AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa inżynierska

System wizyjny śledzący obiekty wykorzystujący ruchomą kamerę zrealizowany w oparciu o heterogeniczny układ Zynq.

An object tracking vision system using a moving camera implemented in a Zynq heterogeneous device.

Autor:	*Marcin Kowalczyk*
Kierunek studiów:	*Automatyka i Robotyka*
Opiekun pracy:	*dr inż. Tomasz Kryjak*

Kraków, 2016

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję . . . tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wstęp

1.1. Cel pracy

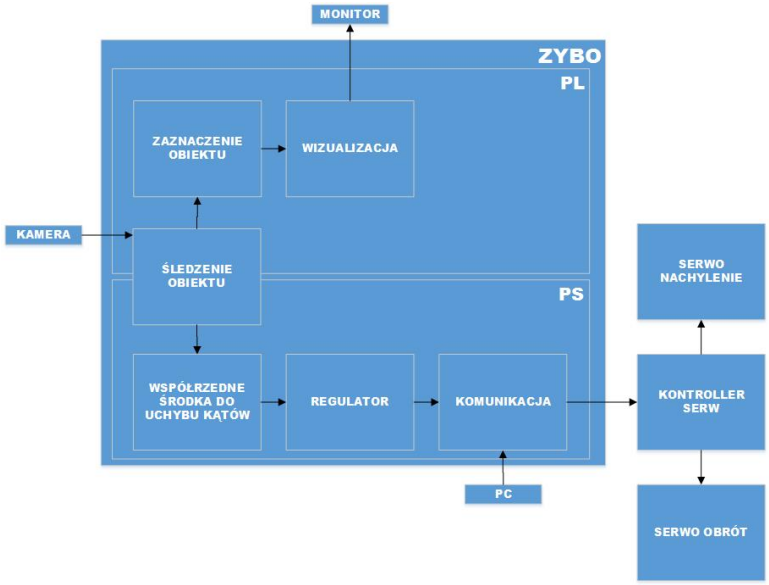
Celem pracy jest stworzenie demonstratora systemu wizyjnego do śledzenia obiektów, przy czym zakłada się, że kamera zamontowana jest na głowicy obrotowej. Praca inżynierska obejmuje skompletowanie, we współpracy z opiekunem, stanowiska testowego składającego się z głowicy ruchomej, kamery, platformy obliczeniowej oraz wskaźnika. Należy przeprowadzić analizę oraz weryfikację różnych koncepcji śledzenia, biorąc pod uwagę skuteczność oraz możliwość implementacji sprzętowej lub sprzętowo-programowej. Wybrane rozwiązanie zostanie zaimplementowane, uruchomione i przetestowane w sprzęcie. Wyjście z modułu śledzenia stanowi podstawę do wypracowania pozycjonowania głowicy obrotowej, takiego, aby utrzymać obiekt w środku kadru oraz oznaczyć go wskaźnikiem.

1.2. Wprowadzenie

Automatyczne śledzenie z pomocą kamery jest wykorzystywane m.in. w zagadnieniach związanych z bezpieczeństwem, inwigilacją lub w zastosowaniach wojskowych. Jest ściśle powiązane z wykrywaniem oraz identyfikacją obiektów. Świadczy to o złożoności tego zagadnienia. Śledzenie różni się od innych algorytmów przetwarzania obrazów głównie tym, że musimy wykorzystywać informacje pochodzące z więcej niż jednego kadru. Uwydatnia się to szczególnie, kiedy kamera rejestruje kilka obiektów podobnych do śledzonego. Jeśli zadaniem jest obserwacja jednego konkretnego obiektu, to aby go wyróżnić musimy wykorzystać położenie śledzonego obiektu w poprzednich ramkach [1]. Znaczące problemy w śledzeniu obiektów mogą być spowodowane zmianą orientacji celu oraz dużą jego szybkością w porównaniu do częstotliwości rejestrowania klatek. W przypadku kamery umieszczonej na głowicy błędy mogą być dodatkowo spowodowane rozmyciem obrazu w trakcie poruszania układu. Istnieją liczne algorytmy służące śledzeniu elementu na obrazach z kamery. Są to m.in:

- Śledzenie przez detekcję
- Mean-shift
- KLT

Algorytmy te zostaną dokładniej omówione w kolejnym rozdziale pracy.



Rys. 1.1. Schemat realizowanego rozwiązania.
Źródło: Własne

1.3. Zawartość pracy

1.3.1. Rozdział 2

Zrealizowano przegląd algorytmów śledzenia i dokonano wyboru algorytmu do implementacji w sprzęcie.

1.3.2. Rozdział 3

Opisano skompletowane stanowisko. Przedstawiono specyfikację użytych elementów oraz uzasadniono ich wybór.

1.3.3. Rozdział 4

Scharakteryzowano komunikację pomiędzy elementami realizowanego systemu (PC-Zynq-Maestro).

1.3.4. Rozdział 5

Przedstawiono realizację podstawowego toru wizyjnego otrzymującego obraz z kamery i wysyłającego go bez zmian na wyjście.

1.3.5. Rozdział 6

Opisano model matematyczny sterowanego obiektu oraz projekt implementowanego regulatora. Krótko omówiono wynik testu regulatora na modelu.

2. Algorytmy

W rozdziale tym omówione zostaną algorytmy śledzenia. Rozważymy również możliwość oraz skuteczność ich implementacji sprzętowej i sprzętowo-programowej.

2.1. Śledzenie przez detekcję

Jest to najłatwiejszy możliwy algorytm śledzenia. Polega on na detekcji obiektu i określeniu jego położenia (np. poprzez wyznaczenie środka ciężkości). Algorytm ten jest efektywny tylko, gdy w kadrze znajduje się maksymalnie jeden wyznaczony obiekt. Ograniczenie to możemy obejść przez zastosowanie dodatkowo algorytmu indeksacji. Wtedy musimy zdecydować który z wyznaczonych obiektów jest tym właściwym. Możemy np. śledzić element znajdujący się najbliżej (mający największą powierzchnię w kadrze). Efektywność implementacji programowo-sprzętowej zależy w większości od wykorzystanego algorytmu detekcji, i/lub segmentacji. Algorytm detekcji na podstawie koloru będzie łatwy i szybki do zaimplementowania, a algorytm bazujący na współczynnikach kształtu wymagać będzie dużo więcej pracy i zasobów.

2.2. Mean-shift

Algorytm ten bazuje na statystycznej metodzie poszukiwania lokalnego maksimum rozkładu prawdopodobieństwa. Polega on na wyznaczeniu maksymalnej wartości prawdopodobieństwa w aktualnym oknie wokół punktu startowego. W celu zastosowania tego algorytmu do śledzenia obiektu należy przedstawić obraz jako rozkład prawdopodobieństwa. W tym celu każdemu pikselowi przypisuje się wartość prawdopodobieństwa. Można to zrobić np. na podstawie koloru, przypisując kolorom wartość prawdopodobieństwa, lub histogramu, porównując histogram otoczenia piksela z histogramem obiektu [2]. Jeśli prawdopodobieństwo obliczone zostaje tylko na podstawie koloru, dobre wyniki można uzyskać, gdy [2]:

- Obiekt ma jednolitą barwę.
- Występują bardzo małe zmiany oświetlenia.
- W kadrze nie ma obiektów podobnych do śledzonego.

– Kolor tła znacznie różni się od koloru obiektu.

– Obiekt nie zostaje całkowicie zasłonięty.

Algorytm ten ma następujący przebieg:

1. Wybierz rozmiar okna.
2. Wybierz początkowe położenie(środek) okna.
3. Wyznacz położenie maksimum wartości prawdopodobieństwa w oknie.
4. Przesuń okno, aby wyznaczone maksimum było jego środkiem.
5. Powtarzaj kroki 3 i 4, aż algorytm będzie zbieżny.

Punktem startowym dla każdego kolejnego obrazu z kamery jest pozycja obiektu w poprzednim obrazie. Poszukiwanie maksymalnej wartości prawdopodobieństwa odbywa się w następujący sposób:

– Obliczamy moment zerowy.

$$M_{00} = \sum_x \sum_y l(x, y) \tag{2.2.1}$$

– Obliczamy moment pierwszy dla osi poziomej.

$$M_{10} = \sum_x \sum_y x \cdot l(x, y) \tag{2.2.2}$$

– Obliczamy moment pierwszy dla osi pionowej.

$$M_{01} = \sum_x \sum_y y \cdot l(x, y) \tag{2.2.3}$$

– Obliczamy środek rozkładu prawdopodobieństwa w danym oknie.

$$x_c = \frac{M_{10}}{M_{00}} \tag{2.2.4}$$

$$y_c = \frac{M_{01}}{M_{00}} \tag{2.2.5}$$

Gdzie $l(x, y)$ jest wartością prawdopodobieństwa dla piksela (x, y)[3].

