

**A G H**

**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W  
KRAKOWIE**

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

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

**Praca dyplomowa magisterska**

*Wbudowany system wizyjny do śledzenia obiektów dla  
potrzeb nawigacji bezzałogowego statku powietrznego  
(UAV)*

*Embedded vision-based tracking system for unmanned  
aerial vehicle (UAV)*

Autor: *Miłosz Mach*

Kierunek studiów: *Automatyka i Robotyka*

Opiekun pracy: *dr inż. Tomasz Kryjak*

Kraków, 2017

*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, videogram 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 godność 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) osobistnie i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

*Serdecznie dziękuję mojemu promotorowi  
Panu dr inż. Tomaszowi Kryjakowi za  
wsparcie merytoryczne, oraz bliskim za  
udzieloną pomoc i wsparcie w trakcie pi-  
sania tej pracy.*



# **Spis treści**

<b>1. Wstęp.....</b>	<b>9</b>
1.1. Cel pracy.....	10
1.2. Struktura pracy .....	10
<b>2. Koncepcja systemu śledzenia obiektów.....</b>	<b>11</b>
2.1. Platforma UAV .....	12
2.2. Autopilot.....	13
2.3. Integracja urządzeń na platformie UAV .....	14
<b>3. Zastosowanie układów rekonfigurowalnych w bezzałogowych statkach powietrznych .....</b>	<b>17</b>
3.1. Układy rekonfigurowalne w roli autopilota platformy UAV .....	18
3.2. Układy rekonfigurowalne w systemach wizyjnych dla platformy UAV .....	20
3.2.1. Zadanie detekcji i śledzenia.....	20
3.2.2. Zadanie omijania przeszkód .....	21
3.2.3. Zadanie lokalizacji drona .....	22
<b>4. Śledzenie obiektów .....</b>	<b>25</b>
4.1. Rodzaje algorytmów śledzących .....	25
4.1.1. Algorytmy różnicowe .....	26
4.1.2. Algorytmy częstotliwościowe .....	26
4.1.3. Algorytmy gradientowe .....	27
4.1.4. Algorytmy korelacyjne .....	27
4.1.5. Algorytmy śledzenia przez detekcję .....	27
4.1.6. Algorytmy śledzenia wykorzystujące filtry cząsteczkowe .....	28
4.1.7. Podsumowanie.....	28
4.2. Algorytm MeanShift.....	29
4.2.1. Konwersja przestrzeni barw RGB->HSV.....	29
4.2.2. Wektor MeanShift.....	30

4.2.3.	Jądro obszaru .....	31
4.2.4.	Gęstość jądra .....	31
4.2.5.	Współczynnik Bhattacharyya.....	33
4.2.6.	Śledzenie.....	33
4.2.7.	Warunki zakończenia algorytmu .....	36
4.3.	Algorytm HOG+SVM .....	36
4.3.1.	Konwersja RGB->GRAY i obliczenie orientacji.....	37
4.3.2.	Histogram gradientów .....	37
4.3.3.	Normalizacja.....	39
4.3.4.	Klasyfikator SVM.....	39
4.3.5.	Detekcja na ramce sekwencji wideo.....	41
<b>5.</b>	<b>Implementacja modelu programowego.....</b>	<b>45</b>
5.1.	Model algorytmu MeanShift .....	45
5.2.	Model algorytmu HoG+SVM.....	46
5.2.1.	Uczenie .....	46
5.2.2.	Rozpoznawanie.....	48
<b>6.</b>	<b>Implementacja sprzętowo-programowa systemu wizyjnego.....</b>	<b>53</b>
6.1.	Układy Zynq.....	53
6.2.	Tor wizyjny w części PL.....	57
6.3.	Implementacja algorytmu MeanShift .....	58
6.3.1.	Konwersja przestrzeni barw RGB->HSV .....	59
6.3.2.	Inicjalizacja.....	60
6.3.3.	Zapis obszaru wideo .....	61
6.3.4.	Funkcja gęstości prawdopodobieństwa .....	63
6.3.5.	Funkcja podobieństwa .....	64
6.4.	Realizacja algorytmu HOG+SVM .....	65
6.4.1.	Konwersja RGB do skali odcieni szarości .....	67
6.4.2.	Skalowanie.....	68
6.4.3.	Obliczanie gradientów .....	70
6.4.4.	Histogram gradientów .....	72
6.4.5.	Uczenie .....	76
6.4.6.	Klasyfikacja .....	77
6.4.7.	Przetwarzanie wyników.....	78

6.5.	Integracja układu FPGA z dronem .....	80
6.5.1.	Zdalne uruchomienie pracy systemu .....	80
6.5.2.	Kontrola pracy algorytmów MeanShift oraz HOG+SVM .....	81
6.5.3.	Komunikacja Zynq <-> autopilot .....	86
6.6.	Podsumowanie stworzonej architektury .....	89
7.	<b>Weryfikacja działania systemu wizyjnego .....</b>	93
7.1.	Testy symulacyjne .....	93
7.2.	Testy w układzie Zynq.....	94
8.	<b>Podsumowanie i możliwości rozwoju pracy.....</b>	99
A.	<b>Spis zawartości płyty CD.....</b>	101
B.	<b>Opis techniczny platformy .....</b>	103
B.1.	Oprogramowanie ArduPilot.....	103
B.1.1.	Tryby lotu autopilota .....	103
B.1.2.	Protokół MAVLink .....	104
B.2.	Konfiguracja aparatury radiowej .....	107
B.3.	Instrukcja powtórzenia eksperymentu śledzenia .....	108

## Streszczenie

W pracy przedstawiona została sprzętowa realizacja detekcji i śledzenia osoby na potrzeby autonomizacji bezzałogowego statku powietrznego (UAV). System wizyjny stworzono przy wieloskalowym wykorzystaniu deskryptora HOG i klasyfikatora SVM, z niezależnym wsparciem algorytmu MeanShift. Zastosowanie potokowego przetwarzania obrazu pozwoliło na realizację obliczeń w czasie rzeczywistym dla sygnału wideo o rozdzielcości  $1280 \times 720$  i 60 klatkach na sekundę dla algorytmu MeanShift i 30 klatkach na sekundę dla metody HOG+SVM. Przedstawiany projekt został napisany w języku opisu sprzętu System Verilog i C++, oraz uruchomiony w układzie Zynq SoC znajdującym się na karcie uruchomieniowej PYNQ.

Dzięki wykorzystaniu heterogenicznego układu Zynq możliwe było zrealizowanie wbudowanego systemu wizyjnego do sterowania bezzałogowym statkiem latającym, realizującym zadanie utrzymywania stałej pozycji względem wykrytej osoby. Poprawne działanie systemu zostało zweryfikowane podczas praktycznych testów.

## Abstract

This paper describes a hardware realization of human detection and tracking for unmanned aerial vehicle (UAV). The vision system – based on a multi-scale approach – was created using HOG descriptor and SVM classifier, with an independent support of a MeanShift algorithm. Hardware implementation and pipelining of image processing allowed to compute  $1280 \times 720$ , 60fps video signal in real time for MeanShift, and 30fps for HOG+SVM method. The project was created in hardware description language – System Verilog – and C++ and was designed for Zynq SoC which is the part of a PYNQ board.

With a deployment on a heterogenous Zynq platform it was possible to create an embedded vision system, which allowed to control an unmanned aerial vehicle with an intention of following the detected person. The correct operation of this system was verified during practical tests.

# 1. Wstęp

Bezzałogowe statki powietrzne, potocznie nazywane dronami, zyskują w ostatnich latach coraz większe zainteresowanie. Według raportu portalu Business Insider z czerwca 2016 roku [1], światowy rynek bezzałogowych statków powietrznych powiększy swoją wartość z niecałych 6 mld \$ w 2013 roku do prawie 13 mld \$ w roku 2024. Biorąc pod uwagę podstawowy podział maszyn – ze względu na przeznaczenie – raport ten wskazuje na rosnący trend zarówno w przypadku przemysłu obronnego, jak i cywilnego. Można uwzględnić zastosowania m.in. w następujących branżach:

- transport – dostarczanie przesyłek,
- przemysł filmowy – realizacja stabilnych ujęć lotniczych,
- rolnictwo – stosowanie oprysków, nadzór nad wypasem zwierząt,
- ekologia – pomiar zanieczyszczeń na danym obszarze,
- ratownictwo medyczne – transport krwi lub urządzeń ratujących życie, poszukiwanie osób zaginionych.

Tak gwałtowny wzrost popularności niesie ze sobą potrzebę tworzenia nowych rozwiązań, którym sprzyja ogólny rozwój technologiczny i dostępność wielu elementów elektronicznych – czujników, modułów komunikacyjnych, autopilotów; stopniowej poprawie ulega też kwestia zasilania baterijnego i związany z nią maksymalny czas pracy.

Jedno z głównych zadań, jakie stoi przed inżynierami, to autonomizacja dronów, czyli nadanie im możliwości pracy bez nadzoru człowieka. Oznacza to wyposażenie ich w odpowiednie urządzenia elektroniczne i rozwiązania służące do realizacji określonego sterowania w oparciu o zebrane informacje. Przykładem takiego zadania jest śledzenie obiektu na podstawie informacji wizywnej. Po inicjalizacji, która polega na wskazaniu celu (lub jego detekcji) następuje określenie jego położenia w kolejnych klatkach obrazu. Na tej podstawie generowane jest sterowanie ruchem drona, które w efekcie ma dążyć do utrzymania obiektu w określonej pozycji na rejestrowanym obrazie.

Biorąc pod uwagę główne systemy elektroniczne drona, większość platform bazuje na podejściu sekwencyjnym, realizowanym na mikrokontrolerach lub procesorach sygnałowych. Dodanie nowych funkcjonalności – zwłaszcza związanych z autonomizacją – często wykracza poza możliwości obliczeniowe wspomnianych układów. Ponadto ograniczenia związane maksymalną ładownością drona i pojemnością baterii wykluczają wykorzystanie większych i często energochłonnych jednostek. Z tego względu naukowcy i firmy związane z rynkiem maszyn bezzałogowych coraz częściej rozważają stosowanie układów rekonfigurowalnych.

## 1.1. Cel pracy

Celem pracy była realizacja systemu nawigacji bezzałogowego, autonomicznego statku powietrznego w oparciu o informację wizyjną. Założeniem zadania było śledzenie obiektu zdefiniowanego podczas inicjalizacji. Pierwszym etapem prac był przegląd artykułów związanych z algorytmami śledzenia zaimplementowanymi we wbudowanych systemach rekonfigurowalnych (FPGA/Zynq), przy czym preferowane były rozwiązania zbliżone do tematu pracy.

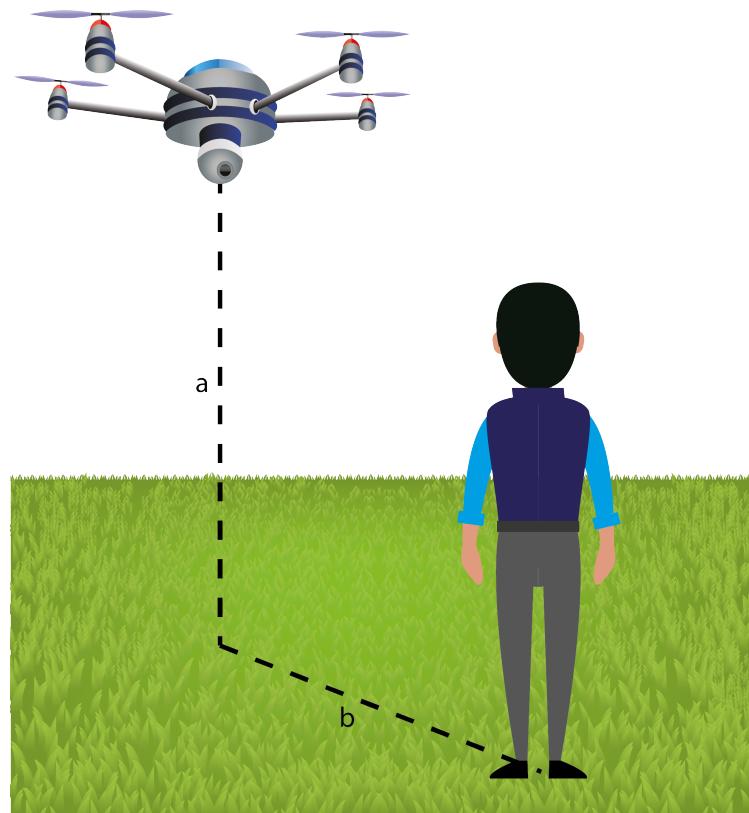
Na tej podstawie wybrano dwa podejścia i zaimplementowano je programowo, analizując ich pracę na materiale wideo zbliżonym do docelowego. Kolejnym etapem była sprzętowa implementacja algorytmów w układzie, której proces był zgodny z typową metodologią obejmującą: stworzenie modelu programowego, rozwój wymaganych modułów, testy symulacyjne oraz testy w sprzęcie. Ostatnie zadanie stanowiła integracja układu z platformą UAV, zrealizowanie komunikacji pomiędzy modułem wizyjnym, układem reprogramowalnym i autopilotem. Docelowo miał powstać prototyp bezzałogowego statku powietrznego, zdolnego podążać za wybraną osobą.

## 1.2. Struktura pracy

Rozdział drugi opisuje koncepcję śledzenia obiektów dla potrzeb bezzałogowego statku powietrznego. W rozdziale trzecim przedstawiono przegląd artykułów naukowych i dostępnych rozwiązań nawiązujących do tematu pracy. Rozdział czwarty poświęcono opisowi algorytmów śledzenia, a następnie skoncentrowano się na dokładnym przedstawieniu zagadnień związanych z metodami HoG+SVM oraz MeanShift. W rozdziale piątym pokrótkę przedstawiono zrealizowane modele programowe, które następnie przeniesiono na system wbudowany – co zostało opisane w rozdziale 5. Ponadto, część 6 zawiera informacje o konfiguracji sprzętowej oraz o wyższej warstwie decyzyjnej. Część 7 prezentuje proces testów poświęconych tworzonemu rozwiązaniu. W ostatnim rozdziale przedstawiono wnioski i wskazano możliwe kierunki rozwoju pracy.

## 2. Koncepcja systemu śledzenia obiektów

Realizacja projektu śledzenia obiektów dla potrzeb nawigacji bezzałogowego statku powietrznego dotyczy detekcji osoby w postawie stojącej. Na podstawie analizy obrazu wideo powinna być określona pozycja osoby względem drona, a w wyniku tego realizowane sterowanie, mające na celu utrzymać platformę w zadanej pozycji. Rysunek 2.1 przedstawia ideę takiego systemu.



**Rysunek 2.1.** Śledzenie osoby realizowane przez platformę UAV

Wymagane było tu zdefiniowanie kilku założeń:

- detekcja osoby znajdującej się jedynie w bezpośrednim otoczeniu drona, wewnątrz okręgu o promieniu do 7 metrów

- śledzenie poprzez ruch całej platformy – kamera jest nieruchoma, a jej pozycję dodatkowo stabilizuje gimbal kompensując wychylenia drona,
- śledzenie w przestrzeni trójwymiarowej, z zadaną pozycją:
  - wysokość  $a$ : około 1.5m od ziemi
  - odległość  $b$ : około 4m od osoby
  - przesunięcie względem osoby  $c$ : 0m, tj. osoba powinna znajdować się w centrum obrazu rejestrowanego przez kamerę
- automatyzacja misji – do zadań użytkownika należy jedynie wydanie rozkazu startu z ziemi i zakończenia pracy, skutkującego lądowaniem
- możliwość awaryjnego przejęcia manualnej kontroli nad dronem
- brak funkcjonalności, która zachowałaby ciągłość detekcji w przypadku pojawienia się drugiej osoby w otoczeniu głównego celu

Dodatkowym warunkiem było oparcie prac na konstrukcji opisanej w kolejnych podrozdziałach.

## 2.1. Platforma UAV

Platforma, na której realizowany jest projekt, składa się z następujących elementów:

- typ obiektu: hexacopter (rama DJI F550),
- rodzaj śmigieł: wzmacnione śmigła o oznaczeniu 9050, czyli o średnicy śmigła równej 9.0" ( 22.86cm) oraz skoku śmigła 5.0" ( 12.7cm).,
- silniki: DJI 2312/960KV sterowane kontrolerami 420 LITE,
- zasilanie: czterokomorowa bateria LiPo o nominalnym napięciu 14.8V (maksymalnym 16.8V) oraz o pojemności 6450mAh,
- kamera: Xiaomi Yi,
- gimbal: Tarot T-2D,
- aparatura radiowa: FrSky Taranis X9D Plus,
- odbiornik: FrSky X8D,

- autopilot: 3DR Pixhawk.
- platforma obliczeniowa: karta PYNQ z układem Zynq 7Z020.

