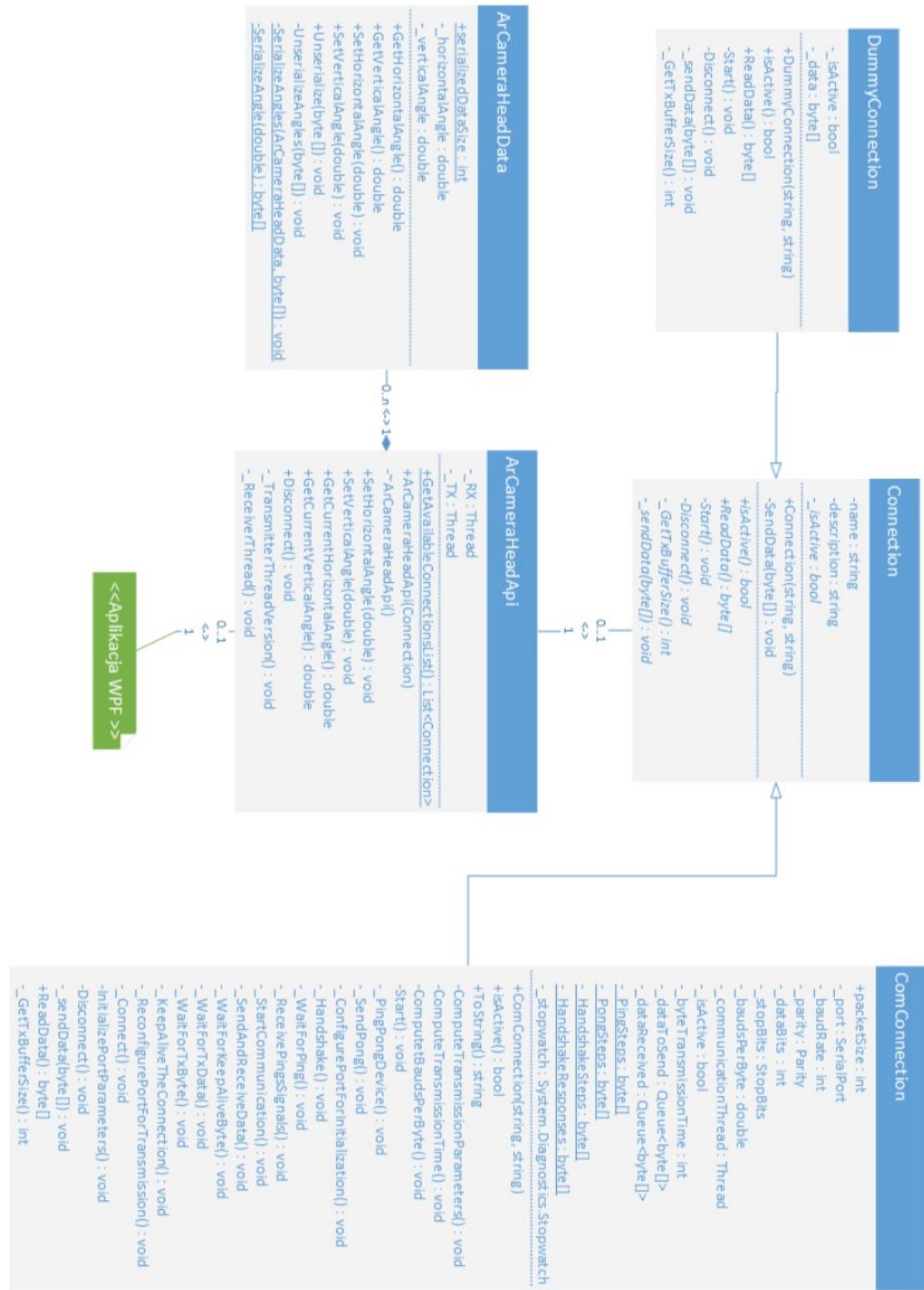
Tabela 3.6. Przypadek użycia – Zmiana obrazu wejściowego

Nazwa	Zmiana obrazu wejściowego
ID	UC–6
Opis	Obrót lub zamiana tekstur odpowiadających za obraz wejściowy z kamer
Warunki początkowe	Aplikacja jest uruchomiona. Kamery zostały wykryte. Obraz widoczny jest w oknie podglądu.
Przebieg	<ol style="list-style-type: none"> 1. Użytkownik za pomocą przycisków zmienia lub obraca tekstury obrazu wejściowego
Warunki końcowe	Tekstury odpowiadające za obraz z kamer są odwrócone i/lub zamienione.
Przebiegi alternatywne	<ol style="list-style-type: none"> 1.1. W przypadku braku wykrytych kamer wcisnięcie kontrolki nie wywołuje akcji. 1.2. W przypadku jednej kamery operacja wykonywana jest tylko dla obrazu z niej pozyskanego.

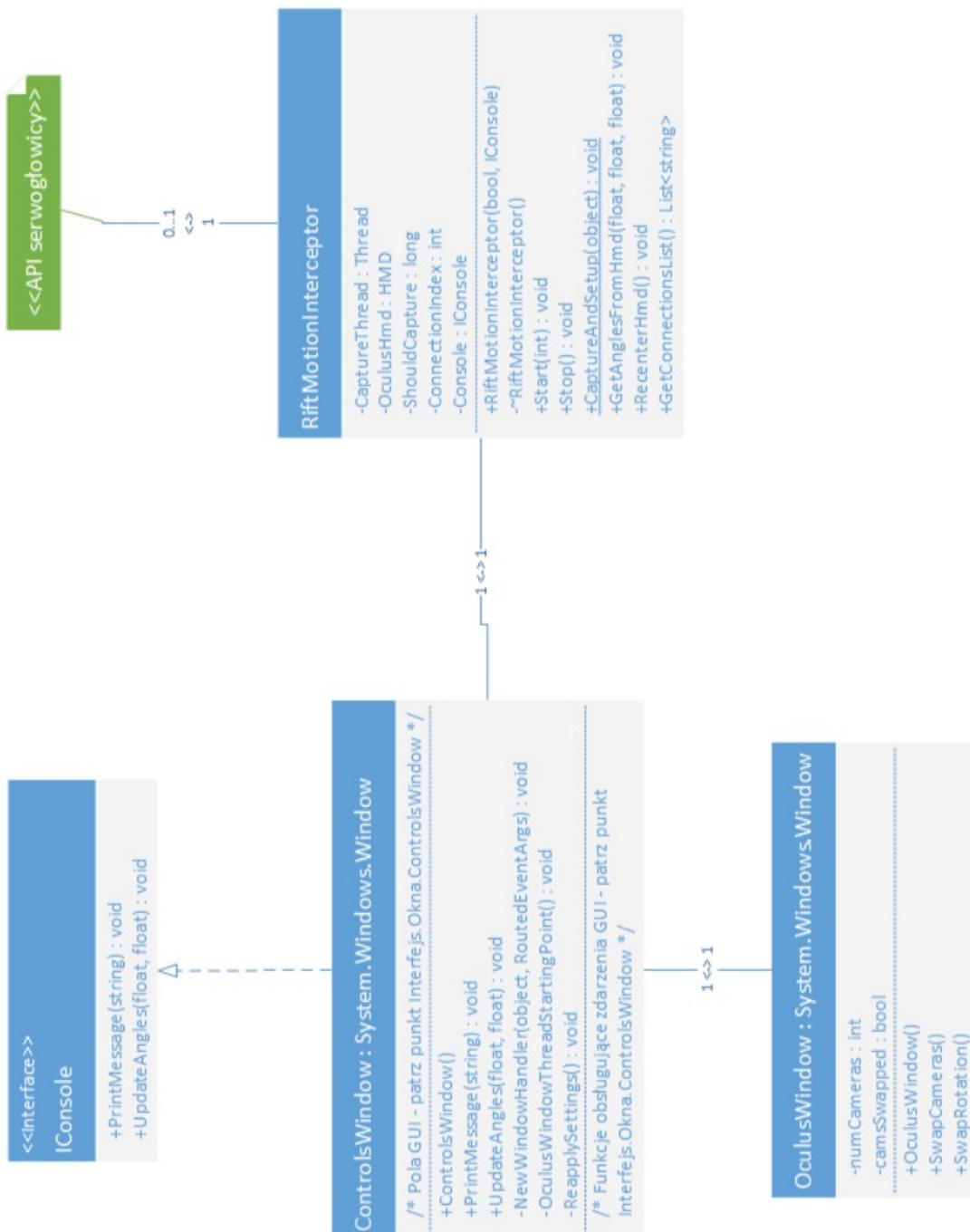
Tabela 3.7. Przypadek użycia – Wybór wektora początkowego

Nazwa	Wybór wektora początkowego
ID	UC–7
Opis	Ustawienie wektora początkowego hełmu przez użytkownika
Warunki początkowe	Aplikacja jest uruchomiona. Hełm rzeczywistości wirtualnej jest połączony do komputera i wykryty przez aplikację. Serwogłówica wraz z kamerami jest połączona do komputera.
Przebieg	<ol style="list-style-type: none"> 1. Użytkownik patrząc w przód wybiera przycisk kalibracji wektora początkowego, 2. Aplikacja ustawia od tej pory kąt serwogłówicy w oparciu o początkową orientację hełmu.
Warunki końcowe	Wartości kątów orientacji hełmu wynoszą 0. Serwogłówica powraca do swojego początkowego stanu. Wyświetlane wartości kątów orientacji hełmu są bardzo zbliżone do zera.
Przebiegi alternatywne	Brak.

3.5. Diagram klas



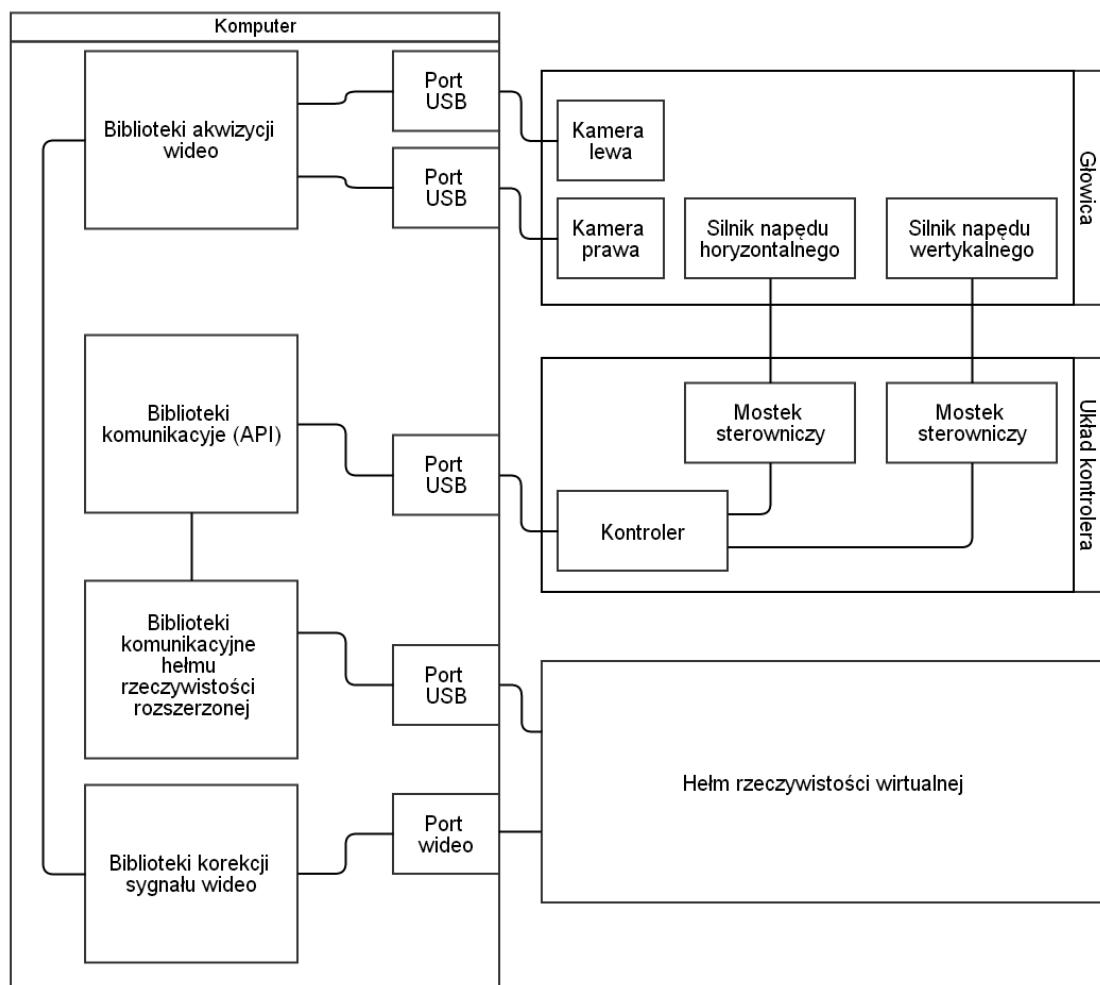
Rys. 3.7. Diagram klas API serwogłówowicy



Rys. 3.8. Diagram klas części integracyjnej oraz graficznej

3.6. Zasada działania

Celem projektu jest stworzenie działającego prototypu systemu składającego się z części sprzętowej: hełmu wirtualnego Oculus Rift, autorskiej serwogłówicy i doczepionych do niej dwóch kamer oraz oprogramowania: aplikacji z graficznym interfejsem użytkownika integrującej aspekt graficzny oraz motoryczny opisywanego zestawu. Obraz z kamer ma być przekazywany na ekran hełmu rzeczywistości wirtualnej, a zmiana orientacji hełmu ma spowodować zmianę orientacji serwogłówicy i w efekcie zmianę wyświetlanego na ekranie hełmu obrazu. Całość ma się składać z jasno wydzielonych modułów, które można będzie ponownie wykorzystać w ramach innych projektów. Modularyzacja systemu bardzo ułatwiała jasny podział odpowiedzialności między poszczególnych członków zespołu.



Rys. 3.9. Wysokopoziomowy schemat ideowy całego systemu.

3.6.1. Część graficzna

Część graficzna ma za zadanie przenieść obraz widziany przez kamery video na ekran hełmu wirtualnej rzeczywistości. Można podzielić ją na trzy moduły:

- Przechwytywanie obrazu. Musimy wybrać odpowiednie kamery pod względem ich parametrów technicznych, rozmiaru oraz możliwości współpracy z wybranym

oprogramowaniem. Ważne jest również ich wzajemne ustawienie, optyka, czy wiele innych aspektów składających się nie tylko na wydajność czy jakość, ale również przystosowanie do widzenia stereoskopowego. Następnie, w możliwie wydajny sposób powinniśmy przechwycić z nich obraz w takiej formie, która dawałaby nam możliwość jego przetwarzania w module drugim,

- Przekształcenie obrazu. Mając do dyspozycji dwa obrazy z kamer, musimy połączyć je, aby uzyskać widzenie stereoskopowe. Ponadto, przed trafieniem na hełm wirtualnej rzeczywistości obraz powinien zostać odpowiednio zniekształcony – mile widziana byłaby również możliwość parametryzacji opcji przekształcania obrazu tak, aby nasz program mógł współpracować z różnymi typami soczewek,
- Przesyłanie obrazu. W pełni przygotowany obraz powinien odpowiednio wyświetlić się na ekranie hełmu wirtualnej rzeczywistości.

Wszystkie te operacje powinny zostać wykonane w możliwie najkrótszym czasie. Wydajność zastosowanych rozwiązań ma bezpośredni wpływ na komfort z użytkowania naszego produktu. Ciąg wykonywanych zadań jest liniowy i nie pozwala na jakiekolwiek zrównoleglenie (poza równoczesnym przechwytywaniem oraz częściowym przekształcaniem obrazów z dwóch kamer przed ich złożeniem), tak więc nie możemy sobie pozwolić na to, by czas wykonywania któregokolwiek z elementów był o rząd wyższy od innych.

3.6.2. Główica

W założeniach wstępnych projektu znajduje się m.in. zastosowanie dwóch kompaktowych kamer z interfejsem USB. Fakt ten jest głównym czynnikiem decydującym o fizycznym kształcie układu i parametrach wykorzystanych elementów. Powinien być on zdolny do poruszania wyżej wspomnianymi sensorami z odpowiednią szybkością kątową. Musi zapewniać wysoki poziom powtarzalności, swobodę ruchów równą co najmniej 180 stopni zarówno wokoło osi pionowej jak i poziomej oraz dużą precyzję ustalania zadanych parametrów. Już na samym początku zdecydowano o zastosowaniu silników krokowych bipolarnych, jako przetworników elektromechanicznych, zdolnych do spełnienia wymienionych wymagań. Należy tutaj zaznaczyć, iż intencją autorów nie było stworzenie perfekcyjnej głowicy o wygórowanych standardach i specyfikacji, a połączenie niedrogich podzespołów, celem udowodnienia, iż tak zwane ciekawostki technologiczne nie muszą pochłaniać wielkich nakładów finansowych, ani wymagać niekończących się prac projektowych.

3.6.3. Kontroler głowicy

Parametry koniecznego do zaprojektowania i stworzenia układu kontrolera wynikają bezpośrednio z wymagań oraz założeń części elektromechanicznej. Są to: wymagana moc wyjściowa sterownika, rodzaj oraz liczba kanałów wyjściowych. Innym ważnym parametrem jest czas reakcji głowicy na zmianę sygnału zadanego poprzez bibliotekę komunikacyjną po stronie komputera, który nie może powodować nieprzyjemnych doznań dla użytkownika hełmu wirtualnego.

Założeniem całego projektu jest modularność oraz możliwość oddzielnego wykorzystywania każdej jednostki projektowej z osobna w różnych innych realizacjach. W związku z tym dodatkowym wymaganiem dla tegoż modułu jest możliwe duży zapas mocy wyjściowej pozwalający na np. wysterowanie głowicy współpracującej z Kinect 2, którego masa wynosi około 800g, a co za tym idzie bazującej na silnikach o wiele większym poborze mocy.

3.6.4.Zestaw bibliotek mikrokontrolera

Na potrzeby tego projektu zdecydowano o stworzeniu zestawu bibliotek oraz sterowników programowych, w tym przede wszystkim: sterownika silników krokowych, sterownika zintegrowanego układu licznika oraz sterownik komunikacji szeregowej wraz z protokołem komunikacyjnym. Nie przewiduje się tutaj stosowania technik i filozofii programistycznych typowych dla kodu tworzonego na potrzeby platform zintegrowanych takich jak np. wstawki assemblera, czy absolutny minimalizm wymagań odnoszących się do zasobów sprzętowych, którym to pierwszą część swojej publikacji poświęcił Tomasz Francuz[12].

3.6.5.Biblioteka komunikacyjna PC – układ wykonawczy

Jako że przedmiot tego podrozdziału jest swoistą linią demarkacyjną całego projektu – oddziela ona fragment czysto informatyczny od części sprzętowo-programistycznej, zatem musiała zostać zdefiniowana na samym początku pracy.

Głównym zadaniem tego modułu, a raczej API jest udostępnienie interfejsu programistycznego umożliwiającego transparentne zestawienie połączenia na wybranym kanale, zadawanie oczekiwanych kątów oraz odczyt aktualnego stanu kontrolowanego układu elektronicznego. Dodatkowym założeniem dla tej części projektu jest modularność, która ma w przyszłości pozwolić na transparentne dodanie innych metod połączenia – np. przez internet.

Należy zaznaczyć, iż technologia wybrana do zaimplementowania tego modułu to C# od Microsoft. Możliwości, które przewidziano to:

- Zadanie oczekiwanej kierunku skierowania kamer za pomocą dwóch kątów
- Odczyt kątów reprezentujących aktualny kierunek skierowania kamer
- Pobranie listy dostępnych połączeń
- Zestawienie połączenia na wybranym kanale

Poniżej zamieszczono ustalony na samym początku interfejs tejże biblioteki:

```
class ARCameraHeadApi
{
    public static List<ARCameraHeadAPI.Connection>
        GetAvailableConnectionsList
    ();
    public ARCameraHeadApi (ARCameraHeadAPI.Connection
    connection);
        public void SetHorizontalAngle (double angle);
        public void SetVerticalAngle (double angle);
        public double GetCurrentHorizontalAngle ();
        public double GetCurrentVerticalAngle();
        public void Disconnect ();
    ~ARCameraHeadApi ()
    {
```

```

        this.Disconnect();
    }
}
class Connection
{
    public readonly string Name;
    public readonly string Description;
}

```

Dokładniejszy opis i specyfikacja tego modułu znajdują się w rozdziale 4 – Realizacja projektu.

3.6.6. Moduł integracyjny

Kolejnym elementem naszego projektu jest komponent synchronizujący obroty serwogłówicy z ruchami hełmu Oculus Rift. Na komponent składać się będzie pojedyncza klasa spełniająca takie założenia:

- Będzie opakowywać główną funkcję realizowaną przez oddzielnego wątek. Funkcja ta ma cyklicznie, z dużą częstotliwością pobierać dane obrotu Oculusa z jego sensorów (akcelerometru, żyroskopu oraz magnetometru), odpowiednio je interpretować, sprowadzać do kątów Eulera (pionowego i poziomego w skali $+/- \pi$ radianów) oraz przekazywać do serwogłówicy poprzez ustalone API.
- Musi być czujna na obecność serwogłówicy – w każdym momencie działania oprogramowania może nastąpić odłączenie urządzenia, co nie powinno powodować nieoczekiwanej zakończenia pracy naszej aplikacji.
- Będzie brać pod uwagę ograniczenia mechaniczne serwogłówicy i powstrzymywać się od przekazywania do serwogłówicy kątów, których nie będzie ona w stanie ustawić.
- Będzie umożliwiać zmianę położenia wektora początkowego – innymi słowy, na kalibrację kątów orientacji hełmu Oculus względem obrotu serwogłówicy. Jest to ważne z punktu widzenia użytkownika – będzie on w stanie dostosowywać wektor swojego widzenia (np., gdy zmieni orientację krzesła, na którym siedzi) z wektorem serwogłówicy.
- Będzie realizować wzorzec projektowy typu Observator – instancja klasy będzie stroną obserwowaną i będzie informować wszystkich nasłuchujących o zmianie kątów orientacji HMD

Zastosowanie osobnego wątku jest konieczne ze względu na potrzebę minimalizacji opóźnień na linii Oculus–serwogłówica oraz zmniejszenia wpływu działania komponentu na szybkość działania całej części software'owej. Dodatkowo rozwiązanie takie pozwala na bezproblemowe strojenie częstotliwości przekazywania kątów między urządzeniami. Ze względu na fakt, że wątek ten działać będzie niezależnie od innych, nie powinna zaistnieć konieczność ich synchronizacji.

3.6.7. Interfejs

Moduł interfejsu jest naszym jedynym komponentem odpowiadającym za kontakt programu z użytkownikiem. Ma on za zadanie nie tylko zezwalać na pełen dostęp do opcji modyfikacji obrazu, ale również być elementem wiążącym wszystkie wymienione wyżej części naszego projektu. Powinien być on intuicyjny, łatwy w obsłudze, niezawodny oraz dostarczać użytkownikowi wszystkich potrzebnych informacji.

Zakładamy, że powinny znaleźć się w nim między innymi:

- Opcje odpowiadające za wszystkie dostępne przekształcenia obrazu,
- Informacje o wspieranych urządzeniach – hełmie wirtualnej rzeczywistości oraz serwogłowicy,
- Wszystkie informacje oraz kontrolki niezbędne do połączenia się z wyżej wymienionymi urządzeniami,
- Jeśli jest to możliwe (oraz nieobciążające nadmiernie platformy testowej) – wyświetlanie zadawanych przez część integracyjną kątów,
- Kontrolki oraz modyfikatory wynikające z dalszej implementacji projektu.

4. REALIZACJA PROJEKTU

W tym rozdziale zostaną szczegółowo opisane zastosowane rozwiązania i wybrane na początku metody implementacji zagadnień wyszczególnionych w rozdziale 3.

4.1. Kamera

4.1.1. Wstęp

Jedną z pierwszych decyzji, które musielismy podjąć, był wybór urządzenia, które przechwytywałoby nasz obraz. Niezbędne było ustalenie tego już na etapie planowania, z uwagi na znaczący wpływ naszej selekcji już w początkowym stadium implementacji.

Ważnymi kryteriami wyboru były między innymi przystępcość cenowa, zadowalające parametry techniczne (rozdzielcość obrazu, akceptowalne poziomy opóźnień, rejestracja wysokiej liczby klatek na sekundę), łatwość pracy z urządzeniem (programowalność, możliwość manipulacji, przechwytywanie obrazu), czy też niewielkie gabaryty (zbyt duże bądź ciężkie urządzenie mogłoby sprawiać problemy przy współpracy z serwogłowicą). Atutami były również solidna budowa urządzenia, czy dobra bądź wymienna optyka pozwalająca na uzyskanie zadowalająco szerokich kątów widzenia.

4.1.2. Wybór – PlayStation Eye

Nasz wybór padł na PlayStation Eye (Rys. 4.1.) – kamerę cyfrową zbliżoną konstrukcją do kamerki internetowej, stworzoną przez Sony z myślą o PlayStation 3. Był to następca wsławionego EyeToy'a – urządzenia wykorzystującego widzenie komputerowe oraz rozpoznawanie gestów pozwalając użytkownikowi na sterowanie grą bez wykorzystania kontrolera – niejako wyprzedzając swoją epokę (kamera ukazała się w październiku 2003 r. – przed pojawiением się popularniejszych rozwiązań takich jak Nintendo Wii, czy Kinecta skierowanego dla Xbox'a 360). Przede wszystkim, dzięki temu, że urządzenie miało swoją premierę w 2007 roku, jest dość tanie – nowy egzemplarz można kupić już za ok. 30 zł. Za tę cenę uzyskujemy solidnie wykonany, relatywnie wysokiej jakości produkt o zadowalających parametrach.

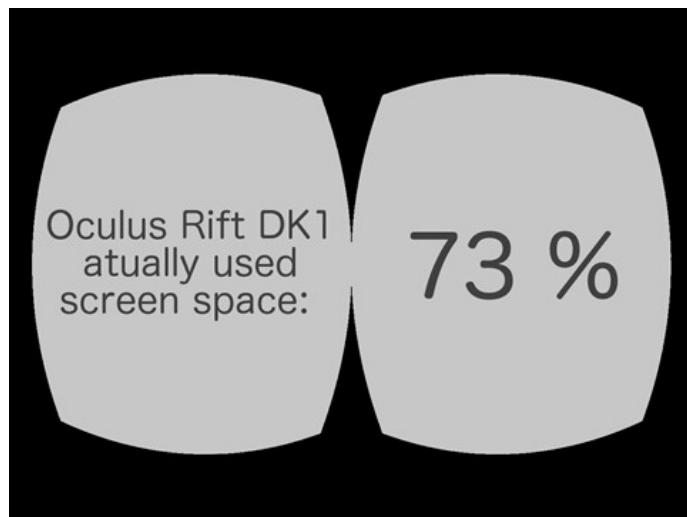


Rys. 4.10. Kamera PlayStation Eye[13]

4.1.3. Parametry techniczne

Otrzymywany obraz ma rozdzielcość VGA, czyli 640 na 480 pikseli. Jest to wartość absolutnie wystarczająca do pracy z Oculus Riftem DK1 (Development Kit 1), którego ekran oferuje nam 1280 na 800 pikseli, czyli 640 na 800 pikseli na każde oko – szczególnie biorąc pod uwagę fakt, że nie wykorzystujemy całego ekranu urządzenia oraz przed trafieniem na niego nasz obraz zostaje poddany przetworzeniu (Rys. 4.2.). Nie mamy do czynienia z rozdzielcością HD, oferowaną przez coraz większą liczbę kamer na rynku, ale w naszym przypadku jest to zaleta – mniejsza liczba pikseli oznacza również mniejszą ilość danych potrzebnych do przesłania oraz przetworzenia, a więc w rezultacie mniejsze wymagania sprzętowe wykorzystywanego komputera.

Ponadto, w przeciwieństwie do większości innych produktów na rynku w tej kategorii cenowej, oferujących przechwytywanie obrazu z prędkością 25 (standard PAL) lub niecałych 30 (standard NTSC) klatek na sekundę, kamery PlayStation Eye mogą rejestrować go z prędkością 60 lub 75 klatek na sekundę (do nawet 187 klatek w rozdzielcości QVGA – 320 na 240 pikseli) [13]. Jest to niezbędne do osiągnięcia pozytywnego odczucia z użytkowania (warto przypomnieć, że ekran Oculus Rifta DK1 odświeżany jest właśnie z częstotliwością 60Hz). Pomimo obecności niedrogich rozwiązań pozwalających na modyfikację FOV (ang. Field of View – pole widzenia) za pomocą nieoficjalnych, domontowywanych soczewek, fabryczna wersja optyki okazała się zadowalająca – każda z kamer oferuje bowiem kąty widzenia o zakresie 75 stopni, co przy percepcej stereoskopowej obrazu z dwóch kamer w pełni spełniło nasze oczekiwania (FOV w Oculus Riffcie DK1 wg. producenta wynosi 110 stopni[14]).

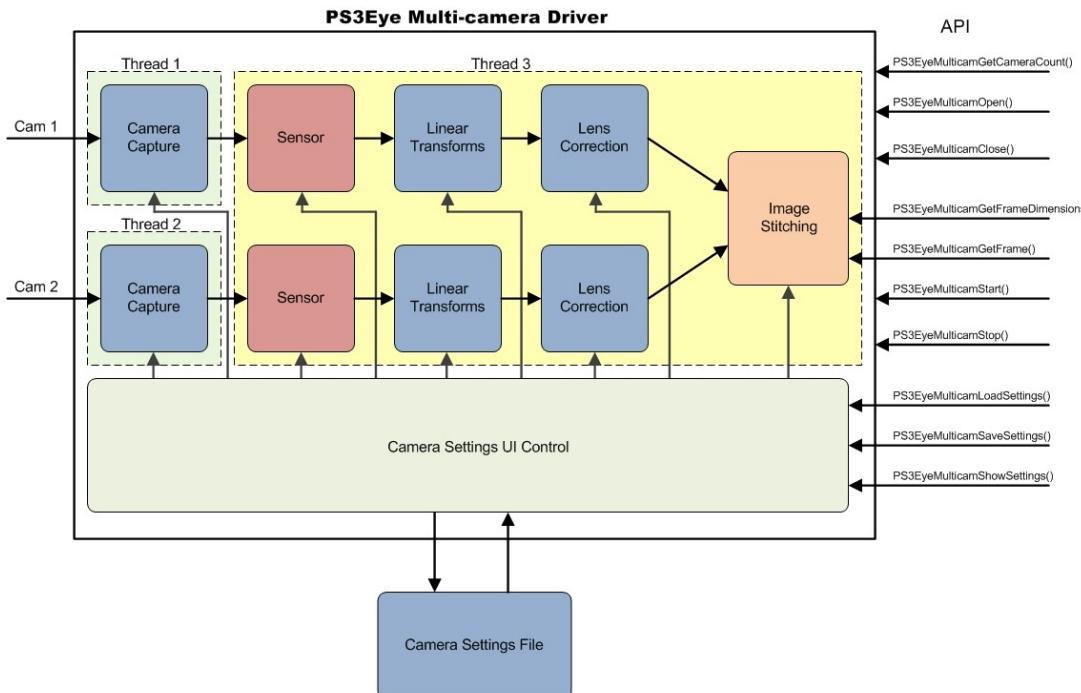


Rys. 4.11. Faktyczne powierzchnia użycia ekranu Oculus Rifta[15]

4.1.4. CL Eye Platform Driver

Mimo że parametry techniczne oraz cena wybranych kamer świetnie wpisują się w nasze wymagania, to nie te czynniki są ich największą zaletą. Firma Code Laboratories, Inc. stworzyła pakiet dedykowany wyłącznie PlayStation Eye o nazwie CL Eye Platform, na który składają się CL Eye Platform Driver (sterownik sprzętowy)[16] oraz CL Eye Platform SDK (software development kit, ang. zestaw narzędzi dla programistów)[17]. Produkt ten daje nam pełną i bezpośrednią kontrolę nad kamerami oraz ich funkcjami, wspiera używanie i rozpoznawanie wielu kamer jednocześnie, jak również udostępnia możliwość manipulacji obrazem bezpośrednio przez sterownik sprzętowy.