2.3. Filtr cząsteczkowy

Filtr cząsteczkowy inaczej nazywany jest sekwencyjną metodą Monte Carlo. Punktem startowym algorytmów tego typu jest model obiektu w postaci równań stanu [4]:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{2.3.1}$$

$$y_k = Cx_k + v_k \tag{2.3.2}$$

x_k jest ukrytym, prawdziwym stanem obiektu.
y_k jest obserwowanym stanem obiektu.
w_{k-1} jest zmienną losową o rozkładzie normalnym reprezentującą szum przetwarzania.
v_k jest zmienną losową o rozkładzie normalnym reprezentującą szum pomiarowy.

Algorytm korzysta z wyprowadzonej z twierdzenia Bayesa rekurencyjnej zależności:

$$p(x_{k+1}|y_{0:k+1}) \propto p(y_{k+1}|x_{k+1}) \int_{x_k} p(x_{k+1}|x_k)p(x_k|y_{0:k})dx_k \tag{2.3.3}$$

$x_{0:k}$ oznacza wektor $(x_0, \ldots, x_k)$. Powyższy wzór opisuje rozkład prawdopodobieństwa, zmiennej x_{k+1}, pod warunkiem, że znane są dotychczasowe pomiary $y_{0:k+1}$. Wartość oczekiwana tego rozkład przyjmowana jest jako wynik zadania śledzenia. W opisywanym algorytmie równanie to rozwiązywane jest metodą Monte Carlo, która skomplikowane problemy przybliża za pomocą zbioru losowo rozmieszczonych cząstek. W omawianym przypadku przybliżaną wielkością jest prawdopodobieństwo $p(x_{k+1}|y_{0:k+1})$. Cząsteczki reprezentują możliwe położenia śledzonego obiektu [5].

Następnie należy każdej cząsteczce przypisać wagę, która reprezentuje prawdopodobieństwo $p(y_{k+1}|x_{k+1})$. Może być określona np. w następujący sposób [5]:

$$w_i = ke^{-\Lambda D^2} \tag{2.3.4}$$

k jest stałą normalizującą sumę wag. Λ jest parametrem algorytmu. D jest dystansem.

Najczęściej stosownym dystansem jest podobieństwo histogramu cząsteczki z histogramem bazowym obiektu. Pozycja obiektu wyznaczana jest następująco:

$$x_{k+1} = \sum_{i=1}^{M} w_{i,k+1} \cdot x_{i,k+1} \tag{2.3.5}$$

Ostatnim etapem jest powielenie cząstek. Losowanych jest M cząstek z prawdopodobieństwem określonym przez wyliczone wagi (cząstka może być wylosowana więcej niż jeden raz). Następnie są one rozrzucane zgodnie z równaniem 2.3.1. Jest to etap predykcji. W wyniku tego działania większość cząstek znowu znajduje się w otoczeniu śledzonego obiektu.

Algorytm ma następujący przebieg [5]:

1. Wyznaczenie histogramu śledzonego obiektu.
2. Rozmieszczenie cząstek wokół początkowego położenia obiektu zgodnie z wybranym rozkładem (najczęściej normalnym).
3. Etap przewidywania. Przesunięcie cząstek zgodnie z równaniami stanu 2.3.1.
4. Obliczenie histogramu w otoczeniu każdej cząstki.
5. Obliczenie wagi każdej cząstki.
6. Obliczenie najbardziej prawdopodobnego położenia obiektu zgodnie z równaniem 2.3.5.
7. Powielenie cząstek.

2.4. KLT

Algorytm opisany przez Kanade'a, Lucas'a i Tomasi'a może posłużyć do śledzenie obiektu na obrazie w skali szarości. Polega on na poszukiwaniu najlepszego dopasowania obrazu referencyjnego $T(x)$ do aktualnej ramki otrzymanej z kamery $I(x)$ [6]. Obrazy te są przesuwane względem siebie o wektor p:

$$W(x,p) = \begin{bmatrix} x_1 + p_1 \\ x_2 + p_2 \end{bmatrix} \tag{2.4.1}$$

Poszukiwane jest więc minimum następującej funkcji:

$$f(x,p) = \sum_x ((I(W(x,p)) - T(x))^2 \tag{2.4.2}$$

Zakłada się, że początkowa wartość przesunięcia p jest znana, a poszukiwana jest najlepsza modyfikacja tego przesunięcia Δp.

$$f(x,\Delta p) = \sum_x ((I(W(x,p+\Delta p)) - T(x))^2 \tag{2.4.3}$$

Po znalezieniu optymalnego przesunięcia Δp aktualizowana jest wartość przesunięcia początkowego:

$$p \leftarrow p + \Delta p \tag{2.4.4}$$

Funkcja $I(x, p + \Delta p)$ linearyzowana jest w otoczeniu punktu (x, p) ze względu na Δp poprzez rozwinięcie w szereg Taylora pierwszego rzędu.

$$f(x,\Delta p) = \sum_x (I(W(x,p)) + \nabla I \cdot \frac{\partial W}{\partial p} \cdot \Delta p - T(x))^2 \tag{2.4.5}$$

$\nabla I = [\frac{\partial I}{\partial x_1}, \frac{\partial I}{\partial x_2}]$ jest transponowanym gradientem obrazu wejściowego w punkcie $W(x, p)$.
$\frac{\partial W}{\partial p}$ jest Jakobianem przesunięcia obrazów.

Obliczamy pochodną funkcji dopasowania $f(x, \Delta p)$ po Δp i przyrównujemy ją do zera.

$$\frac{\partial f(x, \Delta p)}{\partial \Delta p} = 2 \cdot \sum_x (\nabla I \cdot \frac{\partial W}{\partial p})^T \cdot ((I(W(x, p)) + \nabla I \cdot \frac{\partial W}{\partial p} \cdot \Delta p - T(x)) = 0 \tag{2.4.6}$$

Przekształcając powyższy wzór otrzymuje się:

$$\Delta p = H^{-1} \cdot \sum_x (\nabla I \cdot \frac{\partial W}{\partial p})^T \cdot (T(x) - I(W(x, p))) \tag{2.4.7}$$

H jest Hesjanem:

$$H = \sum_x (\nabla I \cdot \frac{\partial W}{\partial p})^T \cdot (\nabla I \cdot \frac{\partial W}{\partial p}) \tag{2.4.8}$$

Aby algorytm dobrze działał musimy odpowiednio wybrać obraz referencyjny. Zauważmy, że we wzorze na Δp jest odwrotność Hesjanu. Odwracanie macierzy może powodować duże błędy numeryczne, gdy macierz ta jest źle uwarunkowana. Oznacza to, że Hesjan musi posiadać odpowiednio duże wartości własne.

$$\lambda(H) > \lambda_{thr} \tag{2.4.9}$$

W praktyce powyższy warunek oznacza, że obraz referencyjny nie może być jednolity. W najlepszym wypadku zawierał będzie on krawędzie śledzonego obiektu.

Algorytm ma następujący przebieg [7]:

1. Znajdź obszary spełniające warunek na wartości własne hesjanu.
2. Wyznacz obraz przesunięty o p.
3. Wyznacz gradient ∇I.
4. Oblicz Jakobian $\frac{\partial W}{\partial p}$ oraz iloczyn $\nabla I \cdot \frac{\partial W}{\partial p}$.
5. Oblicz Hesjan $H = \sum_x (\nabla I \cdot \frac{\partial W}{\partial p})^T \cdot (\nabla I \cdot \frac{\partial W}{\partial p})$.
6. Wyznacz przesunięcie $\Delta p = H^{-1} \cdot \sum_x (\nabla I \cdot \frac{\partial W}{\partial p})^T \cdot (T(x) - I(W(x, p)))$.
7. Zaktualizuj parametr $p \leftarrow p + \Delta p$.
8. Powtarzaj kroki od 2 do 7 do czasu, gdy algorytm będzie zbiegał do punktu.

2.5. Wybór implementowanego algorytmu

Dokonując wyboru algorytmu do implementacji w sprzęcie bazowano na gotowych implementacjach w programie *MATLAB*. Algorytmy uruchamiano dla zarejestrowanych sekwencji testowych, takich jak

nagrania poruszającego się drona. Warto zwrócić uwagę na fakt, że sekwencje te nagrywane były na jednolitym tle, co zwiększało skuteczność rozwiązań. Stwierdzono, że z zadaniem najlepiej poradził sobie algorytm KLT. Oprócz tego postanowiono zaimplementować prosty algorytm śledzenia przez detekcję do celów testowych. Z użyciem tego algorytmu badano pozycjonowanie serwomechanizmów w trakcie testowania i doboru nastaw regulatora. Został również przetestowany gotowy algorytm Mean-shift, aby sprawdzić jego działanie w platformie sprzętowej oraz ocenić działanie posiadanej implementacji.