Wybór komponentów, montaż platformy i jej kalibracja były częścią tej pracy i w ramach projektu SKN AVADER były dofinansowane ze środków Grantu Rektorskiego AGH 2015 oraz funduszy wydziału EAIiIB.

## 2.2. Autopilot

Każdy dron byłby bezużyteczną konstrukcją, gdyby nie serce maszyny – tzw. autopilot. W tym przypadku postanowiono wykorzystać urządzenie Pixhawk. Jest to zgodny ze standardami przemysłowymi moduł na otwartej licencji, stworzony przy współpracy z firmą 3D Robotics oraz ArduPilot Group. Posiada następujące parametry:

- procesor Cortex-M4F taktowany zegarem 168 MHz,
- sensory: trzyosiowy akcelerometr, żyroskop, kompas magnetyczny, barometr i zewnętrzny GPS,
- slot na kartę microSD,
- możliwość połączenia peryferiów (interfejsy: UART, I2C, CAN),
- 14 wyjść PWM (8 głównych z zabezpieczeniami + 6 dodatkowych).



Rysunek 2.2. Autopilot Pixhawk – widok na panel główny oraz we/wy PWM

Powyższy sprzęt jednak nie jest w pełni skonfigurowany do pracy po wyjęciu z pudełka – szczególnie, że jako produkt uniwersalny, bywa montowany na konstrukcjach o szerokim rozrurcie parametrów. Może zapewnić sterowanie śmigłowcom (tzw. multicopterom), samolotom

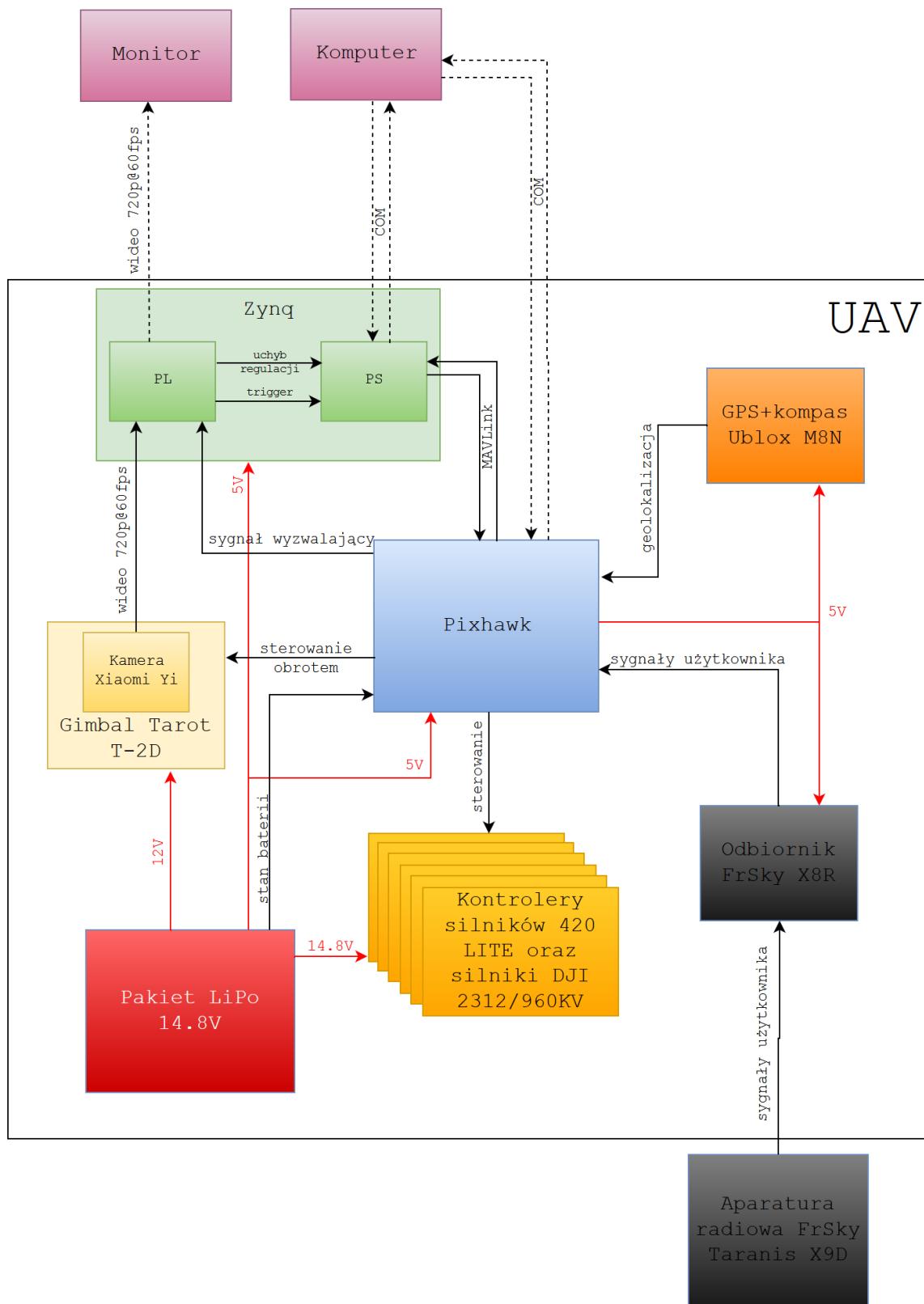
modelarskim oraz nawet łazikom. W przypadku dwóch pierwszych grup konfiguracja wiąże się ze zdefiniowaniem odpowiedniej liczby śmigieł i ich rozstawienia, typu aparatury radiowej oraz zewnętrznych urządzeń geolokalizacyjnych. Tę dość dużą elastyczność mogą zapewnić dwa główne systemy, które są wczytywane z pamięci SD i pracują w czasie rzeczywistym. Dedykowany, PX4 Flight Stack jest stworzony przez twórców modułu, oraz ArduPilot Copter (ArduCopter) – niezależny, otwarty system, który został dostosowany do platformy Pixhawk z wykorzystaniem dostępnych narzędzi deweloperskich. Ze względu na większą bazę użytkowników i dojrzałość projektu, wybrano drugie rozwiązanie.

## 2.3. Integracja urządzeń na platformie UAV

Zbudowanie drona w oparciu o gotowe komponenty nie jest zadaniem skomplikowanym. Jednak z uwagi na wymagania projektu (opisane w 2), należało dokładnie przemyśleć jego rozbudowę o układ PYNQ, zapewnienie zasilania i montaż na konstrukcji. Rysunek 2.3 przedstawia zdjęcie zmodyfikowanej, gotowej do lotu platformy. Z kolei schemat 2.4 opisuje połączenia pomiędzy urządzeniami na platformie UAV. Linie czerwone określają dystrybucję za-



**Rysunek 2.3.** Dron po niezbędnych modyfikacjach. Pod gąbką ochronną znajduje się układ PYNQ



Rysunek 2.4. Relacje pomiędzy urządzeniami na platformie UAV

silania z zamontowanej baterii (pominięto konwertery napięcia), natomiast linie czarne opisują propagację sygnałów pomiędzy urządzeniami. Dodatkowo, liniami przerywanymi zaznaczono sygnały opcjonalne wykorzystywane w trakcie naziemnej analizy systemu. Porty szeregowe komputera służą do komunikacji z konsolą zaimplementowaną na układzie Zynq, oraz do konfiguracji autopilotu poprzez aplikację MISSION Planner.

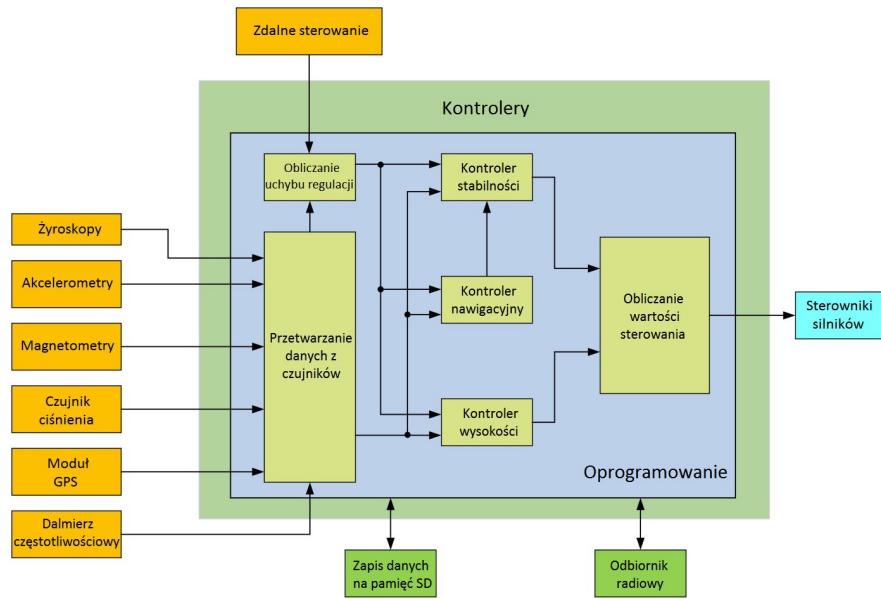
### **3. Zastosowanie układów rekonfigurowalnych w bezzałogowych statkach powietrznych**

Drony są urządzeniami, które zazwyczaj pracują w pewnej odległości od osoby nadzorującej, a często nawet poza jej zasięgiem widzenia. O ile może rodzić to pewne trudności, jest to chociażby jeden z głównych powodów udziału dronów w zastosowaniach militarnych – wówczas operator drona wojskowego nie tylko znajduje się poza obszarem zagrożenia, ale często zdalnie podejmowane przez niego decyzje wiążą się z niższym poziomem stresu. Taka platforma powietrzna musi być jednak wyposażona w elementy elektroniczne pozwalające przesyłać użytkownikowi zestaw danych o statusie drona. Podstawową grupę pełnią informacje telemetryczne z czujników (akcelerometr, żyroskop, GPS, stan baterii lub zapas paliwa w zbiorniku). Ponadto, już nawet komercyjne drony ze średniej półki cenowej umożliwiają przesłanie informacji wizywnej, a niektóre nawet ją przetwarzają. Przykładem może być DJI Spark, miniaturowa konstrukcja, która na podstawie informacji z wbudowanej kamery może wykonywać komendy wydawane gestami [17].

Poprawne działanie platformy latającej wymaga przetworzenia sporej ilości informacji, gwałtownie rosnącej wraz z dodawaniem nowych funkcjonalności. Na etapie projektowania ważne jest zatem dobranie odpowiedniej jednostki obliczeniowej. Schemat 3.1 opisuje podstawowe zależności w systemie, które należy uwzględnić definiując architekturę.

Rynek urządzeń typu UAV obfituje w rozwiązania oparte mikrokontrolery. Popularnymi autopilotami stosowanymi w komercyjnych produktach – i przede wszystkim w ręcznie tworzonych konstrukcjach – są: NAZA-M V2 firmy DJI oraz Pixhawk firmy 3DR. Oba te moduły, bazując na mikrokontrolerach ARM i dużej liczbie peryferiów, w zupełności wystarczają do zastosowań niewymagających przetwarzania dużej ilości danych, zapewniając łączność z aparaturą radiową i realizację misji opartych na predefiniowanej sekwencji ruchów.

Nieco bardziej rozbudowane są rozwiązania wykorzystujące dwa mikrokontrolery – jeden z nich jest skonfigurowany w tzw. trybie „baremetal”, który oznacza uruchomienie aplikacji



**Rysunek 3.1.** Architektura programowo-sprzętowa związana z pracą autopilota [12]

bezpośrednio na procesorze. Umożliwia to wykonywanie niskopoziomowych, krytycznych zadań, jak na przykład stabilizacja lotu uwzględniająca sterowanie pracą silników i pomiar wartości z czujników. Na drugim mikrokontrolerze uruchomiony jest system operacyjny, który stanowi platformę dla aplikacji wysokopoziomowych – takich jak algorytmy planowania trasy, stereowizja czy śledzenie celu. Jedną z takich komercyjnych platform jest „MikroKopter” stworzony przez firmę HiSystems GmbH [25].

Układy rekonfigurowalne, w miarę postępu technologicznego, stają się godną uwagi alternatywą – pomimo często wyższej ceny oferują konkurencyjną wartość poboru mocy i niezrównaną szybszą prędkość działania algorytmów. Szczególnie, jeśli podczas ich implementacji uwzględnia się możliwość zrównolegania obliczeń.

### 3.1. Układy rekonfigurowalne w roli autopilota platformy UAV

Przewaga układów rekonfigurowalnych okazuje się być widoczna już w przypadku zadania stabilizacji, a przykładem może być projekt [19], w którym częstotliwość pracy regulatora PI dla osi obrotu w układzie FPGA osiągnęła wartość 4.3MHz, w porównaniu z 0.71MHz dla rozwiązania programowego na mikrokontrolerze ARM7. Prawdziwym przełomem okazały się być układy SoC, które zintegrowały część procesorową i konfigurowalną, i w efekcie dały sporą

swobodę w sposobie realizacji projektu autopilota. Zespół badawczy odpowiedzialny za opisany wyżej projekt wykorzystał układ Zynq w projekcie kolejnego drona, gdzie w części konfigurowalnej (PL – ang. programmable logic) zaimplementowano regulację PID wymaganej do stabilizacji urządzenia, a niezależnie podejście programowo-sprzętowe pozwoliło zrealizować implementację algorytmu planowania ruchu [20]. Z kolei dla konstrukcji opisanej w publikacji [21], najważniejsze zadania rozdzielono pomiędzy 3 procesory w układzie Zynq: 2 z nich, softprocesory Microblaze, odpowiadały za stabilizację, a wyższa warstwa zadań związanych ze zdefiniowaną misją realizowana była w części procesorowej (PS – ang. processing system) opartej o architekturę ARM.

Istotnym aspektem pracy autopilota powinna być możliwość zapewnienia odpowiedniego interfejsu komunikacji z szeregiem wykorzystywanych czujników. Układy rekonfigurowalne nie tylko udostępniają ogromną liczbę wejść i wyjść, ale są też często wyposażone w sprzętowe kontrolery dla popularnych interfejsów. Ponadto, w przypadku ich niewystarczającej liczby istnieje możliwość sprzętowej implementacji własnego kontrolera. Przede wszystkim jednak, układ rekonfigurowalny pozwala przetworzyć otrzymane wartości w sposób bardziej złożony, niż pozwalałaby na to moc obliczeniowa mikrokontrolera. W publikacji [23] opisano implementację sprzętową cyfrowego kontrolera dla żyroskopów MEMS, który poprzez kwadratową demodulację sygnału wejściowego i wykorzystanie równolegle pętli synchronizacji fazy (PLL) i automatycznej regulacji wzmacnienia (AGC) poprawia dokładność działania czujnika.

Kolejnym, bardzo ważnym zadaniem autopilota jest estymacja stanu, polegająca na kompensowaniu wszelkich zakłóceń pochodzących z pomiarów. Jest ona najczęściej realizowana w formie filtra Kalmana. Jedna z publikacji [22] opisuje sprzętowo-programową implementację bezśadowego filtru Kalmana (UKF – *Unscented Kalman Filter*), której osiągi i zużycie zasobów są konfigurowalne poprzez zdefiniowanie liczby tzw. Bloków Przetwarzania (w liczbach: 1,2,5,10). Algorytm uruchomiony na urządzeniu Zynq XC7Z045 osiągał ponad dwukrotnie większą prędkość działania w porównaniu z rozwiązaniami programowymi i zużywał mniej energii (131mW dla konfiguracji z pojedynczym Blokiem).

Ostatnim aspektem pracy autopilota jest generacja sygnałów sterujących silnikami. W dronach najczęściej montowane są bezszczotkowe silniki prądu stałego, których poprawne działanie wymaga kontrolera sterującego odpowiednim przepływem prądu w uzwojeniach. Zastosowanie układu FPGA pozwala nie tylko generować sygnały PWM wysyłane do kontrolerów prędkości (ESC), ale realizować ich funkcję z pomocą niezależnego obwodu dostarczającego zasilanie. Drugą formę rozwiązania zaprezentowano w pracy [24]. Dzięki niemu osiągnięto wyższą częstotliwością pracy (12.5MHz) niż dla tradycyjnego urządzenia ESC (50Hz), w efekcie tworząc bardziej responsywną maszynę.

Powyższe rozważania przedstawiają potencjał, jaki osiągnąć mogą autopiloty bazujące na układach rekonfigurowalnych – co więcej, jednostki tego typu są już dostępne w sprzedaży.

Jedna z nich steruje quadrotorem „Phenox” [18], w którym część konfigurowalna układu z rodziny Zynq jest odpowiedzialna za generację sygnałów PWM sterujących silnikami, odbiór informacji z czujników oraz przetwarzanie obrazu i dźwięku. Kolejny, ważny przełom został osiągnięty przez firmę Aerotenna, która w 2016 roku rozpoczęła produkcję autopilotów kompatybilnych z niezwykle popularnym oprogramowaniem Ardupilot. Pierwszym z urządzeń był „OcPoc”, z układem Zynq-7000 firmy Xilinx. Kilka miesięcy później firma rozszerzyła portfolio o „OcPoC-Cyclone”, którego sercem został układ Intel FPGA Cyclone V [26].