Poniższy diagram (Rys. 4.3.) pokazuje zastosowany przez twórców ogólny schemat podziału funkcji na wątki pozwalający na efektywne wykorzystanie zasobów sprzętowych oraz minimalizację opóźnień włączając używane przez nas przekształcenia obrazu opisywane w podrozdziałach 4.2.3. oraz 4.9.



Rys. 4.12. Podział na wątki zadań realizowanych przez CL Eye Driver[18]

Poniżej przedstawię fragmenty pliku nagłówkowego CL-Eye Multicam API zawierającego udostępnione przez CL Eye Platform SDK funkcje wraz z wartościami, które możemy poddać zmianom oraz opiszę ich znaczenie bądź działanie[19]:

- Mamy możliwość wyboru trybu, w jakim kamera przechwytuje obraz – kolor bądź skala szarości:

```
// camera color mode
typedef enum
{
    CLEYE_GRAYSCALE,
    CLEYE_COLOR
} CLEyeCameraColorMode;
```

- Możemy przechwytywać obraz w rozdzielcości VGA (640 na 480 pikseli) z prędkościami 15, 30, 40, 50, 60 lub 75 klatek na sekundę albo QVGA (320 na 240 pikseli) z prędkościami 15, 30, 60, 75, 100 lub 125 klatek na sekundę. Warto tym samym nadmienić, że istnieje możliwość wymuszenia maksymalnej liczby klatek na sekundę w trybie QVGA – 187 (poprzez zmianę parametru cameralimage.Device.Framerate na 187 przy upewnieniu się, że jej parametr cameralimage.Device.Resolution ma wartość "QVGA"):

```
// camera resolution
typedef enum
{
    CLEYE_QVGA, // Allowed frame rates: 15, 30, 60, 75, 100, 125
    CLEYE_VGA   // Allowed frame rates: 15, 30, 40, 50, 60, 75
} CLEyeCameraResolution;
```

- Edytowalne parametry urządzenia. Poniższe wartości, w przeciwieństwie do podanych wcześniej, mogą być zmieniane w trakcie funkcjonowania kamery (nie wymagają jej

ponownej inicjalizacji). Należą do nich między innymi wzmocnienie, ekspozycja i balans bieli obrazu (ustawiane ręcznie bądź automatycznie), przewrócenie obrazu w pionie oraz poziomie, wartość zwornika poziomego i pionowego (służących do korekcji zniekształceń trapezowych), przesunięcie w pionie i poziomie, rotacja, powiększenie, jak również – szczególnie dla nas ważna – korekcja soczewki, umożliwiająca stosowanie dystorsji poduszkowej oraz beczkowej przez sterownik:

```
// camera parameters
typedef enum
{
    // camera sensor parameters
    CLEYE_AUTO_GAIN,           // [false, true]
    CLEYE_GAIN,                // [0, 79]
    CLEYE_AUTO_EXPOSURE,       // [false, true]
    CLEYE_EXPOSURE,             // [0, 511]
    CLEYE_AUTO_WHITEBALANCE,   // [false, true]
    CLEYE_WHITEBALANCE_RED,    // [0, 255]
    CLEYE_WHITEBALANCE_GREEN,  // [0, 255]
    CLEYE_WHITEBALANCE_BLUE,   // [0, 255]
    // camera linear transform parameters
    CLEYE_HFLIP,               // [false, true]
    CLEYE_VFLIP,                // [false, true]
    CLEYE_HKEYSTONE,            // [-500, 500]
    CLEYE_VKEYSTONE,            // [-500, 500]
    CLEYE_XOFFSET,               // [-500, 500]
    CLEYE_YOFFSET,               // [-500, 500]
    CLEYE_ROTATION,              // [-500, 500]
    CLEYE_ZOOM,                  // [-500, 500]
    // camera non-linear transform parameters
    CLEYE_LENSCORRECTION1,     // [-500, 500]
    CLEYE_LENSCORRECTION2,     // [-500, 500]
    CLEYE_LENSCORRECTION3,     // [-500, 500]
    CLEYE_LENSBRIGHTNESS,       // [-500, 500]
} CLEyeCameraParameter;
```

- Funkcje zwracające informacje o kamerach – kolejno liczbę aktualnie podłączonych urządzeń oraz UUID (ang. Universally unique identifier – uniwersalny, unikalny identyfikator):

```
// Camera information
CLEYEMULTICAM_API int CLEyeGetCameraCount();
CLEYEMULTICAM_API GUID CLEyeGetCameraUUID(int camId);
```

- Funkcje inicjalizujące i niszące instancję kamery:

```
// Library initialization
CLEYEMULTICAM_API CLEyeCameraInstance CLEyeCreateCamera(GUID
camUUID, CLEyeCameraColorMode mode, CLEyeCameraResolution res,
int frameRate);
CLEYEMULTICAM_API bool CLEyeDestroyCamera(CLEyeCameraInstance
cam);
```

- Funkcje rozpoczynające i kończące pracę kamery:

```
// Camera capture control
CLEYEMULTICAM_API bool CLEyeCameraStart(CLEyeCameraInstance cam);
CLEYEMULTICAM_API bool CLEyeCameraStop(CLEyeCameraInstance cam);
```

- Settery i gettery dla wymienionych wcześniej parametrów urządzenia:

```
// Camera settings control
```

```

CLEYEMULTICAM_API bool
CLEyeSetCameraParameter(CLEyeCameraInstance cam,
CLEyeCameraParameter param, int val);
CLEYEMULTICAM_API int CLEyeGetCameraParameter(CLEyeCameraInstance
cam,CLEyeCameraParameter param);

```

- Funkcje pobierające wymiary ramki oraz obraz:

```

// Camera video frame image data retrieval
CLEYEMULTICAM_API bool
CLEyeCameraGetFrameDimensions(CLEyeCameraInstance cam, int&width,
int &height);
CLEYEMULTICAM_API bool CLEyeCameraGetFrame(CLEyeCameraInstance
cam, PBYTE pData,int waitTimeout = 2000)

```

4.1.5. Przystosowanie kamer

Chcąc uzyskać uczucie widzenia stereoskopowego, wzajemne umiejscowienie kamer powinno jak najbardziej odzwierciedlać położenie ludzkich oczu. W zaawansowanej, profesjonalnej kinematografii 3D używane są zestawy składające się z dwóch kamer poruszających się synchronicznie po szynach, mogące zmieniać kąty, pod którymi obserwują obiekt, naśladowując tym samym ludzkie oko.

Nasze rozwiązanie niestety w tej kwestii będzie uproszczone z kilku powodów. Po pierwsze, wymagałoby to śledzenia ruchów oka – funkcjonalności niewspieranej przez Oculus Rift DK1. Ponadto, nawet zakładając posiadanie takich danych, byłoby to trudne do odpowiedniego oprogramowania, nie mówiąc o mechanicznym oraz elektronicznym skomplikowaniu serwogłów. Opisywany proces kręcenia filmów oraz odpowiedniego operowania kamerami jest wcześniej skrupulatnie planowany przez ekipę realizatorską i całość odbywa się za pośrednictwem człowieka – odpowiednio programującego (lub manualnie sterującego) położeniem urządzeń. Warto nadmienić również, że na rynku pojawiły się konsumenckie kamery 3D, w których wzajemne położenie układów optycznych jest stałe (Rys. 4.4).



Rys. 4.13. Profesjonalny zestaw do filmowania 3D (po lewej)[20] oraz konsumencka kamera video umożliwiająca filmowanie w 3D (po prawej)[21]

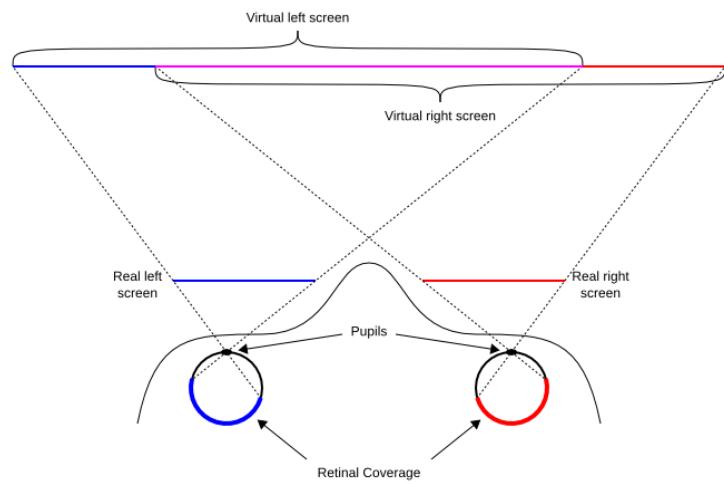
Jak zatem powinny być położone nasze kamery? Założyliśmy, że aby najlepiej odwzorować stereoskopowe postrzeganie świata, należałoby ustawić je tak, by naśladowały oczy patrzące do przodu, gdzie obiekt skupienia powinien być położony 12 stóp (około 3,65

metrów) od obserwatora – zgodnie z ustaniem konwergencji w Oculus Riftie DK1[22]. Ponadto, odległość pomiędzy obiektywami kamer powinna być równa średniemu rozstawowi oczu. Z uwagi na duży mikrofon stereo zamocowany na górze każdej z kamer, ich ustawienie poziome zgodne z założeniami byłoby niemożliwe. W tym celu musielibyśmy ustawić kamery bokiem do siebie (złożyć podstawkę do podstawki) (Rys. 4.5.). Ustawienie to okazało się mieć niemal same zalety – kosztem obliczeniowym obrócenia obrazu z kamery o 90 lub 270 stopni (na który jak najbardziej było nas stać), otrzymaliśmy idealną odległość między urządzeniami PlayStation Eye (około 62 mm pomiędzy środkami soczewek). Zmiana była podyktowana również względem proporcji obrazu – na 800 pikseli ekranu Oculus Rifta przypadało 640 pikseli obrazu z kamery, a na 640 pikseli ekranu – 480 pikseli obrazu. Rozwiązanie okazało się być również bardzo wygodne, dając nam pełną swobodę przy obracaniu kamer w osiach x i y, jak ludzkim okiem.

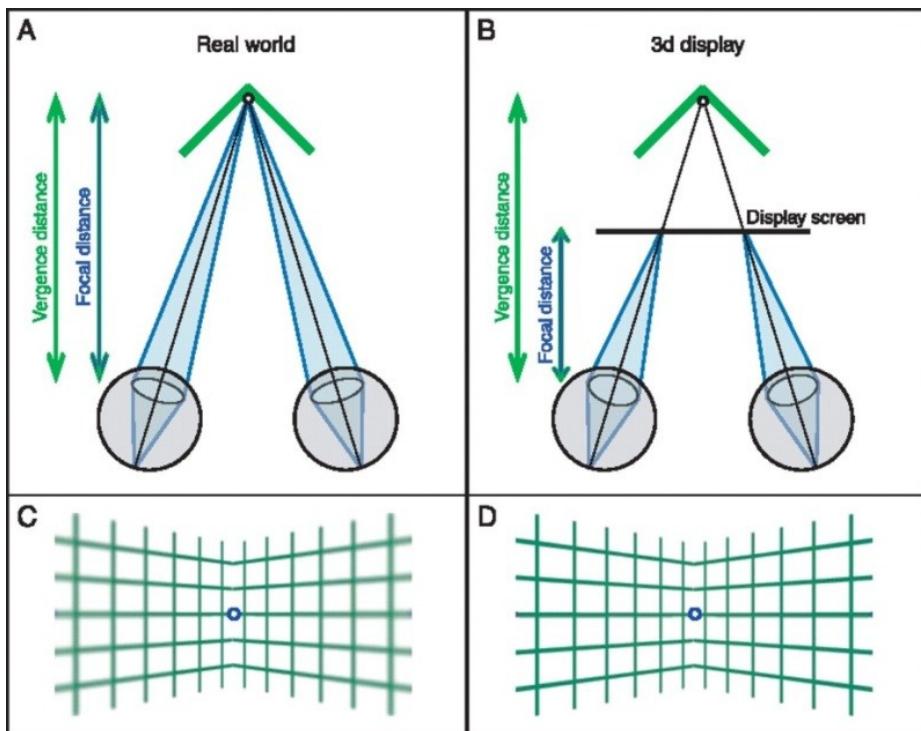


Rys. 4.14. Zastosowany przez nas układ wzajemnego położenia kamer

W zestawie Development Kit 1, wraz z hełmem wirtualnym otrzymujemy między innymi 3 soczewki asferyczne, oznaczone "A", "B" i "C", skierowane dla osób bez wady wzroku, dalekowidzów i krótkowidzów. Ich wartości to 22.5, 23.5 oraz 24.5 dioptrii (informacje nieoficjalne)[22]. Chcąc otrzymać odpowiednie ustawienia rozstawu kamer, wartości tablicy współczynników przekształcenia (o której mowa w punkcie 4.2.3), wartości zbieżności i inne parametry, musielibyśmy więc wziąć wszystkie rodzaje soczewek pod uwagę. Zgodnie z założeniem, część obrazu powinna być widoczna dla obydwu oczu, jak przedstawiono na Rys. 4.6. Niestety, problemem aktualnie nierozwiązywalnym, związanym z wciąż dość prymitywną budową hełmów wirtualnej rzeczywistości, jest różnica pomiędzy zbieżnością i głębią obrazu na ekranie HMD oraz w rzeczywistości (Rys. 4.7.). Niestety, dokumentacja oraz sieć milczą na temat orientacyjnych wartości tablicy współczynników przekształcenia lub sposobu ich wyznaczenia – byliśmy więc zmuszeni do oszacowania tych wartości samodzielnie metodą prób i błędów (przy odpowiednio dużej próbce testowej), wspierając się między innymi klasyczną szachownicą, udostępnioną nawet w CL Eye SDK.



Rys. 4.15. Wizualizacja obrazu widzianego przez hełm wirtualny[23]



Rys. 4.16. Różnica pomiędzy zbieżnością a głębią obrazu na ekranie HMD oraz w rzeczywistości[24]



Rys. 4.17. Ochotnicy testujący nasz projekt podczas prezentacji prototypu na Politechnice Gdańskiej

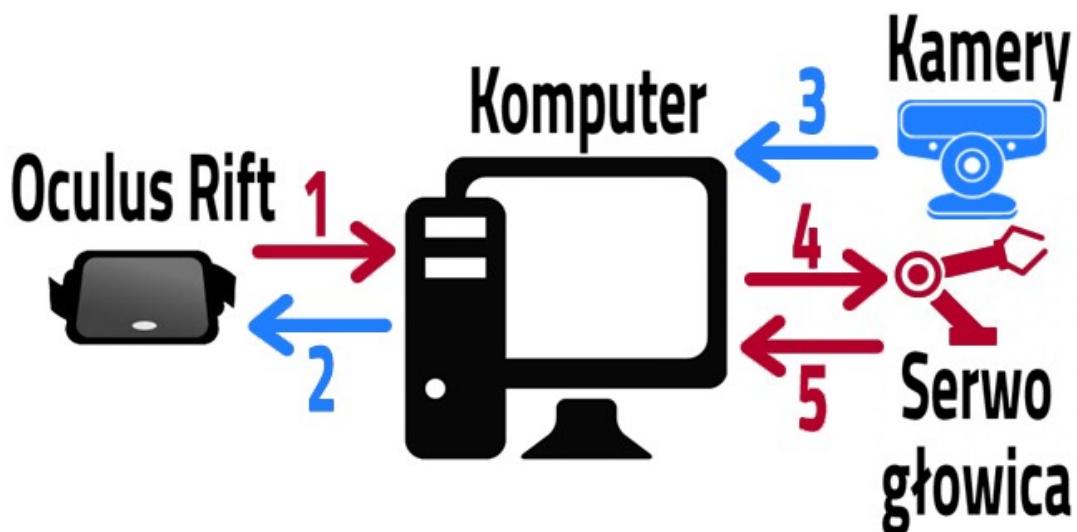
Wybrane wartości tablicy współczynników przekształcenia sprawdziliśmy eksperymentalnie wraz z innymi konfiguracjami i zgodnie z założeniem okazało się to być najbardziej uniwersalne i efektywne położenie. Przy grupie testowej składającej się z ponad 10 osób o różnej budowie ciała (oraz oka, wliczając jego wady), wszyscy uznali obraz za zaskakująco naturalny.

4.2. Przesyłanie obrazu do Oculus Rifta

4.2.1. Wstęp

Jednym z kluczowych problemów naszej pracy było przesłanie obrazu z kamer do Oculus Rifta. Zadanie to, na pozór łatwe, wiązało się jednak z kilkoma wyzwaniami, którym podołanie było niezbędne do prawidłowego działania projektu oraz wygody użytkownika.

Głównym problemem było zminimalizowanie opóźnień. Aby korzystanie z naszego układu było przyjazne użytkownikowi, droga obrazu od soczewki kamery aż do ludzkiego oka powinna zająć jak najmniej czasu. Latencja była w naszym przypadku składową kilku elementów – opóźnienia własnego kamer, odbioru, przetworzenia i wysłania obrazu do hełmu wirtualnego, na latencji wewnętrznej Oculus Rifta kończąc. O ile nie mieliśmy wpływu na ostatnie z wymienionych (według producenta wynoszą one od 40 do 50 ms[25]), od naszego wyboru zależały opóźnienia kamer (przy czym opisane powyżej Playstation Eye mogą pochwalić się nienajgorszym wynikiem 30 ms – wartość zmierzona na platformie testowej 4a opisanej w punkcie 5.2.4.) oraz opóźnienia wynikające przetworzenia obrazu. Nie dążymy jednak do najniższych fizycznie osiągalnych opóźnień – optymalnymi wartościami byłyby zbliżone do tych na drodze sensory ruchu (Oculus Rift) – serwogłówica (Rys. 4.9.). Możemy więc pozwolić sobie na rozsądne rozłożenie nakładu pracy, czy implementację rozwiązań nieznacznie wolniejszych, ale stabilniejszych, czy też będących źródłem innych korzyści.



Rys. 4.18. Schemat przedstawiający dwie grupy opóźnień – 2,3 – niebieski oraz 1,4,5 – czerwony

4.2.2. Dlaczego obraz wymaga przetworzenia?

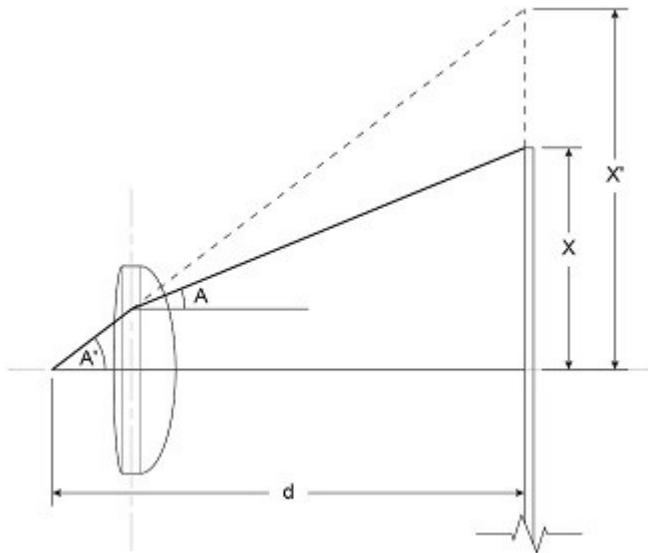
Zanim jednak przejdziemy do opisu implementacji, powinniśmy zrozumieć jaki obraz trafia na Oculus Rifta i jak (oraz dlaczego) jest przetwarzany. Oculus Rift posiada jeden ekran, podzielony na dwie części plastikową podziałką – każde z oczu widzi dokładnie połowę ekranu. Oprócz połączenia obrazu z dwóch kamer, które jest trywialną transformacją (zarówno pod względem implementacyjnym, jak i obliczeniowym), ważnym elementem była dystorsja

beczkowa – kompensująca zniekształcenie soczewek wykorzystywanych w Oculus Riftie. Przyjrzyjmy się ekranowi Oculus Rifta: przy rozmiarze 7 cali, oferuje on rozdzielcość 1280 na 800 pikseli[25] – co oznacza dużo większą liczbę dpi (ang. dots per inch – punktów na cal) niż standardowy monitor i budową przypomina bardziej ekran smartfona (Rys. 4.10.).



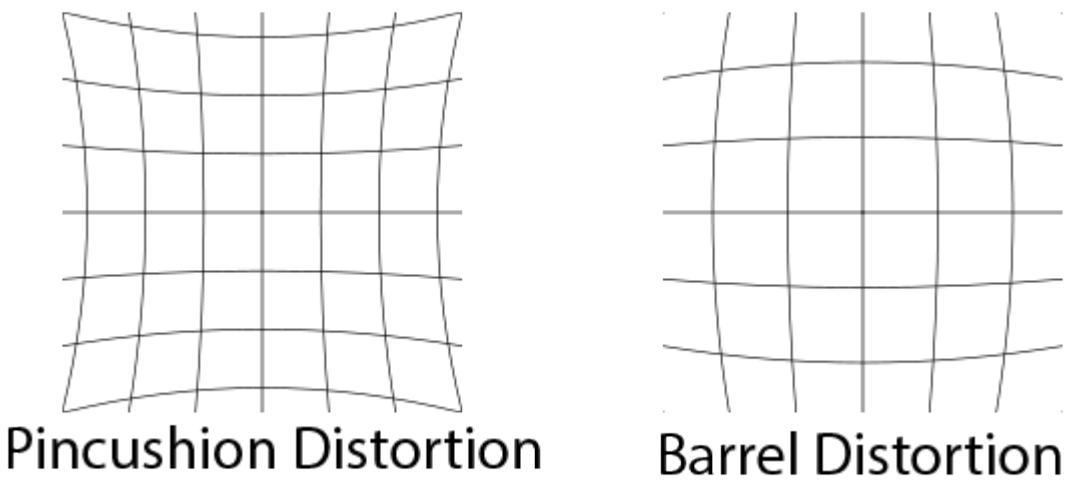
Rys. 4.19. Ekran Oculus Rifta po zdjęciu osłony[6]

Soczewki znajdujące się w Oculus Riftie nadają widzianemu obrazowi dystorsję poduszkową. Rozwiążanie to ma na celu poszerzenie pola widzenia. Centralny punkt obrazu nie jest przekształcany – im natomiast od niego dalej (oraz im większy kąt pomiędzy środkiem soczewki oraz miejscem, na który skierowany jest wzrok użytkownika hełmu), tym bardziej obraz jest zniekształcony. Operując na przykładzie – odwracając wzrok o 60° w prawo, zobaczymy punkt położony na ekranie o 45° od centrum (wartości orientacyjne) (Rys. 4.11.). Tym samym ekran sprawia wrażenie większego, a przede wszystkim położonego dalej od naszych oczu. Właśnie dzięki temu Oculus Rift może być tak lekki, przenośny, jak również zdecydowanie tańszy do wyprodukowania od swoich starszych i większych, wspomnianych we wstępie braci (jak np. Miecz Damoklesa) – oraz nadaje się do „trafienia pod strzechy”.



Rys. 4.20. Zasada działania soczewki w Oculus Riftie[26]

Aby przeciwdziałać nienaturalnie rozciągniętemu obrazowi, przed trafieniem na ekran Oculus Rifta, musi on zostać odpowiednio przekształcony – kompensując tym samym zniekształcenie soczewki. Odwrotnością dystorsji poduszkowej jest dystorsja beczkowa i właśnie to przekształcenie powinniśmy zastosować (Rys. 4.12.).



Rys. 4.21. Dystorsja poduszkowa (po lewej) oraz beczkowa (po prawej)[26]

4.2.3. Dystorsja beczkowa

Jak ustaliliśmy w punkcie 4.2.2., nasz obraz musi zostać poddany dystorsji beczkowej. Dotyczy to obydwu składowych obrazu trafiającego na Oculus Rifta – jako że na każde oko przypada jedna soczewka. Transformacja ta nie jest wymagająca obliczeniowo, chociaż dopiero dzięki ostatniemu gwałtownemu wzrostowi mocy obliczeniowej, zgodnie z prawem Moore'a, możemy wykonywać ją w czasie rzeczywistym przy zadowalająco wysokiej liczbie klatek na sekundę. Algorytm[26] sprowadza się do trzech punktów:

- 1) Znajdź odległość pomiędzy centrum dystorsji oraz aktualnym pikselem (oznaczmy ją jako R , z uwagi na jej tożsamość z promieniem okręgu, na którym leży),
- 2) Z pomocą tablicy współczynników, przemnóż R podniesiony do określonej potęgi dla danego współczynnika z danym współczynnikiem oraz zsumuj wyniki obliczeń. W ten sposób otrzymamy skalę dystorsji, którą oznaczmy jako S ,
- 3) Przemnóż pierwotne wartości położenia punktu x i y przez S . Nowe wartości x' oraz y' oznaczają pozycję punktu znajdującej się na pierwotnym obrazie, który powinniśmy wyświetlić w tym miejscu.

W przypadku Oculus Rifta, chcąc uzyskać dostateczny zakres oraz dokładność przekształcenia, przyjęło się używać trzech współczynników – nazwijmy je $k1$, $k2$ oraz $k3$. Zakładamy również, że punktem środkowym przekształcenia o współrzędnych (x,y) równych $(0,0)$ jest środkowy punkt obrazu (w naszym przypadku – przypadającym na dane oko). Transformację można więc inaczej zapisać:

$$x' = x * (1 + k1 * R^2 + k2 * R^4 + k3 * R^6) \quad (4.1)$$

$$y' = y * (1 + k1 * R^2 + k2 * R^4 + k3 * R^6) \quad (4.2)$$

gdzie:

x' – położenie punktu w poziomie po przekształceniu [px],

y' – położenie punktu w pionie po przekształceniu [px],

x – pierwotne położenie punktu w poziomie [px],

y – pierwotne położenie punktu w pionie [px],

$k1, k2, k3$ – wartości współczynników dystorsji beczkowej,

$R \sqrt{x^2 + y^2}$ – odległość od środka do punktu przekształcenia [px].

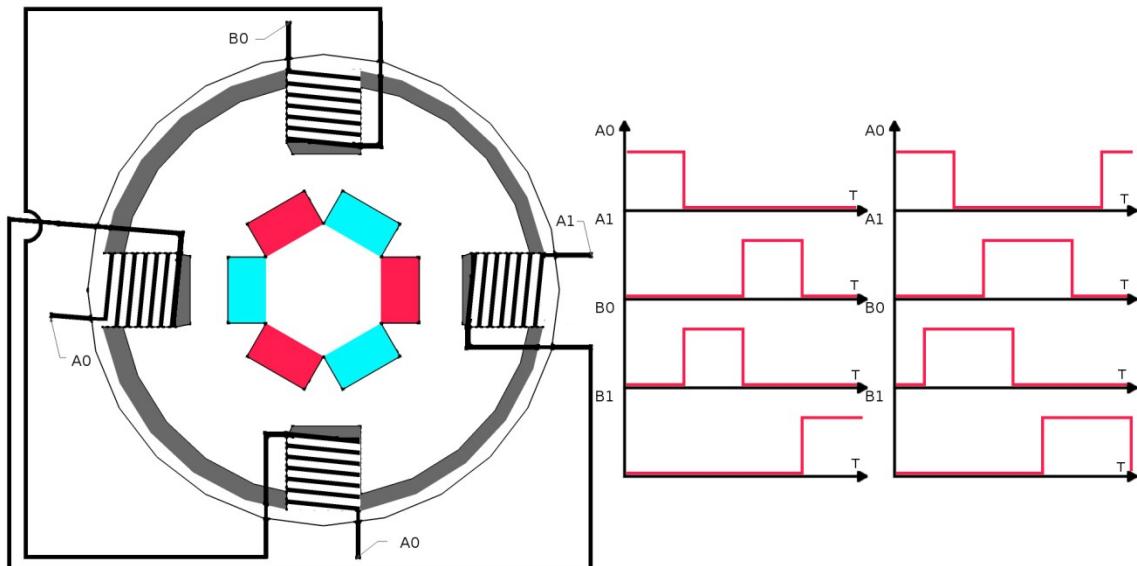
4.3. Główica

Projektowanie tego modułu rozpoczęto od wyboru silników odpowiedzialnych za poruszanie całego układu. Wybór padł na 39BYGH 405B – bipolarny silnik krokowy o standardowych wymiarach 39 mm x 39 mm. Parametrami branymi pod uwagę były: masa własna, generowany moment siły oraz cena. Zadecydowano o wykorzystaniu dwóch identycznych silników do poruszania głowicą wokół obu osi.

4.3.1. Silniki Krokowe

Silnik krokowy to przetwornik energii elektrycznej na ruch obrotowy. Uzyskuje się taki efekt dzięki zastosowaniu zestawu cewek przymocowanych do stojana oraz odpowiednio spolaryzowanych i przymocowanych wokół wirnika magnesów stałych. Ważną cechą tego rodzaju konstrukcji, jest to, że przy zapewnieniu odpowiednich warunków pracy, można być pewnym aktualnego położenia rotora. Cechą ta pozwoliła na rezygnację z zastosowania układu sprzążenia zwrotnego kontrolującego aktualne wychylenie rotora. Na rysunku 4.13 przedstawiono schematyczną budowę takiego silnika z zaznaczonymi cewkami, połączeniami

między nimi oraz magnesami stałymi z zaznaczanymi polaryzacjami symbolizowanymi przez odpowiednie kolory.



Rys. 4.22. Schemat budowy silnika krokowego bipolarnego oraz kształty sygnałów sterujących na zaciskach wejściowych dla przypadku sterowania pełnymi krokami (lewy wykres) oraz pół-krokami (prawy wykres).

Proces sterowania takim urządzeniem polega na zadawaniu na zaciskach wejściowych sekwencji napięć, które to przetwarzane są na pole magnetyczne generowane przez cewki. Wygenerowane w ten sposób pole wymusza obrót wirnika do najbliższej pozycji, w której polaryzacja magnesów stałych jest z nim zgodna. Kształt sekwencji sterującej zależy od wybranego trybu sterowania – pełnymi krokami lub półkrokami. Obydwa rozwiązania posiadają swoje zalety – pierwszy zapewnia generowanie maksymalnej siły napędowej/oporu, drugi zaś dwukrotnie zmniejsza rozmiar kroku, czyli minimalne możliwe do uzyskania przesunięcie kątowe rotora. Kształty sygnałów sterujących dla obydwóch metod kontroli silnika zostały zaprezentowane na rysunku 4.13. Jak łatwo zauważać tryb półkrokowy polega na podwojeniu liczby stanów sygnału sterującego poprzez dodanie kroków pośrednich względem sygnału w trybie pełnych kroków[27].

Silnik krokowy jest urządzeniem wykorzystującym cewki oraz będące w ich bezpośrednim sąsiedztwie ruchome magnesy stałe, dlatego też podczas prac projektowych nad bardzo precyzyjnymi układami sterowania należy brać pod uwagę takie zjawiska jak: generacja prądu w wyniku poruszania się silników oraz impedancyjny charakter całego urządzenia z uwzględnieniem sprzężeń między poszczególnymi elementami układu. W tej pracy zastąpiono długie i żmudne analizy tych zagadnień doborem kluczów sterowników o parametrach kilkukrotnie przekraczających znamionowe parametry silnika.