3. Stanowisko demonstracyjne

Pierwszą częścią pracy jest skompletowanie stanowiska demonstracyjnego, w skład którego wchodzą:

- Kamera
- Platforma obliczeniowa
- Serwomechanizmy
- Sterownik serwomechanizmów
- Zasilanie
- Wskaźnik

3.1. Kamera

Rys. 3.1. Kamera Xiaomi Yi Action.
Źródło: [8]

Postanowiono użyć kamery Xiaomi Yi Action YDXJ01XY. Jest ona bardzo lekka i może wysyłać obraz w wybranej przez nas rozdzielczości z częstotliwością 60 Hz. Oprócz tego można ją przykręcić do użytych elementów połączeniowych głowicy obrotowej. Dane techniczne [8]:

- Maksymalna rozdzielczość: 1920x1080
- Waga: 76.6 g
- Odporność na wstrząsy
- Kąt nagrywania: 155°
- Wymiary: 60.4 mm x 42 mm x 21.2 mm

3.2. Platforma obliczeniowa

Rys. 3.2. Płytka ZYBO Zynq-7000.
Źródło: [9]

Zdecydowano, że platformę obliczeniową stanowić będzie płytka ZYBO. Jest to bogato wyposażone narzędzie zawierające układ programowalny z rodziny Xilinx Zynq-7000. Układ ten oparty jest na architekturze Xilinx All Programable System-on-Chip, w której zintegrowany został dwurdzeniowy procesor ARM Cortex-A9 i układ programowalny FPGA z serii Xilinx 7. Zawiera ona m.in. port HDMI potrzebne

do odbieranie obrazu z kamery oraz port VGA, który umożliwia nam wysyłanie obrazu do monitora [9]. W układzie programowalnym zaimplementowany zostanie tor wizyjny przetwarzający potokowo dane przesyłane z kamery oraz wyznaczający położenie obiektu w danej ramce. Oprócz tego musi on komunikować się ze sterownikiem serwomechanizmów w celu pozycjonowania głowicy w wyznaczonym punkcie oraz z komputerem klasy PC, aby otrzymywać parametry pracy oraz ustawiać początkową pozycję głowicy.

3.3. Serwomechanizmy

Ruchomą głowicę postanowiono skonstruować wykorzystując serwomechanizmy dostępne na rynku. Przy wyborze brano pod uwagę serwomechanizmy analogowe, serwomechanizmy cyfrowe oraz serwomechanizmy smart.

3.3.1. Serwomechanizm analogowy

Serwomechanizm ten sterowany jest impulsami podawanymi z częstotliwością 50 Hz. Podawanie impulsów z większą częstotliwością może doprowadzić do uszkodzenia urządzenia. W zależności od czasu trwania impulsu serwomechanizm ustawia się w odpowiedniej pozycji i ją utrzymuje. Kontrola położenia odbywa się za pomocą potencjometru sprzężonego z kołem zębatym. Serwomechanizm tego typu nie reagują szybko i nie produkują wystarczającego momentu kiedy zadajemy małe przesunięcie.

3.3.2. Serwomechanizm cyfrowy

Jest on, podobnie jak serwomechanizm analogowy, sterowany impulsami. Podawać można je jednak z maksymalną częstotliwością powyżej 300 Hz. W porównaniu do serwomechanizmu analogowego przyspiesza ono szybciej i produkuje stabilniejszy moment obrotowy. Reagują również lepiej na polecenia małych zmian pozycji. Z drugiej strony potrzebują one dużo większej mocy. Prąd, który pobierają w trakcie pracy, może sięgać kilku amperów.

3.3.3. Serwomechanizm smart

Główną różnicą pomiędzy serwomechanizmami standardowymi, a smart, jest sposób komunikacji w celu podania wartości zadanej położenia. Zamiast impulsów o różnej szerokości wykorzystuje się komunikację szeregową. Ich obsługa jest więc bardziej skomplikowana. Najpopularniejszymi protokołami są TTL Half-Duplex, TTL Full-Duplex i RS-485. Za pomocą komunikacji tego typu można zmieniać dużo więcej parametrów niż tylko pozycję zadaną. Można zmieniać nastawy regulatora PID, maksymalną prędkość kątową lub maksymalny moment. W przeciwieństwie do serwomechanizmów klasycznych, możemy również odczytywać aktualną pozycję serwomechanizmu. Minusem tych urządzeń jest ich cena oraz dostępność. Są one kilka razy droższe od cyfrowych serwomechanizmów o podobnych parametrach.

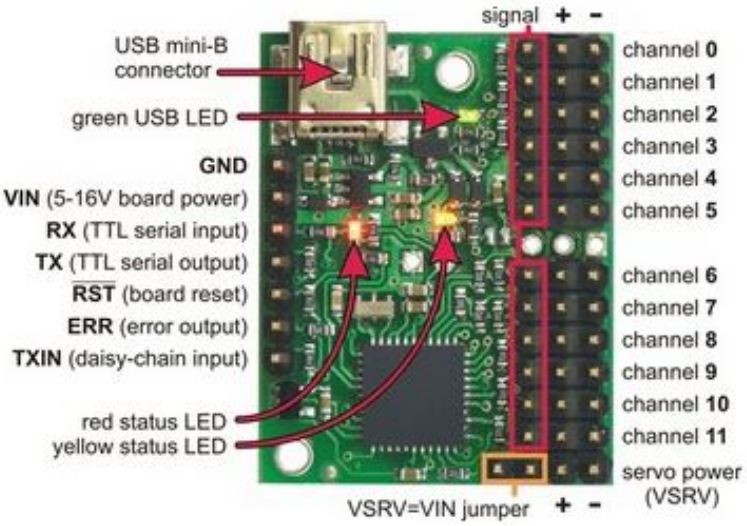
Rys. 3.3. Serwomechanizm cyfrowy PowerHD D-21HV.
Źródło: [10]

Postanowiono użyć dwóch serwomechanizmów cyfrowych PowerHD D-21HV. Są to serwomechanizmy typu standard o dużym momencie obrotowym i wysokiej maksymalnej prędkości obrotowej. Wyposażone jest w łożyska kulkowe oraz tytanowe tryby w celu zwiększenia jego wytrzymałości. Specyfikacja serwomechanizmów [10]:

- Napięcie zasilania: $6.0 - 7.4$ V
- Zakres ruchu: 155°
- Masa: 75 g
- Moment: 21 kg·cm dla napięcie 7.4 V
- Prędkość: 0.12s/60° dla napięcia 7.4 V

Urządzenia te mogą pobierać impulsowy prąd o wartości nawet powyżej 3 A. Sterowane są za pomocą impulsów podawanych z częstotliwością 333 Hz. Pozycję naturalną zadajemy podając impulsy o czasie trwania 1500μs. Serwa zostały połączone w głowicę za pomocą metalowych uchwytów dedykowanych do łączenia serwomechanizmów typu standard w konfigurację PT.

3.4. Sterownik serwomechanizmów



Rys. 3.4. Sterownik serwomechanizmów Pololu Mini Maestro.
Źródło: [11]

Do sterowania serwomechanizmami postanowiono użyć gotowego sterownika produkowanego przez firmę Pololu. 12-kanałowy sterownik Mini Maestro pozwala na równoczesną obsługę obu serwomechanizmów. Oprócz podawania wartości zadanej w postaci impulsów pozwala on również na zmianę ustawianie maksymalnej prędkości obrotowej i maksymalnego momentu obrotowego. Dzięki temu można dużo lepiej kontrolować urządzenia wykonawcze i zapewnić bardziej ciągły ruch głowicy. Komunikacja ze sterownikiem odbywa się za pośrednictwem interfejsu szeregowego UART z prędkością 115200 bitów na sekundę.

3.5. Zasilanie

Rys. 3.5. Układ zasilający serwomechanizmy.
Źródło: Własne

Do zasilania głowicy użyty został zasilacz impulsowy Redox, który na wyjściu daje napięcie 12 V i maksymalny prąd 5 A. Jako, że serwomechanizmy potrzebują napięcia 7.4 V używamy przetwornicy step-down XL4005E1 o regulowanym napięciu wyjściowym. Maksymalny prąd wyjściowy użytej przetwornicy wynosi 5 A. Jest to wartość wystarczająca do zasilenia użytych serwomechanizmów.