## 3.2. Układy rekonfigurowalne w systemach wizyjnych dla platformy UAV

Inną grupę rozwiązań stanowią układy realizujące kontrolę wysokiego poziomu, czyli wykorzystanie dodatkowych informacji w celu zapewnienia określonego poziomu autonomiczności.

### 3.2.1. Zadanie detekcji i śledzenia

Jednym z podstawowych sposobów przetwarzania materiału wideo jest ekstrakcja jego cech w celu detekcji, klasyfikacji i śledzenia obiektów oraz utrzymywania orientacji kamery. W publikacji [27] porównano dwie grupy algorytmów:

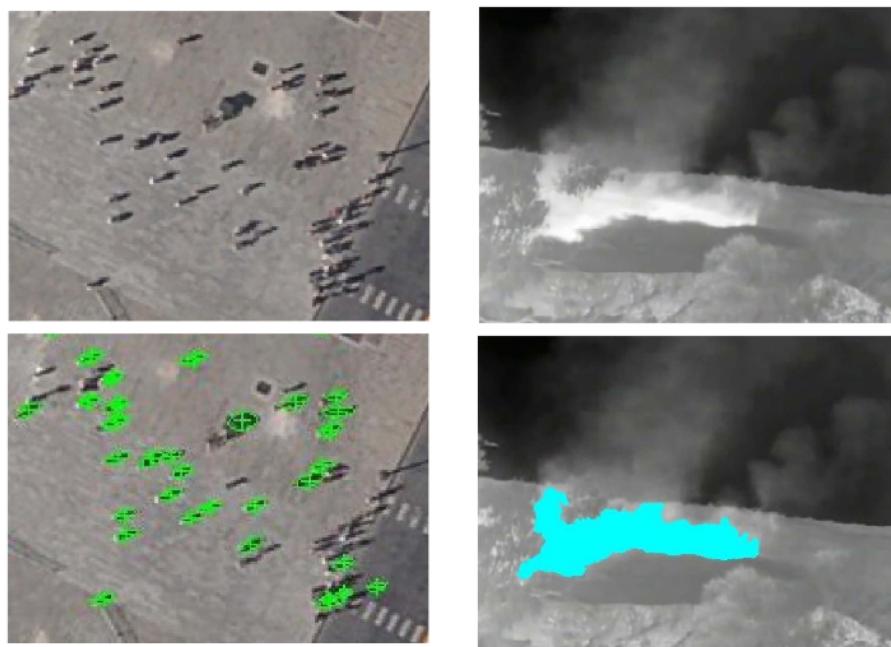
- algorytmy detekcji cech: SIFT, FAST, STAR, SURF, ORB, HCD, D-HCD
- algorytmy opisu cech: SIFT, FEAK, BRIEF, SURF, ORB, HOG, R-HOG.

Następnie w oparciu o algorytmy D-HCD oraz R-HOG stworzono zintegrowany system w układzie Zynq, który na podstawie analizy cech obrazów pochodzących z dwóch kamer ( $1080 \times 1920 @ 30\text{fps}$ ) został wykorzystany w zadaniu trójwymiarowego śledzenia scen – ale może sprawdzić się również w zadaniu śledzenia obiektów, multimodalnej rejestracji obrazów lub generowaniu struktur 3D na podstawie ruchu. System z powodzeniem poddano testom na materiałach zarejestrowanych w trakcie lotu, a przy zapotrzebowaniu na moc na poziomie 4W, częstotliwości odświeżania 30Hz i opóźnieniu wynoszącym mniej niż 3 klatki obrazu, deklasuje rozwiązanie uruchomione na laptopie z 8-rdzeniowym procesorem Intel i7 2.8GHz, na którym częstotliwość pracy uruchomionego algorytmu to zaledwie ok. 2Hz.

Inna praca [31] opisuje system detekcji ognia i ludzi na podstawie obrazów rejestrowanych na dużej wysokości (około 2km). W obu przypadkach przetwarzanie dotyczy obrazów wizyjnych oraz termowizyjnych. Na takich materiałach rzeczywista odległość pomiędzy środkami dwóch sąsiednich pikseli wynosi 12cm–25cm, zatem osoby znajdujące się na ziemi będą przedstawione za pomocą kilku pikseli.

Detekcja ludzi wymaga rozszerzenia analizy o kształt i wielkość cieni oraz charakter ruchu. Wykorzystywane są tu dwa rodzaje obszarów: jeden z nich związany jest z wykrywaną osobą; jest odpowiedzialny za odrzucenie obiektów, których rozmiar nie odpowiada oczekiwanej uśrednionej wielkości człowieka na obrazie. Dodatkową referencję stanowi obraz termowizyjny, gdzie informacja o wydzielanym cieple w określonym miejscu może zwiększyć prawdopodobieństwo obecności człowieka. Drugi obszar reprezentuje cień osoby, którego kierunek powinien być związany z pozycją słońca w danej lokalizacji i określonym momencie dnia. Informacje te można uzyskać poprzez komunikację z urządzeniami działającymi na dronie: GPS i kompasem. Parowanie obszarów obu typów pozwala wykryć osoby na obrazie.

Detekcja ognia polega na wykryciu dużej różnicy w temperaturach pomiędzy obszarem ogarniętym pożarem a jego tłem. Obszar taki jest następnie klasyfikowany pozytywnie lub negatywnie (wykorzystywany jest SVM) w oparciu o wektor cech związanych ze zmianami chromatyczności badanego obszaru. Autorzy pracy opisują układy rekonfigurowalne jako najlepszą platformę do realizacji systemu detekcji – tak ze względu na pobór mocy, wymiary, jak i moc obliczeniową przewyższającą wydajność procesorów wielordzeniowych.

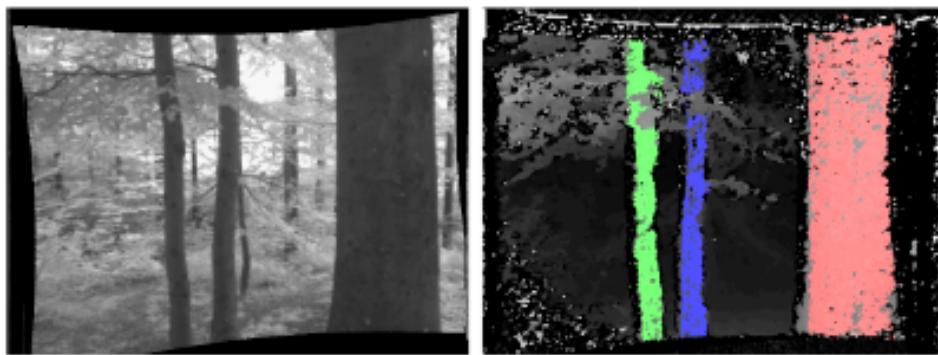


Rysunek 3.2. Przykładowa detekcja ludzi (po lewej) i ognia [31]

### 3.2.2. Zadanie omijania przeszkód

Systemy ostrzeżenia przed kolizją lub omijania przeszkód stają się powoli standardowym elementem wyposażenia dronów. Są one niezastąpione w sytuacji, gdy użytkownik traci maszynę z pola widzenia.

Jedną z najczęściej rozważanych реализаций takiego systemu na platformach latających jest wykorzystanie stereowizji. Jej działanie polega na wyznaczeniu współrzędnych punktów sceny trójwymiarowej na podstawie obrazów uzyskiwanych za pomocą co najmniej dwóch kamer, co w efekcie umożliwia określenie odległości od przeszkód lub celów. Opisana w pracy [28] implementacja algorytmu SGM (Semi-Global Matching) w układzie FPGA (XC7A100T) pozwoliła utworzyć mapę dysparcji w oparciu o obraz o parametrach  $480 \times 752 @ 60\text{fps}$ . Przetworzone dane są przechwytywane przez procesor wykorzystywany zazwyczaj w smartfonach (Samsung Exynos 4412 SoC) i zapisywane w pamięci DDR2. Praca ta została później wykorzystana na małej platformie UAV [29], gdzie spełniała rolę systemu omijania przeszkód o niskiej latencji (poniżej 2ms).



Rysunek 3.3. System omijania przeszkód [29]. Po lewej wejściowy obraz w skali szarości, po prawej mapa dysparcji

Kolejnym rozwiązaniem jest system detekcji linii zasilania [30]. W przypadku małych statków powietrznych, latających zazwyczaj na wysokości kilku, kilkunastu metrów, istnieje zwiększone ryzyko kolizji platformy UAV z instalacją elektryczną. Opracowany system wykorzystuje obraz stereowizyjny, poddając go równolegle transformacie Censusa oraz transformacji Top-Hat i określa stopień (koszt) dopasowania pomiędzy obrazami. Agregacja kosztów jest obliczana z uwzględnieniem obszaru wsparcia o krzyżowej postaci (eng. cross-based support region). Po nieznacznej korekcie informacje są zapisywane w dwuportowej pamięci RAM i przekazywane do niezależnego procesora sygnałowego, który odpowiada za wyższą warstwę logiczną i realizuje funkcje decyzyjne. Przetworzone informacje są ponownie przekazywane poprzez RAM do kontrolera wideo, który generuje obraz wyjściowy z wysegmentowanymi obszarami.

### 3.2.3. Zadanie lokalizacji drona

Inna praca [32] opisuje metodę estymacji pozycji drona w oparciu o obraz z kamery zamontowanej na statku powietrznym i skierowanej pionowo w dół. Pozwala ona wspomóc system



**Rysunek 3.4.** Przykład systemu detekcji linii zasilania zrealizowanego w [30]. Po lewej wejściowy obraz, po prawej mapa dysparcji

nawigacyjny podczas lotu w utrudnionych warunkach (obniżających skuteczność np. GPS). Pierwszym krokiem jest określenie punktów zainteresowań – wykorzystuje się w tym celu metodę Harrisa do wykrycia narożników, a następnie ZNCC (ang. Zero-mean Normalized Cross-Correlation) do określenia podobieństwa pomiędzy kolejnymi obrazami. Otrzymany zestaw informacji zawiera sporo nieprawidłowo skorelowanych par. Korekcja jest dokonywana w trakcie działania metody RANSAC (ang. Random Sample Consensus) opartej o algorytm Levenberga-Marquardta. Implementacja systemu uwzględniała przeniesienie stworzonego wcześniej modelu programowego do części PS i stworzenie sprzętowego akceleratora dla detektora Harrisa. Wymiana informacji jest realizowana przez port ACP, który zapewnia bezpośredni dostęp do pamięci cache procesora. Ostatecznie, porównano sposoby implementacji detekcji narożników metodą Harrisa. Okazuje się, że dla podejścia programowego etap ten trwa ponad 750ms, podczas gdy sprzętowa implementacja skraca ten czas do zaledwie 173ms.



## **4. Śledzenie obiektów**

Zadanie śledzenia obiektów jest zagadnieniem z dziedziny przetwarzania obrazów. Można podzielić je na dwa główne zadania. Pierwszym jest zachowanie ciągłości analizy ruchu obiektów, czyli zdolność określenia ich pozycji na każdej klatce obrazu z wystarczającą dokładnością. Drugie zadanie dotyczy tylko kamer ruchomych i polega na zdefiniowaniu ich ruchu w sposób pozwalający zachować śledzony obiekt w polu widzenia. W praktyce wyznacza się punkt na rejestrowanym obrazie (wartość zadana) i steruje pozycją oraz orientacją kamery w taki sposób, by jego odległość od śledzonego obiektu (również reprezentowanego przez punkt) była jak najmniejsza. Systemy wizyjne tego typu znajdują zastosowanie m.in. w rozpoznawaniu zachowań ludzi, wykrywaniu kolizji z pieszymi (zaawansowane systemy wsparcia kierowcy – ang. advanced driver-assistance systems, ADAS), weryfikacji pracy urządzeń przemysłowych czy nawet śledzeniu pojazdów na polu walki.

### **4.1. Rodzaje algorytmów śledzących**

Algorytmy śledzenia można podzielić na 4 główne kategorie ze względu na metodę identyfikacji obiektu na obrazie. Wyróżnia się metody:

- różnicowe,
- częstotliwościowe,
- gradientowe,
- korelacyjne,
- śledzenia poprzez detekcję,
- śledzenia z wykorzystaniem filtrów cząsteczkowych.

Ponadto, istnieje również podstawowy podział na sceny stacjonarne i ruchome. Wykorzystanie kamery stacjonarnej jest stosunkowo proste w realizacji i nie wymaga dużego nakładu

obliczeniowego (do głównych czynności należy separacja tła i segmentacja obiektów ruchomych/pierwszoplanowych). Śledzenie mogłoby być realizowane tylko w obrębie określonej sceny. W przypadku kamery posiadającej stopnie swobody, pomiędzy obecną i kolejną klatką obrazu zmianie ulec mogą wartości wszystkich pikseli. Wyklucza to stosowanie algorytmów opartych o wyodrębnianie tła. Dodatkowo, jakikolwiek ruch kamery może zmieniać położenie obiektu i jego wielkość na obrazie – wymusza to wybór zdecydowanie bardziej złożonych obliczeniowo metod.

### 4.1.1. Algorytmy różnicowe

Jest to grupa metod, które działają w oparciu o obraz różnicowy, powstały w wyniku odjęcia bieżącego obrazu od poprzedniej klatki lub wcześniej wygenerowanego modelu tła. Pozwala to na wyodrębnienie obszarów, gdzie nastąpiła zmiana, która zwykle związana jest z ruchem. W celu wyeliminowania szumu (minimalnych zmian w wartościach pikseli) stosuje się progowanie [2]. Uzyskuje się maskę binarną – piksele nieruchome (0) i ruchome (1). Analizując taką informację, możliwe jest wyznaczenie nowego położenia obiektu i zdefiniowanie przesunięcia, które nastąpiło pomiędzy klatkami.

Największą zaletą metod różnicowych jest ich prostota w zrozumieniu oraz implementacji, a także niska złożoność obliczeniowa. Z drugiej jednak strony, tak proste metody bywają zawodne w przypadku większych zakłóceń i ruchu w sąsiedztwie obiektu, błędnie interpretowanego jako właściwe przesunięcie. Ponadto, zastosowania tych metod ograniczają się jedynie do obrazów rejestrowanych za pomocą kamery stacjonarnej, co eliminuje je z dalszych rozważań w tej pracy.

### 4.1.2. Algorytmy częstotliwościowe

Metody te opierają się na interpretacji obrazu w dziedzinie częstotliwości. Istnieją różne filtry częstotliwościowe, które pozwalają na wykrycie krawędzi – więc, odpowiednio zaimplementowane, umożliwiają również identyfikację obiektu na kolejnych klatkach wideo. Praca w dziedzinie częstotliwości daje możliwość wykrycia przesunięć obiektu, które mogłyby pozostać niezauważone w przypadku stosowania pozostałych typów metod. Ze względu na swoją wysoką dokładność algorytmy te charakteryzują się nieporównywalnie większą złożonością obliczeniową, związaną z obliczeniem transformaty Fouriera i stosowaniem filtrów częstotliwościowych.

### 4.1.3. Algorytmy gradientowe

Działanie wspomnianych metod śledzenia obiektów opiera się na założeniu niewielkich zmian w luminancji (jasności, oświetleniu) oraz niewielkim przesunięciu obiektu pomiędzy kolejnymi klatkami obrazu. Istotą tych algorytmów jest znalezienie odpowiedniego przesunięcia obiektu pomiędzy kolejnymi klatkami, tak, aby wskaźnik jakości wynikający z podobieństwa obszarów był zminimalizowany. Poszukiwanie obszaru można realizować globalnie, bądź na fragmencie będącym otoczeniem śledzonego obiektu. Algorytm realizujący obliczenia lokalnie może zawodzić w sytuacjach, gdy przesunięcia są zbyt duże i wykraczają poza obszar poszukiwań, jednak taka metodyka obniża złożoność obliczeniową całego algorytmu i znajduje swoje zastosowania w określonych sytuacjach. Spośród wielu algorytmów gradientowych stosowanych do śledzenia obiektów, najpopularniejszymi są MeanShift, Camshift oraz KLT [4].