Parametr silników zastosowanych w projekcie[28]:

- Wymiary obudowy: 39x39x34 mm, wymiary osi: 18x5 mm.
- Rezystancja każdego z uzojeień: 30Ω
- Napięcie pracy cewek: 12V

- Wielkość kroku: 1,8 stopnia
- Szybkość kątowa: 5 obrotów na sekundę przy sterowaniu półkrokowym z częstotliwością zmian sygnału sterującego równą 1 kHz.

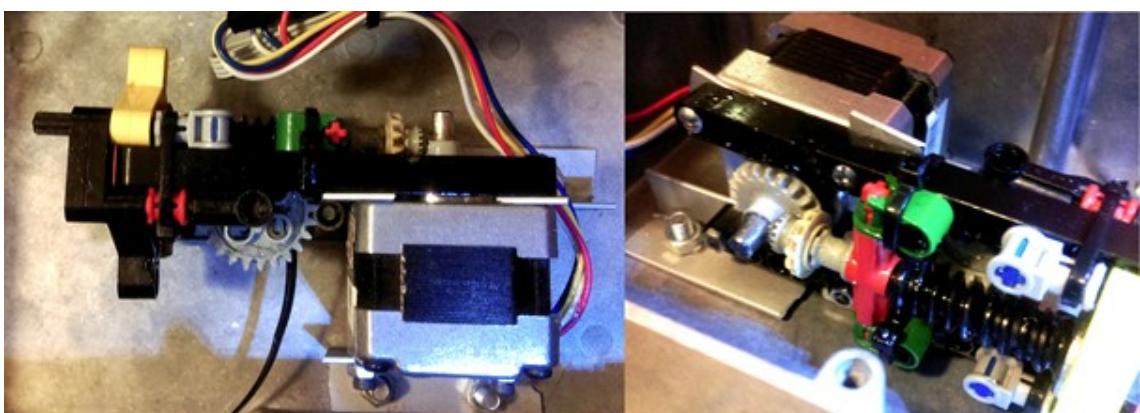
Rozdział został opracowany częściowo na podstawie informacji ze strony hteck.ca [29].

4.3.2. Przekładnie

W realizacji serwomechanizmów zastosowano przekładnie ślimakowe. Ten specyficzny rodzaj przekładni mechanicznej pozwala na zrealizowanie wysokiego współczynnika przełożenia momentu siły w kierunku od zębatki ślimakowej (ślimaka) o wartości równej liczbie zębów podłączonego koła zębatego (ślimacznicy) oraz niemal uniemożliwia transfer sygnału w przeciwnym kierunku. Druga z opisanych tutaj własności stała się głównym powodem zastosowania tego rodzaju układu – służą za mechaniczny odpowiednik diody półprzewodnikowej chroniący silnik krokowy przed zmianą jego stanu w wyniku działania sił zewnętrznych – np. osób "ciekawskich", oraz sił bezwładności. Działanie takie umożliwia uzyskanie o wiele wyższej powtarzalności całego modułu.

4.3.3. Napęd osi poziomej

Układ serwomotoru kontrolującego obrót wokół osi pionowej wykonano stosując metodykę inżynierii empirycznej oraz tanie i łatwo dostępne Lego Technics w charakterze podzespołów. Jako że moduł ten musi poruszać całą głowicą składającą się z kamery oraz drugiego silnika wraz z przekładniami zastosowano przekładnie o kolejnych współczynnikach {20:12, 1:24}, co oznacza iż maksymalna teoretyczna prędkość kątowa na wyjściu tego układu to 25 stopni na sekundę.

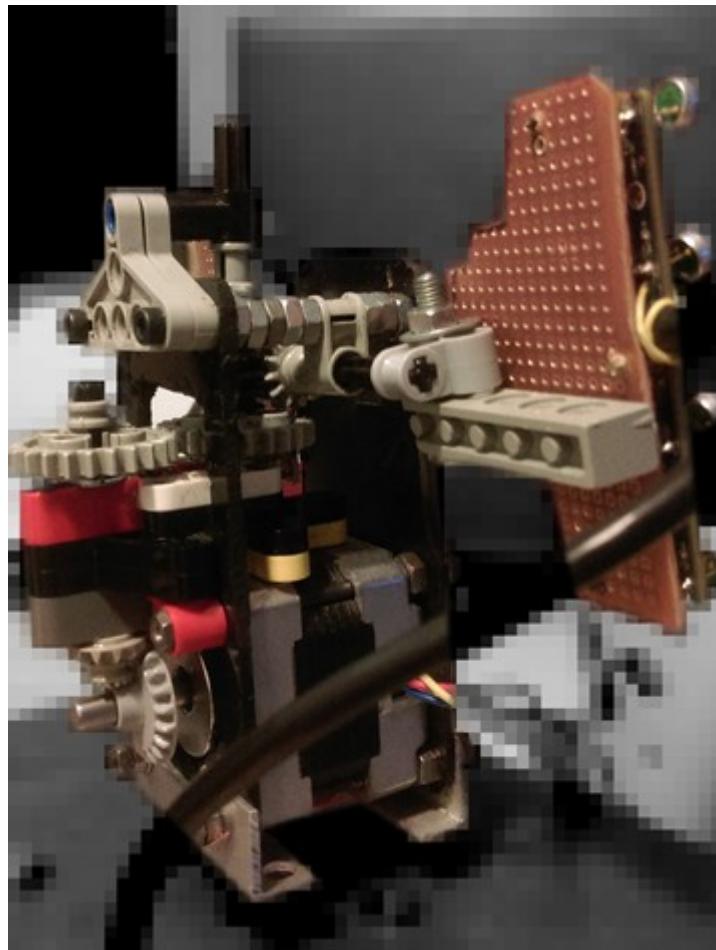


Rys. 4.23. Zdjęcia napędu osi pionowej.

Na zdjęciu 4.14 widać konstrukcję zrealizowanego serwomechanizmu. Są tutaj: przekładnia kątowa {20:12} podłączona bezpośrednio do silnika krokowego oraz ślimakowa {1:24} zamknięte w uniwersalnej obudowie aluminiowej będącej równocześnie podstawą całej konstrukcji.

4.3.4. Napęd osi pionowej

Obciążenie tego modułu stanowią tylko i wyłącznie kamery wraz z okablowaniem, w związku z czym zadecydowano o zastosowaniu przełożenia {20:12, 24:24, 1:12}, głównie ze względu na przyjęte założenie o wykorzystaniu przekładni ślimakowej w charakterze mechanicznej diody półprzewodnikowej. Takie rozwiązanie umożliwia osiągnięcie szybkości kątowej równej 50 stopni na sekundę. Zdjęcie nr 4.15 prezentuje omawiany fragment projektu.



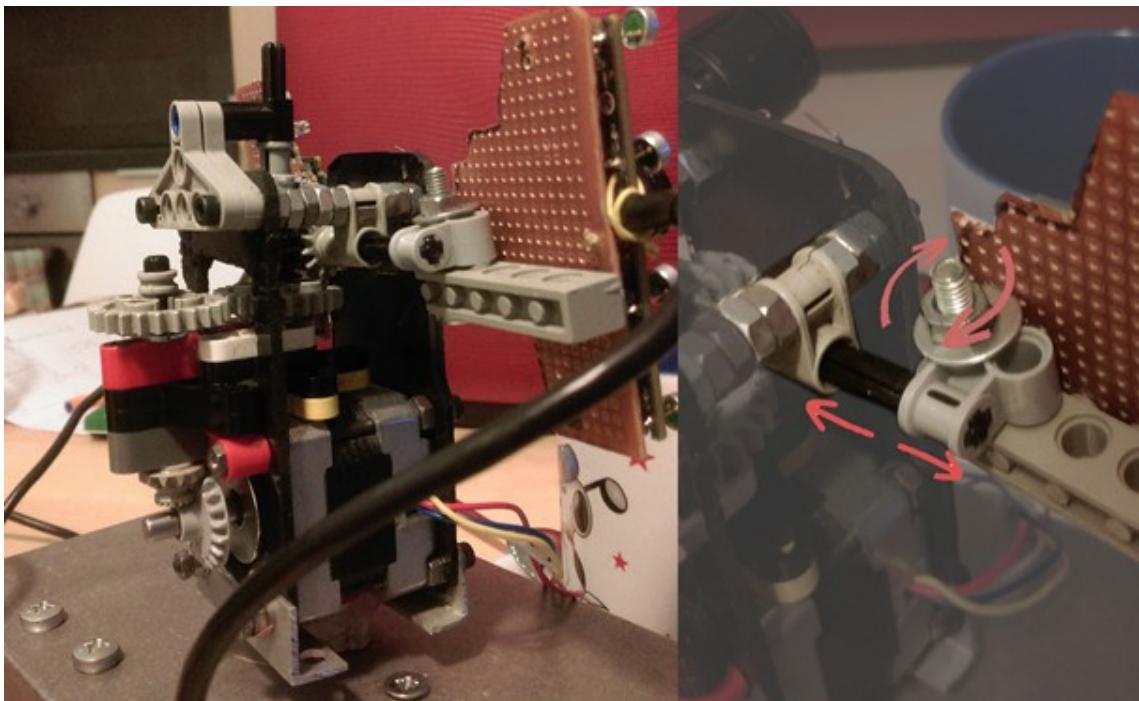
Rys. 4.24. Zestaw przekładni napędu osi poziomej

4.3.5. Montaż kamer wideo

Ten fragment realizacji jest elementem o wyjątkowo wysokiej wadze, gdyż decyduje o odczuciach użytkownika i użyteczności projektu. Ze względu na wymaganą wysoką precyzję ustawienia kamer wideo zadecydowano o stworzeniu ich w taki sposób, aby kalibracja położenia sensorów optycznych była czynnością wykonywaną bezpośrednio przed użyciem głowicy. Jedynym parametrem, który można było uznać za stały, było wychylenie kamer względem zadanego kierunku, które powinno wynosić dokładnie zero – ten fakt zadecydował o zamieszczeniu obydwóch uchwytów na jednej osi. Ustawienia możliwe do modyfikacji po stronie użytkownika to: odległość między kamerami oraz kąty ich wychylenia – do wewnętrz, lub na zewnątrz. Należy tutaj zaznaczyć, iż absolutna precyzja kalibracji nie jest tutaj konieczna,

ponieważ biblioteka przetwarzania wideo umożliwia dokonanie ostatnich poprawek już na poziomie oprogramowania.

Zdjęcie numer 4.16. przedstawia zrealizowany moduł montażowy kamer wideo. Strzałkami zaznaczono możliwe kierunki kalibracji układu. Są to: odległość między kamerami – minimalna: 6,5 cm, maksymalna 7,5 cm oraz kąt wychylenia kamer minimalny: -10 stopni (do wewnątrz), maksymalny: +90stopni (na zewnątrz).



Rys. 4.25. System monału kamer do głowicy wraz ze strzałkami prezentującymi możliwości kalibracyjne położenia sensorów.

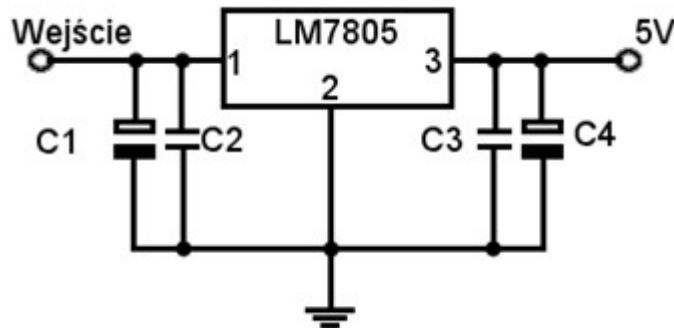
4.4. Kontroler

Układ kontrolera głowicy składa się z dwóch podstawowych elementów – części logicznej, czyli mikrokontrolera oraz części sterownika silników krokowych. Ze względu na decyzję o zasilaniu całego układ z jednej linii 12 V należało zastosować dodatkowe układy liniowego konwertera mocy oraz konwertera poziomów logicznych.

4.4.1. Układ zasilania

Układ kontrolera jako całość wymaga zasilania 12 V przy przewidywanym normalnym poborze prądu równym 1 A. Bezpośrednio do tej linii zasilania został podłączony moduł sterownika silników krokowych, który przy normalnej pracy pobiera z sieci dystrybucji mocy do 800 mA. Podzespoły logiczne wymagają zasilania sygnałem o poziomie 5 V przy poborze prądu do 100 mA, co wymusiło zastosowanie układu konwertera mocy. W roli tej obsadzono klasyczny już układ LM7805, będący liniowym regulatorem napięcia spotykany w niemal każdym prototypie zawierającym mikrokontroler. Takie rozwiązanie, co widać na ilustracji nr 4.17 jest niezwykle proste – wymaga zastosowania tylko czterech kondensatorów filtrujących sygnały wejściowe i wyjściowe – dwóch elektrolitycznych wysokiej pojemności przeciwdziałających

zakłóceniom wolnozmiennym oraz dwóch ceramicznych odpowiedzialnych za wyzerowanie widma dla wysokich częstotliwości.



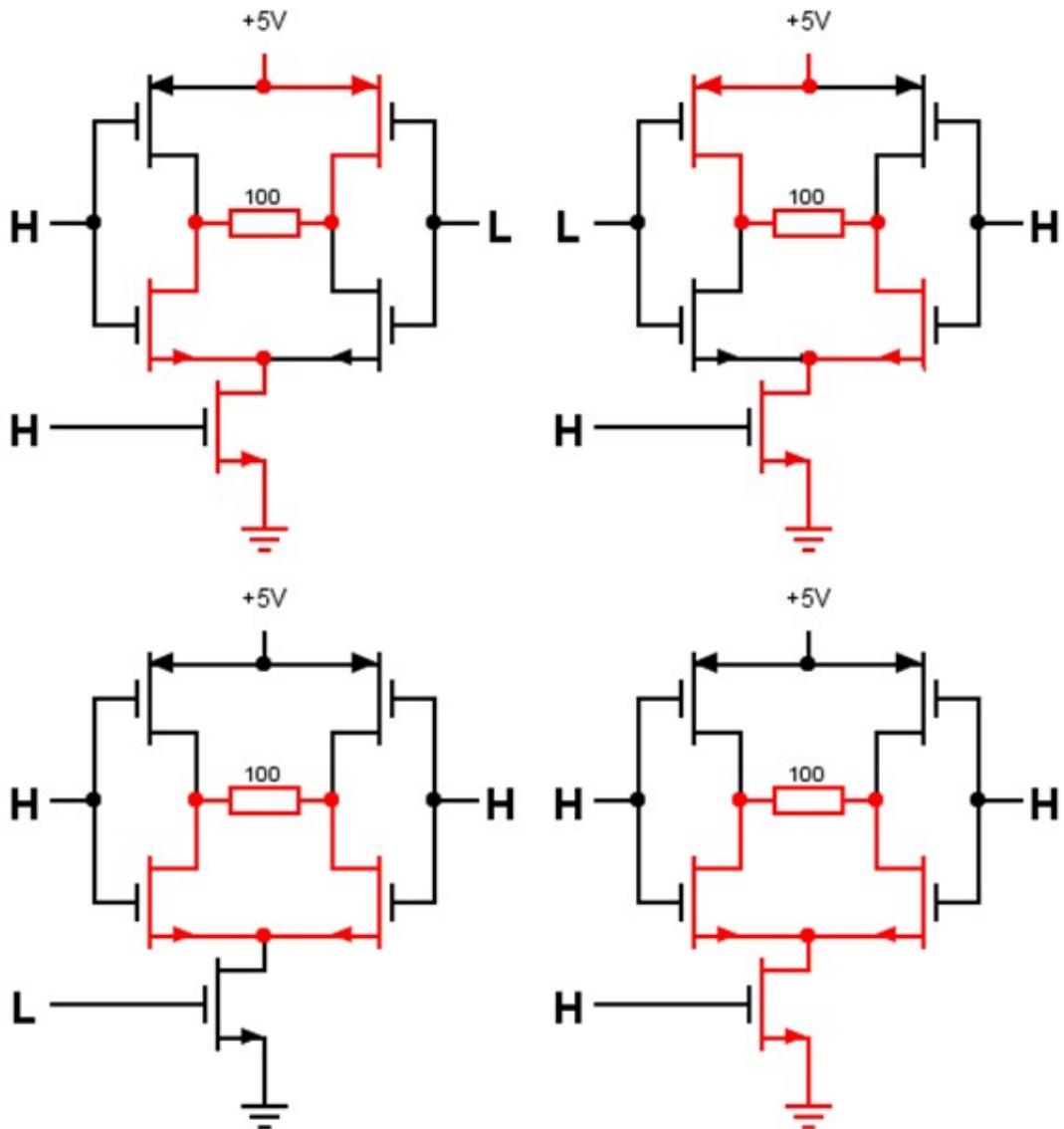
Rys. 4.26. Wykorzystany w realizacji klasyczny układ stabilizacji napięcia oparty na LM7805

Straty w układzie konwersji mocy wynikają bezpośrednio z różnicy napięć na wejściu, prądu obciążenia oraz prądu zużywanego wewnętrznie przez układ[30]. Wynika z tego iż, moc strat, wynosi do 796 mW. Pominięto tutaj straty wynikające z prądów upływu na kondensatorach, które powinny być co najmniej o dwa rzędy mniejsze.

4.4.2. Mostek sterowniczy

Do generacji sygnału sterującego silnikami głowicy zdecydowano się zaprojektować i wykonać własny układ elektroniczny. Zastosowano tutaj baterię czterech klasycznych mostków sterowniczych typu H z dodatkowym kluczem odcinającym masę. W roli kluczy obsadzo no tranzystory MOS ze wzbogaconym kanałem typu n oraz p o nazwach kodowych odpowiednio IRLZ24N oraz IRF9Z34N firmy International Rectifier. Ilustracja nr 4.18 prezentuje schemat ideowy takiego mostka oraz cztery podstawowe tryby jego pracy – zaczynając od lewego górnego rogu są to:

- I. Polaryzacja dodatnia cewki (obciążenia),
- II. Polaryzacja ujemna,
- III. Hamowanie – energia generowana w poruszającym się silniku jest natychmiast do niego zawracana, co powoduje przeciwdziałanie zmianom wychylenia rotora.
- IV. Bieg „luźny” – cała energia generowana w wyniku poruszania się silnika jest odprowadzana do masy układu.



Rys. 4.27. Schematy prezentujące tryby pracy wykorzystanych w projekcie mostków sterowniczych. Kolor czerwony oznacza otwartą ścieżkę dla przepływu prądu sterowanego.

Użyte klucze posiadają izolowaną bramkę, dlatego też, do wspomnianego układu należało dodać rezystory podciągające do napięcia zasilania V_{in} celem zapobieżenia niekontrolowanym zmianom stanu układu w wyniku indukcji zakłóceń na liniach sygnałowych. Wartości tych oporników wynoszące $12\text{ k}\Omega$ dobrano tak, aby natężenie prądu płynącego przez nie wynosiło około 1 mA , co zapewnia pomijalne straty oraz dużą odporność na wspomniane wyżej zagrożenie.

Wyboru tranzystorów dokonano kierując się kryterium zapasu maksymalnej mocy sygnału przełączanego. Przy wykorzystaniu dodatkowego chłodzenia maksymalny prąd na wyjściu każdego z kanałów wynosi 13 A [31] co zdecydowanie spełnia postawione wymaganie. Jako że rezystancja szeregową włączonego klucza wynosi co najwyżej $0,1\text{ }\Omega$, zaś częstotliwość przełączania kluczy wynosi zaledwie 1 kHz pominięto w rozważaniach obliczanie mocy traconej w układzie kluczy sterownika.

Doboru elementów mostka sterowniczego dokonano bazując na propozycjach od autora pozycji „Budowa robotów dla średnio zaawansowanych”[32].

4.4.3. Część logiczna

Za kontrolę działania całego układu odpowiada układ Atmega88 firmy Atmel. Integruje on w jednej obudowie typu DIP28 ośmiorodowy procesor z rodziny AVR oraz zestaw zintegrowanych układów, w tym m.in. przetworniki ADC, generatory PWM, generator zegara 1 MHz, układ USART oraz liczniki[33]. Taki zestaw funkcjonalności umożliwia maksymalne uproszczenie projektowanego układu kontrolera. Cechy, które zadecydowały o wykorzystaniu tego konkretnego mikrokontrolera to:

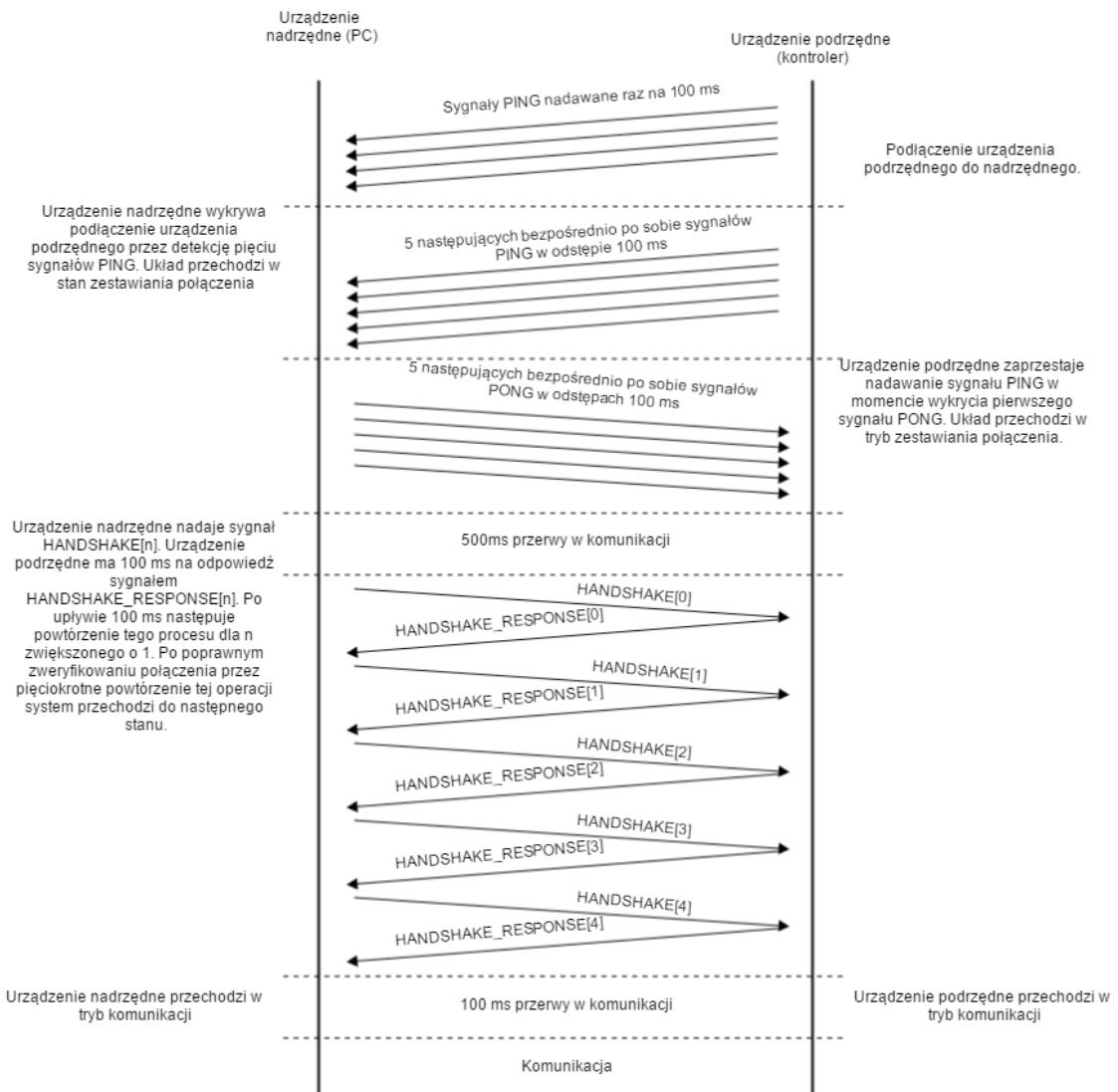
- Niska cena oraz łatwa dostępność.
- Zintegrowany oscylator generujący sygnał zegarowy 1 MHz z możliwością zastosowania zewnętrznego kryształu kwarcowego o częstotliwość znamionowej do 20 MHz. Założenia projektu przewidywały skorzystanie z tej opcji tylko, gdy będzie to konieczne, tzn. zabraknie mocy obliczeniowej.
- Obudowa typu THD ułatwiająca prace prototypowe.
- Duża ilość dostępnych cyfrowych pinów wejścia/wyjścia.
- Zintegrowany układ USART – konieczny do komunikacji z PC.

W praktyce okazało się konieczne wykorzystanie zewnętrznego kryształu kwarcowego 16 Mhz.

Należy tutaj też zaznaczyć, iż wielką zaletą mikrokontrolerów jest możliwość programowej zmiany wykorzystywanych pinów wejściowych w zależności od uznania projektanta. Jest to szczególnie ważne przy projektowaniu płyt drukowanych – w sytuacji, w której wyznaczenie ścieżki między układami scalonymi okazuje się niemożliwe, wystarczy w oprogramowaniu dokonać wyboru innego portu wyjściowego i zaktualizować schemat ideowy. Podczas prac prototypowych wielokrotnie korzystano z tego rozwiązania nie tylko w ramach projektowania PCB, ale także jako narzędzie do weryfikacji i znajdowania błędów.

4.4.4. Komunikacja kontrolera z PC

Elementem projektu o bardzo wysokim znaczeniu dla całej realizacji jest zapewnienie możliwości bezbłędnego przesyłania danych sterujących z komputera PC do kontrolera oraz informacji zwrotnej o aktualnym stanie głowicy z powrotem do oprogramowania. Aby zadośćuczyć tym wymaganiom stworzono specyfikację protokołu komunikacyjnego opartego na dwuprzewodowym łączu asynchronicznym. Ilustracja 4.19 prezentuje proces zestawiania połączenia zgodny z utworzoną specyfikacją.



Rys. 4.28. Schemat prezentujący proces nawiązywania połączenia między urządzeniami

Warstwa fizyczna łącza realizowana jest na trzech przewodnikach przenoszących informację pod postacią odpowiednich napięć – 0 V dla logicznego zera oraz 5 V dla logicznej jedynki. Rzeczywiście trzy przewody służą do przesyłu: sygnału od urządzenia nadzorującego do podporządkowanego – linia TX, podporządkowanego do nadzorującego – linia RX oraz sygnału masy.

Transfer danych odbywa się zgodnie ze standardami UART przy następujących ustawieniach: szybkość transmisji 9600 bitów na sekundę, brak kontroli parzystości, jeden bit stopu, jeden bit startu oraz słowo o rozmiarze 8 bitów.

Protokół przewiduje komunikację między dwoma urządzeniami – jednym nadzorującym oraz podporządkowanym. Umożliwia on automatyczne wykrycie połączenia urządzenia podporządkowanego do obserwowanego portu, zestawienie połączenia oraz transmisję pakietów danych w obie strony. Specyfikacja przewiduje następujące stany, w których może znajdować się układ komunikacyjny:

1. Faza oczekiwania na połączenie – urządzenie nadzorujące nasłuchuje na linii odbiorczej, urządzenie podporządkowane nadaje w 100 ms odstępach czasu sygnał PING.

2. Faza wykrywania podłączenia 1 – w momencie wykrycia pierwszego sygnału PING na linii odbiorczej urządzenie nadzędne przechodzi w ten stan. Przebywa w nim do czasu odebrania pięciu następujących bezpośrednio po sobie w 100 ms odstępach czasu sygnałów PING. W przypadku odebrania mniejszej ilości sygnałów PING lub jakichkolwiek innych danych urządzenie wraca do stanu z punktu 1.
3. Faza wykrywania podłączenia 2 – urządzenie nadzędne nadaje do wykrytego urządzenia podziemnego pięć następujących bezpośrednio po sobie w 100 ms odstępach czasu sygnałów PONG. W momencie odebrania pierwszego sygnału pong urządzenie podziemne zaprzesta nadawania. W przypadku odebrania mniejszej ilości sygnałów PONG, przerwy większej niż 100 ms lub odebrania sygnału innego niż ping urządzenie podziemne wraca do stanu z punktu 1.
4. Przerwa 500 ms – obydwa urządzenia zaprzestają komunikacji na 500 ms. Nadanie jakiejkolwiek informacji przez dowolne urządzenie jest interpretowane jako błąd i powoduje powrót urządzeń do stanów z punktu 1.
5. Faza kontroli poprawności połączenia – urządzenie nadzędne nadaje sygnał HANDSHAKE_SIGNAL[n] oczekuje 100 ms na odpowiedź urządzenia podziemnego, która musi być równa wartości HANDSHAKE_RESPONSE[n]. Urządzenie nadzędne czeka 100 ms i po tym czasie powtarza procedurę dla kolejnego n ze zbioru {1,2,3,4,5}. Nadanie przez urządzenie nadzędne sygnału innego niż HANDSHAKE_SIGNAL[n], odpowiedź urządzenia podziemnego inna niż HANDSHAKE_RESPONSE[n] lub odpowiedź następująca po czasie dłuższym niż 100 ms oraz czas pomiędzy odpowiedzią urządzenia podziemnego, a następnym sygnałem HANDSHAKE_SIGNAL większy niż 100 ms muszą zostać zinterpretowane jako błąd i spowodować powrót urządzenia do stanu z punktu 1.
6. Przerwa 100 ms – obydwa urządzenia zaprzestają komunikacji na 100 ms. Nadanie jakiejkolwiek informacji przez dowolne urządzenie jest interpretowane jako błąd i powoduje powrót urządzeń do stanów z punktu 1.
7. Faza komunikacji – w tej fazie następuje właściwa wymiana danych między urządzeniami. Komunikacja zostaje rozpoczęta przez urządzenie nadzędne przez nadanie pierwszego bajtu pierwszej ramki. Proces nadawania polega na naprzemiennym nadawaniu przez urządzenia kolejnych bajtów. W przypadku, gdy urządzenie nie może nadać kolejnego bajtu przez dłużej niż 500 ms musi w ciągu kolejnych 400 ms nadać sygnał KEEP_ALIVE. Drugie z urządzeń musi w ciągu kolejnych 400 ms odpowiedzieć sygnałem KEEP_ALIVE_RESPONSE. Sytuacja taka oznacza iż dane urządzenie chwilowo ma wszystkie zasoby zajęte w wyniku czego nie jest zdolne do komunikacji. Sytuacje, w których urządzenie po 500 ms milczenia nie nada sygnału KEEP_ALIVE w ciągu 400 ms lub nada w tym czasie inny sygnał, lub urządzenie nada następujące po sobie dwa bajty bez oczekiwania na bajt odebrany, lub drugie z urządzeń nie odpowie w ciągu 400 ms sygnałem KEEP_ALIVE_RESPONSE, lub odpowie jakimkolwiek innym sygnałem, mają zostać zinterpretowane jako błąd i

- spowodować przejście urządzenia do fazy zerwanego połączenia z punktu 8. Oprogramowanie implementujące protokół buforuje m kolejnych odebranych bajtów, po czym wywołuje funkcję nasłuchującą. Zarówno m – rozmiar pakietu, jaki i rzeczona funkcja muszą być znane przed fazą oczekiwania na połączenie (na etapie komplikacji, lub inicjalizacji biblioteki).
8. Faza zerwanego połączenia – urządzenie nie nasłuchuje, ani nie nadaje żadnych danych. W zależności od założeń projektowych, programista może wykryć ten stan i podjąć odpowiednie działania – np. próbę ponownego zestawienia połączenia.