3.6. Wskaźnik

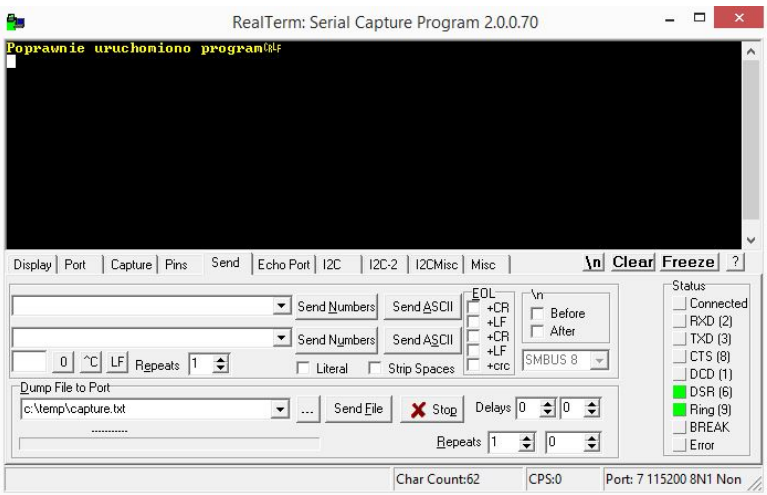
Postanowiono użyć zwykłego wskaźnika laserowego do wskazywania miejsca, w stronę którego zwrócona jest głowica.

4. Komunikacja

Po zbudowaniu stanowiska postanowiono w pierwszej kolejności zaimplementować komunikację między wszystkimi elementami systemu. Można ją podzielić na część PC-ZYNQ oraz ZYNQ-Maestro.

4.1. PC-ZYNQ

Postanowiono wykorzystać UART. Argumentem przemawiającym za tym rozwiązaniem był fakt wyprowadzenia dwóch pinów MIO (48 i 49) procesora płytki ZYBO do złącza mikro USB typu B. Piny te podłączone zostały do jednego z układów procesora odpowiedzialnych za komunikację szeregową - UART1. Złącze USB służy również do programowania układu przez JTAG. Dzięki temu używa się jednego przewodu do programowania układu oraz do komunikacji z nim. Po podłączeniu przewodu oraz włączeniu ZYBO w komputerze pojawia się dodatkowy port COM. Przez ten port można wymieniać dane z platformą obliczeniową. Do wysyłania komunikatów na PC wykorzystano program *Realterm*. Dzięki temu nie było konieczności pisania dodatkowego oprogramowania.



Rys. 4.1. Okno programu Realterm z komunikatem o poprawnym uruchomieniu programu.
Źródło: Własne

Rozmiar jednej komendy wysyłanej do układu ZYNQ wynosi 5 bajtów. Pierwszy informuje o typie rozkazu, a 4 kolejne są danymi do tego rozkazu. Każdy kolejny odebrany bajt umieszczany jest w buforze o rozmiarze 5 bajtów. Włączone zostało przerwanie od pełnego bufora. Po wysłaniu komendy z PC w procesorze układu ZYNQ uruchamiane jest przerwanie, w którym wysłany zostaje odpowiedni rozkaz do sterownika serwomechanizmów lub ustawiana jest zmienna informująca o pracy w trybie autonomicznym. Zaimplementowano obsługę następujących komend w ZYNQ:

– Zmiana pozycji serwomechanizmów: 0x00 0xHH 0xLL 0xHH 0xLL.

– Zmiana maksymalnej prędkości serwomechanizmów: 0x01 0xHH 0xLL 0xHH 0xLL.

– Zmiana maksymalnego momentu serwomechanizmów: 0x02 0xHH 0xLL 0xHH 0xLL.

– Odczyt aktualnej wartości zadanej ze sterownika: 0x03 0x– 0x– 0x– 0x–.

– Rozpoczęcie pracy autonomicznej (śledzenia): 0x04 0x– 0x– 0x– 0x–.

W pierwszych trzech rozkazach pierwsze dwa bajty danych są parametrami dla serwomechanizmu odpowiedzialnego za obrót, a kolejne dwa są parametrami dla serwomechanizmu odpowiedzialnego za nachylenie. Myślniki w kolejnych komendach oznaczają, że wysłane bajty danych nie są istotne (komendy te nie potrzebują danych).

4.2. ZYNQ-Maestro

Komunikacja przez UART została narzucona, gdyż jest to jedyny protokół komunikacyjny w Maestro. Łączymy odpowiednie piny (MIO 14,15) złącza MIO PMOD płytki ZYBO do pinów Maestro odpowiadających za komunikację szeregową. Następnie do użytych pinów MIO podłączamy wyjścia innego układu procesora odpowiadającego za komunikację szeregową - UART0. W dokumentacji Maestro wyszukujemy listę komend i implementujemy wysyłanie potrzebnych w układzie ZYNQ [11]. Są to następujące komendy:

– Zmiana pozycji serwomechanizmów: 0x9F 0x02 0x0A 0xLL 0xHH 0xLL 0xHH.

– Zmiana maksymalnej prędkości serwomechanizmów: 0x87 0x0A/0x0B 0xLL 0xHH.

– Zmiana maksymalnego momentu serwomechanizmów: 0x89 0x0A/0x0B 0xLL 0xHH.

– Odczyt pozycji ze sterownika: 0x90 0x0A/0x0B.

W powyższych komendach A oznacza obrót a B nachylenie (są to kanały sterownika).

4.3. Testy komunikacji

Komunikacja była testowana na układzie złożonym z:

– Komputera klasy PC.

– Płytki ZYBO.

– Komputera Raspberry Pi 2 Model B.

Rys. 4.2. Płytka ZYBO i komputer Raspberry pi 2 B użyty do testów.
Źródło: Własne

W układzie tym Raspberry zastępowało sterownik serwomechanizmów. Zamiana ta została wykonana, by mieć możliwość sprawdzania poprawności danych odbieranych z ZYNQ. W trakcie testów okazało się, że pierwszy bajt wysyłany do ZYBO jest wpisywany do bufora i wskaźnik jest przesuwany, lecz w kontrolerze przerwań nie jest zwiększany licznik ilości bajtów w buforze. W efekcie przerwanie od pełnego bufora danych przychodzących było wywoływane o 1 bajt za późno (po otrzymaniu pierwszego bajtu nowej komendy). Naprawiono to poprzez dodatkowy test w trakcie inicjalizacji komunikacji szeregowej. Test ten polegał na włączenie trybu loopback, wysłaniu i odebraniu kilku bajtów danych, a następnie zresetowaniu bufora. W trybie loopback dane wysyłane przez UART są wysyłane na jego wejście. W ten sposób można sprawdzać zgodność danych. Oprócz tego sprawdzono, czy wysłanie komendy z PC do ZYNQ skutkuje wysłaniem poprawnej komendy z ZYNQ do Maestro. Po naprawieniu opisanego problemu i ponownym przeprowadzeniu testów stwierdzono, że komunikacja działa poprawnie.

5. Tor wizyjny

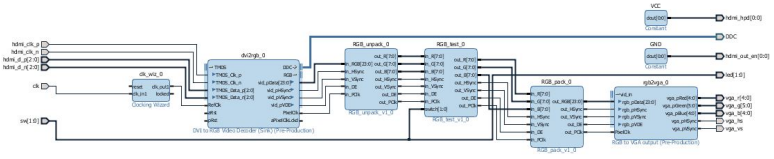
Kolejnym krokiem realizacji projektu była implementacja podstawowych zadań przetwarzania obrazu w logice rekonfigurowalnej układu heterogenicznego. Rozpoczęto od stworzenia podstawowego toru wizyjnego pobierającego obraz z wejścia HDMI i wysyłającego bez zmian na wyjście VGA. Następnie stworzono moduły pozwalające na realizację algorytmu śledzenia przez detekcję.

5.1. Podstawowe przesyłanie obrazu

Tor wizyjny nie wykonujący żadnych zmian w obrazie pozwoli nam na określenie czy dane są poprawnie odbierane z kamery i poprawnie wysyłane do monitora. Stanowić on będzie również bazę dla dowolnego innego realizowanego algorytmu przetwarzania obrazu. Konieczna jest więc pewność, że działa poprawnie. Musi on realizować następujące zadania:

1. Odbieranie danych ze złącza DVI.
2. Konwersja danych odebranych ze złącza DVI do przestrzeni barw RGB i sygnałów synchronizacyjnych.
3. Konwersja przestrzeni barw RGB i sygnałów synchronizacyjnych do VGA.
4. Wysłanie danych do złącza VGA.