### 4.1.4. Algorytmy korelacyjne

Działanie tej grupy algorytmów polega na maksymalizacji funkcji korelacyjnej obliczanej na blokach pikseli. Podstawowym założeniem jest jeden wspólny kierunek przemieszczenia wszystkich punktów obiektu. Istotnym dla złożoności i wydajności parametrem przy implementacji tej grupy metod jest określenie rozmiaru obszaru poszukiwań – w przypadku mniejszych, problemem bywa przemieszczenie obiektu poza obszar; w przypadku tych większych istnieje ryzyko znalezienia maksimum funkcji korelacyjnej, które nie odpowiada obiektyowi. Zaletą metod korelacyjnych jest jednak mniejsza wrażliwość na zachodzące na siebie obiekty, co umożliwia śledzenie bardziej złożonych ruchów. Stosuje się je często, gdy obliczanie pochodnych (dla metod gradientowych) byłoby utrudnione – przykładowo w przypadku gwałtownego ruchu obiektu lub niewystarczającej częstotliwości próbkowania sygnału wideo. Jedną z najbardziej popularnych metod korelacyjnych jest BMA (ang. block-matching algorithm), często używana w kompresji wideo [3].

### 4.1.5. Algorytmy śledzenia przez detekcję

Działanie tej grupy algorytmów polega na niezależnym wykrywaniu określonych obiektów w kolejnych klatkach obrazu. Detekcja jest oparta na ekstrakcji kilku cech charakterystycznych obiektu, na przykład opisujących kształt. W kolejnym etapie tak stworzony deskryptor jest klasyfikowany jako poszukiwany obiekt lub odrzucany. Istotną wadą tych metod śledzenia jest duża wrażliwość na zmianę odległości od obiektu – rozwiązuje się ten problem poprzez analizę kilku przeskalowanych obrazów, jednak podnosi to ogólną złożoność obliczeniową ekstrakcji cech. Odrębną kwestią są parametry klasyfikatora – uzyskiwane w procesie uczenia i mocno wpływające na efekty działania tych metod. W skład tej grupy wchodzą m.in. algorytmy HOG

(ang. Histogram of Oriented Gradients), SIFT (ang. Scale-Invariant Feature Transform) oraz SURF (ang. Speeded-Up Robust Features) [27].

#### **4.1.6. Algorytmy śledzenia wykorzystujące filtry cząsteczkowe**

Idea działania tej grupy metod polega na przedstawieniu pozycji obiektu za pomocą zbioru losowych próbek (cząsteczek). Każda z nich związana jest z prawdopodobieństwem obecności obiektu w pozycji reprezentowanej przez tę cząsteczkę. Po otrzymaniu nowej klatki obrazu, cząsteczkom są przypisywane wagi na podstawie zgodności predykcji. W najprostszym przypadku jest to efekt porównania histogramów interpretowanych jako rozkłady gęstości prawdopodobieństwa. Następnie cząsteczki z „odpowiednio” dobrym wynikiem są pomnażane, natomiast te o złych parametrach są eliminowane z dalszych obliczeń. Wydajność tej grupy algorytmów w dużej mierze zależy od liczby zdefiniowanych cząsteczek, przy czym parametr ten ma również istotny wpływ na skuteczność śledzenia.

#### **4.1.7. Podsumowanie**

Ostatecznie w realizacji projektu postanowiono zaimplementować algorytm gradientowy: MeanShift i metodę śledzenia przez detekcję: HOG+SVM. O takim wyborze zadecydowała możliwość wykorzystania obu rozwiązań dla obrazu pochodzącego z kamery ruchomej, co w przypadku pracy drona było decyzją naturalną. Mimo, że MeanShift jest algorymem iteracyjnym (kryterium zakończenia obliczeń stanowi limit iteracji, bądź uzyskanie określonej dokładności), to odpowiednie zaprojektowanie modułu obliczeniowego w FPGA pozawala przetwarzać obraz w czasie rzeczywistym, tj. ukończyć obliczenia dla aktualnej klatki przed rozpoczęciem przetwarzania kolejnej - tak szybko, jak tylko uzyska się na jej temat komplet informacji. Metoda ta charakteryzuje się dobrymi wynikami w przypadku obiektów zmieniających orientację – jest to szczególnie przydatne w sytuacji, gdy miejscem umiejscowienia kamery jest dron – maszyna podatna na podmuchy wiatru zmieniające chwilowo jej orientację względem obiektu.

Z kolei zastosowanie algorytmu HOG+SVM jest związane z rozpoznaniem kształtu człowieka, co będzie stanowić podstawową metodę potwierdzenia jego obecności na obrazie. Dodatkową korzyścią wynikającą z implementacji algorytmu HOG+SVM jest możliwość pierwotnej detekcji postaci, na podstawie której uruchomiony zostanie system śledzenia. Ponadto, jednoczesna analiza kilku przeskalowanych obrazów na danym obszarze pozwoli wybrać najlepszy wynik i określić na tej podstawie odległość kamery od celu.

Oba algorytmy były wcześniej implementowane w układach rekonfigurowalnych w niezależnych projektach [13],[7], [14].

## 4.2. Algorytm MeanShift

MeanShift jest algorytmem zaprezentowanym po raz pierwszy w 1975 roku przez K. Fukunagę i L. Hostetlera [5]. Jest to metoda znajdowania maksimum rozkładu gęstości na pewnym ograniczonym obszarze. Iteracyjny charakter algorytmu pozwala aktualizować położenie obszaru, przemieszczając go w kierunku maksimum i rozpoczynając ponownie analizę.

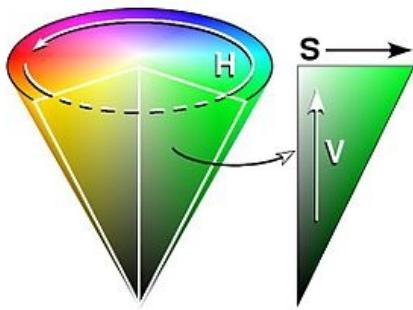
Z czasem stwierdzono, że MeanShift może znaleźć zastosowanie w systemach wizyjnych. Dysponując bowiem obszarem na obrazie, w którym znajduje się śledzony obiekt, należy zapamiętać jego funkcję gęstości prawdopodobieństwa w postaci histogramu barw. Pełnić on będzie rolę wzorca, z którym porównywane będą histogramy obszarów z kolejnych klatek obrazu (kandydaci). Funkcje gęstości prawdopodobieństwa są obliczane z uwzględnieniem jądra o zdefiniowanej postaci, którego wagą faworyzują wartości pikseli w centrum obszaru. Dzięki właściwemu doborowi jądra algorytm z odpowiednią zbieżnością znajduje przesunięcie odpowiadające maksimum rozkładu gęstości (podobieństwa). Następnie obszar śledzenia zmienia swoje położenie zgodnie z obliczonym wektorem MeanShift, by dysponując nowym zestawem pikseli, możliwe było rozpatrzenie nowego kandydata.

### 4.2.1. Konwersja przestrzeni barw RGB->HSV

Algorytm MeanShift użyty w procesie śledzenia wykorzystuje rozkład prawdopodobieństwa wybranej cechy na obrazie – w tym wypadku barwy. Jest to składowa H z przestrzeni barw HSV (ang. *Hue, Saturation, Value*). Domyślną przestrzenią do rejestracji i zapisu obrazów jest jednak RGB, zatem wymagana jest jej konwersja. O ile przestrzeń RGB reprezentowana jest przez sześć parametrami 3 kolorów: czerwonego, zielonego i niebieskiego, to przestrzeń barw HSV opisywana jest przez stożek. Jego podstawą jest koło, którego kąt opisuje barwę (H – *Hue*). Kolor czerwony jest reprezentowany przez kąty  $0^\circ$  (lub  $360^\circ$ ), zielony przez kąt  $120^\circ$ , a kolor niebieski przez  $240^\circ$ . Promień koła barw opisuje nasycenie koloru (S – *Saturation*), zaś za moc światła białego, czyli jasność (V – *Value*) odpowiada wysokość stożka. Model HSV jest lepiej powiązany ze sposobem w jaki postrzega ludzki narząd wzroku, dla którego wszystkie barwy są światłem odbitym od obiektów. Rysunek 4.1 przedstawia interpretację graficzną składowych przestrzeni HSV na stożku.

Poniższe wzory opisują sposób wyznaczenia poszczególnych składowych przestrzeni HSV w oparciu o RGB [8]. Wymagają one kilku względnie złożonych operacji dzielenia.

$$V = \max(R, G, B) \quad (4.1)$$



Rysunek 4.1. Stożkowa przestrzeń barw modelu HSV [33]

$$S = \begin{cases} \frac{V - \min(R, G, B)}{V}, & V \neq 0 \\ 0, & V = 0 \end{cases} \quad (4.2)$$

$$H = \begin{cases} 0, & \text{jeśli } V - \min(R, G, B) == 0 \\ \frac{60(G-B)}{V - \min(R, G, B)}, & \text{jeśli } V == R \\ \frac{60(B-R)}{V - \min(R, G, B)} + 120, & \text{jeśli } V == G \\ \frac{60(R-G)}{V - \min(R, G, B)} + 240, & \text{jeśli } V == B \end{cases} \quad (4.3)$$

#### 4.2.2. Wektor MeanShift

Istotą działania algorytmu MeanShift jest wyznaczanie maksimum funkcji gęstości w kolejnych iteracjach. Dla określonego obszaru obliczany jest środek ciężkości punktów, których nagromadzenie wskazuje na większy stopień podobieństwa rozpatrywanego obszaru z oryginałem (wzorcem). Do nowego środka ciężkości zostaje przesunięte aktualne położenie centrum obszaru po zakończeniu iteracji algorytmu.

To przesunięcie nosi nazwę *wektora MeanShift*, który dla zbioru  $n$  punktów  $\{x_i\}_{i=1..n}$  oraz środka obszaru w punkcie  $y_0$  może zostać zapisany jako:

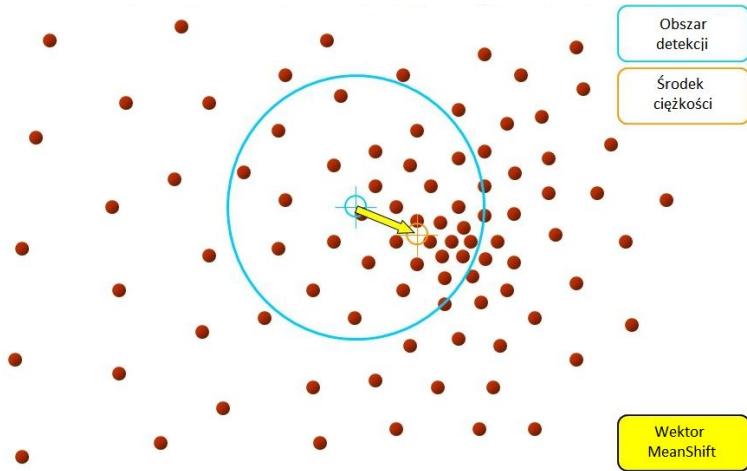
$$M(y) = \left[ \frac{1}{n} \sum_{i=1}^n x_i \right] - y_0 \quad (4.4)$$

Prostota wzoru (4.4) wynika z ujednolicenia wagi dla wszystkich punktów. Wprowadzając wagę punktu zależną od jego odległości od środka obszaru – takiej, która maleje wraz z oddalaniem się od centrum), wzór (4.4) można przepisać jako:

$$M(y) = \frac{\sum_{i=1}^n w_i(y_0) x_i}{\sum_{i=1}^n w_i(y_0)} - y_0 \quad (4.5)$$

W równaniu (4.5),  $w_i(y_0)$  są wagami dla poszczególnych punktów.

Rysunek 4.2 ilustruje przykładowe wygenerowanie wektora MeanShift, gdzie niebieskim okręgiem zaznaczono obszar podlegający działaniu algorytmu. Wszystkie punkty mają tu jednak identyczną wagę.



**Rysunek 4.2.** Graficzna interpretacja wektora MeanShift [6]

### 4.2.3. Jądro obszaru

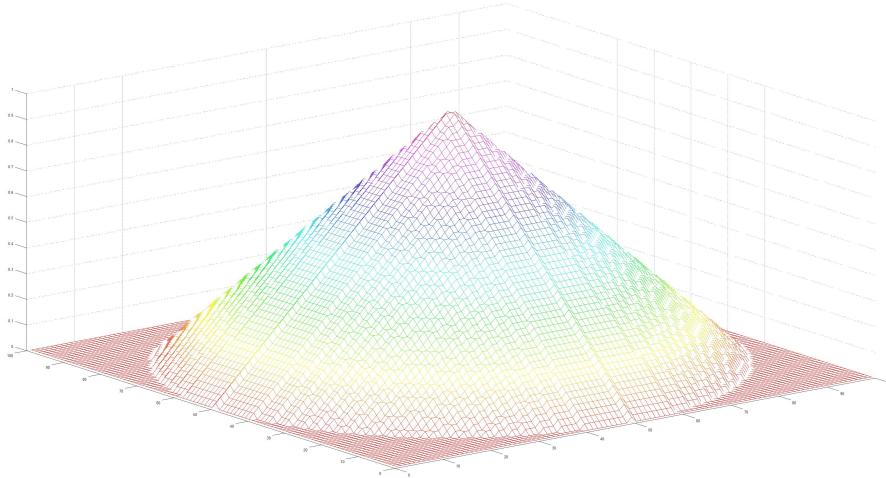
Celem wyznaczenia jądra (ang. kernel) dla obszaru o stałej wielkości jest przypisanie najwyższej wagi punktom, które znajdują się najbliżej środka obszaru. Konsekwencją tego jest nadawanie znakomitych wag punktom na jego brzegach. Postać jądra zwykle przyjmuje klasyczne postacie gęstości rozkładów probabilistycznych: między innymi rozkładu jednorodnego (wagi jednakowe), prostokątnego, Gaussa, Cauchy'ego, Epanechnikova. W niniejszej pracy zdecydowano się wykorzystać jądro trójkątne ze względu na jego dobre wyniki i jednoczesną prostotę implementacji, nie bez znaczenia podczas późniejszej realizacji sprzętowej w układzie FPGA. Jądro takie można opisać poniższym wzorem:

$$K(u) = \begin{cases} 1 - \frac{u}{b}, & \text{jeśli } \frac{u}{b} \leq 1 \\ 0, & \text{jeśli } \frac{u}{b} > 1 \end{cases} \quad (4.6)$$

gdzie:  $u$  jest odległością pomiędzy punktem a środkiem obszaru, a  $b$  jest połową jego boku (zakładając, że obszarem jest kwadrat). Rysunek 4.3 przedstawia jądro o wymiarach  $100 \times 100$  – w punkcie  $(50, 50)$  osiąga maksymalną wartość 1 (natomiast  $u = 0$ ).

### 4.2.4. Gęstość jądra

Mając zbiór  $n \times n$  punktów:  $\{x_i, y_j\}_{i=1..n, j=1..n}$  z przestrzeni  $\mathbb{R}^2$  oraz jądro  $K(u)$  dla punktu centralnego  $P = (x, y)$  można przedstawić funkcję oszacowania gęstości:



**Rysunek 4.3.** Jądro obszaru o wymiarach  $100 \times 100$  (wygenerowane w programie MATLAB)

$$f(P) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n K(\|P - P'(i, j)\|). \quad (4.7)$$

Różniczkując powyższe otrzymuje się:

$$\nabla f(P) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n (P - P'(i, j)) K'(\|P - P'(i, j)\|), \quad (4.8)$$

a po podstawieniu  $g(x)$  za  $-K'(x)$ , wzór (4.8) może zostać zapisany jako:

$$\nabla f(x) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n (P'(i, j) - P) g(\|P - P'(i, j)\|), \quad (4.9)$$

Po przekształceniu wzór może być przedstawiony jako:

$$\begin{aligned} \nabla f(x) = & \frac{1}{n^2} \left[ \sum_{i=1}^n \sum_{j=1}^n g(\|P - P'(i, j)\|) \right] \cdot \\ & \cdot \left[ \frac{\sum_{i=1}^n \sum_{j=1}^n P'(i, j) g(\|P - P'(i, j)\|)}{\sum_{i=1}^n \sum_{j=1}^n g(\|P - P'(i, j)\|)} - P \right], \end{aligned} \quad (4.10)$$

Ze wzoru (4.10) można wyodrębnić człon będący gradientem jądra:  $g(P) = -K'(P)$ . Drugą jego część stanowi wektor MeanShift z wagami odpowiadającymi wartościami gradientu jądra o środku w punkcie  $x$ . Zakładając niezerowość wyrażenia  $\sum_{i=1}^n \sum_{j=1}^n g(\|P - P'(i, j)\|)$ , wektor ten może przyjąć ostateczną formę:

$$M_s(P) = \frac{\sum_{i=1}^n \sum_{j=1}^n P'(i, j) g(\|P - P'(i, j)\|)}{\sum_{i=1}^n \sum_{j=1}^n g(\|P - P'(i, j)\|)} - P. \quad (4.11)$$

### 4.2.5. Współczynnik Bhattacharyya

Zastosowanie algorytmu MeanShift do śledzenia obiektów na sekwencji wideo wymaga znalezienia zależności pomiędzy funkcjami gęstości wzorca oraz kandydata. W tym celu należy zestawić ze sobą oba rozkłady, na przykład za pomocą współczynnika Bhattacharyya, który dla  $m$ -wymiarowych funkcji gęstości wzorca  $q_h$  oraz obszaru-kandydata  $p_h$  ze środkiem w punkcie  $P$  wyraża się wzorem [9]:

$$\rho(P) = \sum_{h=1}^m \sqrt{p_h(P)q_h} \quad (4.12)$$

Wyznaczenie przesunięcia obiektu na obrazie dla kolejnych klatek obrazu wymaga znalezienia maksymalnego współczynnika Bhattacharyya, co jest tożsame (według algorytmu) ze zidentyfikowaniem najbardziej podobnego fragmentu do oryginalnego obszaru śledzonego.