Słowo „sygnał” jest tutaj używane dla odróżnienia rodzaju danych. W przeciwieństwie do pakietu, który składa się z wielu bajtów, sygnał jest oddzielną informacją zawartą w jednym bajcie. Poniżej znajdują się definicje używanych sygnałów:

1. PING = 0x11
2. PONG = 0x11
3. HANDSHAKE[n] = { 0x01 dla n=1, 0x04 dla n=2, 0x10 dla n=3, 0x40 dla n=4, 0x80 dla n=5 }
4. HANDSHAKE_RESPONSE[n] = { 0x02 dla n=1, 0x05 dla n=2, 0x11 dla n=3, 0x41 dla n=4, 0x81 dla n=5 }
5. KEEP_ALIVE = 0x55
6. KEEP_ALIVE_RESPONSE = 0x55

Jak łatwo zauważyc na podstawie opisu fazy komunikacji (punkt 7), mimo iż połączenie fizyczne umożliwia transfer w dwóch kierunkach na raz, dane są przesyłane na przemiennie w trybie half-duplex – jest to wynikiem prac prototypowych w których w wyniku przyjętej filozofii tworzenia oprogramowania okazało się iż mikrokontroler nie zawsze jest w stanie odebrać wszystkie następujące bezpośrednio po sobie bajty ramki. Rozwiążanie to pozwoliło na obejście tego problemu, przy braku zauważalnych wad. W praktyce realizowana szybkość transferu to około 80 pakietów na sekundę, czyli 40 aktualizacji stanu głowicy oraz 40 informacji zwrotnych o aktualnym stanie głowicy.

Zrealizowane oprogramowanie wykorzystuje powyższy protokół do przesyłania 10 bajtowych ramek. Każda ramka składa się z dwóch 32-bitowych liczb bez znaku oraz 16 zarezerwowanych na przyszłość bitów. Rzeczone liczby reprezentują zadany/aktualny kąt wychylenia głowicy, przy czym pierwsza przypisana jest do osi pionowej, zaś druga do osi poziomej. Liczby te są przeliczane na kąty według następujących wzorów:

	$h = (\alpha + \pi) \frac{\text{UINT}32_{\text{MAX}}}{2\pi}$	(4.0)
	$\alpha = h \frac{2\pi}{\text{UINT}32_{\text{MAX}}} - \pi$	(4.0)

Gdzie:

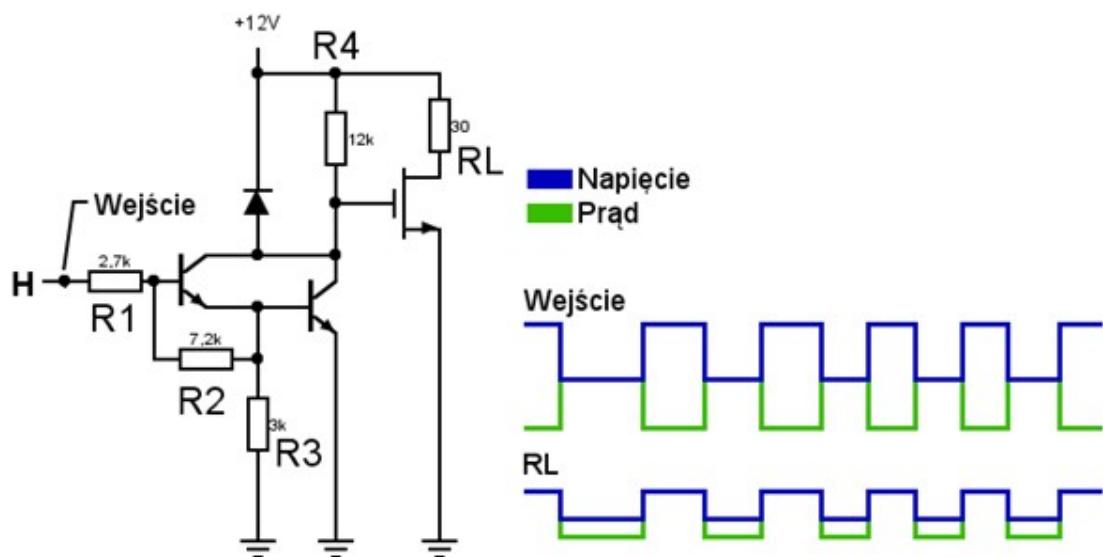
$\text{UINT}32_{\text{MAX}}$ – maksymalna wartość 32 bitowej liczby bez znaku ($2^{32} - 1$)

h – reprezentacja wartości kąta wykorzystywana przez protokół komunikacyjny

α – przeliczany kąt. $\alpha \in [-\pi, \pi]$, wartości spoza zakresu są obcinane.

4.4.5. Separacja układów o różnych napięciach zasilania

Projektowany układ wymaga zastosowania dwóch różnych napięć zasilania oraz wysterowania układu mostka wykorzystującego napięcie 12 V przez mikrokontroler operujący na napięciu 5 V. Ze względu na zależności napięciowe, a dokładniej napięcie progowe, tranzystorów MOS z kanałem typu P, nie jest możliwe bezpośrednie nimi sterowanie z układu Atmega. Problem ten został rozwiązany przez zastosowanie układu uln2803, który integruje w jednej obudowie DIP18 osiem układów Darlingtona, w parze z rezystorami podciągającymi do zasilania. Schemat D prezentuje sposób wykorzystania tego chipu w zaprojektowanym sterowniku wraz z przykładowymi wykresami napięć i prądów na wejściu oraz obciążeniu układu.



Rys. 4.29. Układ wykorzystywany do wysterowania kluczów mostków sterowniczych. Rezystory R1–R3, dioda oraz tranzystory bipolarne są częściami zintegrowanymi wewnętrz uladu ULN2803[34]. Rezystor R4 podciąga bramkę klucza do zasilania

W tak zaprojektowanym układzie bramka sterowanego klucza jest zwierana do masy poprzez tranzystor bipolarny, w chwili gdy na wejście układu zadany jest stan niski. W sytuacji odwrotnej jest ona odcięta od potencjału zero, a za pomocą zwarcia bezprądowego (bramka nie przewodzi prądu) przez rezistor R4 zwierana do napięcia zasilania +12 V. Jak można zauważyć ULN2803 realizuje tutaj nie tylko funkcję konwertera poziomów logicznych, ale także neguje sygnał wejściowy.

4.4.6. Projekt płytki drukowanej

Zrealizowany schemat ideowy kontrolera oraz jego płytki drukowanej zamieszczono w załącznikach do pracy. Ze względu na ograniczenia czasowe nie udało się zrealizować kontrolera w formie gotowego układu, w związku z czym fizycznie nie zweryfikowano poprawności projektu, zaś sam projekt płytki zawiera niewykorzystane w projekcie złącza służące do podłączenia czujników pozycji początkowej, wykorzystane

wyprowadzenia mikrokontrolera różnią się względem tych, wykorzystanych w oprogramowaniu oraz brakuje panelu kontrolnego opisanego w rozdziale 4.9.18.

4.5. Moduł integracyjny

4.5.1. Opis ogólny

Zgodnie z założeniami określonymi w formalnym opisie pracy, powstała klasa RiftMotionInterceptor, która integruje ruchy hełmu Oculus rejestrowane przez jego trzy sensory: akcelerometr, żyroskop i magnetometr, z ruchami serwogłówicy.

Instancja tej klasy stanowi samodzielny łącznik realizujący komunikację na linii hełm Oculus–serwogłówica. W chwili jej tworzenia próbuje nawiązać kontakt z podłączonym do komputera HMD. Następnie należy pobrać za jej pośrednictwem listę wszystkich możliwych połączeń poprzez port szeregowy i wywołać funkcję Start() z parametrami:

- wybrany indeks połączenia z listy możliwych połączeń
- obiekt nasłuchujący (obserwatora) realizujący interfejs IConsole

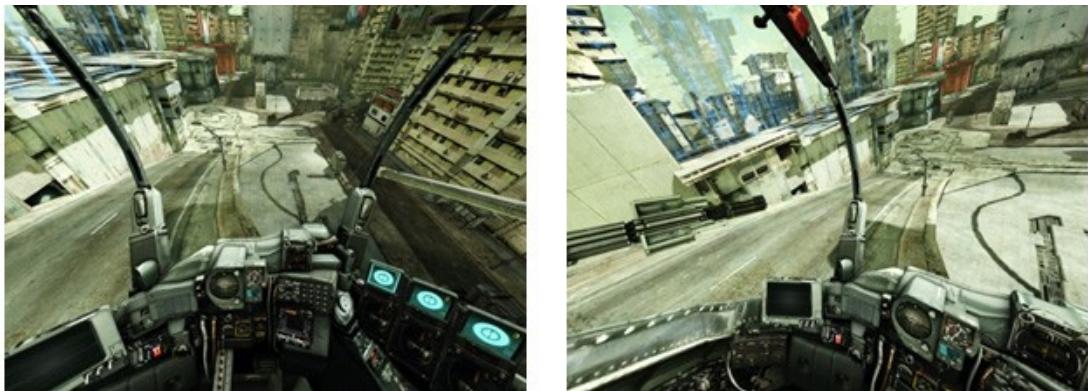
Jeśli którykolwiek z połączeń nie zostanie nawiązane, rzucany zostaje wyjątek z odpowiednią informacją. W przypadku pomyślnego uzyskania połączeń z HMD oraz serwogłówicą, cyklicznie i z dużą częstotliwością odczytywana jest informacja o „stanie śledzenia” hełmu, z której wyciągany jest quaternion jego orientacji, a następnie przekształcany do kątów Eulera w zakresie $+/- \pi$ radianów. Ostatecznie, kąty w tej formie zadawane są poprzez ustalone API do serwogłówicy. Instancja jest czujna na wszelkie wyjątki podczas tej operacji – przede wszystkim na te informujące o niepowodzeniu zadania kąta w sytuacji gdy ów kąt znajduje się poza zakresem możliwości serwogłówicy lub nastąpiło przerwanie z nią połączenia. Aby zakończyć działanie wątku integrującego, należy wywołać na omawianej instancji funkcję Stop().

4.5.2. Zasada działania po stronie Oculus Rifta

Hełm Oculus Rift w celu określania swojego położenia został wyposażony w zestaw sensorów: żyroskop, akcelerometr i magnetometr. Prototyp tego urządzenia otrzymał system śledzący firmy Hillcrest Labs, który zapewniał dane o rotacji HMD w oparciu o 3 stopnie swobody. Częstotliwość z jaką uzyskiwane były odczyty, wynosił pierwotnie 125 Hz i na życzenie osób tworzących hełm Oculus Rift, dzięki wykorzystaniu specjalnego oprogramowania udało się zwiększyć tę wartość do 250 Hz, która pozostała wartością nominalną dla wersji DK1. Wariant DK2 posiada już nowy, wydajniejszy zestaw sensorów, który pozwalał na śledzenie zarówno rotacji jak i pozycji hełmu w przestrzeni 3D z częstotliwością 1000 Hz.

Podstawowym sensorem hełmu Oculus jest żyroskop[35]. Pozwala mierzyć jego orientację w oparciu o zasadę zachowania momentu pędu. Jako samodzielna jednostka, żyroskop nie jest w stanie dostarczyć danych o rotacji hełmu nieobarczonych żadnym błędem. W chwili, gdy różnica czasu nie jest zerowa (sensory musiałyby dostarczać dane z nieskończoną częstotliwością) a dokładność żyroskopu nie jest idealna, po wielu tysiącach aktualizacji położenia zaobserwować można odchylenie wyników obliczeń rotacji od rzeczywistych, dostrzegalnych gołym okiem rzeczywistych kątów obrotu. Poglądający się w ten

sposób błęd ten powszechnie nazywa się błędem całkowania (po ang. drift error – błąd przesunięcia). Jedną z jego składowych jest błąd przechylenia (z ang. tilt error) czyli przesunięcie kątów wokół osi X i Z, które powoduje problemy z określeniem położenia wektora „do góry”. Aby ten błąd wyeliminować, konieczne było zastosowanie drugiego sensora – akcelerometru, na podstawie którego obliczało się ów pionowy kąt oraz łączyło z prędkością kątową na tej samej osi. Drugą ze składowych jest błąd kursu (z ang. yaw error) czyli przesunięcie kąta wokół osi Y, które z punktu widzenia użytkownika zauważalne jest jako „uciekająca północ”. Zjawisko to ilustruje poniższy rysunek.



Rys. 4.30. Prezentacja pierwotnej orientacji pilota helikoptera (po lewej) oraz orientacja z błędem kursu po upływie czasu (po prawej)[36]

W celu minimalizacji wpływu tej składowej, do zestawu sensorów dołączony został trzeci – magnetometr. Dzięki informacji o rzeczywistym polu magnetycznym Ziemi oprogramowanie jest w stanie reagować i korygować to negatywne zjawisko.

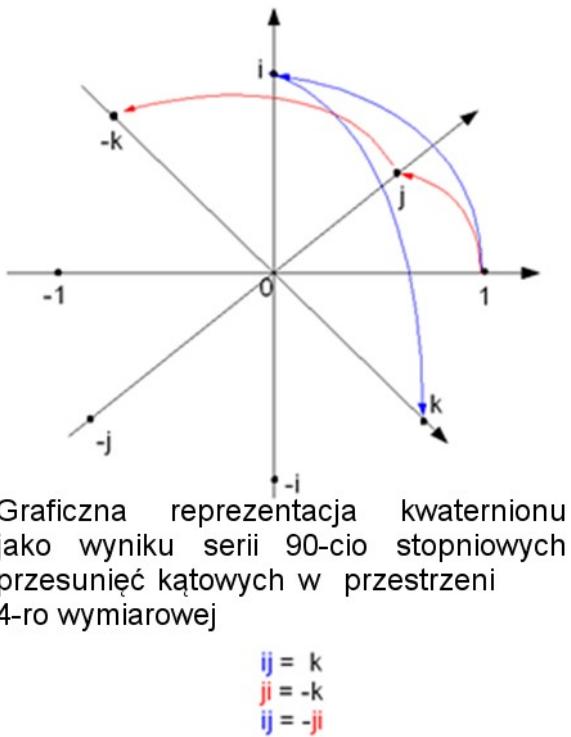
Ostatecznie cały zestaw surowych danych przelicza się w celu uzyskania jak najdokładniejszej orientacji hełmu. Spośród kilku rozwiązań implementacyjnych, wybrano algorytm filtra dopełniającego (z ang. complementary filter), który łączy filtr górnoprzepustowych dla danych żyroskopu i filtr dolnoprzepustowy dla danych akcelerometru. Dzięki wysokiej mocy obliczeniowej współczesnych komputerów filtr ten może być i jest stosowany co każdą iterację odczytu danych i dostosowany do rozwiązywania problemu orientacji z tym konkretnym HMD.

Aplikacja w celu uzyskania jakichkolwiek informacji dotyczących hełmu Oculus musi wykorzystać interfejs wystawiony przez Oculus SDK – oprogramowania dostarczanego przez producenta tego sprzętu. Na żądanie każdej aplikacji ów software odpytuje HMD o zestaw danych dotyczących jego położenia i obrotu, a następnie opakowuje je w czytelniejszą do człowieka strukturę. Ponadto przelicza jałowe wartości sczytane z sensorów (które też wchodzą w skład tej struktury) na pozy (struktury składające się z wektora położenia i obrotów) oraz kwaterniony. Komunikacja między SDK a samym hełmem odbywa się poprzez protokół TCP na poziomie wtyczki nasłuchującej na hoście lokalnym.

4.5.3. Co uzyskujemy, czyli kwaternion w praktyce

Kwaterniony według strictly matematycznej definicji to struktury algebraiczne stanowiące rozszerzenie ciała liczb zespolonych. Opracowane zostały w 1843 przez irlandzkiego

matematyka Williama Hamiltona i pierwotnie służyły do opisywania mechaniki w przestrzeni trójwymiarowej. Współcześnie kwaterniony traktowane są jako czterowymiarowa, unormowana algebra z dzieleniem nad liczbami rzeczywistymi i znajduje zastosowanie zarówno w teorii matematycznej (teorii liczb czy geometrii algebraicznej) jak i w grafice komputerowej (wspomaganie animacji).



Rys. 4.31. Graficzne przedstawienie obrotu kwaternionu o 90° w przestrzeni czterowymiarowej[37]

Z informatycznego punktu widzenia, kwaternion to struktura danych składająca się z czterech liczb dziesiętnych stanowiąca kolejną reprezentację rotacji obiektów w przestrzeni trójwymiarowej. Kwaternion tworzy się z wektora wskazującego oś obrotu oraz kąta obrotu obiektu wokół tej osi.

$$x = \sin\left(\frac{1}{2} * kqt\right) * x_{os} \quad (4.5)$$

$$y = \sin\left(\frac{1}{2} * kqt\right) * y_{os} \quad (4.6)$$

$$z = \sin\left(\frac{1}{2} * kqt\right) * z_{os} \quad (4.7)$$

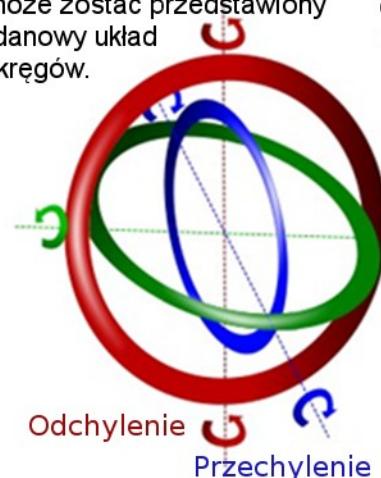
$$w = \cos\left(\frac{1}{2} * kqt\right) \quad (4.8)$$

Kwaternion w porównaniu z wektorem, stanowi strukturę nadmiarową (niedomiaryowe 3 liczby, 4 w przypadku kwaternionu) – tak jak i macierz obrotu 3×3 (9 liczb) – istnieje nieskończenie wiele kwaternionów opisujących ten sam obrót, ale posiadających różną długość.

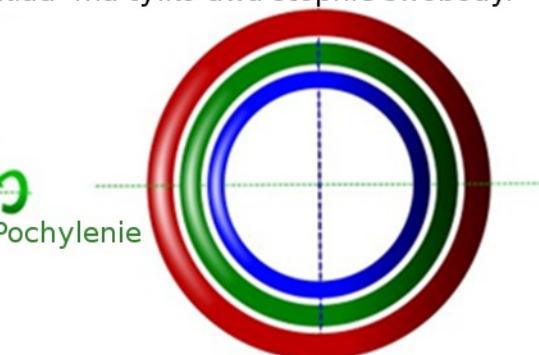
Warto jest zatem, w ramach opisu samej rotacji, je normalizować, czyli sprowadzić składowe wartości kwaternionów do liczb z zakresu <-1, 1>[38].

Owa nadmiarowość daje tej strukturze przewagę nad chociażby kątami Eulera. Po pierwsze, nie istnieje w ich przypadku zjawisko „gimbal lock”. Wykorzystując kąty Eulera do opisu rotacji obiektu, może zaistnieć sytuacja, w której wartość jednego z kątów osiągnie 90 stopni i jedna oś zrówna się z inną. Tracony jest wówczas jeden stopień swobody. Wynika to z faktu, że tworząc macierz w oparciu o kąty Eulera, obroty składane są w określonej kolejności – najczęściej X, Y, Z (np. w programie 3DS Max). Kwaternionów nie dotyczy to ograniczenie. Nie mniej jednak przewagę tą można wykorzystać tylko w sytuacji, gdy dany obiekt w przestrzeni porusza się swobodnie w każdym kierunku, czyli na przykład w sytuacji, gdy opisuje się orientację lecącego samolotu. W przypadku, gdy obiekt uczepony jest jakiejś płaszczyzny (np. samochód czy nawet nasza serwogłówica), kąty Eulera stanowią wystarczającą reprezentację jego orientacji.

1. Kąt obrotu mierzony w kątach Eulera może zostać przedstawiony jako kardanowy układ trzech okręgów.



2. W sytuacji, w której wszystkie trzy okręgi ustawione są równolegle, cały układ ma tylko dwa stopnie swobody.



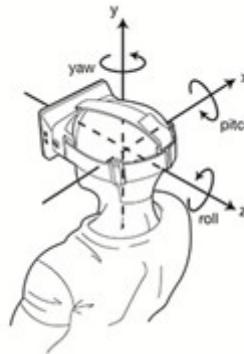
3. Zastosowanie kwaternionów pozwala na obejście tego problemu.

Rys. 4.32. Przedstawienie zjawiska "gimbal lock"[39]

Drugą zaletą kwaternionów jest możliwość interpolacji liniowej, czyli znajdowania punktu pośredniego pomiędzy dwiema orientacjami. Cechą ta nosi nazwę „Slerp” (od skrótu Spherical Linear Interpolation – interpolacja liniowa sferyczna) i wykorzystywana jest najczęściej do tworzenia płynnych animacji poprzez udawanie aficznego konstruktora typu algorytm de Casteljau dla krzywych Béziera[40].

Jako że sam hełm dostarcza informacje o swojej orientacji jedynie w postaci jałowych danych odczytanych z sensorów lub póż i kwaternionów, nie jest możliwe ich przekazanie do serwogłówicy bez „przetłumaczenia” ich do formy dla niej zrozumiałej. Konieczne jest zatem przekształcenie kwaternionu orientacji hełmu do trzech kątów Eulera, z czego serwogłówica wykorzystuje jedynie dwa:

- X (pitch – oś idąca od ucha do ucha użytkownika)
- Y (yaw – oś idąca przez ciało użytkownika po jego długości).



Rys. 4.33. Osie obrotu hełmu Oculus Rift[41]

Konwersja realizowana jest poprzez poniższy pseudokod:

```
void GetEulerAngles(float *yaw, float *pitch, float *roll)
{
    float ww = this.w * this.w;
    float xx = this.x * this.x;
    float yy = this.y * this.y;
    float zz = this.z * this.z;
    float s2 = 2.0f * (this.w * this.y + this.x * this.z);

    if (s2 < -1.0f + 0.000000000001f)
    {
        // Gimbal lock na osi południowej
        *yaw = 0.0f;
        *pitch = -0.5f * 3.14159265358979f;
        *roll = atan2(2.0f * (this.x * this.y + w * this.z), ww + yy - xx -
zz);
    }
    else if (s2 > 1.0f - 0.000000000001f)
    {
        // Gimbal lock na osi północnej
        *yaw = 0.0f;
        *pitch = 0.5f * 3.14159265358979f;
        *roll = atan2(2.0f * (this.x * this.y + w * this.z), ww + yy - xx -
zz);
    }
    else
    {
        *yaw = -1.0f * atan2(-2.0f * (w * x - y * z), ww + zz - xx - yy);
        *pitch = asin(s2);
        *roll = atan2(2.0f * (w * z - x * y), ww + xx - yy - zz);
    }
}
```

Funkcjonalność ta została zaimplementowana w Oculus SDK jako metoda Quat::GetEulerAngles().

4.5.4. Teoria a praktyka – Oculus SDK i SharpOVR

Oculus SDK dostarczane przez firmę produkującą owe hełmy udostępnia funkcje umożliwiające uzyskanie kwaternionu położenia hełmu bez konieczności obliczania i zgrywania danych z poszczególnych sensorów osobno. Ponadto pozwala na przekształcenie kwaternionu na kąty Eulera zgodnie z opisywaną wcześniej teorią. Otrzymane kąty przedstawiane były w postaci znormalizowanej i takie też nadawaliśmy do serwogłówicy poprzez ustalone API.

Nieduży problem stanowił fakt, że Oculus SDK jest przeznaczone dla projektów pisanych w języku C++. Rozwiążaniem było wykorzystanie gotowej biblioteki SharpOVR, będącej tylko i wyłącznie nakładką (wrapperem) na Oculus SDK. Opakowuje ona dynamiczne biblioteki SDK wykorzystując w tym celu mechanizm Platform Invoke co umożliwia korzystanie z owych bibliotek z poziomu aplikacji C#. SharpOVR w porównaniu z Oculus SDK jest stosunkowo okrojona. Pozwala wprawdzie otrzymać kwaternion aktualnego obrotu hełmu, ale nie zawiera żadnego kodu umożliwiającego renderowanie scen na ekranie urządzenia. Mimo wszystko konieczna była instalacja Oculus SDK, aby nakładka w ogóle mogła funkcjonować.

4.5.5. Omówienie modułu

```
public class RiftMotionInterceptor
{
    private Thread CaptureThread = null;
    private HMD OculusHmd = null;
    private ArCameraHeadApi.ArCameraHeadApi ServoHead = null;
    private static long ShouldCapture;
    private int ConnectionIndex = 0;
    private IConsole Console;
    public RiftMotionInterceptor(bool debugMode, IConsole console);
    ~RiftMotionInterceptor();
    public void Start(int connIndex);
    public void Stop();
    public static void CaptureAndSetup(object param);
    public void GetAnglesFromHmd(out float X, out float Y, out float Z);
    public void RecenterHmd();
    public List<string> GetConnectionsList();
}
```

Tabela 4.7. Lista pól klasy RiftMotionInterceptor

Pole	Typ	Opis
CaptureThread	Thread	Obiekt wątku, w którym działać będzie pętla synchronizująca
OculusHmd	HMD	Obiekt hełmu z API Oculus SDK
ServoHead	ArCameraHeadApi.ArCameraHeadApi	Obiekt API serwogłówicy
ShouldCapture	Long	Wartość utrzymująca działanie pętli synchronizującej w wątku CaptureThread. Interpretowana jako wartość logiczna.
ConnectionIndex	int	Indeks połączenia z serwogłówicą wykorzystywany przez wątek synchronizujący. Indeks odpowiada elementowi z listy dostępnych połączeń przez port szeregowy otrzymywanej od funkcji GetConnectionsList()
Console	IConsole	Obserwator; obiekt pełniący rolę konsoli

Tabela 4.8. Lista metod klasy RiftMotionInterceptor

Metoda	Opis
RiftMotionInterceptor(bool)	Konstruktor – umożliwia funkcjonowanie instancji zarówno w normalnym jak i debugowym (ograniczonym) trybie obiektu HMD.
~RiftMotionInterceptor()	Destruktor – w chwili niszczenia obiektu rozłącza aplikację z serwogłówicą i hełmem Oculus Rift
Start()	Tworzy i uruchamia nowy wątek synchronizujący hełm Oculus z serwogłówicą
Stop()	Kończy działanie wątku synchronizującego
CaptureAndSetup(object)	Metoda wątku – realizuje cykliczny odczyt danych (kwaterniony orientacji hełmu), ich konwersję do postaci kątów Eulera (pionowego i poziomego), a następnie przekazuje je do serwogłówicy poprzez jej API w celu zadania odpowiednich kątów obrotu.
GetAnglesFromHmd(out float, out float, out float)	Odczytuje dane orientacji HMD w postaci kwaternionu i przekształca je do postaci kątów Eulera – jednostka: radiany
RecenterHmd()	Ustawia obecną orientację HMD jako początkową
GetConnectionsList()	Zwraca posegregowaną listę nazw dostępnych połączeń poprzez port szeregowy.

Interfejs, który musi spełniać moduł obserwatora wygląda następująco:

```
public interface IConsole
{
    void PrintMessage(string msg);
    void UpdateAngles(float horizontal, float vertical);
}
```

Tabela 4.9. Lista metod interfejsu IConsole

Metoda	Opis
PrintMessage(string)	Metoda, dzięki której przekazywane są komunikaty z modułu synchronizującego
UpdateAngles(float, float)	Metoda, dzięki której przekazywane są aktualnie zadawane na serwogłówicę kąty

4.6. Interfejs i okna – ogólny zarys

4.6.1. Wybór środowiska

Przy renderowaniu grafiki trójwymiarowej, optymalną metodą przetworzenia obrazu jest użycie shadera, wykonującego operacje z pomocą mocy obliczeniowej karty graficznej. Pierwotnie, pomimo renderowania nie sceny 3D, a tekstury dwuwymiarowej, zakładaliśmy wykorzystanie właśnie takiego rozwiązania.

Metoda przewidywała stworzenie sceny 3D w programie Unity3D, wspierającym obsługę wykorzystywanego przez nas hełmu poprzez wtyczkę Oculus Rift SDK. Na scenie stworzone zostałyby dwie prostokątne tekstury 2D, na które naniesione zostałyby obrazy z

kolejnych kamer. Na tekstury zwrócona zostałaby natomiast trójwymiarowa kamera wirtualna Oculus Rifta (po jednej teksturze na oko), a przed trafieniem do urządzenia – obraz zostałby potraktowany napisanym przez nas shaderem.

Pomimo wysokiej efektywności przekształcenia obrazu w powyższy sposób, wszystkie czynności związane z przygotowaniem sceny 3D, pobraniem obrazu, naniesieniem go na tekstury i wyświetlaniem poprzez kamerę wirtualną, generowało duże opóźnienia. Musieliszy więc znaleźć prostszy, bardziej bezpośredni, a przede wszystkim szybszy sposób na przeniesienie widoku z kamery na ekran hełmu rzeczywistości wirtualnej.

Postanowiliśmy więc porzucić środowisko Unity3D i wyświetlić obraz bezpośrednio, jako okno stworzone w języku C# z pomocą WPF (Windows Presentation Foundation).

“Windows Presentation Foundation (WPF, nazwa kodowa Avalon) – nazwa silnika graficznego i API bazującego na .NET 3, wchodzącego w skład WinFX. WPF integruje interfejs użytkownika, grafikę 2D i 3D, multimedia, dokumenty (nazwa kodowa Metro) oraz generowanie/rozpoznawanie mowy (do aplikacji sterowanych głosem).

API w WPF opiera się na języku XML, dokładniej na jego implementacji o nazwie XAML. Całość jest zawarta w nowym API WinFX, zaś graficzna część GUI wykorzystuje grafikę wektorową, budowaną z użyciem akceleratorów grafiki 3D i efektów graficznych udostępnianych przez WGF. Rozwiązanie to jest podobne do Quartz z Mac OS X.”[42]

Rozwiązanie to, poza oczywistym ominięciem wszystkich czynności charakterystycznych dla środowiska Unity3D i zdecydowanie wyższą wydajnością w kwestii pobrania i zwrócenia obrazu na ekran, mogło wykazać się jeszcze jedną, chyba najbardziej znaczącą cechą – możliwością integracji z CL Eye Platform SDK. Interfejs tworzony z pomocą WPF można edytować zarówno poprzez modyfikację pliku xaml, jak i poprzez wbudowany w IDE Microsoft Visual Studio edytor graficzny.

Każde z okien stworzonych z pomocą Windows Presentation Foundation składa się z dwóch plików:

- Pliku .xaml odpowiadającego za elementy znajdujące się w obrębie okna, ich nazewnictwo, położenie (ang. layout) oraz wywoływane zdarzenia (ang. events),
- Klasy–kontrolera .xaml.cs odpowiadającego za jego inicjację okna, obsługę zdarzeń, czy logikę aplikacji.

4.6.2.Zarys interfejsu

Na interfejs programu składają się dwa okna – OculusWindow oraz ControlsWindow. Pierwsze z nich, zgodnie z nazwą, odpowiada za obraz pojawiający się na ekranie Oculus Rifta, drugie zaś zawiera wszystkie opcje związane między innymi z manipulacją kamerami, obrazem, połączeniem i innymi.