Zadania te są realizowane przez moduły dostarczane przez firmę *Digilent*, producenta platformy obliczeniowej. Moduły połączone zostały za pomocą diagramu w programie *Vivado*. Diagram, w porównaniu do pliku języka Verilog, zwiększa przejrzystość połączeń między modułami oraz zmniejsza ryzyko błędu np. w nazwie portu.



Rys. 5.1. Podstawowy tor wizyjny.
Źródło: Własne

Testy rozpoczęto od podłączenie wyjścia HDMI laptopa do wejścia HDMI płytki ZYBO oraz podłączenia monitora do wyjścia VGA ZYBO. W tak połączonym układzie obraz był wyświetlany poprawnie. Następnie źródło obrazu zamieniono na docelową kamerę. Tym razem żadem obraz nie został wyświetlony na ekranie monitora. W celu znalezienia i naprawy problemu postanowiono w torze wizyjnym włączyć opcję *debug* dla części węzłów. Dzięki temu można podglądać sygnały już w działającym układzie. Ustalono, że problem jest z modułem odbierającym dane z wejścia HDMI - *dvi2rgb*. Problem udało się naprawić aktualizując go z wersji 1.2 do wersji 1.6 oraz zmieniając ustawienia zegara taktującego oraz domyślnej rozdzielczości przetwarzanego obrazu.

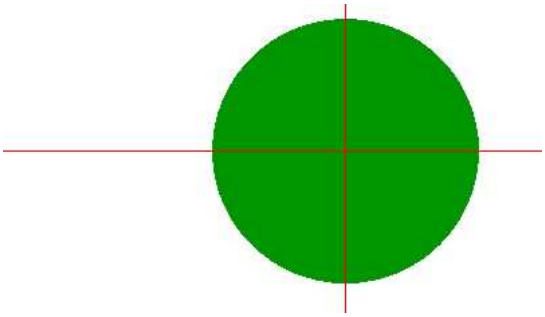
5.2. Śledzenie przez detekcję

Śledzenie przez detekcję zostało zrealizowane na podstawie koloru śledzonego obiektu. Jako parametr modułu detekcji podawany jest zakres kolorów w przestrzeni barw RGB, które mają być traktowane jako należące do obiektu. Pikselowi, którego kolor mieści się w podanym zakresie przypisywana jest wartość 1, a pozostałym 0. Pod nadaniu wartości wszystkim pikselom kadru obliczany jest środek ciężkości obrazu. Wyznaczony środek ciężkości jest środkiem śledzonego obiektu. Punkt ten zaznaczany jest na obrazie wyjściowym za pomocą dwóch czerwonych linii - poziomej i pionowej. Podczas testów wykrywany był następujący zakres przestrzeni barw RGB, odpowiadający kolorowi zielonemu:

$$R \in [0, 30] \tag{5.2.1}$$

$$G \in [40, 255] \tag{5.2.2}$$

$$B \in [0, 30] \tag{5.2.3}$$



Rys. 5.2. Przykładowy wynik działania algorytmu.
Źródło: Własne

Testy były wykonywane zarówno dla obrazu z komputera PC jak i kamery. Należy zwrócić uwagę na niską efektywność tej metody dla obrazu z kamery. Spowodowane jest to szumami oraz bardzo dużą zależnością rozpoznanego koloru od natężenia i rodzaju światła.

5.3. Komunikacja PL-PS

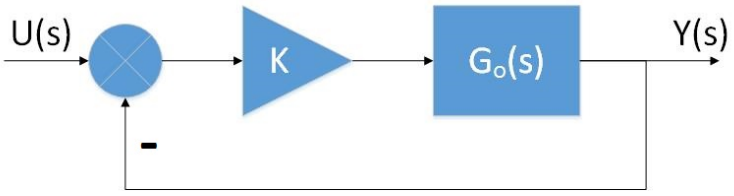
Połączono wykonany w tym rozdziale tor wizyjny z wyznaczaniem środka ciężkości z komunikacją zaimplementowaną w rozdziale 4. Komunikacja pomiędzy procesorem ARM a FPGA odbywa się z użyciem modułu *AXI Lite*. Do procesora wysyłane są współrzędne wyznaczonego środka ciężkości śledzonego obiektu. Kiedy dane są gotowe do odczytu, sygnał oznaczający wysłanie danych do procesora zmienia wartość logiczna z 0 na 1. Od narastającego zbocza tego sygnału zgłaszane jest przerwanie, w którym aktualizowana jest zmienna odpowiedzialna za położenie celu, czyli wartości zadane serwomechanizmów. W procesorze odebrane współrzędne przeliczone zostają na uchyby regulacji kątów na podstawie rozdzielczości obrazu i kątów nagrywania kamery. Założono, że odległość pikseli od środka obrazu jest proporcjonalna do kąta. Założenie to nie jest słuszne, lecz pozwala pominąć znaczną ilość testów, które należałoby przeprowadzić, aby wyznaczyć tą funkcję. Obliczanie skomplikowanych funkcji matematycznych wymagałoby również znacznie więcej czasu procesora.

6. Regulator

W zaprojektowanym układzie kamera pełni rolę sensora uchybu regulacji. Układ nie ma możliwości mierzenia pozycji serwomechanizmów. Aby poprawnie pozycjonować musimy więc zaimplementować regulator, którego wyjściem będzie wartość zadana serwomechanizmów.

6.1. Model układu

Aby mieć możliwość testowania regulatora bez konieczności każdorazowej zmiany kodu źródłowego oraz ryzyka uszkodzenia układu postanowiono wyznaczyć model układu, a następnie przeprowadzić badania symulacyjne w programach *MATLAB* i *Simulink*. Na podstawie modelu możemy również dobrać parametry regulatora, które zapewnią asymptotyczną stabilność oraz szybkie pozycjonowanie układu. Rozpoczęto od wyznaczenia transmitancji serwomechanizmów. Wiadomo, że w serwomechanizmie jest regulator proporcjonalny.



Rys. 6.1. Schemat blokowy serwomechanizmu.
Źródło: Własne

$G_o(s)$ jest transmitancją silnika prądu stałego, gdy wejściem jest napięcie, a położenie kątowe wału silnika:

$$G_o(s) = \frac{1}{s(Ts+1)} \tag{6.1.1}$$

K jest wzmocnieniem regulatora proporcjonalnego.
Transmitancja zastępcza powyższego układu wynosi:

$$G_s(s) = \frac{K}{Ts^2+s+K} \tag{6.1.2}$$

Ze względu na brak enkoderów w układzie, nie można przeprowadzić identyfikacji parametrów serwomechanizmów. Na podstawie obserwacji wiadomo, że w wzmocnienie regulatora jest dobrane tak, by

silnik nie oscylował wokół wartości zadanej. Powinno być ono wzmocnieniem krytycznym dla danego silnika. Aby wzmocnienie K było krytyczne, musi być ono równe:

$$K = \frac{1}{4T} \tag{6.1.3}$$

Podstawiając powyższy wzór do równania 6.1.2 otrzymuje się:

$$G_s(s) = \frac{1}{4T^2s^2 + 4Ts + 1} \tag{6.1.4}$$

Wartość stałej czasowej jest przyjęta na podstawie odpowiedzi skokowej serwomechanizmu.

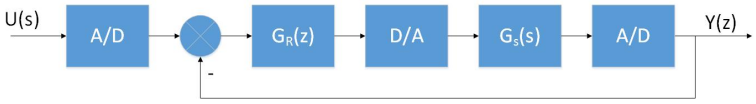
$$T = 0.05 \tag{6.1.5}$$

Następnie wykonano schemat blokowy kompletnego układu (z kamerą i implementowanym regulatorem).

Rys. 6.2. Schemat blokowy kompletnego układu.
Źródło: Własne

$G_R(z)$ jest transmitancją projektowanego regulatora dyskretnego.

Rolę przetwornika A/D pełni w układzie kamera, a przetwornika D/A sterownik serwomechanizmów. Schemat ten można przedstawić w następującej, równoważnej postaci.



Rys. 6.3. Równoważny schemat kompletnego układu.
Źródło: Własne

Transmitancję opisaną równaniem 6.1.2 można zapisać za pomocą równań stanu:

$$\dot{x} = Ax + Bu \tag{6.1.6}$$

$$y = Cx \tag{6.1.7}$$

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{K}{T} & -\frac{1}{T} \end{bmatrix} \tag{6.1.8}$$

$$B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{6.1.9}$$

$$C = \begin{bmatrix} \frac{K}{T} & 0 \end{bmatrix} \tag{6.1.10}$$

Wzmocnienie krytyczne regulatora w serwomechanizmach dobrane jest dla silników bez obciążenia. Jeśli silnik zostaje obciążony, to zwiększa się jego stała czasowa. Wzmocnienie więc również powinno być zmienione. Wziąć należy jednak pod uwagę, że masa użytej kamery jest mała w porównaniu do maksymalnej masy obciążenia tych urządzeń. Z tego względu można przyjąć, że stała czasowa nie zmienia się, a wzmocnienie nadal jest krytyczne. Nadal zachodzi więc równanie 6.1.3. Podstawiono je do powyższych macierzy.