### 4.2.6. Śledzenie

Opisane wyżej jądro i jego gradient stanowią inicjalizację całego algorytmu i są obliczane jeszcze przed zdefiniowaniem wzorca. Cechą obrazu, dla której będzie liczona funkcja prawdopodobieństwa, jest kolor (H z przestrzeni barw HSV, jest to liczba z zakresu 0-359). Jeśli położenie piksela na obrazie wzorca oznaczone jest jako  $\{x_i, y_j\}_{i=1..n, j=1..n}$ , niech zdefiniowana będzie funkcja  $b : \mathbb{R}^2 \rightarrow \{1..m\}$ , która odwoływać się będzie do składowej  $H$  danego piksela.

Barwa piksela stanowi argument funkcji prawdopodobieństwa utworzonego dla  $H$  na danym obszarze detekcji. Współczynnik Bhattacharyya jest *de facto* porównaniem dwóch funkcji prawdopodobieństwa (wzorca i kandydata) – dwóch histogramów o 360 przedziałach. Podczas obliczania prawdopodobieństwa wystąpienia określonego koloru, istotną rolę odgrywać musi wartość jądra, które zwiększa wagę pikseli znajdujących się w centrum obszaru, marginalizując znaczenie tych brzegowych – które mogłyby być częścią zmennego w czasie tła. O ile w tradycyjnym histogramie wartości przedziałów zwiększa się poprzez inkrementację, to w tym przypadku zdecydowano się na powiększenie o wartość jądra odpowiadającego położeniu piksela na obszarze  $100 \times 100$ . Dla przykładowej barwy  $h$ , funkcja gęstości prawdopodobieństwa może być zdefiniowana jako:

$$q_h = C \sum_{i=1}^n \sum_{j=1}^n K(\|P - P'(i, j)\|) \delta[b(P'(i, j)) - h], \quad (4.13)$$

gdzie:  $P$  to nadal środek obszaru detekcji, a symbol  $\delta$  jest deltą Kroneckera. Współczynnik  $C$  odpowiada za normalizację  $q_h$ :  $\sum_{h=1}^m q_h = 1$ . Po przekształceniu okazuje się, że:

$$C = \frac{1}{\sum_{i=1}^n \sum_{j=1}^n K(\|P - P'(i, j)\|)} \quad (4.14)$$

W każdej iteracji dla kandydata wyznacza się funkcję gęstości prawdopodobieństwa. Uwzględniając obszar ze środkiem w punkcie  $P$ , wzór prezentuje się następująco:

$$p_h(P) = C_k \sum_{i=1}^n \sum_{j=1}^n K(\|P - P'(i, j)\|) \delta[b(P'(i, j)) - h] \quad (4.15)$$

Współczynnik  $C_h$  wyznacza się podobnie, jak dla wzorca, jednak z uwzględnieniem położenia środka obszaru kandydata,  $P$ :

$$C_k = \frac{1}{\sum_{i=1}^n \sum_{j=1}^n K(\|P - P'(i, j)\|)} \quad (4.16)$$

Następnie należy zbadać podobieństwo obu rozkładów gęstości – wykorzystywany jest w tym celu współczynnik Bhattacharyya, opisany w rozdziale 4.2.5. Rozwijając wzór (4.12) w szereg Taylora ze środkiem obszaru wzorca równym  $P_0$  można otrzymać:

$$\rho(p(P), q) \approx \frac{1}{2} \sum_{u=1}^m \sqrt{p_u(P_0)q_u} + \frac{1}{2} \sum_{u=1}^m p_u(P) \sqrt{\frac{q_u}{p_u(P_0)}} \quad (4.17)$$

Podstawienie równania (4.15) do powyższego wyrażenia daje:

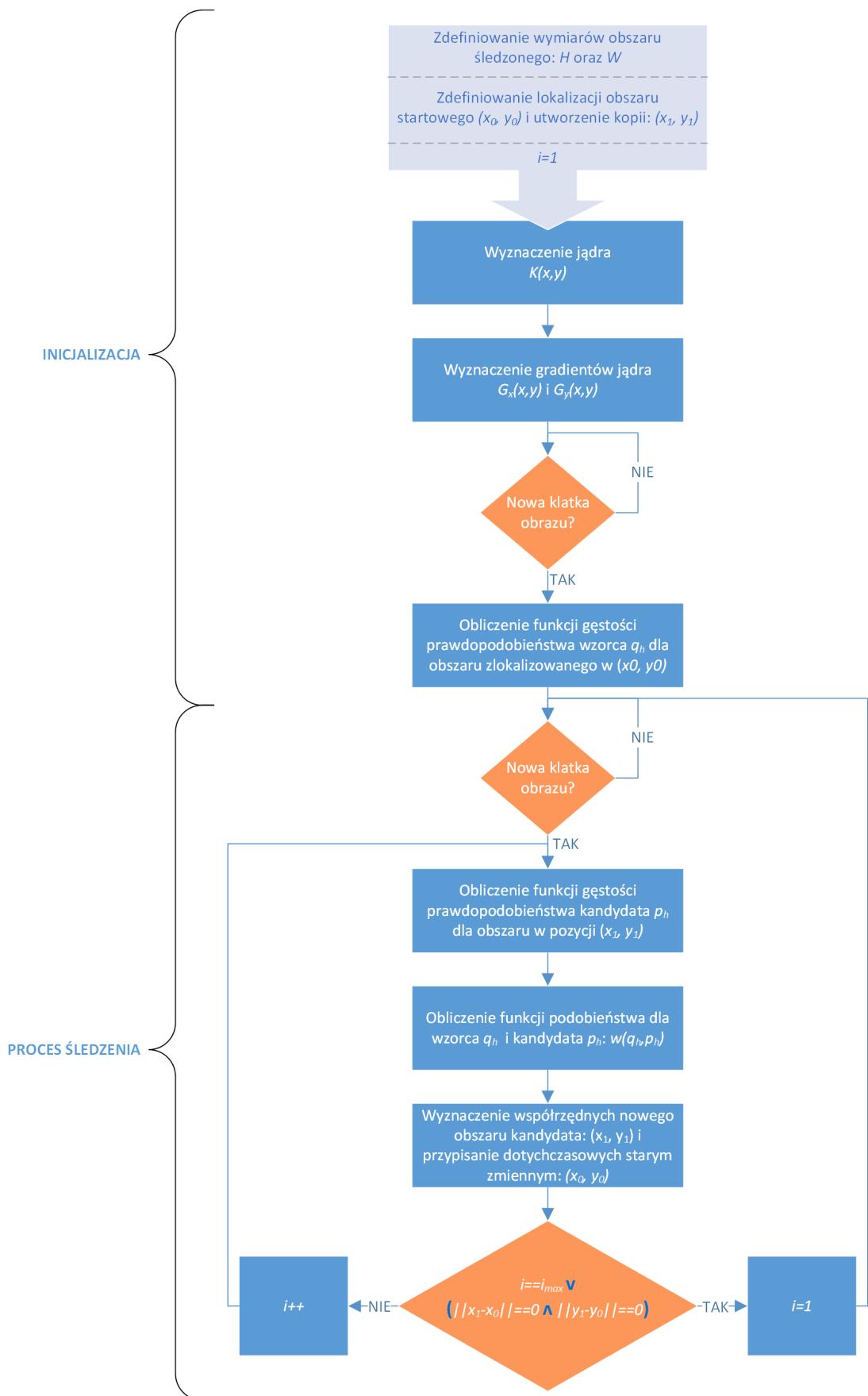
$$\begin{aligned} \rho(p(P), q) &\approx \frac{1}{2} \sum_{u=1}^m \sqrt{p_u(P_0)q_u} + \\ &\frac{C_k}{2} \sum_{i=1}^n \sum_{j=1}^n \sum_{u=1}^m \sqrt{\frac{q_u}{p_u(P_0)}} \delta[b(P'(i, j)) - h] k(\|P - P'(i, j)\|) \end{aligned} \quad (4.18)$$

Stąd można wyodrębnić:

$$w_{i,j} = \sum_{u=1}^m \sqrt{\frac{q_u}{p_u(P_0)}} \delta[b(P'(i, j)) - h] \quad (4.19)$$

We wzorze (4.18) pierwsza składowa jest niezależna od położenia środka obszaru kandydata  $P$ . Maksymalizując funkcję podobieństwa, należy skupić się na poszukiwaniu największej wartości drugiego członu. Wykorzystując wektor MeanShift z rozdziału 4.2.2 z wagami równymi wartościami funkcji podobieństwa wzorca i kandydata oraz wzór (4.19), dla każdej klatki obrazu i w każdej iteracji należy wyliczyć aktualne położenie obszaru w oparciu o względne przesunięcie dane poniższym wzorem.

$$\Delta P = \frac{\sum_{i=1}^n \sum_{j=1}^n P(i, j) \cdot w_{i,j} \cdot g(P - P'(i, j))}{\sum_{i=1}^n \sum_{j=1}^n w_{i,j} \cdot g(\|P - P'(i, j)\|)} \quad (4.20)$$



Rysunek 4.4. Schemat blokowy algorytmu MeanShift

#### 4.2.7. Warunki zakończenia algorytmu

W przypadku algorytmu iteracyjnego konieczne jest zdefiniowanie celów, które należy osiągnąć. Uzyskanie całkowitego podobieństwa pomiędzy wzorcem i kandydatem w zmiennej sekwencji obrazów jest w realnych warunkach nie do uzyskania. Chcąc przetwarzać obraz w czasie rzeczywistym, należy zakończyć przetwarzanie klatki przed momentem, w którym dane z następnej będą gotowe. Stąd dwa podstawowe warunki zakończenia algorytmu to:

1. numer iteracji == maksymalna liczba iteracji
2.  $\|P_{nm} - P_{n_{m-1}}\| < \epsilon$ ,

gdzie:  $n$  jest liczbą porządkową klatki,  $m$  numerem iteracji algorytmu MeanShift dla konkretnej klatki, a  $\epsilon$  stanowi akceptowalnie małą wartość. W momencie spełnienia jednego z powyższych warunków, akceptuje się dotychczasowo uzyskaną zmianę położenia obszaru detekcji i rozpoczyna obliczenia następnej klatki. Rysunek 4.4 przedstawia schemat blokowy algorytmu MeanShift dla sygnału wideo.

### 4.3. Algorytm HOG+SVM

Jedną z metod śledzenia przez detekcję jest algorytm wykorzystujący zestaw cech obrazu – konkretnie histogram zorientowanych gradientów (ang. *Histogram of Oriented Gradients - HOG*); rolę decyzyjną pełni klasyfikator nazywany maszyną wektorów nośnych (ang. *Support Vector Machine - SVM*). Pierwotnie, rozwiązanie to zostało przedstawione w pracy [10] z zastosowaniem w detekcji pieszych; po nieznacznych zmianach powinno ono osiągnąć dobre wyniki w realizowanym projekcie. Metodę klasyfikacji w oparciu o zestaw cech można również zastosować do detekcji innych obiektów – w szczególności takich, które charakteryzują się dość unikalnym układem krawędzi.

Założeniem metody HOG+SVM jest przedstawienie fragmentu wejściowego obrazu (okna detekcji) w formie tzw. deskryptora – wektora cech, który podlegać będzie klasyfikacji. Proces generowania deskryptora polega na podzieleniu okna detekcji na mniejsze, jednakowej wielkości obszary (komórki). Dla każdego z pikseli wewnętrz okna obliczana jest wartość gradientu, jego orientacja i moduł, po czym piksele przydzielone do jednej komórki tworzą histogram zorientowanych gradientów. Okno detekcji jest następnie reorganizowane w formę bloków składających się z współdzielonych pomiędzy sobą 4 komórek (histogramów). Każdy z bloków jest niezależnie normalizowany, po czym wszystkie łączone są w postać wektora cech.

Klasyfikator SVM działa w oparciu o współczynniki otrzymane na etapie uczenia. Proces uczenia maszyny wektorów nośnych polega na wyznaczeniu hiperpłaszczyzny, która rozdzieli deskryptory należące do dwóch klas z maksymalnym marginesem. Wymaga to obliczenia

zbioru wektorów cech na oknach detekcji, które zostały już przyporządkowane poszczególnym klasom. Klasyfikacja jest etapem prostszym, w trakcie którego obliczana jest pozycja analizowanego wektora cech względem zdefiniowanej hiperpłaszczyzny (na tej podstawie określana jest jego klasa).

### 4.3.1. Konwersja RGB->GRAY i obliczenie orientacji

Pierwszym etapem jest dostosowanie pierwotnej palety barw RGB do 8-bitowej skali szarości. Dokonuje się tego, przekształcając każdy piksel według wzoru:

$$P_{GRAY} = 0.299P_R + 0.587P_G + 0.114P_B \quad (4.21)$$

Następnie przeprowadzana jest operacja kontekstowa w celu obliczenia gradientów kierunkowych  $g_x$  oraz  $g_y$ ; używane maski to odpowiednio:  $[-1, 0, 1]$  oraz  $[-1, 0, 1]^T$ . Mając gradienty, dla każdego piksela (o koordynatach  $[i, j]$ ) obliczany jest moduł i kąt ze wzorów:

$$\begin{aligned} m(i, j) &= \sqrt{g_x(i, j)^2 + g_y(i, j)^2} \\ \theta(i, j) &= \arctg\left(\frac{g_y(i, j)}{g_x(i, j)}\right) \end{aligned} \quad (4.22)$$

### 4.3.2. Histogram gradientów

W kolejnym etapie poddawany przetwarzaniu obraz jest dzielony na kwadratowe obszary (dla jasności nazywane od teraz *komórkami*) o wymiarach  $4 \times 4$ ,  $8 \times 8$  lub  $16 \times 16$  pikseli. W każdej z komórek zostanie wyliczony niezależny histogram na podstawie orientacji gradientu. Oprócz rozmiaru komórki, kluczowym okazuje się również drugi parametr, czyli liczba przedziałów pojedynczego histogramu – w tym wypadku zdecydowano, by rozpatrywany kąt przyporządkowywać jednemu z 9 przedziałów określających kierunek (z pominięciem zwrotu). Dzielą one równo zakres kątów  $[0^\circ, 180^\circ]$  – i odpowiednio  $[-180^\circ, 0^\circ]$ . Wynika z tego, że każdy kolejny fragment o szerokości  $a = 180^\circ / 9 = 20^\circ$  będzie stanowić osobny przedział histogramu.