Interfejs nie tylko, zgodnie z założeniami projektu, pozwala na pełen dostęp do opcji obrazu i jego poprawne wyświetlanie, ale jest również punktem splotu pracy wszystkich realizatorów projektu, inicjatorem wszystkich wątków, czy – zwyczajowo – punktem odpowiadającym za kontakt programu z użytkownikiem. Dlatego też dołożyliśmy wszelkich

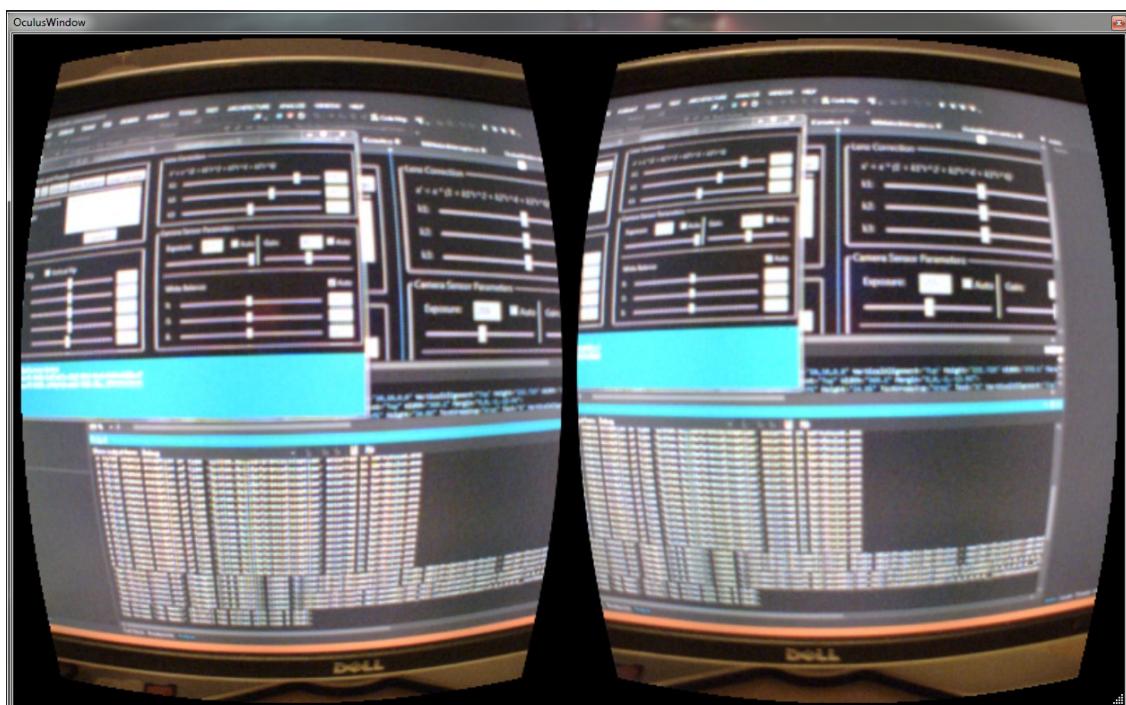
starań, aby był możliwie najbardziej przejrzysty, intuicyjny, stabilny, prosty w obsłudze, niezawodny oraz odporny na wszelkiego typu sytuacje losowe. Chcieliśmy umożliwić użytkownikowi kompleksowy dostęp do wszystkich funkcji programu, nie przytłaczając go tym samym ich mnogością komunikatów.

4.6.3. Wielowątkowość

Głównym wątkiem naszej aplikacji jest ten odpowiadający za ControlsWindow. To właśnie on rozpoczyna pracę wszystkich procesów, o których wspomnimy poniżej. Zgodnie z Rys. 4.3., obsługa kamer zajmuje 3 wątki – po jednym na każdą z kamer wraz z wątkiem wspólnym, służącym za transformację i obsługę obrazu[16]. Są one podległe wątkowi OculusWindow (obsługi obrazu trafiającego na ekran Oculus Rifta), bezpośrednio rozpoczętym przez ControlsWindow. Oddzielny, równoległy wątek przynależy również do części integracyjnej – odpowiadającej za pobieranie kątów bezpośrednio z Oculus Rifta, przetwarzanie ich, oraz przesyłanie zarówno do serwogłówicy, jak i do nadrzędnego okna kontrolera (oczywiście, w tym przypadku w odpowiednio większych odstępach czasu).

Daje to, nie licząc wątków systemowych i towarzyszących, sześć wątków. Naszym zdaniem jest to optymalna wartość, biorąc pod uwagę, że większość współczesnych procesorów oferuje nam ich osiem lub więcej – możemy w ten sposób wykorzystać ich potencjał, zostawiając wolne miejsce na wątki towarzyszące, systemowe, czy działające w tle.

4.7. OculusWindow

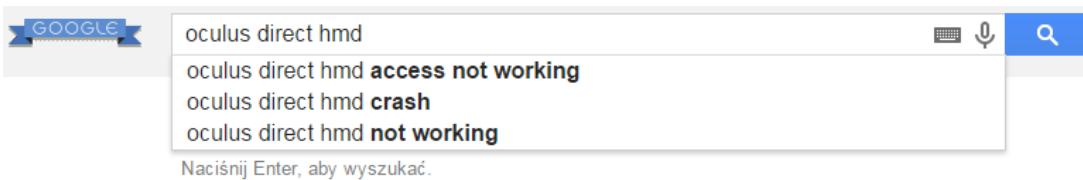


Rys. 4.34. Zrzut okna OculusWindow

4.7.1. Direct HMD Access czy Extended Desktop to the HMD?

OculusWindow jest tym oknem, które trafia na ekran hełmu rzeczywistości wirtualnej. Dzięki wyborowi trybu wyświetlania “Extended Desktop to the HMD” (rozszerzenie pulpu do hełmu wirtualnego) zamiast “Direct HMD Access from Apps” (bezpośredni dostęp do hełmu z poziomu aplikacji), jest to “normalne”, WPF–owe okno, widoczne na ekranie monitora – jako że w tym trybie ekran Oculus Rifta widziany jest przez system właśnie w taki sposób. Zdecydowaliśmy się na to rozwiązanie z kilku powodów:

- Awaryjność rozwiązania. Tryb “Direct HMD Access from Apps” jest wciąż niedopracowany, dla specjalistyczne są pełne wątków dotyczących rozwiązywania problemów z ich (nie)działaniem, a pierwszym wynikiem zwracanym w wyszukiwarkach po wpisaniu frazy “oculus direct hmd” jest “oculus direct hmd access not working” (Rys. 4.26.),



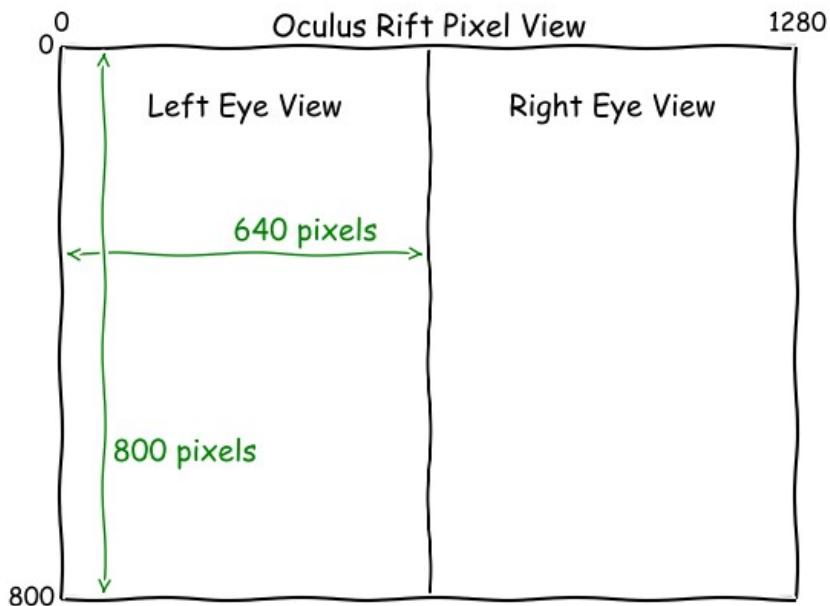
Rys. 4.35. Wyniki wyszukiwania po wpisaniu frazy "oculus direct hmd" w google[43]

- W naszym wypadku istnieje niewielka różnica pomiędzy wydajnością w trybach “Extended Desktop to the HMD” oraz “Direct HMD Access from Apps”. Dane dotyczące położenia i tak pobierane są bezpośrednio poprzez SharpOVR (nakładkę na Oculus SDK), a różnica opóźnień pomiędzy obydwooma trybami jest o rząd niższa od opóźnień wprowadzanych przez samą serwogłówicę,
- Dzięki zastosowaniu “Extended Desktop to the HMD” otrzymujemy rozwiązanie generyczne, współpracujące z niemal każdym hełmem rzeczywistości wirtualnej wspierającym ten tryb wyświetlania – szczególnie dzięki rozwiązaniom opisanym poniżej,
- Wyłącznie renderowana grafika potrafi w pełni wykorzystać potencjał trybu “Direct HMD Access from Apps”. Mając do czynienia nie z rzeczywistością wirtualną i obrazem generowanym komputerowo, a pozyskiwanym przez kamery, zalety wykorzystania tego trybu są niewspółmierne do ich jego niestabilności oraz wad.

4.7.2. Zawartość oraz położenie okna – OculusWindow.xaml

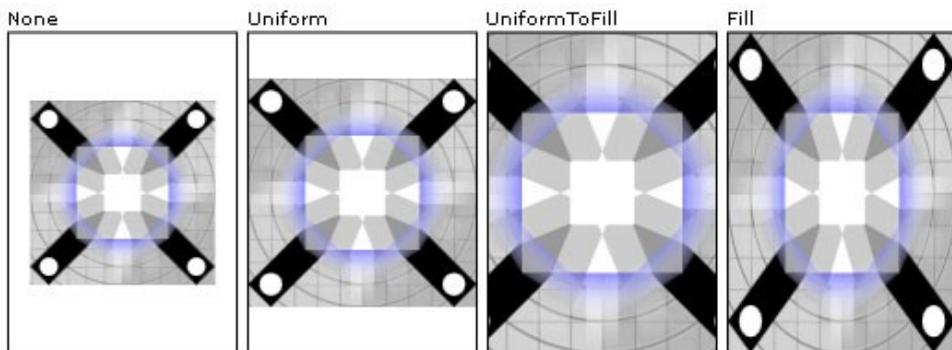
Jak wspomnieliśmy w poprzednim punkcie, z uwagi na wybór trybu rozszerzenia pulpu do hełmu wirtualnego, nasze okno stworzone zostało w Windows Presentation Foundation. Umożliwiło to nam, po odpowiednich ustawieniach, dowolną manipulację jego rozmiarem, przy zachowaniu adekwatnych proporcji. Dzięki temu otrzymujemy rozwiązanie generyczne – które od strony graficznej będzie współpracowało z każdym hełmem rzeczywistości wirtualnej

wspierającym ten tryb wyświetlania – od modeli już leciwych do najnowszych prototypów lub nawet urządzeń, które jeszcze nie pojawiły się na rynku.



Rys. 4.36. Podział ekranu hełmu rzeczywistości wirtualnej Oculus Rift[26]

Aby to osiągnąć, niezbędne było zastosowanie nie konkretnych wartości definiujących wielkość okna (np. dla Oculus Rifta DK1 jest to 1280x800 pikseli, czyli 640x800 pikseli na oko, Rys. 4.27.), a wartości relatywnych. Na okno OculusWindow składa się siatka (ang. grid) zawierająca dwie sąsiadujące horyzontalnie ze sobą komórki dzielące siatkę poziomo w skali 1:1. Każda z nich zawiera teksturę typu CLEyeCameralImage, zainportowaną z CLEye Platform SDK, wyświetlającą obraz na żywo z kamery. Tekstury te są odpowiednio rozciągnięte w pionie oraz wyśrodkowane w poziomie przy zachowaniu proporcji (kolejno opcje VerticalAlignment: Stretch, HorizontalAlignment: Center, Stretch: UniformToFill, StretchDirection: Both). Tylko powyższa kombinacja ustawień okazała się być skuteczna dla każdego z testowanych rozmiarów oraz dawała nam pełną możliwość manipulacji teksturami CLEyeCameralImage z pomocą modyfikatorów ujętych w oknie ControlsWindow.



Rys. 4.37. Rodzaje rozciągnięcia obrazu w Windows Presentation Foundation[44]

4.7.3. Kontroler okna OculusWindow – OculusWindow.xaml.cs

Jak wspomnieliśmy w punkcie 4.7.1., to w kontrolerze opisana jest cała logika okna. W przypadku OculusWindow, odpowiada on nie tylko za wyświetlenie obrazu na ekranie hełmu rzeczywistości wirtualnej, ale również obsługę kamer PlayStation Eye. To tutaj są one tworzone, inicjowane, czy ustawiane. Klasa ta jako jedyna w naszym projekcie bezpośrednio odwołuje się do CLEye Platform SDK i posiada do niego referencje. Staraliśmy się jednak, aby jej implementacja była jak najprostsza z uwagi na profilowanie wydajności pod kątem niskich opóźnień przechwytywania, przetwarzania oraz wyświetlania obrazu – a nie responsywności modyfikatorów przekształceń. W związku z tym obsługa wszystkich kontrolek odpowiedzialnych za parametry przekształcenia obrazu znajduje się w innym oknie, kontrolerze oraz wątku, a implementacja OculusWindow jest dość skromna i prezentuje się następująco (warto nadmienić, że definicje niektórych elementów znajdują się nie w pliku .xaml.cs, a w pliku .xaml, więc nie są widoczne we wklejonym fragmencie kodu):

```
public partial class OculusWindow : Window
{
    ControlsWindow controlsWindow;
    int numCameras = 0;
    bool camsSwapped = false;

    public OculusWindow(ControlsWindow window)
    void OculusWindow_Loaded(object sender, RoutedEventArgs e)
    public void OculusWindow_Closing(object sender,
                                      System.ComponentModel.CancelEventArgs
                                      e)
    public void SwapCameras()
    public void SwapRotation()
    private void Window_Closed(object sender, EventArgs e)
}
```

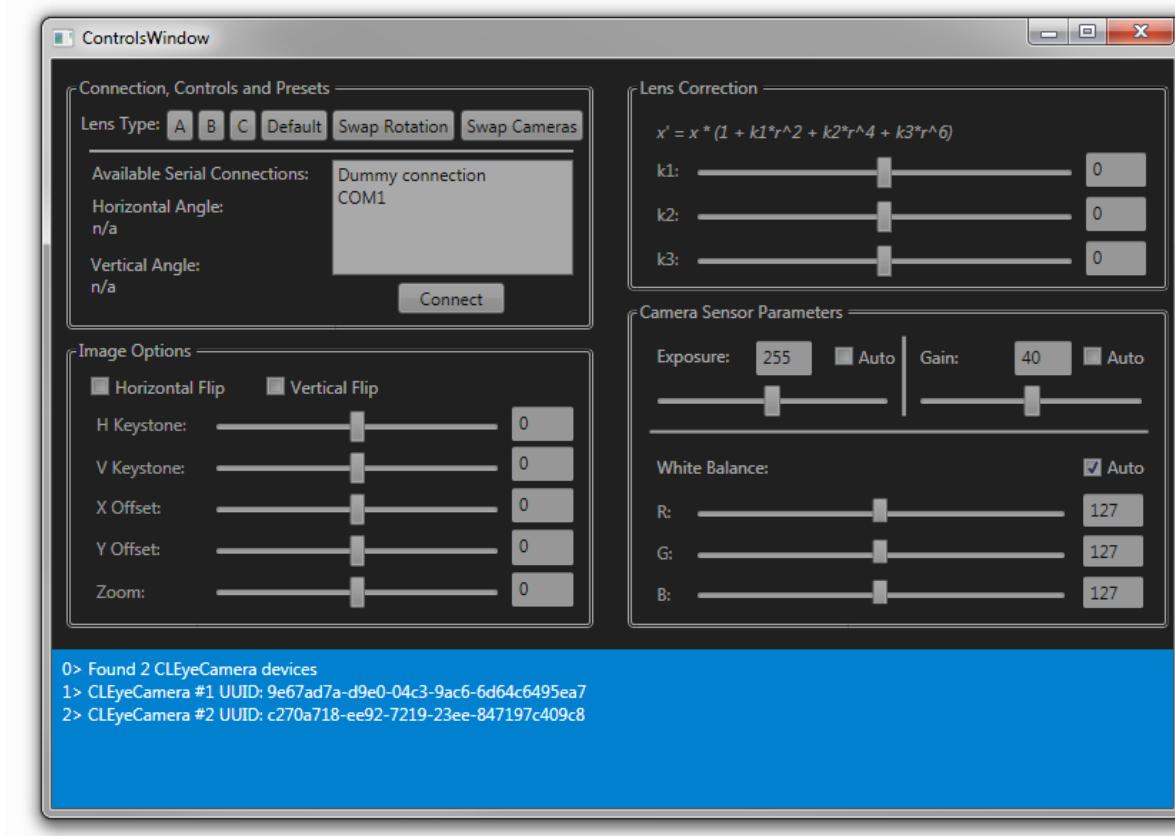
Tabela 4.10. Lista wybranych pól klasy OculusWindow

Pole	Typ	Opis
controlsWindow	ControlsWindow	Obiekt nadzawanego, inicjującego nasz obiekt, okna interfejsu ControlsWindow, na które przesyłamy wszystkie komunikaty.
numCameras	int	Liczba podłączonych do komputera kamer, zwracana przez CLEyeCameraDevice.CameraCount.
camsSwapped	bool	Pomocnicza zmienna logiczna tożsama z tym, czy kamery zostały zamienione.
cameralImageLeft	CLEyeCameralImage	Tekstura odpowiadająca za lewą kamerę.
cameralImageRight	CLEyeCameralImage	Tekstura odpowiadająca za prawą kamerę.

Tabela 4.11. Lista wybranych metod klasy OculusWindow

Metoda	Opis
OculusWindow(ControlsWindow window)	Konstruktor okna – jako parametr podajemy nadzędne okno typu ControlsWindow, które przypisujemy do zmiennej controlsWindow.
OculusWindow_Loaded(object sender, RoutedEventArgs e)	W tej funkcji przeprowadzane są wszystkie akcje mające miejsce po załadowaniu okna. Czytana jest ilość podłączonych kamer, tworzone oraz ustawiane są obiekty typu CLEyeCameralmage oraz wysyłana jest wiadomość o statusie wykonanego zadania.
SwapRotation()	Funkcja zamieniająca rotację kamer o 180°.
SwapCameras()	Funkcja zamieniająca przypisanie kamer do tekstur. Aby to zrealizować, kamery są zatrzymywane oraz niszczone jak przy wyjściu, a następnie podmieniane i ponownie inicjalizowane.
OculusWindow_Closing(object sender, System.ComponentModel.CancelEventArgs e)	Funkcja wywoływana podczas zamykania okna. Przed wyjściem zatrzymywane oraz niszczone są obiekty typu CLEyeCameralmage.
Window_Closed(object sender, EventArgs e)	Funkcja wywoływana po zamknięciu okna. Aplikacja jest zamykana.

4.8. ControlsWindow



Rys. 4.38. Zrzut okna ControlsWindow

ControlsWindow to okno główne programu, spinające wszystkie części projektu. To ono inicjalizowane jest jako pierwsze, rozpoczyna i zarządza wszystkimi wątkami, czy odpowiedzialne jest za komunikację między nimi. Jest to również jedyne okno odpowiadające za kontakt z użytkownikiem – to tutaj znajdują się wszystkie opcje, wyświetlane są informacje, czy komunikaty o błędach. Interfejs podzielony jest na pięć części – cztery grupy ustawień znajdujące się w oddzielnego GroupBox'ach oraz StatusBar na dole okna informujący użytkownika o błędach i komunikatach.

4.8.1. Zawartość oraz położenie okna – ControlsWindow.xaml

Podobnie jak OculusWindow, nadzorującym elementem ControlsWindow jest Grid (pol. siatka). Dzieli ona okno na 4 (efektywnie 3) części: poziomo w stosunku 1:1 oraz pionowo per pixel: 102 pikseli dla części górnej zawierającej GroupBox'y oraz 31 dolnych pikseli zarezerwowanych dla StatusBar'a. W WPF siatka najpierw jest definiowana, a następnie w jej kolejnych komórkach umieszczane są elementy interfejsu. Grid jest wymaganym elementem interfejsu w każdym przypadku, gdy chcemy umieścić na zadanym obszarze więcej niż jedną kontrolkę. Przykładowo, GroupBox "Connection Controls and Presets" zawiera się w Gridzie, gdyż okno mieści więcej niż jednego GroupBoxa – ale wspomniany GroupBox nie jest tylko

dzieckiem typu Grid, ale i rodzicem kolejnej siatki zawierającej wewnątrz liczne kontrolki typów Button, Label, ListBox i innych.

W przypadku ControlsWindow, specjalnie okrojona (wycięte fragmenty reprezentuje "(.....)") i opatrzona komentarzami (<!-- Komentarz -->) definicja głównych elementów siatki prezentuje się następująco:

```
(.....)
<Grid Background="#FF1C1C1C" RenderTransformOrigin="0.531,0.518">
    <!-- Definicje układu siatki głównej -->
    <Grid.RowDefinitions>
        <RowDefinition Height="102*" />
        <RowDefinition Height="31*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <!-- Zawartość elementów siatki głównej -->
    <GroupBox Header="Connection, Controls and Presets" (.....) >
        <GroupBox.BorderBrush>
            <SolidColorBrush Color="#FF151515"/>
        </GroupBox.BorderBrush>
        <Grid HorizontalAlignment="Left" Height="156"
VerticalAlignment="Top"
            Width="359" Margin="0,0,-2,0">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="201*" />
                <ColumnDefinition Width="158*" />
            </Grid.ColumnDefinitions>
            <!--Zawartość GroupBoxa Connection, Controls and Presets -->
            (.....)
        </Grid>
    </GroupBox>
    <GroupBox Header="Lens Correction" (.....)">
        <Grid HorizontalAlignment="Left" Height="141.719"
VerticalAlignment="Top"
            Width="369.1" Margin="0,0,-2,-12.96">
            <!-- Zawartość GroupBoxa Lens Correction -->
            (.....)
        </Grid>
    </GroupBox>
    <GroupBox Header="Camera Sensor Parameters" (.....)>
        <Grid HorizontalAlignment="Left" Height="211.355"
VerticalAlignment="Top"
            Width="369.1" Margin="0,0,-2,0">
```

```

        <!-- Zawartość GroupBoxa Camera Sensor Parameters -->
        (.....)
    </Grid>
</GroupBox>
<!-- StatusBar – szeroki na 2 rzędy: Grid.RowSpan="2" -->
(.....)
<StatusBar x:Name="StatusBar" Grid.ColumnSpan="2"
Background="#FF007ACC"
    Foreground="#FFE6E6E6" Grid.Row="1" Grid.RowSpan="2">
<ListBox x:Name="statusBox" (.....)>
    <!-- Zawartość ListBoxa statusBox -->
    (.....)
</ListBox>
</StatusBar>
<GroupBox Header="Image Options" (.....)>
    <Grid HorizontalAlignment="Left" Height="194"
VerticalAlignment="Top"
        Width="359" Margin="0,0,-2,-12">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <!-- Zawartość GroupBoxa Image Options -->
        (.....)
    </Grid>
</GroupBox>
</Grid>
</Window>
(.....)

```

4.8.2. Inicjalizacja okna oraz wątków

Jak wspomnieliśmy na początku punktu 4.7.2., ControlsWindow jest oknem głównym projektu, rozpoczynającym oraz zarządzającym wszystkimi wątkami. Poniżej przedstawimy oraz opiszemy kolejne kroki wykonywane przy rozpoczęciu programu. Konstruktor klasy prezentuje się następująco:

```

public ControlsWindow()
{
    InitializeComponent();
    NewWindowHandler(null,null);
}

```

Wywoływana jest metoda InitializeComponent, będąca domyślnym konstruktorem okna w Windows Presentation Foundation, inicjująca część klasy, odpowiadającą za typy Window oraz UserControl. Kolejna funkcja, NewWindowHandler, została zaimplementowana przez nas, a jej zadaniem jest przygotowanie do działania wszystkich pozostałych komponentów.

```

private void NewWindowHandler(object sender, RoutedEventArgs e)
{

```

```

        oculusWindowThread = new Thread(
            new ThreadStart(OculusWindowThreadStartingPoint));
        oculusWindowThread.SetApartmentState(ApartmentState.STA);
        oculusWindowThread.IsBackground = true;
        oculusWindowThread.Name = "Oculus window thread";
        oculusWindowThread.Start();
        // Debug mode when TRUE (uses dummy object instead of real
        Oculus HMD)
        interceptor = new RiftMotionInterceptor(false, this);

        List<String> connectionList = interceptor.GetConnectionsList();
        foreach (string connName in connectionList)
        {
            comboBox.Items.Add(connName);
        }
    }
}

```

Pierwsze pięć linii metody odpowiada za ustawienie parametrów oraz inicjalizację wątku oculusWindowThread, który, jak wskazuje jego nazwa, odpowiada za OculusWindow. Kolejno, tworzony jest obiekt klasy odpowiadającej za część integracyjną. Następnie pobieramy z niego obiekt connectionList oraz z pomocą ostatniej pętli wyświetlamy jego zawartość – listę dostępnych portów do połączenia z serwogłówicą. Poniżej znajduje się implementacja metody OculusWindowThreadStartingPoint wołanej przy tworzeniu wątku w pierwszej linii opisywanej wyżej funkcji.

```

private void OculusWindowThreadStartingPoint()
{
    oculusWindow = new OculusWindow(this);
    oculusWindow.Show();
    System.Windows.Threading.Dispatcher.Run();
}

```

4.8.3. Komunikacja pomiędzy wątkami

Jak wspomnieliśmy w punkcie 4.8.3., OculusWindow jest jedynym oknem odwołującym się do biblioteki CLEye Platform, a interfejs oraz wartość kontrolek znajdują się w ControlsWindow. Dlatego też wszystkie ustawienia dotyczące przekształceń obrazu wykonywanych po stronie sterownika, wywoływane są ze strony ControlsWindow poprzez referencję do obiektu, na przykład:

```

private void gainSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
{
    try
    {
        gainTBox.Text = gainSlider.Value.ToString();
        oculusWindow.cameraImageLeft.Device.Gain =
(int)gainSlider.Value;
        oculusWindow.cameraImageRight.Device.Gain =
(int)gainSlider.Value;
    }
    catch (Exception)
    {

```

```
        }  
    }
```

Można zauważyc, że jawnie odwołujemy się poprzez stworzony przy inicjalizacji obiekt oculusWindow, zarówno do lewej, jak i prawej kamery. W większości przypadków zadawane wartości są identyczne, jednak w niektórych przypadkach zadawane są wartości odwrotne – na przykład przy ustawianiu parametru HorizontalKeystone:

```
oculusWindow.cameraImageLeft.Device.HorizontalKeystone =  
    (int)hKeystoneSlider.Value;  
oculusWindow.cameraImageRight.Device.HorizontalKeystone =  
    (int)(-hKeystoneSlider.Value);
```

Nie wszystkie wartości umożliwiają zastosowanie tak bezpośredniego podejścia. Niejednokrotnie nie było to możliwe np. z uwagi na brak możliwości dostępu do danych wartości innego wątku lub wywoływanie jego metod. W tym wypadku konieczne było wykorzystanie mechanizmu Dispatcher'a (pol. dyspozytora) – operacje takie były umieszczone w bloku analogicznym do poniższego:

```
private void resetAnglesButton_Click(object sender, RoutedEventArgs  
e)  
{  
    Dispatcher.BeginInvoke((Action)() =>  
    {  
        interceptor.RecenterHmd();  
    });  
}
```

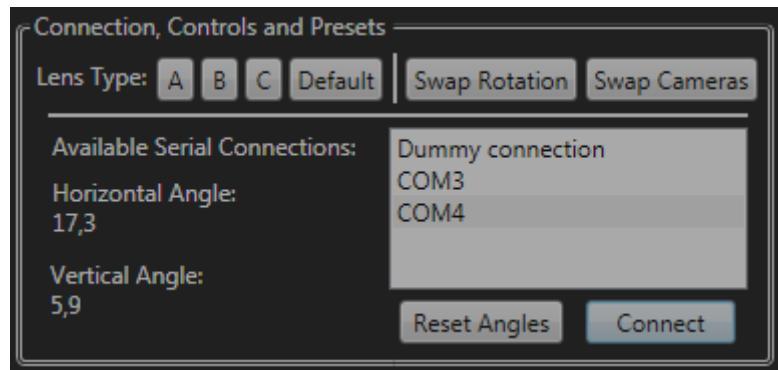
Ważnym elementem towarzyszącym wielowątkowości naszej aplikacji jest znajdujący się na dole ControlsWindow pasek statusu. Pojawiające się na nim komunikaty pochodzą nie tylko od naszego, nadziednego wątku, ale również wszystkich wątków podrzędnych. Aby ułatwić korzystanie z niego oraz stworzyć kod czytelniejszym, zdecydowaliśmy się zaimplementować specjalnie do tego służącą funkcję typu public oraz umieścić wewnątrz niej dispatcher (zamiast umieszczać ją wewnątrz jego lub starać się dodawać komunikaty bezpośrednio odwołując się do elementu interfejsu).

```
public void PrintMessage(string msg)  
{  
    Dispatcher.BeginInvoke((Action)() =>  
    {  
        statusBox.Items.Add(statusBox.Items.Count + "> " + msg);  
        statusBox.ScrollIntoView(  
            statusBox.Items[statusBox.Items.Count - 1]);  
    });  
}
```

Pierwsza z linii wewnątrz Dispatcher'a dodaje elementy do statusBox'a, druga natomiast ustawia widok w taki sposób, by ostatnio dodany komunikat był widoczny.

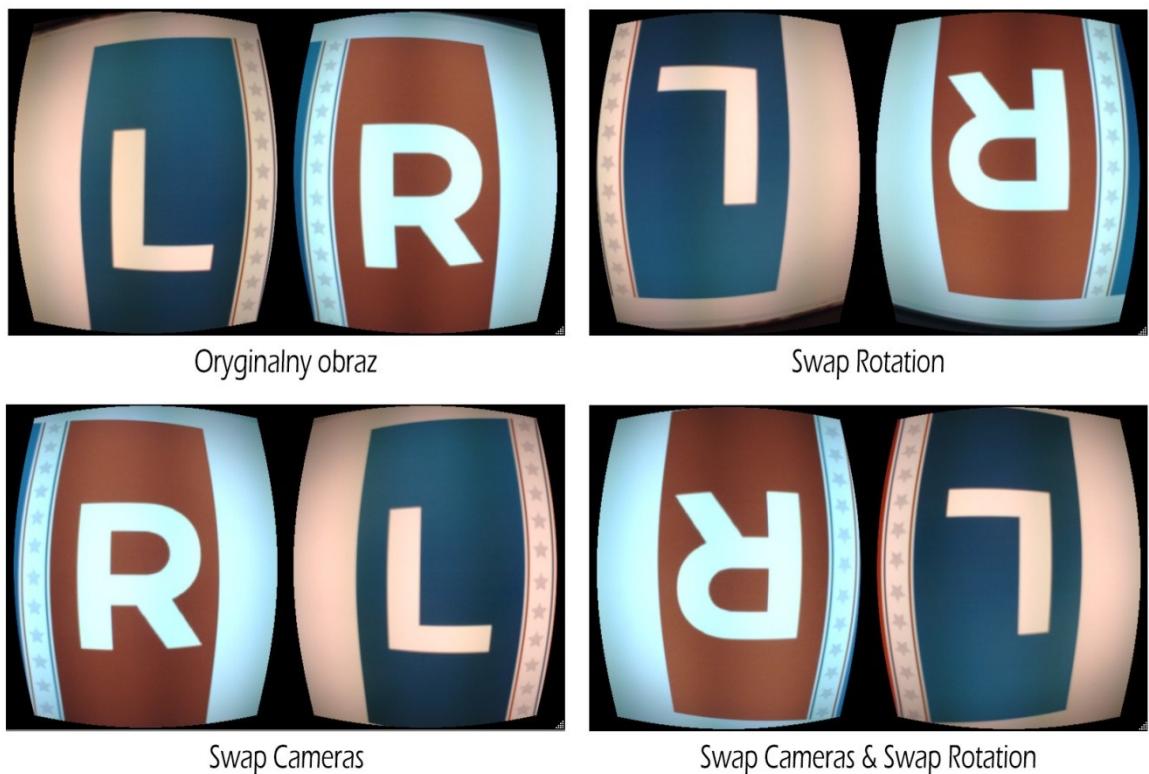
W poniższych punktach bliżej przyjrzymy się się kolejnym grupom interfejsu oraz realizowanym przez nie funkcjom.