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{1}{4T^2} & -\frac{1}{T} \end{bmatrix} \tag{6.1.11}$$

$$B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{6.1.12}$$

$$C = \begin{bmatrix} \frac{1}{4T^2} & 0 \end{bmatrix} \tag{6.1.13}$$

Obiekt ten można razem z przetwornikiem A/D i przetwornikiem D/A pierwszego rzędu przedstawić można jako jeden równoważny obiekt dyskretny. Wyznaczono następnie równania stanu tego obiektu [12].

$$A^+ = e^{hA} \tag{6.1.14}$$

$$B^+ = \int_0^h e^{tA} B dt \tag{6.1.15}$$

$$C^+ = C \tag{6.1.16}$$

h oznacza okres próbkowania dyskretnej części układu.

$$h = \frac{1}{60} \tag{6.1.17}$$

Obliczenia rozpoczęto od wyliczenia macierzy e^{tA}.

$$e^{tA} = \mathcal{L}^{-1}[(sI - A)^{-1}] \tag{6.1.18}$$

$$e^{tA} = \mathcal{L}^{-1} \begin{bmatrix} \frac{s+\frac{1}{T}}{(s+\frac{1}{2T})^2} & \frac{1}{(s+\frac{1}{2T})^2} \\ \frac{-\frac{1}{4T^2}}{(s+\frac{1}{2T})^2} & \frac{s}{(s+\frac{1}{2T})^2} \end{bmatrix} \tag{6.1.19}$$

$$e^{tA} = \begin{bmatrix} (1+\frac{t}{2T})e^{-\frac{t}{2T}} & te^{-\frac{t}{2T}} \\ -\frac{t}{4T^2}e^{-\frac{t}{2T}} & (1-\frac{t}{2T})e^{-\frac{t}{2T}} \end{bmatrix} \tag{6.1.20}$$

Podstawiono 6.1.20 do 6.1.14 i 6.1.15.

$$A^+ = \begin{bmatrix} (1+\frac{h}{2T})e^{-\frac{h}{2T}} & he^{-\frac{h}{2T}} \\ -\frac{h}{4T^2}e^{-\frac{h}{2T}} & (1-\frac{h}{2T})e^{-\frac{h}{2T}} \end{bmatrix} \tag{6.1.21}$$

$$B^+ = \int_0^h \begin{bmatrix} te^{-\frac{t}{2T}} \\ (1-\frac{t}{2T})e^{-\frac{t}{2T}} \end{bmatrix} dt \tag{6.1.22}$$

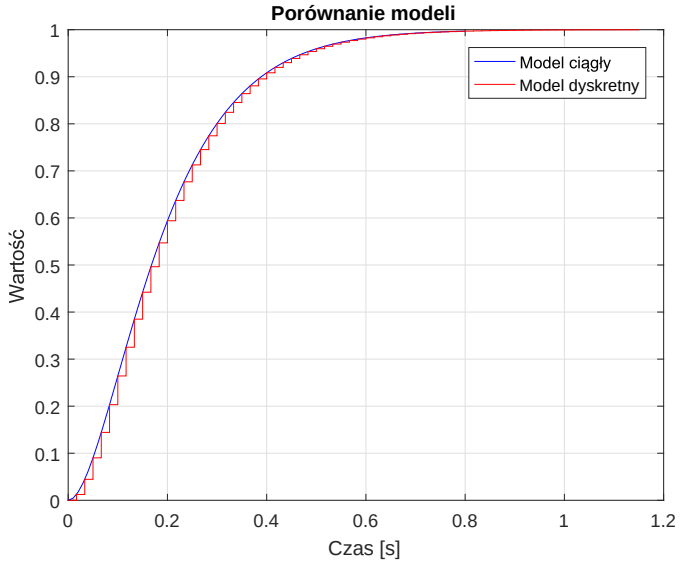
$$\int te^{-\frac{t}{2T}} = -2Tte^{-\frac{t}{2T}} + 2T\int e^{-\frac{t}{2T}}dt \tag{6.1.23}$$

$$\int te^{-\frac{t}{2T}} = -2Tte^{-\frac{t}{2T}} - 4T^2e^{-\frac{t}{2T}} \tag{6.1.24}$$

$$B^+ = \begin{bmatrix} -2T(h+2T)e^{-\frac{h}{2T}} + 4T^2 \\ he^{-\frac{h}{2T}} \end{bmatrix} \tag{6.1.25}$$

$$C^+ = \begin{bmatrix} \frac{1}{4T^2} & 0 \end{bmatrix} \tag{6.1.26}$$

Aby sprawdzić poprawność przeprowadzonych obliczeń postanowiono porównać odpowiedzi skokowe układu ciągłego i dyskretnego. Do tego celu zostało wykorzystane środowisko *MATLAB*.

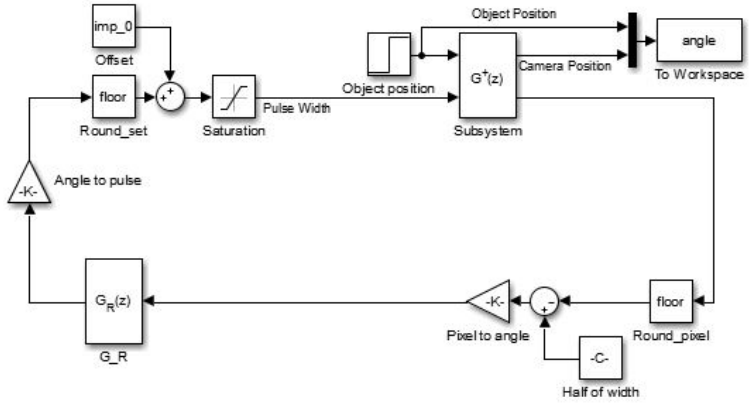


Rys. 6.4. Porównanie modelu ciągłego oraz odpowiadającego mu dyskretnego.
Źródło: Własne

Na podstawie powyższego wykresu można wnioskować, że wyliczony model jest poprawny. Należy jednak zwrócić uwagę na fakt, że w rzeczywistym układzie występują występują dodatkowo opóźnienia, które nie zostały uwzględnione w modelu. Z tego względu model nie będzie idealnie odpowiadał działaniu rzeczywistego układu.

Kompletny model układu postanowiono zaimplementować w programie *Simulink*. Uwzględnia on:

- Nasycenie wartości zadanej serwomechanizmów
- Przeliczanie współrzędnych śledzonego obiektu na kąty
- Obliczanie szerokości impulsów wysyłanych do serwomechanizmu
- Kwantyzację liczby pikseli i szerokości impulsów



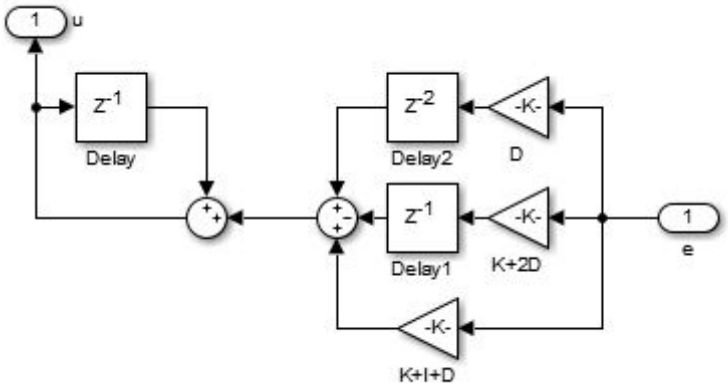
Rys. 6.5. Model układu zrealizowany w programie *Simulink*.
Źródło: Własne

6.2. Projekt regulatora

W układzie nie ma czujnika pozycji serwomechanizmów, więc algorytm regulacji musi bazować na wyjściu z poprzedniej iteracji. Algorytmem, który działa w ten sposób jest prędkościowy (przyrostowy) regulator PID. Zaimplementowany regulator ma następującą postać:

$$u(k) - u(k-1) = P \cdot (e(k) - e(k-1)) + I \cdot e(k) + D \cdot (e(k) - 2e(k-1) + e(k-2)) \quad (6.2.1)$$

Został on zrealizowany w programie *Simulink*.