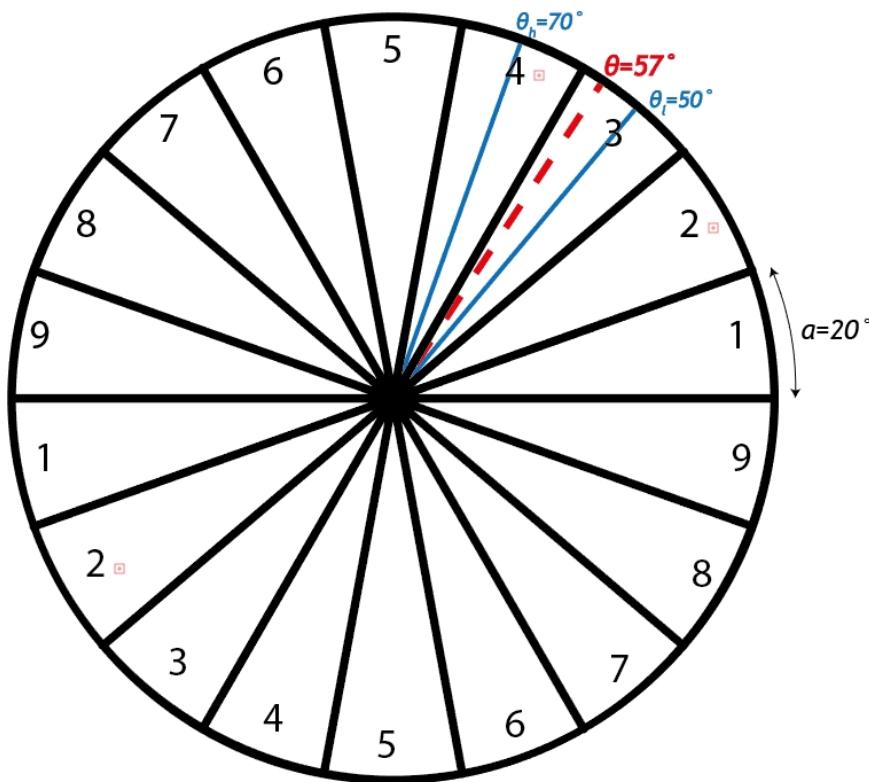
O ile typowy histogram tworzony jest poprzez inkrementację odpowiedniego licznika dla przedziału o 1, to tworzona struktura będzie wykorzystywać obliczony wcześniej moduł z gradientu. Dodatkowo, autorzy publikacji wspominają o możliwości interpolacji pomiędzy dwoma sąsiednimi przedziałami i inkrementacji poszczególnych wartości o proporcjonalne części modułu – dowiedziono eksperymentalnie, że takie działanie pozytywnie wpływa na wyniki detekcji. Jeśli zatem przyjąć, że  $\theta_h$  i  $\theta_l$  to wartości kątów będących środkami dwóch sąsiadujących ze sobą przedziałów, którym jednocześnie najbliżej do badanego kąta  $\theta$ , wówczas będzie można zdefiniować elementy  $M_h$  i  $M_l$  jako:

$$\begin{aligned} M_h(i, j) &= m(i, j) \left( \frac{\theta - \theta_l}{a} \right) \\ M_l(i, j) &= m(i, j) \left( \frac{\theta_h - \theta}{a} \right) \end{aligned} \quad (4.23)$$

Ostatecznie, następuje powiększenie wartości dwóch odpowiednich przedziałów histogramu  $H$  tworzonego w obrębie komórki zawierającej piksel o współrzędnych  $(i, j)$ :

$$\begin{aligned} H(\theta_h, i, j) &= H(\theta_h) + M_h(i, j) \\ H(\theta_l, i, j) &= H(\theta_l) + M_l(i, j) \end{aligned} \quad (4.24)$$

Koncepcję zilustrowano na rysunku 4.5. Przedstawia on okrąg z ponumerowanymi przedziałami, każdy o szerokości  $a$ . Jeśli orientacja obliczonego gradientu jest reprezentowana przez kąt  $\theta = 57^\circ$ , to przedziałami uwzględnionymi w interpolacji będą te najbliższe, o numerach 3 oraz 4. Bazując na wartości kątów w ich środkach,  $\theta_h$  i  $\theta_l$ , obliczane zostaną  $M_h$  i  $M_l$ , jako odpowiednio: 35% i 65% wartości modułu zgodnie z wzorami (4.23). Wyjątkowym zdarzeniem jest takie, w którym kąt  $\theta$  będzie znajdował w okolicy kąta  $0^\circ = 360^\circ$  (z przedziałami 1 oraz 9). Wówczas, dla wygody, w równaniach (4.23) należy użyć kątów  $\theta_l = 350^\circ$  oraz  $\theta_h = 370^\circ$ .



Rysunek 4.5. Przykładowy problem liniowej interpolacji

### 4.3.3. Normalizacja

Zazwyczaj, analizowany materiał wideo będzie rejestrowany w warunkach, w których ciężko zagwarantować równomierny poziom oświetlenia poszczególnych fragmentów obrazu. Zakłócenia te negatywnie wpływają na wyniki działania algorytmu. Z tego powodu proponuje się stosowanie normalizacji blokowej.

Podejście zakłada utworzenie struktur nazywanych *blokami*, gdzie każdy z nich obejmować ma  $2 \times 2$  sąsiednie komórki. Dla obrazu, na bazie którego utworzono  $N \times M$  komórek, powinno powstać  $(N - 1) \times (M - 1)$  bloków – przy ich generowaniu wykonuje się krok o jedną komórkę w poziomie i/lub w pionie względem poprzedniego obszaru. Rozmiar pojedynczego bloku sugeruje, że będzie się on składał z 4 histogramów, w formie wektora:

$$v = [H_{i,j}, H_{i,j+1}, H_{i+1,j}, H_{i+1,j+1}], \quad (4.25)$$

gdzie:  $i$  wiersz,  $j$  - kolumna. Następnie należy dokonać normalizacji, wykorzystując jedną z sugerowanych zależności:

$$L1 = \frac{v}{\sum_i^n v_i + \epsilon} \quad (4.26)$$

$$L1_{sqrt} = \frac{v}{\sqrt{\sum_i^n v_i + \epsilon}} \quad (4.27)$$

$$L2 = \frac{v}{\sqrt{\sum_i^n v_i^2 + \epsilon^2}} \quad (4.28)$$

$$L2_{hys} = \frac{v}{\sqrt{\sum_i^n v_i^2 + \epsilon^2}}, v \leq 0.2 \quad (4.29)$$

W powyższych równaniach  $\epsilon$  to stała o małej wartości, a  $n$  to liczba elementów w bloku (4 histogramy po 9 przedziałów  $\rightarrow 36$ ). Wektory  $L$  zebrane z całego okna detekcji budują ostateczny wektor cech, na którym pracować będzie klasyfikator.

Niech podsumowaniem rozważania będzie przykład – okno detekcji o rozmiarze  $200 \times 400$  i komórka o wielkości  $8 \times 8$ . Wynikiem opisywanej procedury będzie utworzenie aż  $25 \cdot 50 = 1250$  komórek i  $24 \cdot 49 = 1176$  bloków. Po normalizacji blokowej klasyfikator otrzymały aż  $1176 \cdot 4 \cdot 9 = 42336$  wartości do przetworzenia. Obsługa tak dużej ilości danych niosłaby za sobą konieczność pewnych optymalizacji, co będzie rozważone w rozdziale poświęconym implementacji algorytmu.

### 4.3.4. Klasyfikator SVM

Maszyna wektorów nośnych to klasyfikator binarny, który dzięki swej prostocie i skuteczności jest powszechnie wykorzystywany w procesie odróżniania klas w różnych aplikacjach,

w tym wizyjnych [11]. Jego działanie tradycyjnie można podzielić na dwie fazy: uczenie i klasyfikację. Etap uczenia to relatywnie dłuższy czasowo proces wyznaczenia hiperpłaszczyzny, która z jak najlepszym marginesem rozdzieli wejściowe wektory cech obiektów klasyfikowanych jako  $1$  od tych określonych jako  $0$  (umownie). Stosowane podeście nie powinno odbiegać od klasycznej metodologii uczenia maszynowego – dysponując zbiorom danych wejściowych, należy dokonać podziału na próbki uczące oraz testowe (w stosunku około 70:30). Każdy z tych podzbiorów (a przede wszystkim uczący) powinien posiadać zarówno obiekty zdefiniowane jako pozytywne, jak i negatywne. W niniejszej pracy, która ma na celu rozpoznanie osoby w postawie stojącej, wymagane będzie użycie obrazów o rozmiarach  $128 \times 64$  pikseli, przedstawiających taką postać w jak największej liczbie kombinacji (ale w podobnej skali), lecz także obrazów ze zróżnicowaną scenerią, na których żadnych osób jednak nie ma. Dla poprawienia ostatecznych wyników należy zadbać również o zróżnicowanie obrazów pod kątem poziomu oświetlenia, a nawet tła otaczającego postać. Etapem klasyfikacji nazwać można wykorzystanie otrzymanych w procesie uczenia parametrów do wyznaczenia położenia badanego wektora cech względem hiperpłaszczyzny. Na tej podstawie do obiektu zostanie przydzielona odpowiednia klasa.

#### 4.3.4.1. Uczenie

Aby detektor mógł działać poprawnie, klasyfikator musi dysponować odpowiednimi parametrami. Uzyskuje się je na etapie uczenia, używając odpowiedniego zestawu danych. Wzorując się na pracy Dalala oraz Triggsa, postanowiono skorzystać ze zbioru ponad 5000 obrazów udostępnionych przez twórców. Specyfikacja tego zestawu pozwala nauczyć klasyfikator rozpoznawania osoby na obrazie o wielkości  $128 \times 64$  pikseli. Wysokość przedstawionej osoby, wyrażona w pikselach, powinna oscylować wokół 95 ( $\pm 5$ ). Zgodnie z przyjętą metodologią, zestaw ten zawiera gotowy podział na próbki pozytywne i negatywne, dalej pogrupowane na zestaw uczący oraz testowy w stosunku zbliżonym do 70:30. Przykłady przedstawiono na ilustracji:

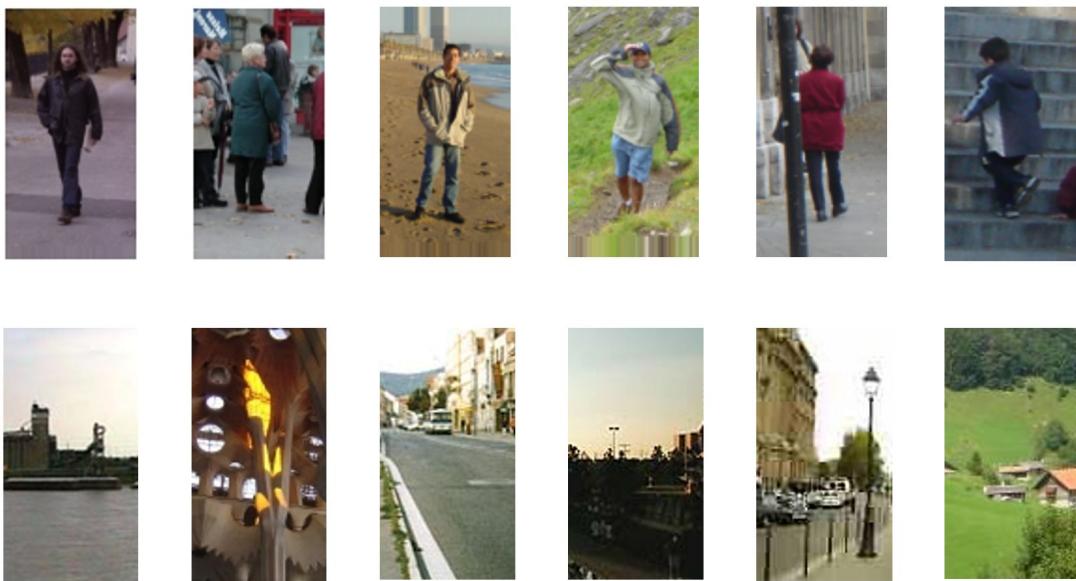
Wśród próbek pozytywnych połowa plików to duplikaty odwrócone horyzontalnie. Ma to zapewnić poprawne uwzględnienie cech osób niezależnie od orientacji względem kamery.

#### 4.3.4.2. Klasyfikacja

Etap ten jest stosunkowo prosty i stanowi główną zaletę klasyfikatora SVM. Wymaga bowiem wcześniejszego przeprowadzenia etapu uczenia oraz dostarczenia wektora cech klasyfikowanego obiektu. Obliczane jest wówczas równanie:

$$r = \sum_{i=1}^N (a_i \cdot l_i) + b, \quad (4.30)$$

gdzie:  $l_i$  to elementy wektora cech, natomiast  $a_i$  i  $b$  to parametry wyliczone na etapie uczenia.



**Rysunek 4.6.** Przykłady obrazów ze zbioru wykorzystanego przy uczeniu – na górze próbki pozytywne, na dole negatywne

W projekcie, proces uczenia i klasyfikacji oparto o funkcje dostępne w środowisku MATLAB, gdzie to równanie jest nieco zmodyfikowane:

$$r = - \sum_{i=1}^N a_i(l_i + b_i) - c, \quad (4.31)$$

gdzie:  $a$ ,  $b$  oraz  $c$  to parametry hiperpłaszczyzny wyliczone na etapie uczenia ( $a$  i  $b$  – wektory,  $c$  – skalar).

Dla obu przypadków, detekcja wskazuje obiekt jako przynależący klasy jeśli  $r > 0$ , w przeciwnym wypadku go odrzuca.

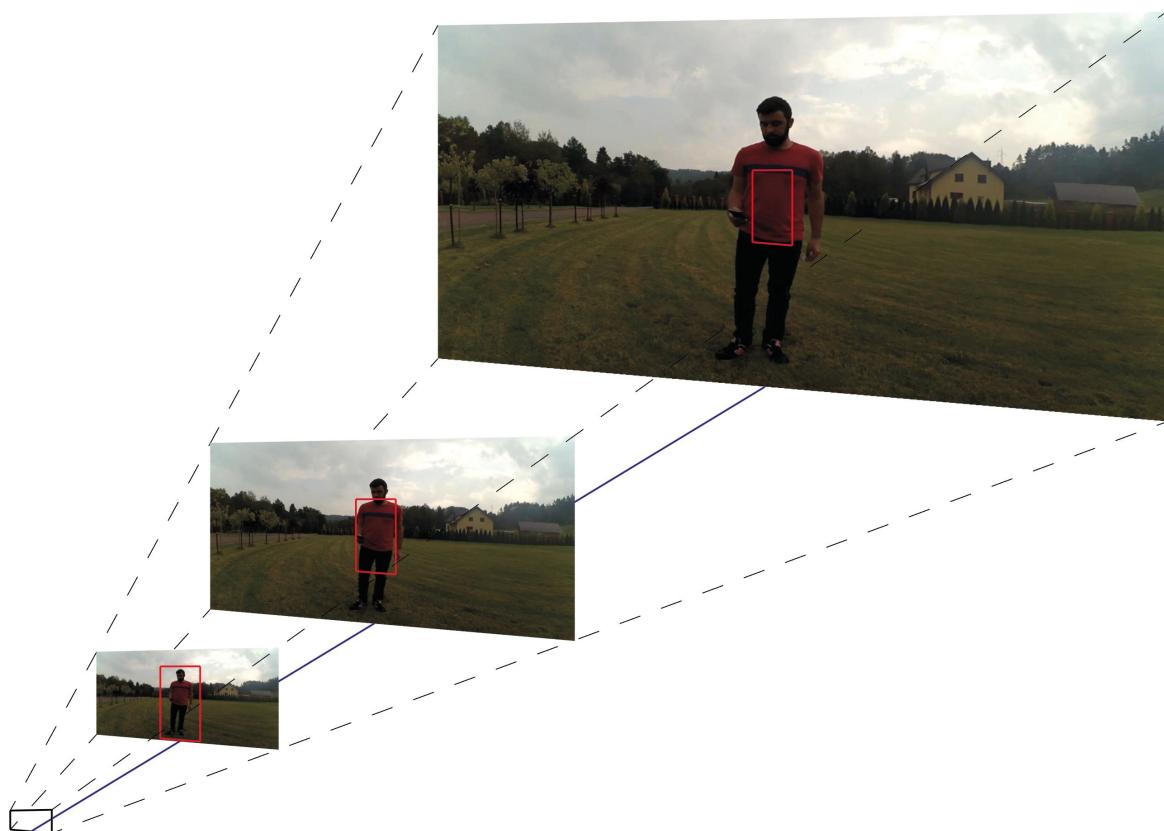
### 4.3.5. Detekcja na ramce sekwencji wideo

Omówiony klasyfikator jest w stanie rozpoznać postać znajdująca się w centrum próbki o rozmiarze  $128 \times 64$  pikseli.

Wykonanie detekcji na obrazie o dowolnym rozmiarze np.  $1280 \times 720$  wymaga zastosowania mechanizmu okna przesuwnego. Dysponując określonym obszarem zainteresowań, algorytm powinien sprawdzić każdą możliwą konfigurację położenia okna  $128 \times 64$  wewnętrz tego obszaru. Przykładowo oznacza to przesuwanie okna o 1 piksel w bok, a po dotarciu do prawej linii – o 1 piksel w dół. Takie podejście skutkuje bardzo dużą złożonością obliczeniową – dla obszaru zainteresowań o wymiarach  $144 \times 96$  i rozpatrywanego okna istnieje 512 możliwości (512 wektorów cech). W praktyce zwiększa się wykonywany krok o kilka lub kilkanaście pikseli – dla obrazu o dość dużej rozdzielczości  $1280 \times 720$  jest to akceptowalny poziom pominięcia

niektórych szczegółów. Przykład realizacji okna przesuwnego jest przedstawiony na rysunku 6.5.

Metoda HOG+SVM jest mało odporna na zmianę wielkości wykrywanego obiektu na oryginalnym obrazie, a porównując wymiary ramki ( $1280 \times 720$ ) z wielkością okna detekcji, może wydawać się to poważnym ograniczeniem. Skutecznym rozwiązaniem okazuje się być zastosowanie algorytmu HOG+SVM na kilku kopiach obrazu z różnym współczynnikiem skalowania. Metodyka detekcji w wielu skalach stanowi dobrą odpowiedź na jedno z założeń systemu, mówiące o konieczności określenia odległości kamery od rozpoznanej osoby, by poprzez odpowiednie sterowanie utrzymać zadany dystans. Z każdej skali należy wydobyć próbki obrazów o wymiarach  $128 \times 64$  i znaleźć najlepszą pozytywną detekcję. Opisują to rysunki 4.7.