4.8.4. Connection, Controls and Presets



Rys. 4.39. Grupa ustawień "Connection, Controls and Presets"

Connection, Controls and Presets to grupa odpowiadająca za połączenie z serwogłówką poprzez część integracyjną. W jej lewej górnej części mamy możliwość wyboru predefiniowanych ustawień (ang. presets) soczewki – A, B, C oraz Default – przywrócenia wartości do domyślnych. Na prawo znajdują się przyciski odpowiadające za zamianę rotacji (ang. swap rotation) oraz kamer (ang. swap cameras). Konieczne było wprowadzenie tych opcji z uwagi, że kamery są na stałe przymocowane do serwogłówicy, a ich przyporządkowanie do tekstur odbywa się losowo, w zależności od ich UUID. Z uwagi na ich położenie boczne, lewa tekstura powinna być obrócona od 270° , a prawa o 90° – stąd konieczność wprowadzenia nie tylko opcji zamiany źródła obrazu, ale i jego rotacji.



Rys. 4.40. Porównanie działania opcji "Swap Rotation" oraz "Swap Cameras"

Obydwie ze wspomnianych funkcji są zaimplementowane wewnątrz OculusWindow i zostały opisane w punkcie 4.7.3. O ile opcja zamiany obrotu odbywa się bardzo prosto, poprzez zmianę wartości parametru Rotation obiektu typu CLEyeCameralmage jak w pierwszym przypadku omawianym w punkcie 4.9.3., to zamiana kamer okazała się być bardziej problematyczna. Nie chodzi tutaj wyłącznie o trudności opisane wcześniejszej w punkcie 4.8.3., ale również związane ze stosowaniem tej metody efekty uboczne – po reinicjalizacji kamer, ich ustawienia wracają do wartości domyślnych. Dlatego też niezbędna była implementacja mechanizmu odświeżającego wszystkie ustawienia obrazu zmienione przez użytkownika.

```
private void swapCamerasButton_Click(object sender, RoutedEventArgs e)
{
    Dispatcher.BeginInvoke((Action)() =>
    {
        oculusWindow.SwapCameras();
        Thread.Sleep(1000);
        reapplySettings();
    });

    statusBox.Items.Add(
        statusBox.Items.Count + "> " + "Cameras swapped");
    statusBox.ScrollIntoView(
        statusBox.Items[statusBox.Items.Count - 1]);
}
```

Jak widać w powyższym fragmencie kodu, wywoływane przez nas funkcje zawarte są w mechanizmie Dispatcher. Wywołujemy metodę SwapCameras na obiekcie OculusWindow, czekamy 1 sekundę na przeprowadzenie odpowiednich operacji, a następnie przywracamy ustawienia obrazu z pomocą metody reapplySettings. Działanie tej metody sprowadza się do wymuszenia eventu zmiany wartości dla każdej z kontrolek, co skutkuje odświeżeniem każdego z modyfikatorów z osobna.

```
private void reapplySettings()
{
    if (autoGainCheckBox.IsChecked.Value)
        autoGainCheckBox_Checked(null, null);
    else
        autoGainCheckBox_Unchecked(null, null);

    if (autoExposureCheckBox.IsChecked.Value)
        autoExposureCheckBox_Checked(null, null);
    else
        autoExposureCheckBox_Unchecked(null, null);

    if (whiteBalanceCheckBox.IsChecked.Value)
        whiteBalanceCheckBox_Checked(null, null);
    else
        whiteBalanceCheckBox_Unchecked(null, null);

    if (horizontalFlipCheckBox.IsChecked.Value)
        horizontalFlipCheckBox_Checked(null, null);
    else
        horizontalFlipCheckBox_Unchecked(null, null);

    if (verticalFlipCheckBox.IsChecked.Value)
        verticalFlipCheckBox_Checked(null, null);
    else
```

```

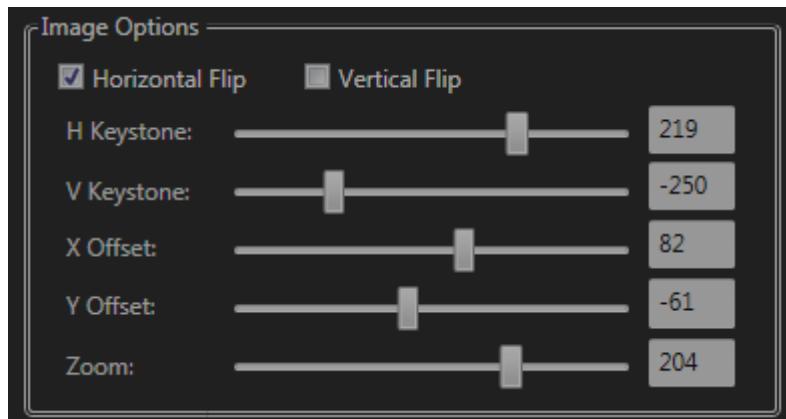
        verticalFlipCheckBox_Unchecked(null, null);

lensCorrectionSlider1_ValueChanged(null, null);
lensCorrectionSlider2_ValueChanged(null, null);
lensCorrectionSlider3_ValueChanged(null, null);

hKeystoneSlider_ValueChanged(null, null);
vKeystoneSlider_ValueChanged(null, null);
xOffsetSlider_ValueChanged(null, null);
yOffsetSlider_ValueChanged(null, null);
zoomSlider_ValueChanged(null, null);
}

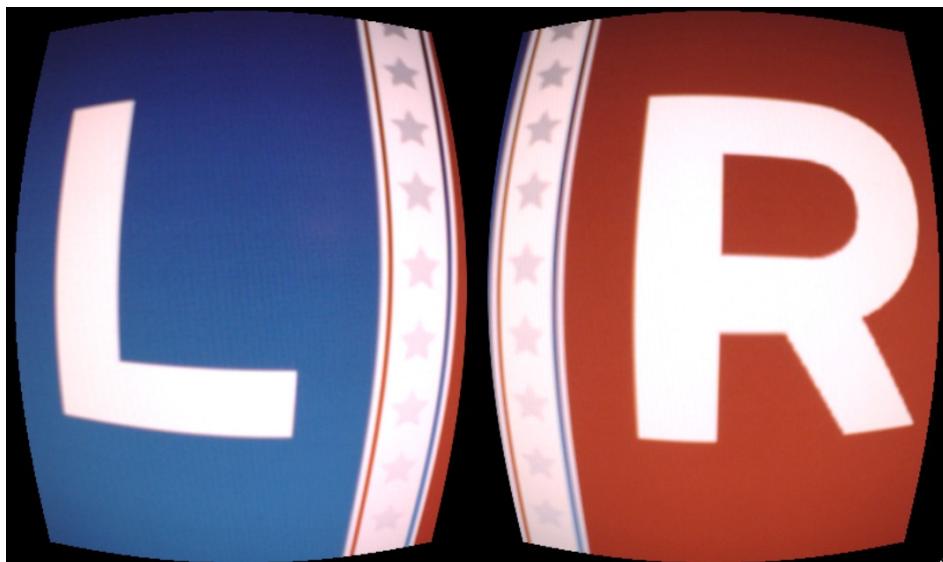
```

4.8.5. Image Options



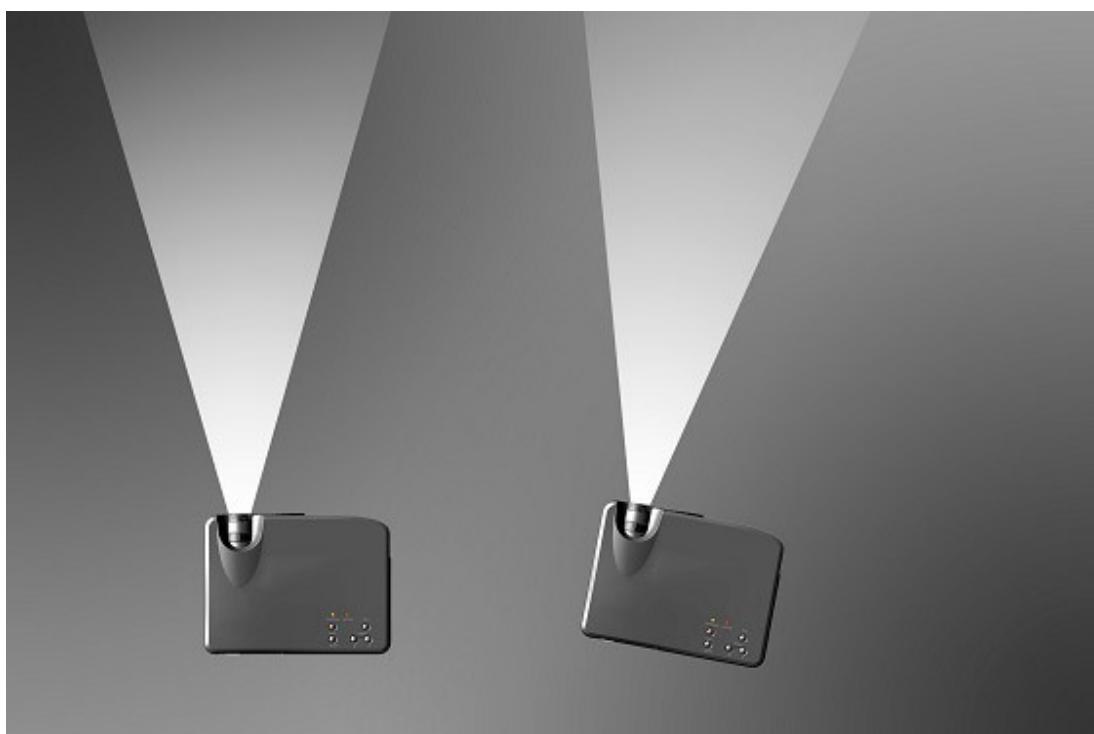
Rys. 4.41. Grupa ustawień "Image Options"

W GroupBoxie Image Options (pol. opcje obrazu) dostępne są wszystkie opcje liniowego przekształcenia obrazu wykonywane po stronie sterownika kamer. Poniżej przyjrzymy się po kolejnej każdej z nich. Wszystkie prezentowane obrazy posiadają już odpowiednie zniekształcenie soczewki, tak jakby obraz miał być prezentowany na ekranie Oculus Rifta. Warto również nadmienić, że wszystkie opisywane transformacje liniowe mają miejsce przed zastosowaniem korekcji soczewki, co ma duże znaczenie praktyczne, szczególnie biorąc pod uwagę intuicyjność wprowadzania ustawień oraz zupełną niezależność przekształceń liniowych od korekcji soczewki – w ten sposób po zmianie jakiegokolwiek z parametrów, nie musimy ustawiać jej na nowo.



Rys. 4.42. Oryginalny obraz (z ustawioną korekcją soczewki)

- H Keystone oraz V Keystone (skrót od Horizontal/Vertical Keystone – pozioma oraz pionowa korekcja zniekształceń trapezowych) – rozwiązanie najczęściej znajdujące zastosowanie w projektorach. Jeżeli wyświetlany obraz nie pada na powierzchnię bezpośrednio pod kątem 90° (Rys. 4.34.), zostaje zniekształcony (Rys. 4.35.) – aby tego uniknąć, stosuje się przekształcenie odwrotne. W przypadku niektórych HMD może to mieć duże znaczenie, jeśli np. ekran nie jest ustawiony pod kątem 90°, centralnie przed okiem użytkownika.



Rys. 4.43. Projektor skierowany prostopadle do powierzchni (po lewej) oraz rzucający obraz wymagający korekcji zniekształceń trapezowych (po prawej)[45]



Rys. 4.44. Obraz zniekształcony (po lewej) oraz po zastosowaniu korekcji zniekształceń trapezowych (po prawej) – obrazy bez transformacji beczkowej[45]



Rys. 4.45. Obrazy po przekształceniu: H Keystone = 280 (po lewej) oraz H Keystone = -500 (po prawej)



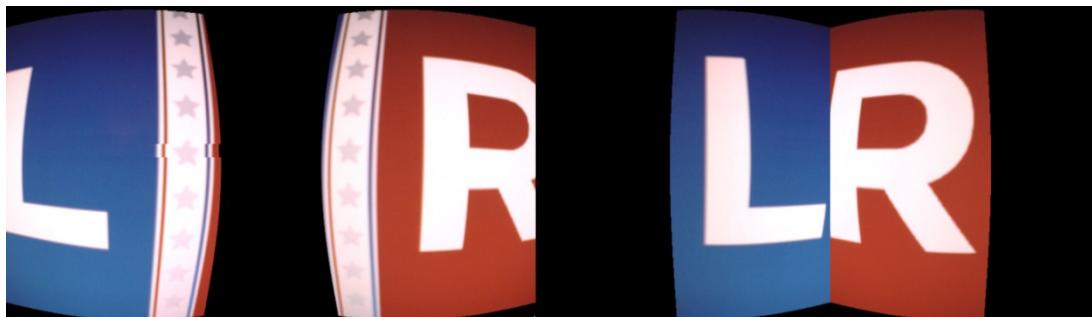
Rys. 4.46. Obrazy po przekształceniu: V Keystone = 500 (po lewej) oraz V Keystone = -200 (po prawej)

- X Offset oraz Y Offset (przesunięcie w poziomie oraz pionie) – proste przesunięcie obrazu w osi X i Y. W naszym przypadku jest to szczególnie ważne w przypadku nierównego ustawienia kamery, drobnych dekalibracji, problemów z wyświetlaczem, czy po prostu dostosowaniem sposobu

wyświetlania pod konkretnego użytkownika (ze względu na np. jego rozstaw oczu). Z pomocą tego przekształcenia możemy zmienić wzajemne położenie kamer przesuwając tekstury wyżej i niżej, jak również zmieniając poziomą odległość pomiędzy nimi. Jak można zauważyć, X Offset tak naprawdę modyfikuje położenie w pionie, a Y Offset w poziomie – czyli odwrotnie, niż wskazywałoby ich nazewnictwo. Postanowiliśmy jednak trzymać się tych nazw, gdyż właśnie tak zostały one określone przez twórców CL Eye Device SDK.



Rys. 4.47. Obraz po przekształceniu X Offset



Rys. 4.48. Obrazy po przekształceniu: $Y\ Offset = 100$ (po lewej) oraz $Y\ Offset = -200$ (po prawej)

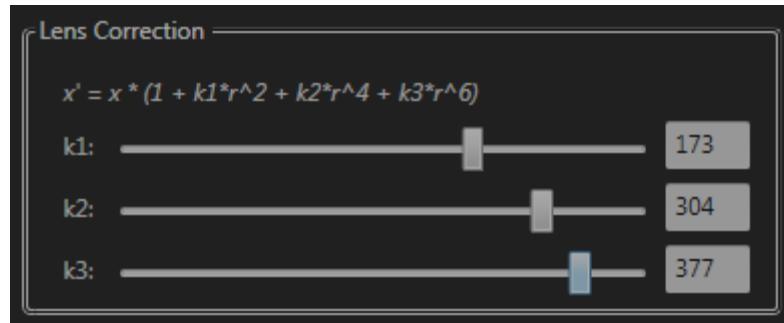
- Zoom (powiększenie) – kolejne proste, często wykorzystywane przez nas przekształcenie obrazu. Polega ono na powiększaniu i zmniejszaniu tekstur – co pozwala odpowiednio dobrą ich rozmiar w zależności od wymienionych wyżej przekształceń oraz zwiększenia soczewki, aby osiągnąć optymalne pole widzenia przy zachowaniu zadowalającej rozdzielczości tekstur.



Rys. 4.49. Obrazy po przekształceniu: Zoom = 200 (po lewej) oraz Zoom = -300 (po prawej)

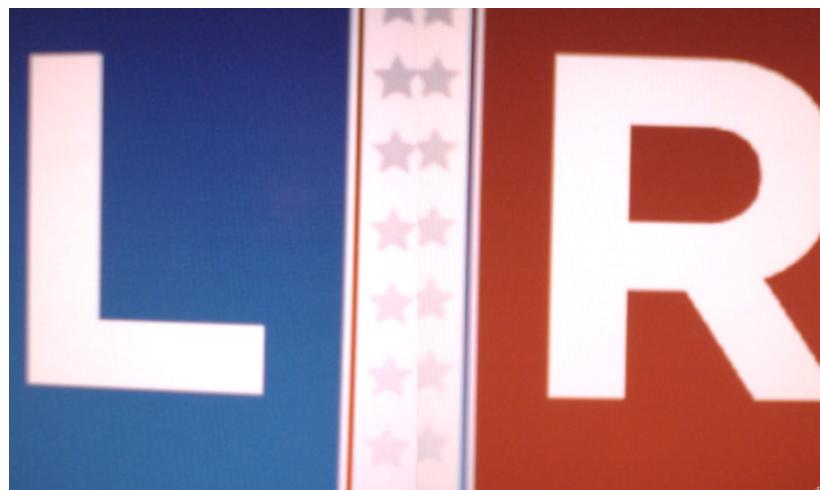
- Horizontal Flip oraz Vertical Flip (symetria osiowa względem osi pionowej oraz poziomej) – prosta transformacja obrazu polegająca na odwróceniu jego względem osi X lub Y.

4.8.6. *Lens Correction*

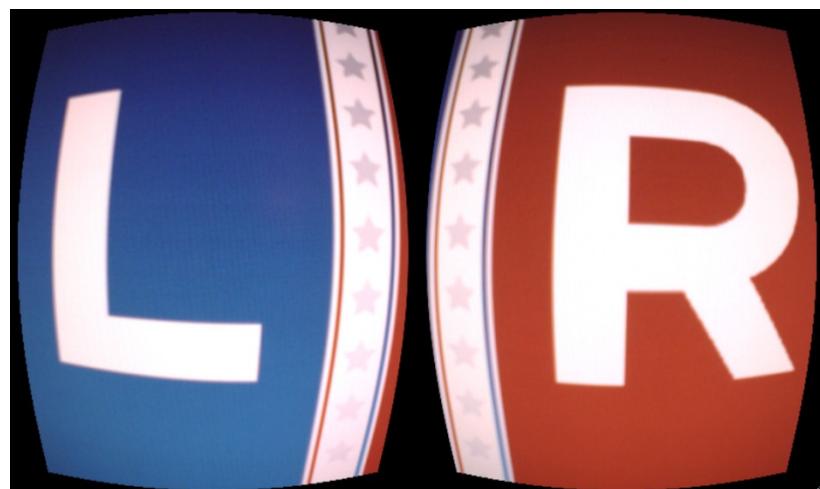


Rys. 4.50. Grupa ustawień "Lens Correction"

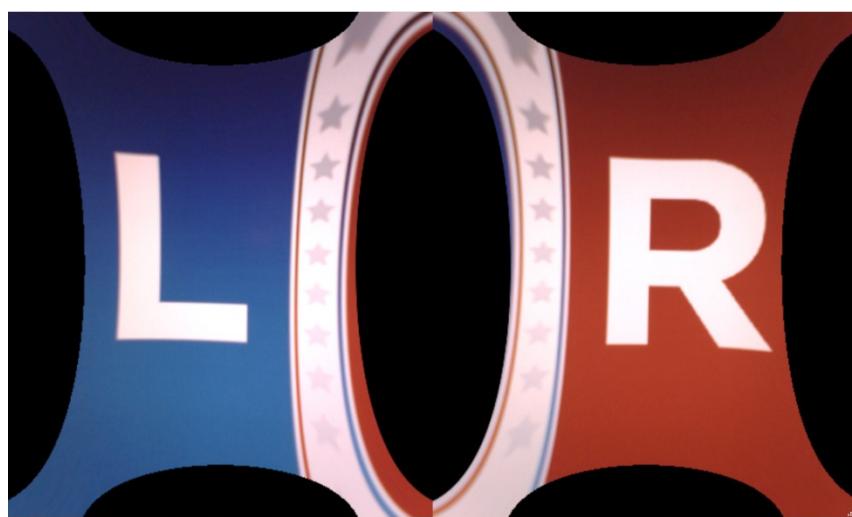
Funkcjom realizowanym w GroupBoxie Lens Correction zostały poświęcone działy 4.2.2. oraz 4.2.3. Wartości k1, k2 oraz k3 reprezentują kolejne parametry podstawiane do wzorów 4.1 oraz 4.2. Dzięki zakresie wartości od -500 do 500 możliwa jest realizacja zarówno dystorsji beczkowej (dla wartości dodatnich, Rys 4.43.), jak i poduszkowej (dla wartości ujemnych, Rys 4.44.).



Rys. 4.51. Obraz bez jakichkolwiek przekształceń

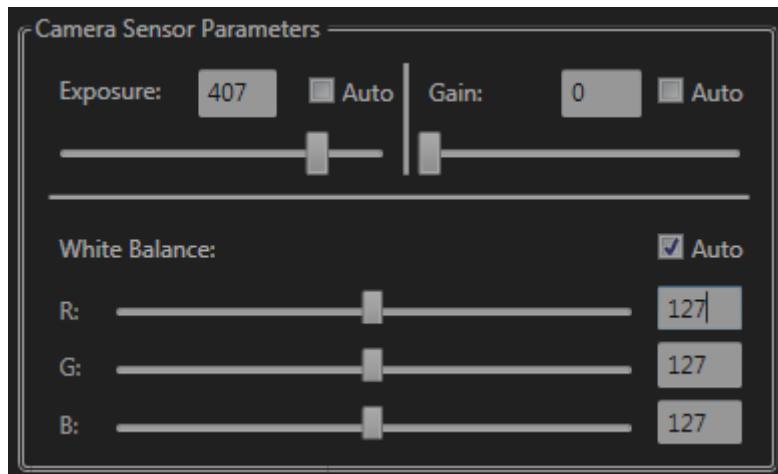


Rys. 4.52. Obraz przystosowany do soczewki A: Zoom = -128, LensCorrection k1 = 256 (dystorsja beczkowa)



Rys. 4.53. Obraz po korekcji poduszkowej: Zoom = -500, LensCorrection k1 = -350

4.8.7. Camera Sensor Parameters



Rys. 4.54. Grupa ustawień "Camera Sensor Parameters"

Grupa Camera Sensor Parameters (parametry sensora kamery), w przeciwieństwie do wymienionych wyżej, jako jedyna oferuje nam nie przekształcenia kształtu obrazu, a ich koloru i barwy. Mamy możliwość edycji trzech opcji: Exposure (pol. ekspozycja), Gain (pol. przyrost) oraz White Balance (pol. balans bieli). Na ostatnią z nich składają się trzy składowe parametry odpowiadające za kolory główne: R (red – czerwony), G (green – zielony) oraz B (blue – niebieski). Każdą z nich można zarówno ustawić ręcznie, jak i włączyć tryb automatycznego doboru parametrów. Warto jednak nadmienić, że wartości automatyczne dobierane są indywidualnie dla każdej z kamer, co szczególnie znaczenie może mieć przy ustawianiu balansu bieli.

Ekspozycja określa czułość sensora odpowiedzialnego za przechwytywanie obrazu. Im wyższa jej wartość, tym jaśniejszy obraz otrzymujemy. Przyrost realizuje dokładnie to samo zadanie, ale w inny sposób. O ile ekspozycja wpływa bezpośrednio na ilość przechwytywanego światła, przyrost odpowiednio wzmacnia otrzymany już przez sensor obraz. Jest to obarczone jednak wadą – im wyższa wartość przyrostu, tym większa ilość generowanego szumu. W ogólnym przypadku nie używamy opcji Gain, chyba że ekspozycja została już ustalona na maksymalną wartość.

4.8.8. Ekran Statusu



Rys. 4.55. Ekran statusu z przykładowymi komunikatami

Ekran statusu ma za zadanie dostarczać użytkownikowi wszystkich informacji związanych z działaniem programu. Wyświetlane są tutaj komunikaty, błędy, czy instrukcje dla

użytkownika. W celu ułatwienia nawigacji oraz umożliwiania wyszukiwania komunikatów przez wewnętrzny silnik WPF, linie są numerowane.

Istnieją dwa sposoby na przesyłanie komunikatów na ekran statusu. Pierwszym oraz najbliższym z nich jest bezpośrednie dodawanie elementów do obiektu wewnętrz ControlsWindow.

```
statusBox.Items.Add(statusBox.Items.Count + "> " + "Komunikat");  
statusBox.ScrollIntoView(statusBox.Items[statusBox.Items.Count -  
1]);
```

Pierwsza z linii dodaje komunikat wraz z poprzedzającym go numerem linii wziętym poprzez właściwość obiektu statusBox.Items.Count. Druga z nich przewija widok do ostatnio dodanego komunikatu korzystając z tej samej (zaktualizowanej już) właściwości.

Drugi sposób umożliwia dodawanie komunikatów przez wszystkie elementy oraz wątki zewnętrzne. Jest to funkcja publiczna przyjmująca argumenty typu string korzystająca z mechanizmu Dispatcher, o zasadzie działania analogicznej do powyższego fragmentu kodu – metoda ta została przedstawiona oraz opisana w punkcie 4.9.3.

4.9. Biblioteki mikrokontrolera

Realizacja postawionego zadania wymagała stworzenia oprogramowania odpowiedzialnego za komunikację z oprogramowaniem zainstalowanym na PC oraz kontrolę mostków sterowniczych. Zrezygnowano z wykorzystania gotowych rozwiązań jak np. Arduino na rzecz zaprojektowanego na potrzeby projektu i dopasowanego do jego potrzeb zestawu klas i funkcji, który zostanie opisany w poniższych podrozdziałach.

Jako że kod powstawał w myśl zasady, iż „najlepszy komentarz to ten, którego nie musisz napisać”, dokumentacja w większości wypadków ograniczona została do prezentacji definicji klasy, ogólnego jej opisu oraz skomentowania jedynie najważniejszych metod i pól.

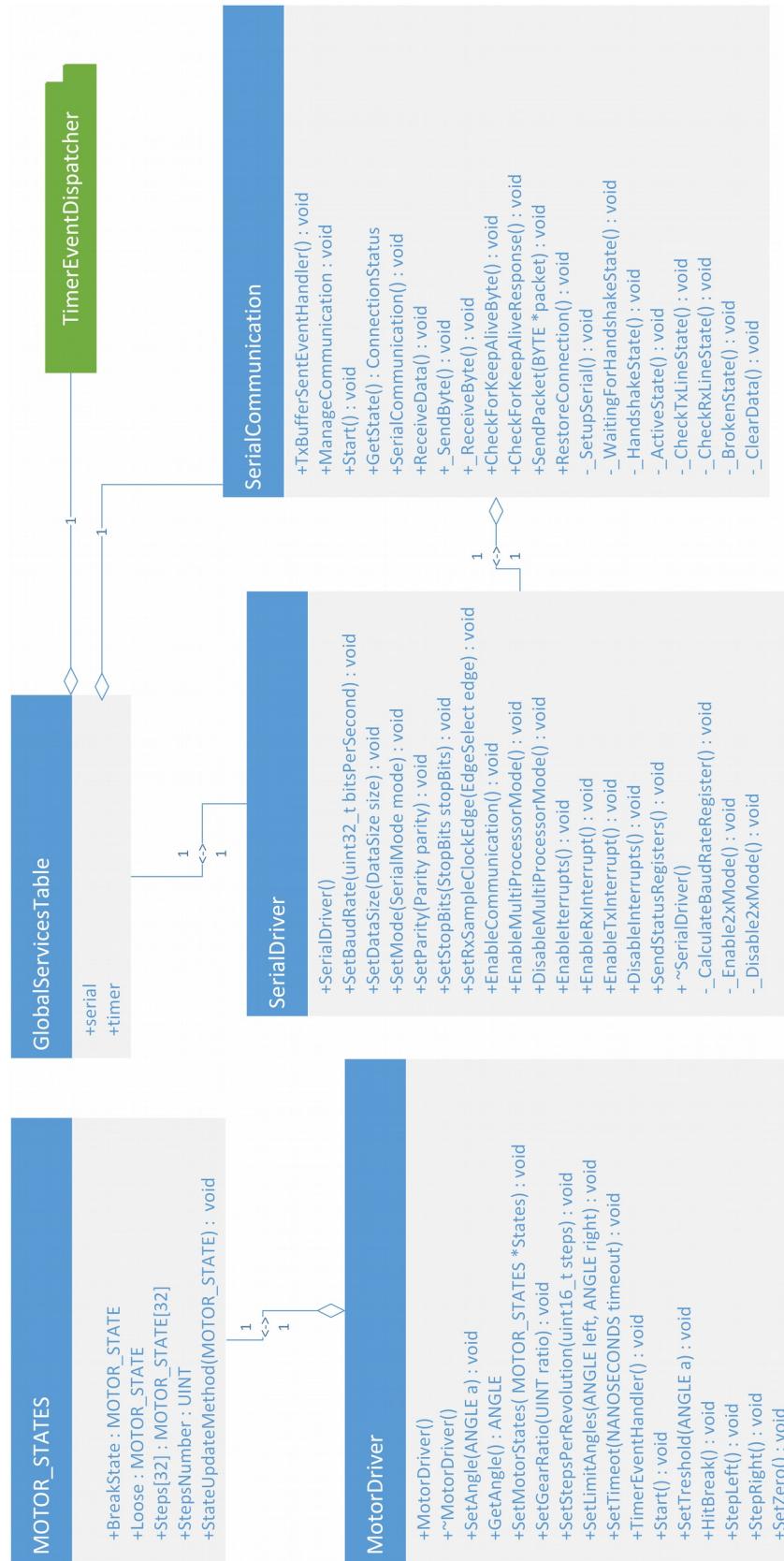
4.9.1. Technologia oraz filozofia wykonania

Zadecydowano o zrealizowaniu oprogramowania mikrokontrolera w języku C++ oraz wykorzystaniu kompilatora GNU/Avr rozpowszechnianym jako zintegrowany element środowiska AVR Studio 6.1.