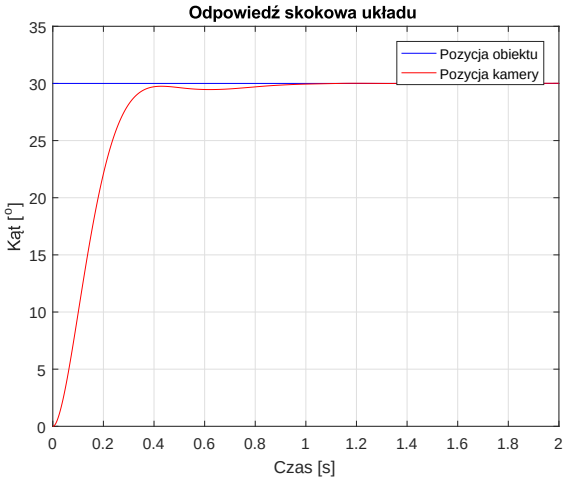
Rys. 6.6. Realizacja przyrostowego regulatora PID.
Źródło: Własne

Na podstawie odpowiedzi modelu dobrano następujące wartości parametrów:

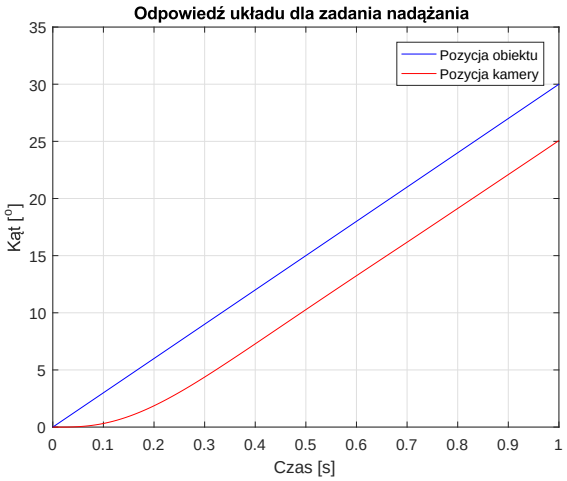
$$P = 1 \quad (6.2.2)$$

$$I = 0.1 \quad (6.2.3)$$

$$D = 0.2 \tag{6.2.4}$$



Rys. 6.7. Odpowiedź skokowa układu z dobranym regulatorem.
Źródło: Własne



Rys. 6.8. Odpowiedź układu dla zadania nadążania.
Źródło: Własne

Na wykresie 6.7 widać, że układ szybko reaguje na skokowe zmiany pozycji śledzonego obiektu. Z kolei na wykresie 6.8 zauważyć można, że obecny jest uchyb ustalony dla zadania nadążania. Aby się go pozbyć konieczne byłoby zastosowanie w układzie dodatkowego członu całkującego szeregowo użytym regulatorze, co zwiększyłoby rząd stosowanego rozwiązania. Użyte parametry mogą zostać skorygowane w rzeczywistym układzie w zależności od rezultatów i wpływu pominiętych efektów.

7. Tor wizyjny z algorytmem Mean-shift

Przekształcono tor wizyjny zbudowany w rozdziale 5 tak, by źródłem położenia obiektu był gotowy algorytm Mean-shift, który został opisane w sekcji 2.2. Rozpoczęto od przystosowania otrzymanego modułu tak, by bardziej nadawał się do celów projektu. Najpierw zmieniono sposób podawania rozmiaru okna oraz początkowego jego położenia z inicjalizacji rejestrów odpowiednimi wartościami na parametry modułu. Dzięki temu podczas testów nie trzeba generować ponownie modułu w celu zmiany powyższych parametrów. Oprócz tego dodano sygnał wyjściowy informujący o wyznaczeniu nowego położenia środka obiektu. Używany on jest do wysyłania współrzędnych środka do procesora oraz do zgłoszenia przerwania w procesorze.

Użyty algorytm Mean-shift wylicza wartości prawdopodobieństwa dla pikseli na podstawie histogramu ich otoczenia i histogramu początkowej ramki. Działa on na obrazie w skali szarości. Jego działanie może być więc w dużej mierze zależne od użytej metody konwersji przestrzeni barw RGB. Postanowiono najpierw użyć wartości składowej Y z przestrzeni barw YUV. Obliczana jest ona następująco:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{7.0.1}$$

Bierze się w ten sposób pod uwagę, że oko ludzki jest najbardziej wyczulone na kolor zielony, a najmniej na kolor niebieski. Do wykonania powyższej konwersji zostały użyte 3 moduły mnożarki w układzie FPGA. Oprócz wartości piksela i sygnałów synchronizacyjnych na wejście podawany jest również sygnał ustawienia histogramu obiektu. Zapisywany jest wtedy histogram obiektu aktualnie znajdującego się w oknie. Sygnał ten został podłączony do przełącznika znajdującego się na płytce ZYBO. Wyjście modułu realizującego śledzenie zostało wyprowadzone do modułu, który na obrazie wyjściowym zaznacza aktualne okno.

Najpierw sprawdzono działanie zaimplementowanego systemu podając na wejście HDMI obraz z komputera PC. Było nim czarne koło na białym tle. Przesuwano koło i sprawdzano, czy zaznaczone okno będzie się przesuwać razem z nim. Na podstawie wyników stwierdzono, że algorytm działa, lecz gubi obiekt, kiedy ten porusza się zbyt szybko, lub wykona jakiś gwałtowny ruch. Może to być bardzo problematyczne w trakcie śledzenia rzeczywistego obiektu. Dla obrazu z kamery nie będzie również tak dobrze rozróżniony obiekt od tła. Można się więc domyślać, że algorytm ten dużo gorzej będzie działać w rzeczywistym układzie.

Następnie na wejście HDMI podano obraz z kamery używanej w projekcie. Śledzono zielony prostokątny element. Problemem okazał się wspomniany brak dobrego rozróżnienia obiektu od tła. Zmiana położenia obiektu, a w efekcie zmiana oświetlenia powodowała zmianę śledzonego obiektu na element otoczenia.

Aby zwiększyć efektywność testowanego algorytmu zmieniono sposób wyznaczania obrazu w skali szarości. Zamiast obliczania składowej Y przestrzeni YUV do modułu śledzącego podano składową H przestrzeni barw HSV. Odpowiada ona za kolor obiektu, więc powinna dać lepsze wyniki gdy obiekt ma inny kolor niż otoczenie [13].

$$V = max(R, G, B) \tag{7.0.2}$$

$$S = \begin{cases} \frac{V - min(R,G,B)}{V} & \text{jeśli } V \neq 0 \\ 0 & \text{jeśli } V = 0 \end{cases} \tag{7.0.3}$$

$$H = \begin{cases} 0 & \text{jeśli } V - min(R,G,B) = 0 \\ \frac{60(G-B)}{V - min(R,G,B)} & \text{jeśli } V = R \\ \frac{60(B-R)}{V - min(R,G,B)} + 120 & \text{jeśli } V = G \\ \frac{60(R-G)}{V - min(R,G,B)} + 240 & \text{jeśli } V = R \end{cases} \tag{7.0.4}$$

$$H \leftarrow \begin{cases} \frac{H+360}{360} & \text{jeśli } H < 0 \\ \frac{H}{360} & \text{jeśli } H \geqslant 0 \end{cases} \tag{7.0.5}$$

Zmiana ta podyktowana jest faktem, że śledzony obiekt jest koloru zielonego, a w otoczeniu występuje niewiele elementów tego koloru. W wyniku opisanej zmiany algorytm Mean-shift zaczął dużo lepiej śledzić testowy obiekt. Gubiony był on w wyniku zbyt nagłego ruchu obiektu lub gdy zmieniał się kolor tła.

8. Implementacja algorytmu KLT

Jak opisywano w sekcji 2.5 najlepiej z zadaniem śledzenia w modelu programowym poradził sobie algorytm KLT. W związku z tym postanowiono przeprowadzić próbę jego implementacji w układzie Zynq. Należy zauważyć, że celem autora było zaimplementowanie jak największej części algorytmu w logice rekonfigurowalnej, aby zmniejszyć ilość obliczeń potrzebnych do wykonania w procesorze oraz jak najwięcej z nich wykonywać równolegle. W efekcie system ma działać szybko, wymagać małej ilości energii oraz będzie go można łatwo rozbudować lub zmienić zamieniając tylko odpowiednie moduły w diagramie blokowym programu *Vivado*.