**Rysunek 4.7.** Skalowanie 1:1, 1:2 oraz 1:4 z naniesionym przykładowym oknem detekcji

$128 \times 64$

Po zakończeniu klasyfikacji dla wszystkich skal następuje grupowanie wyników – w efekcie wyłaniane jest najlepiej sklasyfikowane okno i określana jest jego pozycja na oryginalnym obrazie. W przypadku przedstawionym na rysunku, klasyfikator prawdopodobnie rozpozna postać na oknie detekcji z obrazu przeskalowanego w stosunku 1:4. Stała wielkość okna detekcji  $128 \times 64$  każe wnioskować, że osoba wykryta na oryginalnym obrazie mogłaby być znacząco

oddalona od kamery. Znając przybliżoną odległość pomiędzy kamerą tak wykrytą osobą i stojącą proste twierdzenie Talesa, można oszacować jej rzeczywistą odległość dla pozytywnej detekcji w jednej ze skal. Opisuje to poniższy wzór:

$$d_r = \frac{d_o}{s_c}, \quad (4.32)$$

gdzie:  $d_r$  to aktualna odległość pomiędzy osobą i kamerą,  $d_o$  to odległość do wykrytej osoby na obrazie oryginalnym (zmierzona), a  $s_c$  to skala obrazu (w postaci ułamkowej), w którym okno detekcji uzyskało najlepszy wynik.

Szczegóły realizacji algorytmu opisano w podrozdziale 6.4.



## **5. Implementacja modelu programowego**

Pierwszym etapem weryfikującym poprawność przedstawionej koncepcji było stworzenie modeli programowych algorytmów MeanShift oraz Hog+SVM, bez warstwy nadzorującej/sterującej dronem. Wybrano środowisko MATLAB ze względu na jego powszechnie zastosowanie w branży naukowej/inżynierskiej, co skutkuje olbrzymią bazą bibliotek i materiałów pomocniczych. W tym przypadku MATLAB ułatwia pracę na obrazach lub sekwencjach wideo poprzez: wczytywanie materiału, wyświetlanie, dostęp do poszczególnych klatek oraz zapewnia wiele wbudowanych funkcji, m.in. do konwersji określonych przestrzeni barw oraz klasyfikacji z wykorzystaniem maszyny wektorów nośnych.

### **5.1. Model algorytmu MeanShift**

Skrypt rozpoczyna działanie od wyznaczenia jądra obszaru (4.6) o wymiarach  $100 \times 100$  (zgodnie ze schematem 6.4) oraz jego gradientów. Po tym następuje właściwa część algorytmu, która pracuje na wczytanym materiale wideo, przekonwertowanym do przestrzeni HSV. Obszarem śledzonym jest kwadrat  $100 \times 100$ , który dla pierwszej klatki obrazu jest zlokalizowany w miejscu obecności śledzonego obiektu. Dla tego fragmentu obliczany jest histogram barw wzorca. Następnie, dla kolejnych klatek, obliczany jest histogram kandydata. Na podstawie dwóch takich zestawów danych obliczany jest wektor MeanShift, czyli przemieszczenie maksymalizujące wartość funkcji podobieństwa obu obszarów. Dla każdej z klatek operacja ta jest przeprowadzana 10 razy, poprawiając precyzyję ostatecznego przesunięcia. Przed rozpoczęciem przetwarzania kolejnej klatki, następuje aktualizacja pozycji śledzonego obszaru. Po wykonaniu algorytmu dla zdefiniowanej liczby klatek, skrypt wyświetla film w oryginalnych barwach RGB z dorysowaną czerwoną ramką otaczającą śledzony obszar, co pozwala wizualnie sprawdzić poprawność działania całego kodu. Niestety, czas wykonania tej symulacji jest nieporównywalnie dłuższy od rzeczywistego trwania sekwencji i, przykładowo, na komputerze wyposażonym w procesor klasy i7 materiał o długości 685 klatek (ok. 11 sekund filmu) jest przetwarzany w czasie ok. 100 sekund.

Rysunek 5.1 przedstawia śledzenie obiektu w postaci czerwonej koszulki z ciemnym poziomym paskiem. Kolejne zrzuty są realizowane z odstępem 100 klatek. Zauważać można prawne działanie algorytmu nawet pomimo zmiany odległości obiektu od kamery.

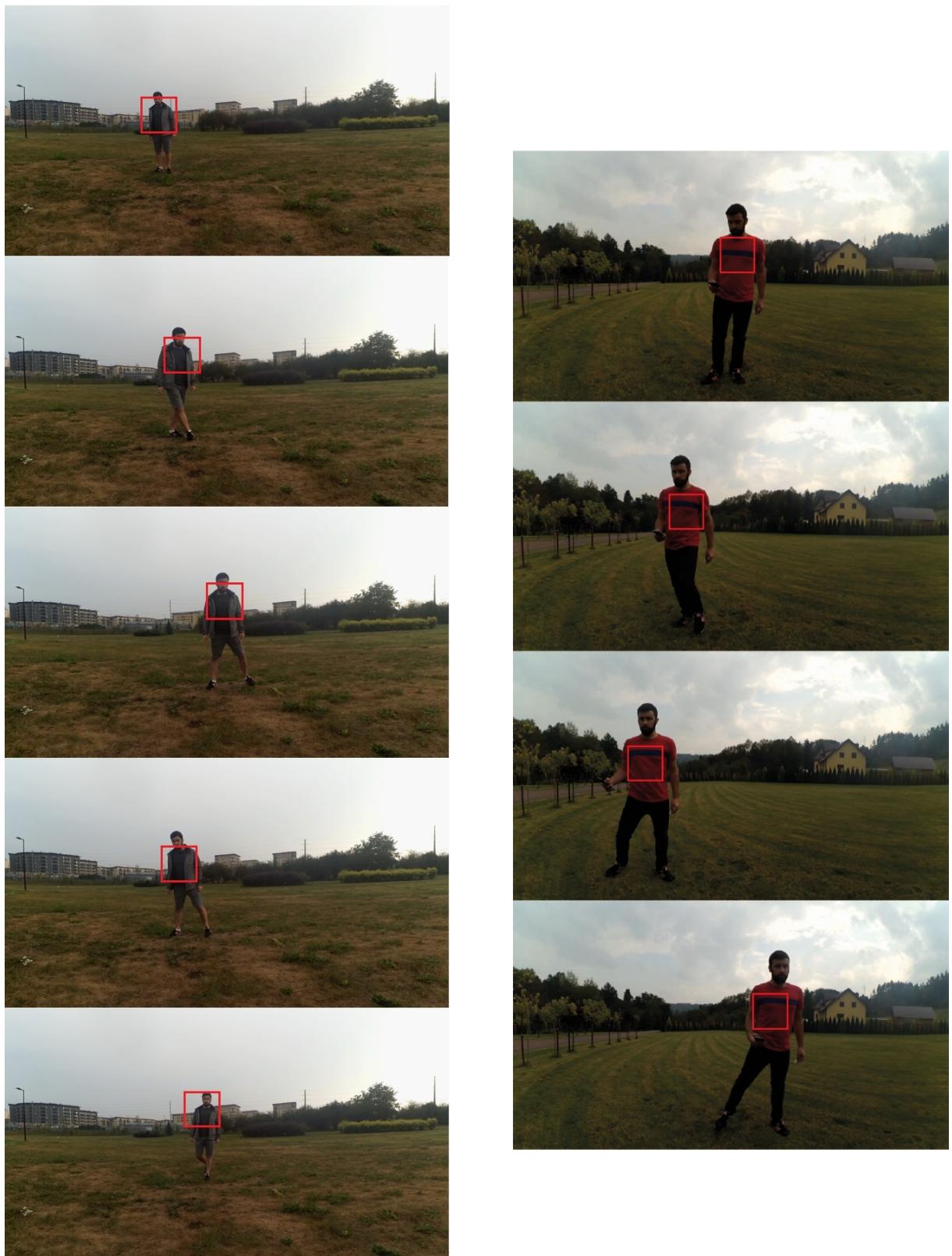
## 5.2. Model algorytmu HoG+SVM

Funkcję modelu programowego można podzielić na dwa zadania. Pierwszym jest proces uczenia klasyfikatora SVM, którego współczynniki zostaną użyte nie tylko w drugiej części, ale również w trakcie implementacji sprzętowej. Drugim zadaniem jest realizacja detekcji osoby na wielu skalach obrazu wejściowego, w oparciu o wskazany przez użytkownika punkt środkowy obszaru zainteresowań.

### 5.2.1. Uczenie

Uczenie jest dość złożonym obliczeniowo procesem przetworzenia obrazów i ekstrakcji wektorów cech, na podstawie których powstanie płaszczyzna rozdzielająca próbki pozytywne od negatywnych. Przejście przez tak wielką liczbę obrazów ma charakter jednorazowy, po którym będzie dysponować się parametrami umożliwiającymi klasyfikację. Z tego względu nie zdecydowano się na implementację modułu uczącego w układzie FPGA – tak ze względu na złożoność tego etapu, czas trwania procedury, jak i prawdopodobnie duży udział modułu w zużyciu zasobów logiki.

Wybór padł na środowisko MATLAB dysponujące funkcjami wspierającymi metodę SVM. Skrypt korzysta z plików tekstowych zawierających listy próbek do wczytania, które ostatecznie muszą być w rozmiarze  $128 \times 64$ . Wymiary próbek negatywnych mocno odbiegają od założonego wymiaru klasyfikowanego obrazu ( $128 \times 64$ ). Zdecydowano, iż w ich przypadku uczeniu poddany będzie obszar o wymiarach  $128 \times 64$  wycięty ze środka oryginałów. Proces tworzenia wektorów cech przeprowadzono zgodnie z informacjami zawartymi w rozdziale 4.3; za normę blokową obrano L2 (4.28). Na bazie tak przygotowanych próbek powstaje tablica deskryptorów oraz druga, która przechowuje odpowiadające im wartości klas (1 lub 0). Z obu tablic korzysta dostępna w pakiecie MATLAB funkcja *svmtrain*, która zwraca strukturę *svm\_struct* ze współczynnikami wykorzystywanymi w klasyfikacji. Z kolei funkcja *svmclassify* służy do klasyfikacji obrazu (a właściwie jego deskryptora). Funkcję tę użyto w procesie testowania niezależnego zestawu danych złożonego z manualnie sklasyfikowanych obrazów, by sprawdzić skuteczność klasyfikacji wyuczonego SVM. Etap testowania jest dodatkowo przydatny – pozwala eksperymentalnie zdefiniować optymalny rozmiar komórki (kwadratu pikseli), na bazie których liczony jest pojedynczy histogram. Tabela 5.1 prezentuje wyniki, wśród których uwzględniony jest błąd,



Rysunek 5.1. Śledzenie MeanShift

liczony jako stosunek liczby niepoprawnych klasyfikacji (0 zamiast i 1 zamiast 0) do liczby testowanych obrazów. Eksperyment przeprowadzono na komputerze wyposażonym w procesor Intel klasy i7, trzeciej generacji.

Rozmiar komórki [px]	Liczba elementów wektora cech	Błąd klasyfikacji [%]	Czas trwania obliczeń [s]
[2 × 2]	70308	3.4069	52min 42s
[4 × 4]	16740	2.3344	14min 08s
[8 × 8]	3780	3.3438	05min 01s
[16 × 16]	756	5.1735	02min 50s

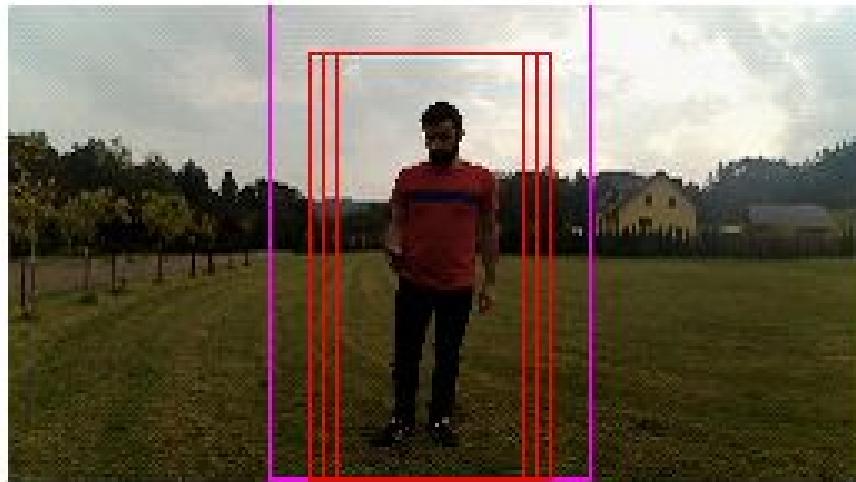
**Tablica 5.1.** Skuteczność algorytmu na zbiorze testowym w zależności od wielkości pojedynczej komórki

Okazuje się, że algorytm wykazuje największą skuteczność dla komórek o rozmiarze  $4 \times 4$ . Zdecydowano się zatem na implementację algorytmu HOG w układzie FPGA właśnie dla tej wartości parametru. Szybkie obliczenia pokazują, że dla obrazu o wymiarach  $128 \times 64$  powstanie  $32 \times 16$  komórek, a idąc dalej,  $31 \times 15$  bloków po 4 histogramy po 9 wartości – długość wektora cech będzie wynosić 16740. Oznacza to również, że po zakończonym procesie uczenia powstanie wektor współczynników o takiej długości.

### 5.2.2. Rozpoznawanie

Po uzyskaniu struktury ze współczynnikami możliwa jest klasyfikacja obszarów z obrazów wczytanych przez użytkownika. Napisany w tym celu skrypt rozpoczyna działanie od wczytania obrazu wejściowego. Po zdefiniowaniu punktu będącego środkiem obszaru zainteresowań następuje realizacja algorytmu HOG+SVM dla każdej z 5 zdefiniowanych wcześniej skal. Wielkość obszaru zainteresowań określono na podstawie potrzeb i możliwości związanych z implementacją sprzętową - dla każdej ze skal ma on rozmiar  $144 \times 96$ . Pierwszym krokiem jest obliczenie  $5 \times 9$  wektorów cech na fragmencie  $144 \times 96$  – celem jest znalezienie jak najlepszego wyniku detekcji. Wykorzystując strukturę ze współczynnikami i wzór 4.31, następuje klasyfikacja każdego obliczonego wektora cech. Proces ten jest powtarzany dla każdej z wcześniej określonych skal. Końcowym etapem jest wyświetlenie oryginalnego obrazu w kolorze, z naniesionymi konturami: obszarów zainteresowań  $144 \times 96$  oraz pozytywnych detekcji o wymiarach  $128 \times 64$  – przeskalowanych do rzeczywistego rozmiaru. Finalna detekcja jest obliczana na podstawie najlepszego (najniższego) wyniku klasyfikacji z zebranych wyników analiz wszystkich skal.

Przykłady detekcji w pojedynczych skalach zaprezentowano na rysunkach 5.2, 5.3 oraz 5.4. Różowym konturem oznaczono okno detekcji  $144 \times 96$ , natomiast na czerwono zakreślono ostateczne obszary  $128 \times 64$  pozytywnie wybrane przez klasyfikator. Przedstawiono również tabele z wynikami klasyfikacji poszczególnych deskryptorów. Pogrubione zostały pozytywne wyniki detekcji.