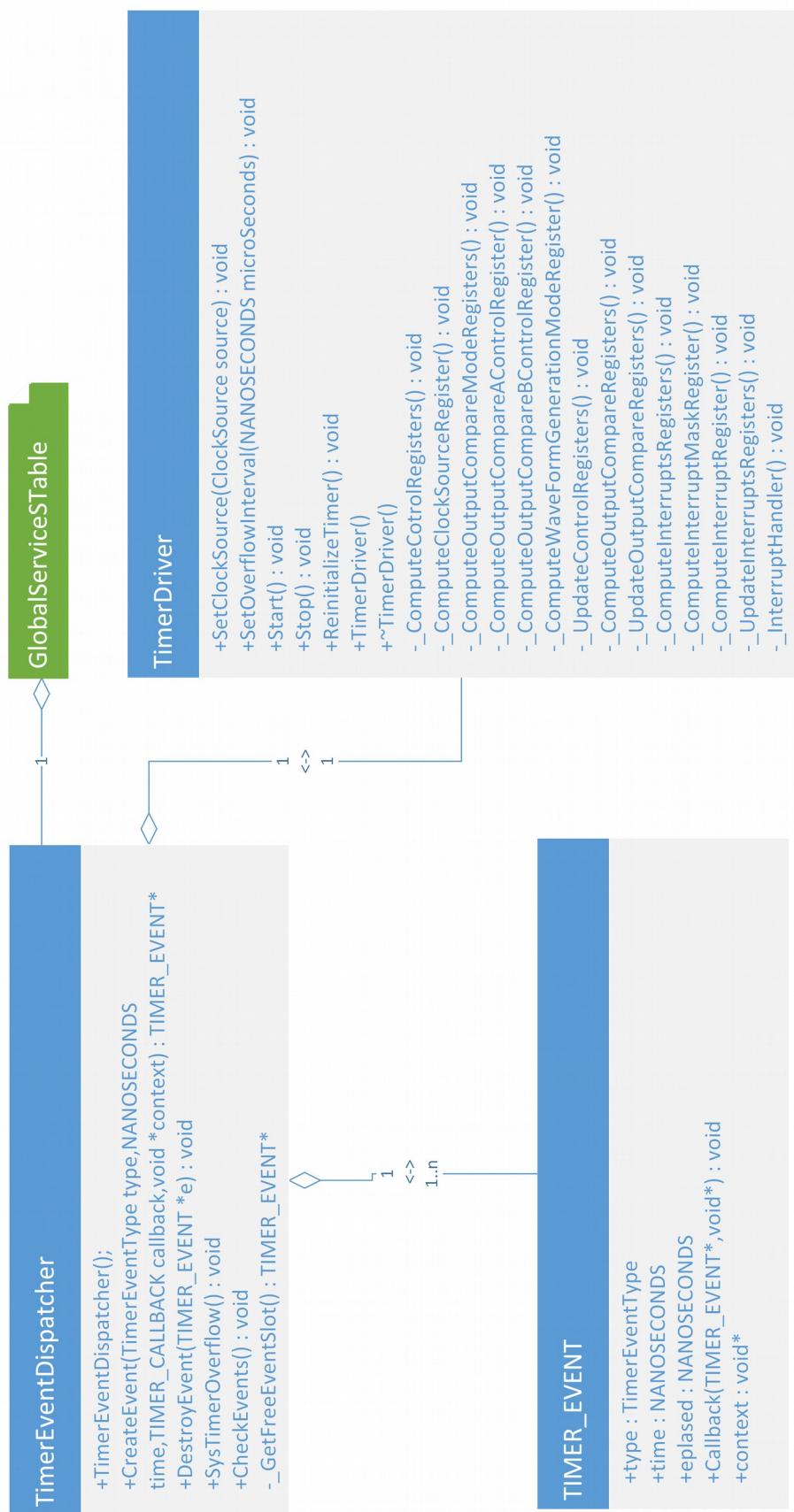
Filozofia tworzenia bibliotek zakładała, iż klasyczne podejście do programowania mikrokontrolerów polegające na ostrożnym zarządzaniu mocą obliczeniową i wszelkimi zasobami nie ma tutaj najmniejszego sensu. Założono, iż w wypadku, gdy układ scalony nie podoła stawianym mu zadaniom, koszt jego wymiany na nowy, spełniający wysokie wymagania jest pomijalnie mały. Z tego właśnie powodu w czasie planowania oraz samych prac projektowych przedkładano wygodę programisty oraz czytelność kodu ponad ekstremalne formy optymalizacji.

4.9.2. Struktura bibliotek

Diagramy oznaczone numerami 4.56 i 4.57 prezentują strukturę klas stworzonego oprogramowania mikrokontrolera. Ze względu na czytelność schematu zrezygnowano z zamieszczania w nim większości pól, które w zdecydowanej większości są zewnętrznie reprezentowane przez publiczne metody Get*/Set*.



Rys. 4.56 Diagram klas bibliotek mikrokontrolera, część pierwsza



Rys. 4.57 Diagram klas bibliotek mikrokontrolera, część 2

4.9.3. Biblioteka nagłówkowa Standarts.h

Plik ten zawiera zestaw typów wyliczeniowych, aliasów typów wbudowanych oraz struktur będący podstawą dla pozostałych bibliotek. Poniższa tabela jest wykazem najważniejszych umieszczonych tam definicji.

Tabela 4.12. Najważniejsze typy zdefiniowane w bibliotece Standarts.h

Nazwa	Definicja	Opis
MOTOR_STATE	uint8_t	Typ używany jako pole do zapisu stanu portu mikrokontrolera skorelowanego z konkretnym stanem silnika podłączonego do tego portu
NANOSECONDS	uint32_t	Typ używany do obliczeń bazujących na czasie
BYTE	uint8_t	Alias stworzony dla czytelności kodu
UINT	uint8_t	Liczba całkowita bez znaku o rozmiarze natywnym dla aktualnej platformy
ANGLE	int32_t	Typ używany do obliczeń związanych z kątami wychylenia – w tym projekcie jest typem wykorzystywanym przez funkcje powiązane z zadaniem sterowania silnikiem krokowym
PrescalerValues	Enum { NoPrescaling=1, TwoToThePowerOfThree, TwoToThePowerOfSix TwoToThePowerOfEight, TwoToThePowerOfTen };	Typ wyliczeniowy wykorzystywany do ustawienia parametrów preskalera zintegrowanych liczników w mikrokontrolerze
Status	enum { Success, Fail, OutOfResources, UnknownError, InvalidParameter };	Status zwracany przez niektóre funkcje celem weryfikacji wyników operacji oraz identyfikacji przyczyn ewentualnych błędów
ConnectionStatus	enum { Stopped, WaitingForHandshake, Handshaking, Active, Broken, WaitingForKeepAliveResponse, WaitingForKeepAliveByte };	Typ używany do identyfikacji stanu w którym znajduje się obiekt odpowiedzialny za komunikację z innymi urządzeniami. Wykorzystywany przez klasę SerialConnection
TIMER_CALLBACK	typedef void (*TIMER_CALLBACK)(TIMER_EVENT*,void*)	Funkcja zwrotna używana przez strukturę TIMER_EVENT, wywoływana, gdy zdarzenie przejdzie w stan aktywny
TIMER_EVENT	struct{ TimerEventType type; NANOSECONDS time; NANOSECONDS eplased; TIMER_CALLBACK Callback; void *context; }	Struktura reprezentująca zdarzenie licznika. Przechowuje funkcję zwrotną, kontekst funkcji zwrotnej, informacje o oczekiwany czasie wywołania funkcji zwrotnej oraz o typie zdarzenia
TimerEventType	enum{ Closed=0, Interval, Timeout, ExternalInteval };	Typ wyliczeniowy używany do oznaczania typu zdarzenia licznika. Dostępne rodzaje: <ul style="list-style-type: none">• Closed – zdarzenie nie aktywne• Interval – zdarzenie cykliczne• Timeout – zdarzenie jednorazowe• ExternalInteval – zdarzenie cykliczne o niskim priorytecie wywoływanie bez blokowania globalnego wektora przerwań.

Dodatkowo zdefiniowano tutaj zestaw podstawowych makr, często używanych w programowaniu niskopoziomowym:

```
#define CP_BITS(source,destination,mask) ((destination) & (~(mask))) | ((source)&(mask));
#define SET(byte,mask) (byte)|(mask)
#define CLR(byte,mask) (byte)&(~(mask))
#define TOGGLE(byte,mask) (byte)^~(mask)
#define NUMBER2MASK(number) (0x1 << (number))
#define BIT_VAL(byte,bitNumber) (((1<<(bitNumber))&(byte))!=0)
#define MICROSECONDS_PERIOD(number)      (NANOSECONDS)1000      *
(NANOSECONDS)number
#define MILISECONDS_PERIOD(number)      (NANOSECONDS)1000000      *
(NANOSECONDS)number
#define SECONDS_PERIOD(number)          (NANOSECONDS)1000000000      *
(NANOSECONDS)number
```

4.9.4. Klasa SerialDriver

Klasa SerialDriver odpowiedzialna jest za kontrolę działania zintegrowanego w układzie Atmega podzespołu USART. Umożliwia ona ustawienie typu połączenia – synchroniczne/asynchroniczne, zbocza aktywnego zegara w trybie synchronicznym, rozmiaru słowa, kontroli parzystości, aktywowanie/dezaktywowanie przerwań oraz zadawanie oczekiwanej szybkości transmisji. Poniżej znajduje się deklaracja publicznych metod tej klasy. Cały kod powstał w oparciu o rozdział 22.9 noty katalogowej układu Atmega88[46].

```
class SerialDriver
{
public:
    SerialDriver();
    void SetBaudRate(uint32_t bitsPerSecond);
    void SetDataSize(DataSize size);
    void SetMode(SerialMode mode);
    void SetParity(Parity parity);
    void SetStopBits(StopBits stopBits);
    void SetRxSampleClockEdge(EdgeSelect edge);
    void EnableCommunication();
    void EnableMultiProcessorMode();
    void DisableMultiProcessorMode();
    void EnableInterrupts();
    void EnableRxInterrupt();
    void EnableTxInterrupt();
    void DisableInterrupts();
    void SendStatusRegisters();
    ~SerialDriver();
    BYTE _controlRegisters[3];
```

};

Implementacja tej klasy zgodnie z zaleceniami projektowymi dla mikrokontrolerów firmy Atmel zarządza błędem wyznaczenia szybkości transmisji i w razie potrzeby wymusza na układzie wykorzystanie podwójnej precyzyji licznika generującego sygnał zegarowy dla układu USART. Proces decyzyjny wygląda następująco[46]:

1. Oblicz wartość rejestru konfiguracyjnego UBRR0 według następującego wzoru:

$$UBRR0 = \frac{F_{CPU}}{16 * desiredBaud} - 1 \quad (4.0)$$

Gdzie:

F_{CPU} – aktualna częstotliwość taktowania procesora w Hz.
 $desiredBaud$ – zadana szybkość transmisji.

2. Oblicz uzyskaną, rzeczywistą szybkość transmisji $calculatedBaud$:

$$calculatedBaud = \frac{F_{CPU}}{16(UBRR0+1)} \quad (4.0)$$

Gdzie:

$calculatedBaud$ – rzeczywiście uzyskana szybkość transmisji

Ze względu na błędy zaokrągleń i skończony rozmiar rejestru UBRR0 będzie ona różna od szybkości zadanej.

3. Oblicz błąd:

$$err = \left| \frac{desiredBaud - calculatedBaud}{desiredBaud} \right| * 100 \% \quad (4.0)$$

4. Jeśli błąd nie przekracza 2 % zakończ.

5. Wymuś użycie podwójnej precyzyji i ponownie oblicz wartość rejestru UBRR0 z uwzględnieniem tego faktu:

$$UBRR0 = \frac{F_{CPU}}{8 * desiredBaud} \quad (4.0)$$

Korzystanie z tej klasy polega na stworzeniu obiektu, wywołaniu sekwencji metod Set*/Enable*/Disable* powodujących zmianę tylko i wyłącznie stanu wewnętrznego obiektu, zakończonej wywołaniem metody **SendStatusRegisters()**, która dokonuje aktualizacji rejestrów kontrolujących pracę układu USART. Operacja ta powoduje natychmiastowe uruchomienie/deaktywację odpowiednich podzespołów oraz przerwań. Całą sekwencję można powtarzać wielokrotnie w celu zmiany zachowania kontrolowanego podzespołu.

4.9.5. Klasa SerialCommunication

Klasa SerialCommunication jest programową implementacją zaprojektowanego protokołu komunikacyjnego zdefiniowanego w rozdziale 4.4.4 wykorzystującą obiekt SerialDriver do kontroli sposobu działania modułu USART. Poniżej zamieszczono deklaracje publicznych metod tego modelu:

```
typedef void (*TX_BUFFER_SENT)();  
class SerialCommunication{  
public:  
    TX_BUFFER_SENT TxBUFFER_SENTEventHandler;  
    SerialCommunication();  
    Status Start();  
    ConnectionStatus GetState();  
    void ManageCommunication(TIMER_EVENT *e, void *c);
```

```

        void ReceiveData();
        void _SendByte();
        void _ReceiveByte();
        void CheckForKeepAliveByte();
        void CheckForKeepAliveResponse();
        void SendPacket(BYTE *packet);
        void RestoreConnection();
};


```

Typ `TX_BUFFER_SENT` jest definicją funkcji wywoływanej przez klasę w momencie nadania przez nią pełnego pakietu. Jej zadaniem jest pobranie/wygenerowanie następnych danych do wysłania oraz przekazanie ich metodzie `SendPacket`. Rozmiar pakietu jest ustalony w czasie komplikacji na 10 bajtów.

Metoda `ManageCommunication`, wywoływana co każde 100 ms, jest maszyną stanów kontrolującą połączenie, a wykorzystującą do tego celu typ wyliczeniowy `ConnectionStatus`, którego wartości odpowiadają fazom protokołu komunikacyjnego według następującego klucza:

- Faza oczekiwania na połączenie, faza wykrywania połączenia 1 – `WaitingForHandshake`
- Faza wykrywania połączenia 2, przerwa 500 ms, faza kontroli poprawności połączenia, przerwa 100 ms – `Handshaking`
- Faza komunikacji – `Active`, `WaitingForKeepAliveResponse`, `WaitingForKeepAliveByte`
- Faza zerwanego połączenia – `Broken`

Metoda `GetState` może być wykorzystana do wykrycia zerwanego połączenia i ponownego jego zestawienia. Poniższy fragment kodu wykorzystany w projekcie prezentuje takie zachowanie:

```

int main(void)
{
    SetupPorts();
    InitGlobalServicesTable();
    SetUpMotors();
    //pętla, której każda następna iteracja następuje tylko w wypadku
    //zerwania połączenia
    while(true){
        //inicjalizacja protokołu
        gST.serial->Start();
        //przekazanie uchwytu do funkcji generującej pakiety do nadania
        gST.serial->TxBufferSentEventHandler = SendHeadStatus;
        //wymuszenie generacji pierwszego pakietu
        SendHeadStatus();
        //główna pętla programu, wykonuje się aż do momentu zerwania połączenia
    }
}

```

```

        while(gST.serial->GetState() != Broken){
            TheLoop();
        }
        //zerowanie parametrów układu
        horizontalMotor.SetAngle(0);
        verticalMotor.SetAngle(0);
        _delay_ms(3000);
    }
}

```

Metoda `ReceiveData()` jest odpowiedzialna za sprawdzenie dostępności danych w buforze odbiorczym UDR0, ich odczyt, formowanie pakietów oraz wywoływanie funkcji `TxBufferSentEventHandler` przekazującej odebrane pakiety do przetwarzania przez odpowiedni moduł. Wywołanie tej operacji ma miejsce w głównej pętli programu, nie zaś w ramach przerwania `ISR(USART_RX_vect)`. Rozwiążanie to, chociaż ogranicza szybkość komunikacji układu, pozwala wykorzystywać skomplikowane/czasochłonne funkcje obsługi danych odebranych, bez blokowania globalnego wektora przerwań.

4.9.6. Klasa *TimerDriver*

Typ `TimerDriver` został stworzony w celu kontroli zachowania zintegrowanego układu ośmiobitowego licznika. Poniżej zamieszczono definicję tej klasy z pominięciem pól prywatnych:

```

class TimerDriver{
public:
    void SetClockSource(ClockSource source);
    void SetOverflowInterval(NANOSECONDS microSeconds);
    void Start();
    void Stop();
    void ReinitializeTimer();
    TimerDriver();
    ~TimerDriver();
private:
    void _ComputeCotrolRegisters();
    void _ComputeClockSourceRegister();
    void _ComputeOutputCompareModeRegisters();
    void _ComputeOutputCompareAControlRegister();
    void _ComputeOutputCompareBControlRegister();
    void _ComputeWaveFormGenerationModeRegister();
    void _UpdateControlRegisters();
    void _ComputeOutputCompareRegisters();
    void _UpdateOutputCompareRegisters();
    void _ComputeInterruptsRegisters();
}

```

```

    void _ComputeInterruptMaskRegister();
    void _ComputeInterruptRegister();
    void _UpdateInterruptsRegisters();
    void _InterruptHandler();
};


```

Metody `Start()` oraz `ReinitializeTimer()` odpowiedzialne są odpowiednio za wystartowanie licznika oraz zmianę jego parametrów pracy. Wykorzystuję one zestaw metod prywatnych `_Compute*Register()` przetwarzających stan wewnętrzny obiektu, czyli przede wszystkim interwału między kolejnymi przerwaniami, na wartości konfiguracji licznika. Obliczone w ten sposób wartości są następnie przenoszone do odpowiednich rejestrów za pomocą metod `_Update*Registers()`.

Należy tutaj zauważyć, iż logika zawarta w klasie kontroluje również preskaler sygnału zegarowego na wejściu licznika. Algorytm doboru parametrów tego podukładu został zaprojektowany w taki sposób, by wybrana konfiguracja powodowała minimalne odchylenie interwału przerwań otrzymanego względem interwału zadanego.

Celem stworzenia omawianego typu jest implementacja prostego sterownika licznika, umożliwiającego za pomocą trzech instrukcji uruchomienia przerwań skorelowanych z tym licznikiem uruchamianych z określonym interwałem czasowym. Poniższy kod pochodzący z konstruktora klasy `TimerEventDispatcher` prezentuje takie podejście:

```

TimerDriver _driver;
_driver = TimerDriver();
_driver.SetClockSource(Internal);
_driver.SetOverflowInterval(_sysTimerTimeout);
_driver.Start();

```

Powyższy zestaw instrukcji powoduje sygnalizowanie przepelnienie licznika co każde `_sysTimerTimeout` mikrosekund. Użytkownik klasy `TimerDriver`, powinien samodzielnie zaimplementować obsługę tego przerwania, np. poprzez wykorzystanie makra `ISR(TIMER0_COMPA_vect) { . . . }`.

4.9.7. Klasa `TimerEventDispatcher` oraz struktura `TIMER_EVENT`

Klasa `TimerEventDispatcher` jest typem wykorzystującym obiekt typu `TimerDriver`, a stworzonym celem zarządzania zadaniami wymagającymi kontroli zależności czasowych. Realizowane jest to przez udostępnienie zestawu metod pozwalających na wywoływanie zadanych funkcji w trybie cyklicznym z określonym interwałem lub jednorazowe po określonym czasie. Informacje o tych funkcjach przechowywane są w strukturach `TIMER_EVENT`. Poniżej zamieszczono definicje obydwóch typów:

```

class TimerEventDispatcher{
public:
    TimerEventDispatcher();
    TIMER_EVENT *CreateEvent( TimerEventType type, NANoseconds time,
    TIMER_CALLBACK callback, void *context );

```

```

    void DestroyEvent(TIMER_EVENT *e);
    void SysTimerOverflow();
    void CheckEvents();

private:
    TimerDriver _driver;
    UINT _eventListLength = TIMER_EVENTS_LIST_LENGTH;
    NANoseconds _sysTimerTimeout = TIMER_TIMEOUT;
    TIMER_EVENT _events[TIMER_EVENTS_LIST_LENGTH];
    TIMER_EVENT* _GetFreeEventSlot();

};

typedef struct _TIMER_EVENT{
    TimerEventType type;
    NANoseconds time;
    NANoseconds eplased;
    TIMER_CALLBACK Callback;
    void *context;
}TIMER_EVENT;

```

Struktura `TIMER_EVENT` służy do przechowywania informacji o zdarzeniu licznika, tzn. czas przypisany do tego zdarzenia, jego typ, czas do najbliższego przejścia w stan aktywny oraz funkcji zwrotnej wywoływanej w momencie przejścia w stan aktywny wraz z jej argumentem `*context`.

Konstruktor klasy `TimerEventDispatcher` jest odpowiedzialny za wykorzystanie implementacji typu `TimerDriver` do zainicjalizowania zintegrowanego układu licznika do generacji przerwań co każde `_sysTimerTimeout` mikrosekund.

Metoda `CreateEvent()` odpowiedzialna jest za alokowanie przestrzeni potrzebnej dla struktury `TIMER_EVENT` wypełnionej wartościami przekazanymi jako argumenty wywołania oraz dodania do jej listy zdarzeń oczekujących na przejście w stan aktywny.

Metoda `SysTimerOverflow()` przewidziana jest jako instrukcja obsługująca przerwanie `TIMER0_COMPA_vect`. Zadaniem tej funkcji jest aktualizacja stanów obiektów typu `TIMER_EVENT` zgromadzonych w tablicy `_events` celem zmniejszenia zmiennej `eplased`. Jeśli wartość po aktualizacji jest równa zero, a zdarzenie jest typu `Interval` lub `Timeout`, następuje wywołanie przypisanej do niej funkcji zwrotnej, po której następuje aktualizacja stanu obiektu zdarzenia – dla zdarzeń cyklicznych następuje zrestartowanie wartości pola `eplased`, zaś w przypadku zdarzeń jednorazowych typ zostaje zmieniony na `Closed`. Jako że funkcja jest wywoływana w ramach obsługi przerwania powoduje ona zablokowanie globalnego wektora przerwań.

Metoda `CheckEvents()` zaprojektowana została celem wywoływania jej w głównej pętli programu. Jej zadaniem jest sprawdzenie czy istnieją zdarzenia typu `ExternalInterval` gotowe do aktywowania, tzn. charakteryzowane przez pole `eplased` mniejsze lub równe `_sysTimerTimeout`. Jeśli tak, funkcja zwrotna zostaje uruchomiona, a zawartość struktury

zdarzenia zostaje zaktualizowana adekwatnie do jej typu. Rozwiążanie to pozwala na obsługę skomplikowanych/czasochłonnych instrukcji bez blokowania globalnego wektora przerwań.

4.9.8. Struktura GLOBAL_SERVICES_TABLE

Typ używany do przechowywania wskaźników do implementacji klas `TimerEventDispatcher` oraz `SerialCommunication`. Wykorzystywany jest do tworzenia globalnego obiektu `gST`, który ma służyć za wygodny skrót do najczęściej używanych funkcji, a którego inicjalizacja jest dokonywana za pomocą funkcji `InitGlobalServicesTable()`.

4.9.9. Klasa MotorDriver oraz struktura MOTOR_STATES

Dwa wymienione w tytule typy zostały zaprojektowane celem implementacji programowego sterownika silników krokowych podłączonych do mikrokontrolera. Poniżej znajdują się ich definicje zawierające publiczne metody oraz najważniejsze pola:

```
typedef void (*MOTOR_STATE_UPDATE_METHOD)(UINT state);

typedef struct _MOTOR_STATES{
    MOTOR_STATE BreakState;
    MOTOR_STATE Loose;
    MOTOR_STATE Steps[32];
    UINT StepsNumber;
    MOTOR_STATE_UPDATE_METHOD StateUpdateMethod;
} MOTOR_STATES;

class MotorDriver
{
    void SetAngle(ANGLE a);
    ANGLE GetAngle();
    void SetMotorStates( MOTOR_STATES *States);
    void SetGearRatio(UINT ratio);
    void SetStepsPerRevolution(uint16_t steps);
    void SetLimitAngles(ANGLE left, ANGLE right);
    void SetTimeot(NANOSECONDS timeout);
    void TimerEventHandler();
    void Start();
    void SetTreshold(ANGLE a);
    void HitBreak();
    void StepLeft();
    void StepRight();
    void SetZero();
};

;
```

Obiekty typu `MOTOR_STATES` mają za zadanie przechowywać podstawowe informacje niezbędne do kontrolowania konkretnego silnika podłączonego do mikrokontrolera. Są tutaj

takie informacje jak: lista kolejnych, możliwych do zadania stanów logicznych na wyprowadzenia sterujące, liczba tych stanów, stany hamowania i bieg „luźny” (patrz: 4.4.2 Mostek sterowniczy) oraz funkcja przekazująca zadane stany na wyprowadzenia mikrokontrolera.

Poniżej znajduje się lista metod klasy `MotorDriver` wraz z krótkimi ich opisami:

- `SetAngle` – zadaje sterownikowi kąt wychylenia rotora, do którego ma dążyć.
- `GetAngle` – pobiera aktualny kąt wychylenia rotora.
- `SetMotorStates` – przyjmuje i zapisuje strukturę typu `MOTOR_STATES`.
- `SetGearRatio`, `SetStepsPerRevolution` – metody mające ułatwić sterowanie silnikami ze zintegrowanymi układami przekładni. Nie zostały poprawnie zaimplementowane.
- `SetLimitAngles` – określa maksymalne kąty wychylenia rotora w obydwu kierunkach.
- `SetTimeot` – określa interwał z jakim sygnały sterujące mogą się zmieniać.
- `TimerEventHandler` – metoda używana jako funkcja zwrotna w obiekcie typu `TIMER_EVENT`.
- `Start` – rozpoczyna kontrolowanie silnika. Wykorzystuje `gST.timer->CreateEvent()` (patrz 4.9.8: Struktura `GLOBAL_SERVICES_TABLE`).
- `SetTreshold` – ustanawia maksymalną różnicę kątów między kątem rzeczywistym a zadanym, na którą sterownik nie reaguje. Wywołanie z argumentem zerowym powoduje wyłączenie funkcjonalności.
- `HitBreak` – powoduje przejście silnika w stan hamowania. Funkcja nie powinna być wykorzystywana z zewnątrz obiektu.
- `StepLeft`, `StepRight` – powodują przesunięcie kątowe rotora o jeden krok w zadanym kierunku. Funkcje nie powinny być wykorzystywane z zewnątrz obiektu.
- `SetZero` – powoduje zinterpretowanie aktualnego kąta wychylenia rotora jako kąta odniesienia.

4.9.10. Plik `AtmegaRom.c`

Główny plik źródłowy zawierający funkcję `main()` oraz całą logikę spinającą opisane wcześniej biblioteki w użyteczną całość. Znajdują się w nim funkcje inicjalizujące: `SetupPorts` – inicjuje rejesty kontrolne cyfrowych portów wejścia/wyjścia, `setUpMotors`, która inicjalizuje obiekty sterujące silnikami głowicy oraz główna pętla programu `TheLoop()`.

4.9.11. API komunikacyjne na komputer PC

Realizacja projektu wymagała stworzenia zestawu bibliotek umożliwiających nawiązanie i utrzymanie wiarygodnego połączenia między komputerem, a kontrolerem głowicy oraz przesyłanie w obie strony danych z szybkością akceptowlaną dla użytkownika głowicy. Zadanie to zrealizowano za pomocą środowiska .NET oraz języka programowania C#.

4.9.12. Struktura bibliotek

Biblioteki zostały stworzone i udostępnione jako projekt programu Visual Studio 2013 zawierający trzy solucje:

- ArCameraHeadApi – implementacja api komunikacyjnego
- ArCameraHeadApiTester – przykładowy program prezentujący działanie API.
- DebugLib – zestaw narzędzi pomocniczych wykorzystywanych w procesie tworzenia aplikacji.

Struktura klas omawianych modułów została umieszczona na schematach oznaczonych numerami 3.2 oraz 3.3.

4.9.13. API komunikacyjne – klasa ArCameraHeadApi oraz struktura Connection

Z punktu widzenia programisty integrującego głowicę do swojego projektu ważne są jedynie klasy [ArCameraHeadApi](#) oraz [Connection](#) będące implementacją API zdefiniowanego w rozdziale 3.5. Typ [ArCameraHeadApi](#) udostępnia następujące metody:

- GetAvailableConnectionsList – statyczna metoda służąca do pobrania listy dostępnych połączeń. Zwrotna lista zawiera obiekty dziedziczące po abstrakcyjnej klasie [Connection](#).
- Konstruktor – za argument przyjmuje obiekt typu [Connection](#), który jest wykorzystywany do automatycznego zestawienia połączenia na danym kanale.
- SetVerticalAngle, SetHorizontalAngle – metody służące do zadania żądanego kąta wychylenia głowicy. Za argument przyjmują liczbę z zakresu od $-\pi$ do $+\pi$. Wartości spoza zakresu są obcinane.
- GetCurrentHorizontalAngle, GetCurrentVerticalAngle – metody służące do odczytu aktualnego kąta wychylenia głowicy. Zwracają liczbę z zakresu od $-\pi$ do $+\pi$.

Typ [Connection](#) jest klasą abstrakcyjną, pomyślaną jako szkielet do tworzenia klas implementujących rzeczywiste połączenia. Zdefiniowany został w sposób następujący:

```
public abstract class Connection
{
    public readonly string name;
    public readonly string description;
    public Connection(string name, string description)
    {
        this.name = name;
        this.description = description;
    }

    abstract internal int _GetTxBufferSize();
    abstract internal void Start();
    abstract internal void Disconnect();
    internal void SendData(byte[] data)
```

```

    {
        if (isActive)
            _sendData(data);
        else
            throw new ConnectionDisabled();
    }
    public abstract byte[] ReadData();
    abstract internal void _sendData(byte[] data);
    abstract public bool isActive { get; }
}

```

Jak widać jest to bardzo ogólna definicja umożliwiająca implementację niemal dowolnego typu połączenia. Z punktu widzenia programisty implementującego zaprojektowane API do swojego projektu ważne są tylko pola `name` oraz `description` zawierające informację na temat obiektu w formie czytelnej dla użytkownika aplikacji. Może to być np. para wartości `{"COM3", „Zestaw połączenia z wykorzystaniem portu komunikacji asynchronicznej”}`.

Programista tworzący klasę implementującą nowy rodzaj połączenia musi zaimplementować wyżej zdefiniowane metody w sposób następujący:

- `Start()` – metoda ma za zadanie zestawienie połączenia.
- `Disconnect()` – metoda komplementarna do `Start()`
- `_sendData(byte[] data)` – metoda dodająca dane do bufora nadawczego
- `ReadData()` – metoda zwracająca najstarszy odebrany pakiet z bufora odbiorczego
- `isActive` – pole tylko do odczytu zwracające informację o tym, czy połączenie jest zestawione

4.9.14. Klasa DummyConnection

Implementacja klasy abstrakcyjnej `Connection` stworzona w celach testowych. Zestawia wirtualne połączenie typu echo z nieistniejącą głowicą kamerową. Pozwala na testowanie aplikacji korzystającej z API bez dostępu do sprzętu.

4.9.15. Klasa ComConnection

Implementacja klasy abstrakcyjnej `Connection`. Implementuje zdefiniowany w rozdziale 4.4.4 protokół komunikacyjny w celu zestawienia wiarygodnego połączenia między kontrolerem głowicy a komputerem PC.

4.9.16. Plik Exceptions.cs

Plik ten zawiera listę wyjątków możliwych do wygenerowania przez API. Zawiera on następujące definicje klas:

```

class ConnectionOpenException : Exception { }
class HandshakeException : Exception { }

```

```
class ConnectionDisabled : Exception { }
class BrokenConnection : Exception { }
```

4.9.17. Klasa ArCameraHeadData

Klasa ta służy do przechowywania oraz konwersji do/z formatu zgodnego ze specyfikacją protokołu komunikacyjnego danych o zadanym/ aktualnym stanie głowicy. Poniżej zamieszczono definicję omawianego typu.

```
class ArCameraHeadData
{
    public static readonly int serializedDataSize = 10;
    private double _horizontalAngle;
    private double _verticalAngle;
    public double GetHorizontalAngle();
    public double GetVerticalAngle();
    public void SetVerticalAngle(double angle);
    public void SetHorizontalAngle(double angle);
    public void Unserialize(byte[] data);
    private void DeserializeAngles(byte[] data);
    explicit operator byte[](ArCameraHeadData data);
    byte static void SerializeAngles(ArCameraHeadData data, byte[]
byteArr);
    private static byte[] SerializeAngle(double angle);
}
```

4.9.18. Korekcja pozycji zerowej oraz ręczna kontrola głowicy.

Kontroler głowicy podczas włączania zakłada iż kamery znajdują się w pozycji zero – tzn. są skierowane do przodu, a ich wektor normalny jest równoległy do powierzchni ziemi. Z różnych względów nie zawsze jest to prawda, dlatego też przewidziano możliwość ręcznej korekcji tego błędu. Dokonano tego przez wyprowadzenie klawiatury składającej się z sześciu przycisków służących do: obracania kamerami w góre/dół, lewo/prawo, ustawiania aktualnego wychylenia jako pozycji zerowej oraz wymuszenia powrotu do aktualnie zapamiętanej pozycji zero.

Opisywana metoda kontroli może również służyć do ręcznego sterowania głowicą kamerową bez dostępu do komputera PC oraz/lub OculusRift.

4.10. Wymagania programowe

Z uwagi na współpracę z dużą ilością niezależnych urządzeń, nasz projekt korzysta z wielu bibliotek, sterowników programowych oraz innych. Poniżej przedstawiamy listę oprogramowania niezbędnego do poprawnego działania naszego programu:

- System operacyjny Windows 7 lub nowszy,
- Oculus Rift Runtime,
- CL Eye Driver,

- Microsoft .NET Framework 4.5,
- Microsoft Visual C++ 2012 Redistributable Package (x86).