Metoda śledzenia KLT działa na obrazie w skali szarości. Konwersję przeprowadzamy tak jak dla metody Mean-shift, z użyciem wzoru 7.0.1. Zgodnie z przebiegiem algorytmu przestawionym w sekcji 2.4, działanie algorytmu rozpoczyna się od wyboru punktów charakterystycznych spełniających warunek na wartości własne hesjanu. Aby to zrobić najpierw musimy wyliczyć gradient obrazu wejściowego. Jest on obliczany jako konwolucja dwuwymiarowa obrazu wejściowego z następującymi maskami:

$$M_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{8.0.1}$$

$$M_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{8.0.2}$$

Wyniki powyższych splotów są następnie podnoszone do kwadratów i mnożone przez siebie. W wyniku otrzymuje się macierz hesjanu dla aktualnego piksela. Zdecydowano, że powyższe wartości będą wyliczane dla całego obrazu w sposób potokowy. Aby to zrealizować należy zaimplementować również linię opóźniającą wartości pikseli oraz sygnały synchronizacyjne z użyciem pamięci BRAM, gdyż do wykonania powyższych działań potrzebujemy wartości pikseli z wcześniejszych linii obrazu wejściowego. Należy również wyznaczyć latencję użytych sumatorów i mnożarek, by opóźnić sygnały synchronizacyjne o odpowiednią liczbę taktów zegara.

Aby wyznaczyć punkty charakterystyczne następnie należy wyliczone wartości hesjanów poddać filtracji filtrem Gaussowskim. Jądro dwuwymiarowego filtru Gaussowskiego o wymiarze 3x3, wartości

oczekiwanej równej $(0, 0)$ i odchyleniu standardowym równym 1 wynosi:

$$M_G = \begin{bmatrix} 0.0751 & 0.1238 & 0.0751 \\ 0.1238 & 0.2042 & 0.1238 \\ 0.0751 & 0.1238 & 0.0751 \end{bmatrix} \tag{8.0.3}$$

Zwykły filtr Gaussowski wymagałby jednak dużej ilości zasobów (kilku mnożarek), więc postanowiono zaimplementować tzw. dyskretną wersję powyższego jądra, która ma postać [14]:

$$M_G = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{8.0.4}$$

Powyższy filtr w porównaniu do 8.0.3 nie wymaga użycia mnożarek. Zamiast tego wykorzystujemy przesunięcia bitowe. Implementacja wymaga kolejnej linii opóźniającej oraz dodatkowego opóźnienia sygnałów synchronizacyjnych.

Na podstawie wyników opisanego powyżej splotu obliczamy funkcję celu, czyli wartość mniejszej wartości własnej hesjanu. Hesjan ma postać:

$$H = \begin{bmatrix} H_{xx} & H_{xy} \\ H_{xy} & H_{yy} \end{bmatrix} \tag{8.0.5}$$

Wyznaczono analitycznie jego wartości własne.

$$\begin{vmatrix} H_{xx} - \lambda & H_{xy} \\ X_{xy} & H_{yy} - \lambda \end{vmatrix} = 0 \tag{8.0.6}$$

$$(H_{xx} - \lambda)(H_{yy} - \lambda) - H_{xy}^2 = 0 \tag{8.0.7}$$

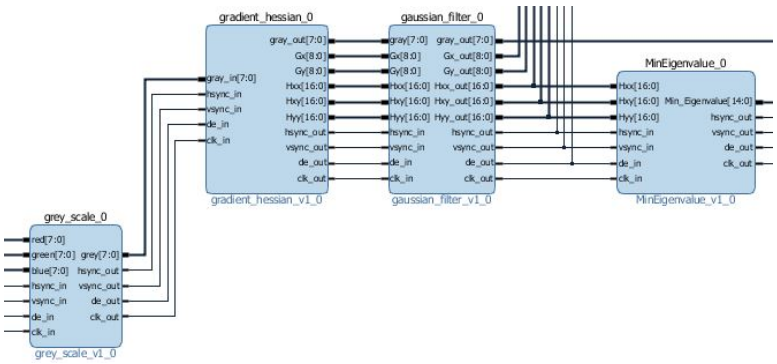
$$\lambda^2 - \lambda(H_{xx} + H_{yy}) + H_{xx}H_{yy} - H_{xy}^2 = 0 \tag{8.0.8}$$

$$\Delta = (H_{xx} - H_{yy})^2 + 4H_{xy}^2 \geqslant 0 \tag{8.0.9}$$

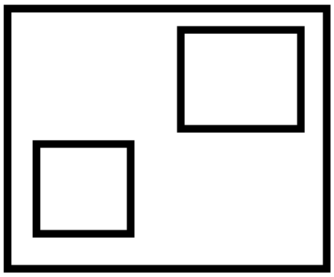
Mniejsza z jego wartości własnych wynosi więc:

$$\lambda = \frac{H_{xx} + H_{yy} - \sqrt{\Delta}}{2} \tag{8.0.10}$$

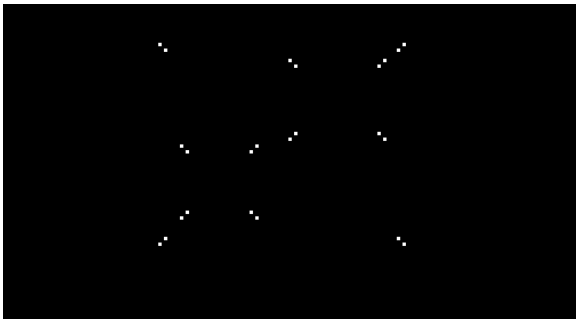
Piksele dla których obliczona wartość funkcji celu jest największa są punktami charakterystycznymi. Ze wzorów 8.0.9 i 8.0.10 widać, że obliczenie wartości własnej wymagać będzie modułów mnożarki, sumatorów oraz obliczania pierwiastka podanej wartości. Obliczenia te zostały zrealizowane za pomocą modułów dostępnych w programie *Vivado*.



Rys. 8.1. Fragment schematu blokowego, realizujący opisane powyżej operacje.
Źródło: Własne



Rys. 8.2. Przykładowy obraz wejściowy.
Źródło: Własne



Rys. 8.3. Obraz wyjściowy odpowiadający wejściowemu.
Źródło: Własne

Nie udało się dokończyć implementacji algorytmu KLT. Potrzeba jeszcze zaimplementować operacje realizujące śledzenie okna wokół wybranych punktów charakterystycznych, a więc obliczające wartość przesunięcia z równania 2.4.7 oraz iteracyjne wyznaczanie przesunięcia za pomocą wspomnianego wzoru.

9. Podsumowanie

Bibliografia

[1] Wikipedia. *Video Tracking*. *https://en.wikipedia.org/wiki/Video_tracking*. Dostęp: 12.10.2016.

[2] Nicole M. Artner. *A Comparison of Mean Shift Tracking Methods*. 2008.

[3] Gary R. Bradski. „Computer Vision Face Tracking For Use in a Perceptual User Interface". W: *Intel Technology Journal* (1998).

[4] Amir Mukhtar i Likun Xia. „Target Tracking Using Color Based Particle Filter". W: *IEEE* (2014).

[5] Tomasz Meresiński. *Implementacja algorytmu śledzenia obiektów w heterogenicznym układzie Zynq*. Kraków, 2016.

[6] Tomas Svoboda. *Kanade–Lucas–Tomasi Tracking*. *http://cmp.felk.cvut.cz/cmp/courses/Y33ROV/Y33ROV_ZS20082009/Lectures/Motion/klt.pdf*. Dostęp: 19.11.2016.

[7] Kris Kitani. *KLT Tracker*. *http://www.cs.cmu.edu/~16385/lectures/Lecture23.pdf*. Dostęp: 19.11.2016.

[8] *YI Action Camera Technical Specs*. *http://www.xiaoyi.com/en/specs_en.html*. Xiaomi, Dostęp: 26.12.2016.

[9] *ZYBO FPGA Board Reference Manual*. Xilinx. 2016.

[10] Hobbyking. *PowerHD D-21HV specification*. *https://hobbyking.com/en_us/power-hd-durable-d-21hv-high-voltage-digital-car-servo-w-titanium-alloy-gears-21kg-75g-12sec.html*. Dostęp: 31.12.2016.

[11] *Pololu Maestro Servo Controller User's Guide*. Pololu. 2014.

[12] Jerzy Baranowski i in. *Teoria Sterowania. Materiały pomocnicze do ćwiczeń laboratoryjnych.* Kraków: Wydawnictwo AGH, 2015.

[13] Mateusz Komorkiewicz i Tomasz Kryjak. *Projektowanie struktury układów FPGA. Skrypt do ćwiczeń laboratoryjnych.* Kraków: Wydawnictwo AGH, 2014.

[14] Lukas Benda. *Hardware Acceleration for Image Processing*. EPFL, I&C, BIRG, 2008.