5. TESTOWANIE

5.1. Założenia

Zakres testowania, który zamierzamy pokryć to jedynie testy akceptacyjne pomagające stwierdzić czy udało nam się stworzyć produkt o odpowiedniej, zakładanej przez nas jakości. Nasz prototyp nie był projektowany jako rozwiązanie komercyjne ani system czasu rzeczywistego, dlatego na opóźnienie jego działania ma wpływ duża liczba czynników, których minimalizacja nie była zadaniem krytycznym dla realizacji tego projektu, w tym:

- szybkość przekazania obrazu z kamer do komputera,
- szybkość przetwarzania obrazu,
- szybkość przekazania obrazu na ekran hełmu wirtualnego,
- opóźnienia wątków nadawczo–odbiorczych na porcie szeregowym,
- opóźnienia działania silników krokowych,
- stopień precyzji działania elementów mechanicznych.

W związku z tym faktem chcemy skupić się na subiektywnym odczuciu użytkownika, czy system jako całość działa dla niego dostatecznie szybko, czyli czy opóźnienie działania systemu jest na tyle małe, że nie przeszkadza w jego normalnym wykorzystywaniu. W przypadku części graficznej postanowiliśmy przeprowadzić test na trzech platformach testowych w celu sprawdzenia w jakim stopniu moc obliczeniowa sprzętu PC ma wpływ na szybkość przetwarzania obrazu otrzymywanego z kamer.

5.2. Ustalenie wymagań sprzętowych

Część graficzna została przetestowana w sumie przez kilkanaście osób podzielonych na zespoły oraz jednostki już na etapie implementacyjnym (Rys. 4.8.). Wyniki testów były zadowalające – szybkość działania, jakość obrazu, predefiniowane ustawienia soczewek oraz subiektywne odczucia z korzystania z projektu zostały określone jako bardzo dobre. Wszystkie wyżej wymienione testy były przeprowadzone na platformie testowej opisanej w punkcie 5.2.1. i to ona będzie naszym punktem odniesienia.

Wykonywane przez nas obliczenia obciążają przede wszystkim procesor, ale powinniśmy zadbać również o to, by karta graficzna nie była wąskim gardłem przy przesyłaniu przetworzonego już obrazu na ekran hełmu rzeczywistości wirtualnej. Aby ustalić minimalne oraz zalecane wymagania sprzętowe sprawdziliśmy działanie naszego projektu na kilku dostępnych nam komputerach o zróżnicowanych parametrach. Poniżej opiszemy nasze spostrzeżenia dotyczące kilku wybranych przypadków.

5.2.1. Platforma 1 – laptop HP EliteBook 8540w

Pierwszą z testowanych platform był niewielki komputer przenośny klasy notebook. Na jego pokładzie znajdowały się między innymi:

- Procesor: Intel Core i5–540M:

- o Taktowanie: 2,53 GHz,
- o Liczba rdzeni / wątków: 2 / 4,
- o Pamięć L1 / L2 / L3: 128 KB / 512 KB / 3072 KB.
- 4GB RAM DDR3,
- Karta graficzna: nVidia Quadro FX 880M.

Powyższa konfiguracja pozwoliła na sprawne oraz komfortowe korzystanie z systemu. Opóźnienia części graficznej utrzymywały się na niskim, zadowalającym poziomie, a przesyłany w czasie rzeczywistym obraz osiągał maksymalne 60 FPS (ang. frames per second – klatek na sekundę). Obraz był płynny niezależnie od działania wyłącznie części graficznej, czy też całego projektu.

5.2.2. Platforma 2 – netbook Acer AO722

Kolejną testowaną platformą był niewielki komputer przenośny klasy netbook. Na jego parametry techniczne składały się między innymi:

- Procesor: AMD C-60:
 - o Taktowanie: 1,33 GHz,
 - o Liczba rdzeni / wątków: 2 / 2,
 - o Pamięć L1 / L2 / L3: 128 KB / 1024 KB / brak.
- 4GB RAM DDR3,
- Karta graficzna: AMD Radeon HD 6290.
 - o Pamięć karty graficznej współdzielona z systemową.

Na jednostce tej widoczny był spadek wydajności. Testując wyłącznie część graficzną, przesyłany obraz był płynny jedynie przy obsłudze jednej z kamer – przy czym osiągał pełne 60 kl./s wyłącznie bez zastosowania korekcji soczewkowej. Widoczne były również nieznacznie większe opóźnienia niż na platformie 1 (opisywanej w punkcie 5.2.1.). Dodając do układu drugą kamerę, niestety wydajność zauważalnie spadła, a obraz przestał być wyświetlany płynnie. Po uruchomieniu części odpowiadającej za przeliczanie oraz zadawanie kątów serwogłówicy wydajność spadła jeszcze bardziej. Korzystanie z programu było na tej platformie możliwe, jednak nie należało do komfortowych, a po dłuższym czasie mogło być męczące.

5.2.3. Platforma 3 – Laptop Acer Aspire 5740G

Następną platformą testową był niewielki komputer klasy notebook. Jego parametry sprzętowe to:

- Procesor: Intel Core i5-430M:
 - o Taktowanie: 2,26 GHz,
 - o Liczba rdzeni / wątków: 2 / 4,
 - o Pamięć L1 / L2 / L3: 128 KB / 512 KB / 3072 KB.
- 4GB RAM DDR3,
- Karta graficzna: ATI Mobility Radeon HD5650

Jest to konfiguracja trochę słabsza niż platforma 1, a mimo to nie zauważono względem niej spadku wydajności. Obraz jest przetwarzany i wyświetlany płynnie, niezależnie od działania części graficznej projektu, czy całego systemu.

5.2.4. Platforma 4a – podkręcony komputer stacjonarny z dedykowanym układem graficznym

Kolejną z testowanych platform był komputer stacjonarny klasy PC. Na jego pokładzie znajdowały się między innymi:

- Procesor: Intel Core i5 2500K:
 - Taktowanie: 3,3 GHz (podkręcony do 4.8 GHz),
 - Liczba rdzeni / wątków: 4 / 8,
 - Pamięć L1 / L2 / L3: 128 KB / 1024 KB / 6MB.
- 16GB RAM DDR3,
 - 4x4GB Quad Channel, 1600Mhz, 9–9–9–27.
- Karta graficzna: AMD Radeon HD 6850.

Jest to konfiguracja zdecydowanie mocniejsza od platformy 1 (opisywanej w punkcie 5.2.1.). Różnica w wydajności jest jednak minimalna – opóźnienia są nieznacznie mniejsze, ale nie ma to żadnego realnego ani widocznego wpływu na odczucie z użytkowania naszego projektu.

5.2.5. Platforma 4b – komputer stacjonarny ze zintegrowanym układem graficznym

Postanowiliśmy zbadać wpływ częstotliwości taktowania procesora oraz zastosowanej karty graficznej na wydajność programu. Sprawdziliśmy więc działanie naszego projektu jeszcze raz na tym samym komputerze po wprowadzeniu kilku zmian. Po pierwsze, zmniejszyliśmy taktowanie procesora na standardowe, a po drugie – zastosowaliśmy zintegrowaną kartę graficzną.

- Procesor: Intel Core i5 2500K:
 - Taktowanie: 3,3 GHz,
 - Liczba rdzeni / wątków: 4 / 8,
 - Pamięć L1 / L2 / L3: 128 KB / 1024 KB / 6MB.
- 16GB RAM DDR3,
 - 4x4GB Quad Channel, 1600Mhz, 9–9–9–27.
- Karta graficzna: Intel HD Graphics 3000.

Nie zaobserwowaliśmy żadnych różnic w wydajności programu w porównaniu do platformy 4a (opisywanej w punkcie 5.2.4.) pomimo zastosowania dużo mniej wydajnej zintegrowanej karty graficznej dla układów Sandy Bridge Intela. Możemy więc założyć, że jest ona w zupełności wystarczająca do naszych zastosowań.

5.3. Wymagania sprzętowe

Wyniki testów na wielu platformach (z wybranymi opisanyimi w punkcie 5.2.) pozwoliły nam sformułować poniższe wymagania sprzętowe (zakładając, że minimalne wymagania oznaczają akceptowalne korzystanie z systemu, a zalecane komfort oraz niskie poziomy opóźnień).

Minimalne wymagania sprzętowe:

- System operacyjny Windows 7 lub nowszy,
- Procesor dwurdzeniowy (4-wątkowy),
- 2GB pamięci RAM,
- Karta graficzna klasy Intel HD Graphics 3000 lub wyższa.

Zalecane wymagania sprzętowe:

- System operacyjny Windows 7 lub nowszy,
- Procesor czterordzeniowy (8-wątkowy),
- 4GB pamięci RAM,
- Karta graficzna klasy Intel HD Graphics 3000 lub wyższa.

Powyższe wymagania zostały dobrane arbitralnie w oparciu o wyniki przeprowadzonych testów. Zdecydowaliśmy się na zdefiniowanie wymagań w zależności od liczby rdzeni oraz wątków (a nie taktowania procesora) z uwagi na specyfikę wykonywanych przez nasz program operacji – uznaliśmy, że właśnie ten parametr będzie miał tutaj największy wpływ. Podana karta graficzna została wyselekcjonowana z uwagi na jej dużą popularność, niską wydajność oraz ważną w naszym wypadku – obsługę dwóch lub więcej ekranów (niedbanych do współpracy w trybie „Extend Desktop to the HMD”.

6. PODSUMOWANIE

6.1. Wnioski

Realizacja naszego projektu przebiegła pomyślnie. Udało nam się stworzyć działający prototyp oraz w pełni go oprogramować spełniając wcześniej ustalone przez nas założenia. Na poziomie “proof-of-concept” system jest w pełni funkcjonalny, a jego szybkość działania przekroczyła oczekiwania zespołu. Moduł mechaniczny obraca się z akceptowalną prędkością w odpowiednim zakresie kątów, a część informatyczna osiąga maksymalną płynność przy relatywnie niskich wymaganiach sprzętowych (dużo mniejszych niż typowy komputer domowy klasy PC).

W trakcie realizacji projektu wystąpił szereg problemów z działaniem serwogłówicy, przez co nie udało się dotrzymać terminów zakładanych w harmonogramie. Ostatecznie udało się im zaradzić i dodatkowo usprawnić funkcjonowanie całego prototypu.

6.2. Rozwój

Nasz projekt ma w sobie potencjał, może stanowić tańsze niż inne dostępne na rynku rozwiązanie teleobecności, które znajdzie zastosowanie w sytuacjach i miejscach dla człowieka niebezpiecznych, bądź fizycznie nieosiągalnych. W celu usprawnienia jego działania, uzyskania większej stabilności oraz poszerzenia zakresu jego zastosowań można wykonać szereg usprawnień:

- Uodpornić prototyp na działanie czynników zewnętrznych poprzez stworzenie bardziej niezawodnej konstrukcji (pełnoprawnej płytce PCB zamiast płytki prototypowej, trwałej obudowy itp.),
- Zastosować lepsze silniki krokowe zmniejszające czas reakcji serwogłówicy na ruchy hełmu rzeczywistości wirtualnej,
- Zapewnić wsparcie dla systemu Linux,
- Zapewnić możliwość zdalnej obsługi serwogłówicy przez sieć Ethernet.

7. WYKAZ LITERATURY

1. The Father Of Virtual Reality <http://www.mortonheilig.com/>, (data dostępu 20.11.2014 r.).
2. The sights and scents of the Sensorama Simulator, <http://www.engadget.com/2014/02/16/morton-heiligs-sensorama-simulator/>, (data dostępu: 2.12.2014 r.).
3. InventorVR, <http://www.mortonheilig.com/InventorVR.html>, (data dostępu: 20.11.2014 r.).
4. Virtual Reality History – HowStuffWorks, <http://electronics.howstuffworks.com/gadgets/other-gadgets/virtual-reality8.htm>, (data dostępu: 2.12.2014 r.).
5. Section 17: Virtual Reality, <https://design.osu.edu/carlson/history/lesson17.html>, (data dostępu 30.12.2014 r.).
6. Oculus Rift Teardown – iFixit, <https://www.ifixit.com/Teardown/Oculus+Rift+Teardown/13682>, (data dostępu: 16.10.2014 r.).
7. A DIY 3-D Viewer for Remote Piloting, <http://spectrum.ieee.org/geek-life/hands-on/a-diy-3d-viewer-for-remote-piloting>, (data dostępu: 30.12.2014 r.).
8. Head-mounted display controls video camera, keeps you painfully single, <http://www.engadget.com/2010/07/28/head-mounted-display-controls-video-camera-keeps-you-painfully/>, (data dostępu: 30.12.2014 r.).
9. Teleobecność – Wikipedia, wolna encyklopedia, <http://pl.wikipedia.org/wiki/Teleobecno%C5%9B>, (data dostępu: 30.12.2014 r.).
10. Otworzyliśmy Laboratorium Zanurzonej Wizualizacji Przestrzennej – Aktualności – Politechnika Gdańska, http://pg.edu.pl/aktualnosci/-/asset_publisher/hWGncmoQv7K0/content/otworzylyismy-laboratorium-zanurzonej-wizualizacji-przestrzennej, (data dostępu: 15.12.2014 r.).
11. Wikipedia.org – KiCad, <http://pl.wikipedia.org/wiki/KiCad>, (data dostępu: 29.11.2014 r.).
12. Tomasz Francuz, *Język C dla mikrokontrolerów AVR. Od podstaw do zaawansowanych aplikacji.* ISBN 978-83-246-3064-6, 9788324630646,
13. PlayStation Eye – Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/PlayStation_Eye, (data dostępu: 11.12.2014 r.).
14. Meant to be Seen – View topic – Some people complain "Rift has low FOV", <http://www.mtbs3d.com/phpbb/viewtopic.php?f=140&t=17479>, (data dostępu: 11.12.2014 r.).
15. PreviewLabs» Blog Archive » Would a 4K Resolution Oculus Rift be Possible?, <http://www.previewlabs.com/would-a-4k-resolution-oculus-rift-be-possible/>, (data dostępu: 11.12.2014 r.).
16. CL – About CL Eye Platform – PS3 Eye, <https://codelaboratories.com/products/eye/driver>, (data dostępu: 16.10.2014 r.).
17. CL – About CL Eye Platform – PS3 Eye, <https://codelaboratories.com/products/eye/sdk>, (data dostępu: 16.10.2014 r.).
18. CL – Research – PS3 Eye, <https://codelaboratories.com/research/view/cl-eye-multicamera-api>, (data dostępu: 16.10.2014 r.).
19. CL – Research – PS3 Eye, <https://codelaboratories.com/research/view/cl-eye-multicamera>, (data dostępu: 16.10.2014 r.).
20. Bi2-Vision Intros New 3D Camera Rig at NAB | Below the Line, <http://www.btlnews.com/crafts/camera/bi2-vision-intros-new-3d-camera-rig-at-nab/>, (data dostępu: 11.12.2014 r.).
21. Daily Motion viewers to explore forests in 3D | 3D printing, 3D TV, virtual reality, AR and Ultra HD news, <http://www.3dfocus.co.uk/3d-news-2/3d-broadcasting/daily-motion-viewers-to-explore-forests-in-3d/7518>, (data dostępu: 11.12.2014 r.).
22. vrwiki: Oculus Rift (development kit) – Edits, [http://vrwiki.wikispaces.com/Oculus+Rift+\(development+kit\)](http://vrwiki.wikispaces.com/Oculus+Rift+(development+kit)), (data dostępu: 29.11.2014 r.).

- 23.Trends in 3D Technology: Where Is 3D Gaming Now?, <https://www.noscopeglasses.com/blog-2/66-where-is-3d-gaming-now>, (data dostępu: 11.12.2014 r.).
- 24.VR Sickness, The Rift, and How Game Developers Can Help | Oculus Rift – Virtual Reality Headset for 3D Gaming, <https://www.oculus.com/blog/vr-sickness-the-rift-and-how-game-developers-can-help/>, (data dostępu: 11.12.2014 r.).
- 25.Oculus Rift: DK1 vs DK2 comparison, <http://in2gpu.com/2014/08/10/oculus-rift-dk1-vs-dk2/>, (data dostępu: 16.10.2014 r.).
- 26.Oculus Rift in Action: Understanding the Oculus Rift Distortion Shader, <http://rifty-business.blogspot.com/2013/08/understanding-oculus-rift-distortion.html>, (data dostępu: 16.10.2014 r.).
- 27.AN235 Application note – Stepper motor driving, Thomas Hopkins, http://www.st.com/web/en/resource/technical/document/application_note/CD00003774.pdf, (data dostępu: 29.11.2014 r.).
- 28.WObit, nota katalogowa silników krokowych z serii 39BYGH, <http://www.micropik.com/PDF/39BYGH.pdf>, (data dostępu: 29.11.2014 r.).
- 29.hteck.ca, Stepper motor driver, <http://www.hteck.ca/electronics/stepper-motor-drv/sm-driver.html>, (data dostępu: 29.11.2014 r.).
- 30.STMicroelectronics, nota katalogowa układu LM7805C, <http://datasheet.octopart.com/L7805CV-STMicroelectronics-datasheet-7264666.pdf>, (data dostępu: 29.11.2014 r.).
- 31.INTERNATIONAL RECTIFIER, nota katalogowa układu IRLZ24NPBF, <http://www.tme.eu/pl/Document/33aff840278099c6edad7df22c8d95bb/irlz24n.pdf>, (data dostępu: 29.11.2014 r.).
- 32.David Cook, *Budowa robotów dla średnio zaawansowanych*. Wydanie II. ISBN: 978–83–246–5529–8, 9788324655298.
- 33.Atmel corporation, dokumentacja układu Atmega88p Rev. 8025M–AVR–6/11, <http://www.atmel.com/images/doc8025.pdf>, (data dostępu: 29.11.2014 r.).
- 34.Toshiba, nota katalogowa ULN2803APG, http://www.toshiba-components.com/docs/linear/ULN2804APG_en_datasheet.pdf, (data dostępu: 29.11.2014 r.).
- 35.Sensor Fusion: Keeping It Simple, <https://www.oculus.com/blog/sensor-fusion-keeping-it-simple/>, (data dostępu: 2.12.2014 r.).
- 36.Help! My Cockpit Is Drifting Away, <https://www.oculus.com/blog/magnetometer/>, (data dostępu 30.12.2014 r.).
- 37.Quaternion – Wikipedia, the free encyclopedia, <http://en.wikipedia.org/wiki/Quaternion>, (data dostępu: 15.12.2014 r.).
- 38.Kwaterniony w praktyce, <http://warsztat.gd/wiki/Kwaterniony+w+praktyce>, (data dostępu: 15.12.2014 r.).
- 39.Avoid Gimbal Lock for Rotation/Direction Maya Manipulators, <http://around-the-corner.typepad.com/adn/2012/08/avoid-gimbal-lock-for-rotationdirection-maya-manipulators.html>, (data dostępu: 11.12.2014 r.).
- 40.Slerp – Wikipedia, the free encyclopedia, <http://en.wikipedia.org/wiki/Slerp>, (data dostępu 1.12.2014 r.).
- 41.Oculus Developer Guide v.0.4.4, http://static.oculus.com/sdk-downloads/documents/Oculus_Developer_Guide_0.4.4.pdf, (data dostępu: 27.12.2014 r.).
- 42.Windows Presentation Foundation – Wikipedia, wolna encyklopedia, http://pl.wikipedia.org/wiki/Windows_Presentation_Foundation, (data dostępu: 29.11.2014 r.).
- 43.oculus direct hmd – Google Search, <https://www.google.co.uk/webhp?#q=oculus+direct+hmd>, (data dostępu: 11.12.2014 r.).
- 44.Imaging Overview, <http://msdn.microsoft.com/en-us/library/ms748873%28v=vs.110%29.aspx>, (data dostępu: 11.12.2014 r.).
- 45.Keystone Correction, http://www.theprojectorpros.com/learn-s-learn-p-keystone_correction.htm, (data dostępu: 29.11.2014 r.).

46. Atmel corporation, dokumentacja układu Atmega88p Rev. 8025M–AVR–6/11:
<http://www.atmel.com/images/doc8025.pdf>, (data dostępu: 29.11.2014 r.).

8. WYKAZ TABEL

Tabela 3.1. Przypadek użycia – Użytkowanie.....	25
Tabela 3.2. Przypadek użycia – Kalibracja.....	25
Tabela 3.3. Przypadek użycia – Uruchomienie.....	26
Tabela 3.4. Przypadek użycia – Korekcja obrazu.....	26
Tabela 3.5. Przypadek użycia – Wybór profilu soczewki.....	26
Tabela 3.6. Przypadek użycia – Zmiana obrazu wejściowego.....	27
Tabela 4.1. Lista pól klasy RiftMotionInterceptor.....	64
Tabela 4.2. Lista metod klasy RiftMotionInterceptor.....	65
Tabela 4.3. Lista metod interfejsu IConsole.....	65
Tabela 4.4. Lista wybranych pól klasy OculusWindow.....	70
Tabela 4.5. Lista wybranych metod klasy OculusWindow.....	71
Tabela 4.6. Najważniejsze typy zdefiniowane w bibliotece Standarts.h.....	89

9. WYKAZ RYSUNKÓW

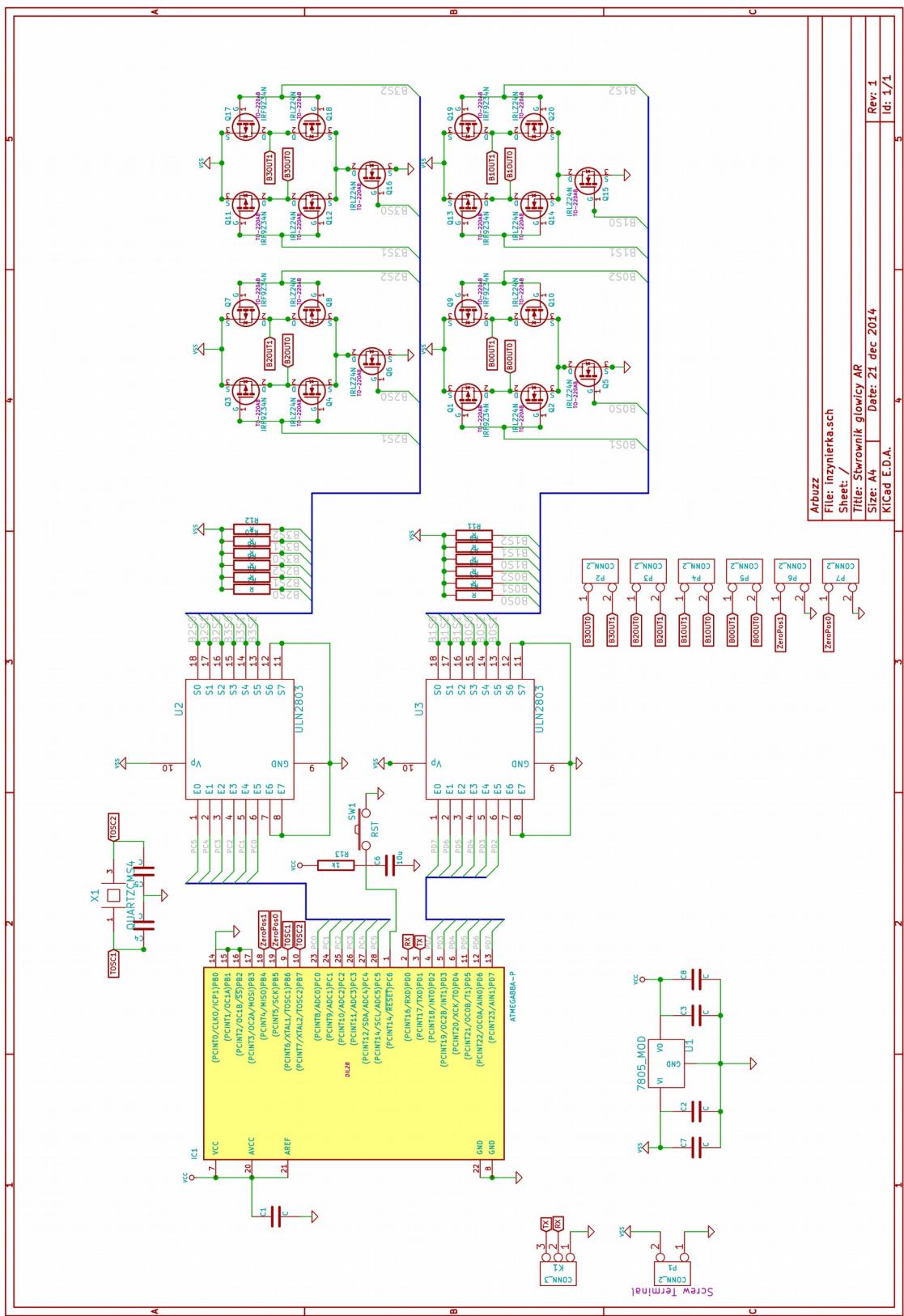
Rys. 1.1. Plakat przedstawiający Sensorame[3].....	10
Rys. 1.2. Konstrukcja Sutherlanda nosiła nazwę "Miecz Damoklesa", prawdopodobnie ze względu na swój nietypowy wygląd.[5].....	11
Rys. 1.3. Oculus Rift DK1 rozebrany na czynniki pierwsze [6].....	13
Rys. 2.1. Profesjonalna sala do wideokonferencji[9].....	15
Rys. 2.2. Laboratorium Zanurzonej Wizualizacji Przestrzennej na Wydziale ETI Politechniki Gdańskiej[10].....	16
Rys. 3.1. Diagram przypadków użycia naszego systemu.....	24
Rys. 3.2. Diagram klas API serwogłówicy.....	28
Rys. 3.3. Diagram klas części integracyjnej oraz graficznej.....	29
Rys. 3.4. Wysokopoziomowy schemat ideowy całego systemu.....	30
Rys. 4.1. Kamera PlayStation Eye[13].....	36
Rys. 4.2. Faktyczne powierzchnia użycia ekranu Oculus Rifta[15].....	37
Rys. 4.3. Podział na wątki zadań realizowanych przez CL Eye Driver[18].....	38
Rys. 4.4. Profesjonalny zestaw do filmowania 3D (po lewej)[20] oraz konsumencka kamera wideo umożliwiająca filmowanie w 3D (po prawej)[21].....	40
Rys. 4.5. Zastosowany przez nas układ wzajemnego położenia kamer.....	41
Rys. 4.6. Wizualizacja obrazu widzianego przez hełm wirtualny[23].....	42
Rys. 4.7. Różnica pomiędzy zbieżnością a głębią obrazu na ekranie HMD oraz w rzeczywistości[24].....	42
Rys. 4.8. Ochotnicy testujący nasz projekt podczas prezentacji prototypu na Politechnice Gdańskiej.....	43
Rys. 4.9. Schemat przedstawiający dwie grupy opóźnień – 2,3 – niebieski oraz 1,4,5 – czerwony.....	44
Rys. 4.10. Ekran Oculus Rifta po zdjęciu osłony[6].....	45
Rys. 4.11. Zasada działania soczewki w Oculus Riffcie[26].....	46
Rys. 4.12. Dystorsja poduszkowa (po lewej) oraz beczkowa (po prawej)[26].....	46
Rys. 4.13. Schemat budowy silnika krokkowego bipolarnego oraz kształty sygnałów sterujących na zaciskach wejściowych dla przypadku sterowania pełnymi krokami (lewy wykres) oraz pół-krokami (prawy wykres).....	48
Rys. 4.14. Zdjęcia napędu osi pionowej.....	49
Rys. 4.15. Zestaw przekładni napędu osi poziomej.....	50
Rys. 4.16. System monażu kamer do główicy wraz ze strzałkami prezentującymi możliwości kalibracyjne położenia sensorów.....	51
Rys. 4.17. Wykorzystany w realizacji klasyczny układ stabilizacji napięcia oparty na LM7805.	52
Rys. 4.18. Schematy prezentujące tryby pracy wykorzystanych w projekcie mostków sterowniczych. Kolor czerwony oznacza otwartą ścieżkę dla przepływu prądu sterowanego.....	53
Rys. 4.19. Schemat prezentujący proces nawiązywania połączenia między urządzeniami.....	55
Rys. 4.20. Układ wykorzystywany do wysterowania kluczy mostków sterowniczych. Rezystory R1–R3, dioda oraz tranzystory bipolarne są częściami zintegrowanymi wewnętrz układy ULN2803[34]. Rezystor R4 podciąga bramkę klucza do zasilania.....	58
Rys. 4.21. Prezentacja pierwotnej orientacji pilota helikoptera (po lewej) oraz orientacja z błędem kursu po upływie czasu (po prawej)[36].....	60
Rys. 4.22. Graficzne przedstawienie obrotu kwaternionu o 90° w przestrzeni czterowymiarowej[37].....	61
Rys. 4.23. Przedstawienie zjawiska "gimbal lock"[39].....	62
Rys. 4.24. Osie obrotu hełmu Oculus Rift[41].....	63
Rys. 4.25. Zrzut okna OculusWindow.....	67
Rys. 4.26. Wyniki wyszukiwania po wpisaniu frazy "oculus direct hmd" w google[43].....	68

Rys. 4.27. Podział ekranu hełmu rzeczywistości wirtualnej Oculus Rift[26].....	69
Rys. 4.28. Rodzaje rozciagnięcia obrazu w Windows Presentation Foundation[44].....	69
Rys. 4.29. Zrzut okna ControlsWindow.....	72
Rys. 4.30. Grupa ustawień "Connection, Controls and Presets"	77
Rys. 4.31. Porównanie działania opcji "Swap Rotation" oraz "Swap Cameras".....	77
Rys. 4.32. Grupa ustawień "Image Options"	79
Rys. 4.33. Oryginalny obraz (z ustawioną korekcją soczewki).....	80
Rys. 4.34. Projektor skierowany prostopadle do powierzchni (po lewej) oraz rzucający obraz wymagający korekcji zniekształceń trapezowych (po prawej)[45].....	80
Rys. 4.35. Obraz zniekształcony (po lewej) oraz po zastosowaniu korekcji zniekształceń trapezowych (po prawej) – obrazy bez transformacji beczkowej[45].	81
Rys. 4.36. Obrazy po przekształceniu: H Keystone = 280 (po lewej) oraz H Keystone = -500 (po prawej).....	81
Rys. 4.37. Obrazy po przekształceniu: V Keystone = 500 (po lewej) oraz V Keystone = -200 (po prawej).....	81
Rys. 4.38. Obraz po przekształceniu X Offset.....	82
Rys. 4.39. Obrazy po przekształceniu: Y Offset = 100 (po lewej) oraz Y Offset = -200 (po prawej).....	82
Rys. 4.40. Obrazy po przekształceniu: Zoom = 200 (po lewej) oraz Zoom = -300 (po prawej).83	83
Rys. 4.41. Grupa ustawień "Lens Correction"	83
Rys. 4.42. Obraz bez jakichkolwiek przekształceń.....	84
Rys. 4.43. Obraz przystosowany do soczewki A: Zoom = -128, LensCorrection k1 = 256 (dystorsja beczkowa).....	84
Rys. 4.44. Obraz po korekcji poduszkowej: Zoom = -500, LensCorrection k1 = -350.....	84
Rys. 4.45. Grupa ustawień "Camera Sensor Parameters".....	85
Rys. 4.46. Ekran statusu z przykładowymi komunikatami.....	85
Rys. 4.47 Diagram klas bibliotek mikrokontrolera, część pierwsza.....	87
Rys. 4.48 Diagram klas bibliotek mikrokontrolera, część 2.....	88

10. ZAŁĄCZNIKI

Załącznik A: Płyta CD zawierająca źródła oprogramowania oraz projekt płytka kontrolera

Załącznik B: Schemat ideowy kontrolera



Załącznik C: Projekt płytki PCB układu kontrolera wydrukowany w powiększeniu.