**Rysunek 5.2.** Detekcja na oknie wewnętrz obrazu (skala: 5)

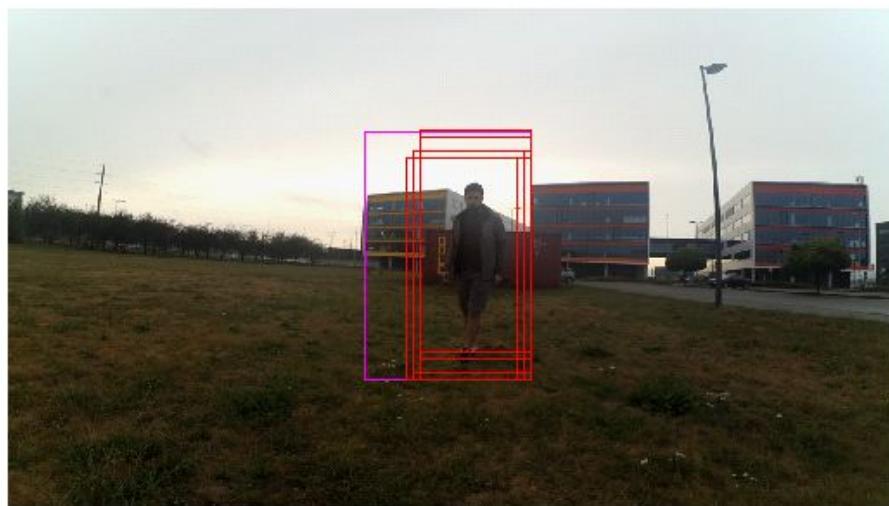
**Tablica 5.2.** Tabela z wynikami dla  $5 \times 9$  detekcji z rysunku 5.2

1.4357	1.0343	1.4887	1.1960	1.1270	1.2742	0.6366	0.9465	0.8936
0.9876	1.1790	1.7361	1.4707	1.0850	1.2012	1.2482	1.5096	1.2834
1.4188	1.3449	2.1487	1.8528	0.9389	0.7458	1.0194	1.5121	1.7702
1.5967	2.0461	1.7974	1.1585	0.6553	0.6955	0.5781	1.6717	1.3235
1.2971	2.0113	1.2289	<b>-0.0191</b>	<b>-0.5477</b>	<b>-1.0460</b>	0.2423	1.0235	1.7798

Następnie wykonano eksperyment pozwalający potwierdzić działanie detekcji dla implementowanych sprzętowo skal: 2/2.5/3/3.5/4. W tym celu nagrano sekwencję wideo, która przedstawia postać zbliżającą się w stronę kamery z odległości ok. 10m. Skrypt z wczytanym

**Tablica 5.3.** Tabela z wynikami dla  $5 \times 9$  detekcji z rysunku 5.3

2.1223	1.9234	1.8072	1.8854	1.6932	1.5320	0.5443	0.2648	<b>-0.0366</b>
1.9547	2.5649	1.6400	1.7075	1.1940	1.5821	0.6492	0.2804	<b>-0.0880</b>
2.5165	2.7328	2.3592	2.2650	2.1196	1.1509	0.8788	0.6314	0.1222
2.4744	2.8618	2.7310	2.6338	2.1627	1.4763	0.4520	<b>-0.7166</b>	<b>-1.1757</b>
2.6806	2.4200	2.2032	1.6812	1.9216	0.9355	<b>-0.0505</b>	<b>-1.3537</b>	<b>-1.3712</b>



Rysunek 5.3. Detekcja na oknie wewnętrz obrazu (skala: 2.5)



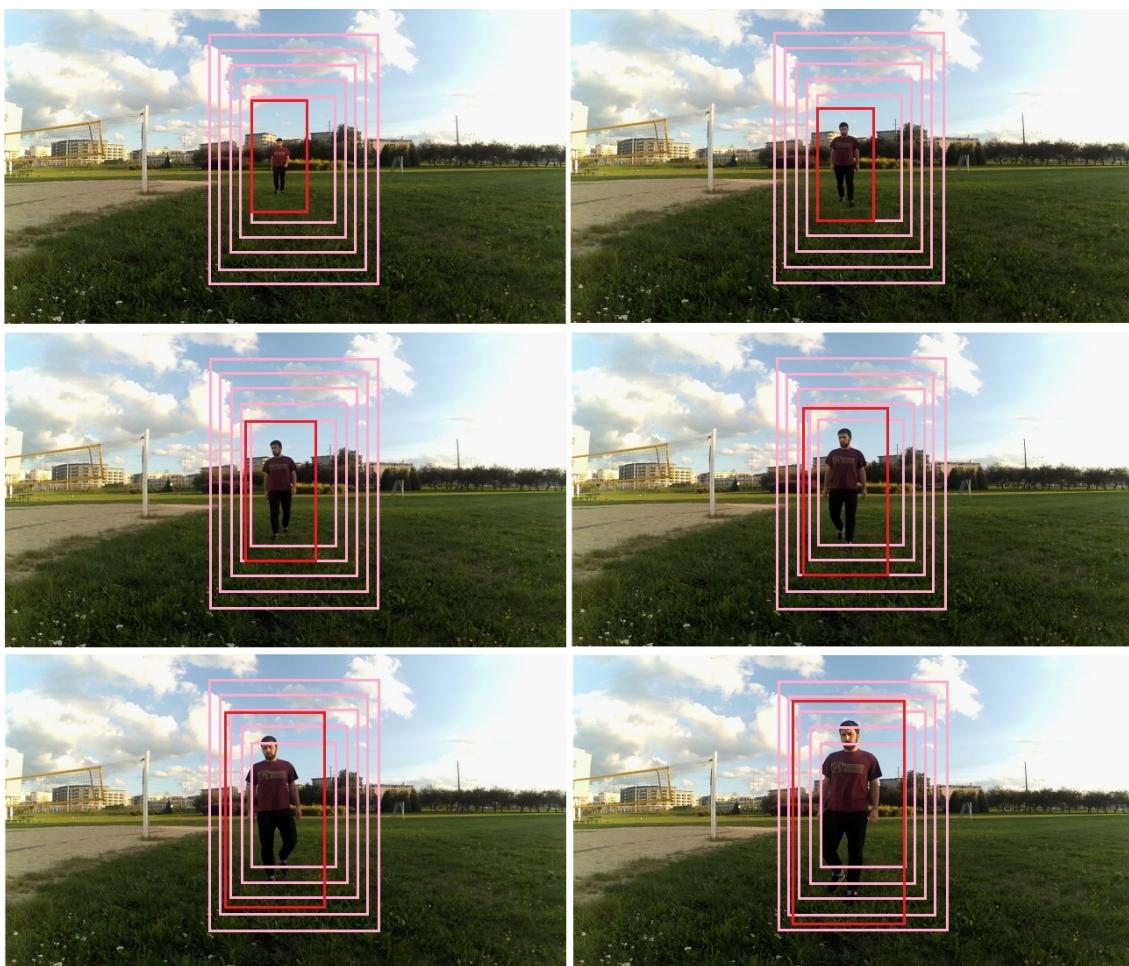
Rysunek 5.4. Detekcja na oknie wewnętrz obrazu (skala: 3)

Tablica 5.4. Tabela z wynikami dla  $5 \times 9$  detekcji z rysunku 5.4

1.8159	1.8308	1.7164	1.4868	1.1955	1.1542	0.8346	0.4625	0.6144
2.3357	2.6091	2.0017	2.0776	2.2638	1.8270	0.6915	<b>-0.0323</b>	0.6554
2.3233	2.4182	1.9292	2.1127	1.8924	1.7307	0.8044	0.3019	0.7611
1.5705	1.6276	1.5120	1.7848	1.6619	1.7323	0.4511	0.8625	1.1717
1.4874	1.8153	1.0768	1.2884	1.0186	1.8310	0.8986	0.6349	1.1386

materiałem wideo realizuje detekcję na wielu skalach i zaznacza na ramkach obrazu wszystkie obszary zainteresowań oraz najlepsze okno detekcji. Wyniki przedstawia rysunek 5.5. Dla obu detekcji z pierwszego rzędu, najlepsze wyniki osiągnięto na obszarze w skali równej 2. Należy zwrócić uwagę, że ze względu na brak uwzględnienia mniejszych skal w teście, na obrazie przeskalowanym ze współczynnikiem 2 może zostać wykryta osoba o małych rozmiarach względem okna detekcji (przypadek w lewym górnym rogu rysunku). Rzędy drugi oraz trzeci przedstawiają poprawnie wykrytą postać w pozostałych skalach: odpowiednio 2.5/3/3.5/4.

Eksperyment posłużył również do oszacowania odległości postaci od kamery, które należałyby przyporządkować detekcjom w poszczególnych skalach. Na podstawie minimalnej i maksymalnej odległości osoby od kamery w sekwencji wideo (ok. 2m – 10m) można oszacować, że detekcja w skali 4 odpowiada odległości ok. 2m, natomiast każda kolejna skala powiększa tę wartość o ok. 1 – 1.5m, aż do odległości ok. 7m dla detekcji w skali 2. Tak jak wspomniano wcześniej, w skali tej wykrywana jest również dalej stojąca osoba (nawet 10m), choć nie jest to regułą.



Rysunek 5.5. Detekcja dla wielu skal



## 6. Implementacja sprzętowo-programowa systemu wizyjnego

Jednym z pierwszych zadań związanych z realizacją projektu było określenie sposobu implementacji systemu wizyjnego. Po uwzględnieniu wysokiej złożoności obliczeniowej algorytmów, a także ograniczeń związanych z miejscem działania systemu wizyjnego (platforma UAV) – takich jak rozmiar układu i pobór mocy – postanowiono wybrać podejście oparte o implementację sprzętowo-programową. W przeciwieństwie do programu komputerowego, urządzenie realizujące odpowiednio zaimplementowany algorytm ze wsparciem sprzętowym cechuje się większą szybkością działania. Wadą takiego rozwiązania jest koszt i czas jego przygotowania.

W pracy zdecydowano się wykorzystać heterogeniczny układ Zynq opisany w dalszej części rozdziału.

### 6.1. Układy Zynq

Najważniejszym i najbardziej czasochłonnym elementem pracy była sprzętowa implementacja algorytmów detekcji i śledzenia. W dzisiejszych czasach istnieje możliwość skorzystania z przeróżnych platform obliczeniowych, które dzieli się, jak przedstawiono poniżej:

- brak możliwości zmiany funkcjonalności po wyprodukowaniu układu:
  - procesory ASIP (ang. *Application Specific Instruction Set Processor*) – zaprojektowane z dedykowanym zestawem instrukcji
  - układy ASIC (ang. *Application Specific Integrated Circuit*) – zaprojektowane do realizacji określonej z góry zadań
  - procesory ogólnego przeznaczenia GPP (ang. *General Purpose Processor*)
  - procesory graficzne GPU (ang. *Graphics Processing Unit*)
  - procesory sygnałowe DSP (ang. *Digital Signal Processor*)

- z możliwością konfiguracji po procesie produkcyjnym - układy FPGA (ang. *Field-Programmable Gate Array*) oraz CPLD (ang. *Complex Programmable Logic Device*)

Specjalizowane układy ASIC są niestety drogie w prototypowaniu. Proces ten oznacza stworzenie układu scalonego od podstaw, i biorąc pod uwagę brak możliwości rekonfiguracji, musi uwzględniać szereg działań związanych z zapewnieniem poprawności działania (tj. testowania) przed przekazaniem do produkcji. Procesory CPU, czy DSP nie pozwalają na przetworzenie tak dużej ilości danych w czasie rzeczywistym – szczególnie dla obrazów o wysokiej rozdzielczości. Ponadto charakteryzują się względnie dużym zużyciem energii.

Z kolei druga z wymienionych grup urządzeń ma atut polegający na możliwości zmiany architektury układu oraz zrównoleglenia obliczeń tak bardzo, jak tylko pozwala na to liczba dostępnych zasobów. Dodatkowo – i paradoksalnie zarazem – urządzenia te charakteryzują się stosunkowo niskim zapotrzebowaniem na energię. Ma to istotne znaczenie, mając na uwadze światowy trend poszukiwań energooszczędnego rozwiązań w każdej branży – a zwłaszcza na rynku dronów, których czas lotu jest przeważnie ograniczony pojemnością baterii do kilkunastu minut. Obszary, w których FPGA znajduje zastosowanie to między innymi:

- studia dźwiękowe,
- telekomunikacja,
- przemysł motoryzacyjny/zbrojeniowy/lotniczy/kosmiczny,
- szyfrowanie i przetwarzanie danych,
- aparatura medyczna,
- systemy wizyjne.

To wszystko nie oznacza jednak, że układy FPGA nie są pozbawione wad. Przykładowo, realizacja niektórych rodzajów algorytmów (głównie rekurencyjnych) jest utrudniona i mało efektywna. Ponadto, sam proces rozwoju konkretnej architektury nie należy do najprostszych i może wymagać potwierdzenia funkcjonalności na kilka sposobów:

- porównanie wyników z modelem stworzonym w innym, konwencjonalnym środowisku (MATLAB, OpenCV). Przykładowo, pozwala to zweryfikować poprawność logiki w układzie rekonfigurowalnym, która jest tworzona w oparciu o zapis stałoprzecinkowy. Dla wspólnego scenariusza testowego, różnice pomiędzy poszczególnymi wartościami nie powinny być większe niż akceptowalny poziom błędu.
- symulacja HDL na komputerze – umożliwia weryfikację projektu lub jednego z modułów poprzez określenie wymuszeń sygnałów w testowym module zewnętrznym, tzw. *testbenchu*. Proces ten aktualizuje stan logiczny wszystkich obiektów (modułów) w określonym odstępie czasu, przykładowo 1 ps.

- weryfikacja wybranych sygnałów w układzie przy użyciu zintegrowanego analizatora logicznego – SignalTap dla urządzeń firmy Intel (dawniej Altera), ILA dla układów firmy Xilinx. Oznaczając w budowanym projekcie odpowiednie sygnały i wykorzystując protokół JTAG, możliwe jest skomunikowanie się z układem i akwizycja zdefiniowanej liczby próbek każdego z elementów. Na szczególną uwagę zasługuje opcja wyzwolenia procesu akwizycji spełnieniem określonego warunku (powiązanego ze stanem dostępnych sygnałów).

Część problemów bywa z czasem rozwiązywana przez producentów. Stopień skomplikowania projektu można znacznie zredukować, wykorzystując dostarczaną z oprogramowaniem własność intelektualną – odpowiednio skonfigurowane tzw. IP Core'y – i łącząc sygnały pomiędzy nimi na diagramach graficznych. Co więcej, narzędzie Vivado High-Level Synthesis (HLS) oferuje możliwość konwersji języka C/C++ do technologii FPGA, upraszczając proces powstawania architektury.

Postępująca zwłaszcza w ostatnich latach integracja i zmniejszanie procesu technologicznego pozwoliły na stworzenie układów łączących logikę programowalną (FPGA) oraz dwurdzeniowy procesor ARM. Układ, na którym oprócz jednostki obliczeniowej znajdują się różne peryferia, nosi nazwę układu heterogenicznego, w branży bardziej znanego jako *System on a Chip* (SoC). Dzięki magistrali (przykładowo AXI) zapewniającej szybką wymianę danych pomiędzy blokami, taki układ pozwala łączyć możliwość zrównoleglania obliczeń, którą daje logika programowalna (ang. *PL - Programmable Logic*) oraz prostotę rozwoju oprogramowania uruchamianego na procesorze (ang. *PS - Processing System*). W tej ostatniej części istnieje możliwość pracy w systemie operacyjnym (jednym ze specjalnych dystrybucji systemu Linux) lub nawet stworzenia konfiguracji opartej o wieloprocesorowość asynchroniczną (ang. AMP – *asynchronous multiprocessing*), która pozwala wykorzystać oba dostępne rdzenie procesora do niezależnych celów. Poza prostym przypadkiem jak praca z dwiema równoległymi aplikacjami, możliwe jest nawet działanie w konfiguracji Linux+RTOS (ang. real-time operating system – system operacyjny czasu rzeczywistego) [34]. Warto tu zauważyć, że narzędzia syntezы logiki od dawna oferowały możliwość stworzenia softprocesorów w układach FPGA (Altera – Nios, Xilinx – MicroBlaze), jednak te rozwiązania istotnie ograniczały dostępne zasoby, a częstotliwość taktowania takich komponentów rzadko przekraczała 200 MHz.

Powyższa charakterystyka przekonuje, że układy FPGA, a zwłaszcza SoC mogą być szczególnie docenione na platformie latającej, gdyż były w stanie sprostać wymaganiom stawianym przez zadanie przetwarzania obrazu w czasie rzeczywistym, a dzięki niewielkim wymiarom i niskiemu zużyciu energii nie stanowiłyby wielkiego obciążenia przy i tak już mocno ograniczonym czasie lotu na jednej baterii.

W pracy zdecydowano się wykorzystać układ Zynq SoC firmy Xilinx. O wyborze zadecydował nie tylko charakter projektu wymagający zrównoleglenia wykonywanych obliczeń, ale

też niski pobór mocy oraz obecność dwóch rdzeni procesora ARM do realizacji części zadań. Sledzenie za pomocą algorytmów MeanShift oraz HOG+SVM wymaga bowiem przetwarzania danych w czasie rzeczywistym, ale już sterowanie pracą obu metod może być wykonywane w sposób programowy. Oczywiście, ze względu na poziom skomplikowania montażu układu Zynq nie podjęto decyzji o prototypowaniu własnego obwodu drukowanego, lecz wykorzystano niewielki, lecz bogaty w peryferia zestaw PYNQ z układem Zynq SoC XC7Z020. Oprogramowaniem wspierającym rozwój architektury na to urządzenie jest pakiet Xilinx Vivado + SDK. Dostarczana z nim własność intelektualna (tzw. moduły IP) jest tworzona w oparciu o popularny standard wymiany danych - magistralę AXI (ang. Advanced eXtensible Interface). Pozwala ona stworzyć jednolitą architekturę tworzonego systemu bazującego na adresowym dostępie do danych.

Część PS układu XC7Z020 jest wyposażona w:

- dwurdzeniowy procesor Cortex-A9 taktowany zegarem 650MHz,
- kontroler pamięci DDR3 z 8 kanałami DMA (ang. Direct Memory Access) oraz 4 wydajnymi portami AXI3,
- wydajne kontrolery 1Gb Ethernet, USB 2.0, SDIO (ang. Secure Digital Input Output),
- kontrolery SPI, UART, CAN, I2C.

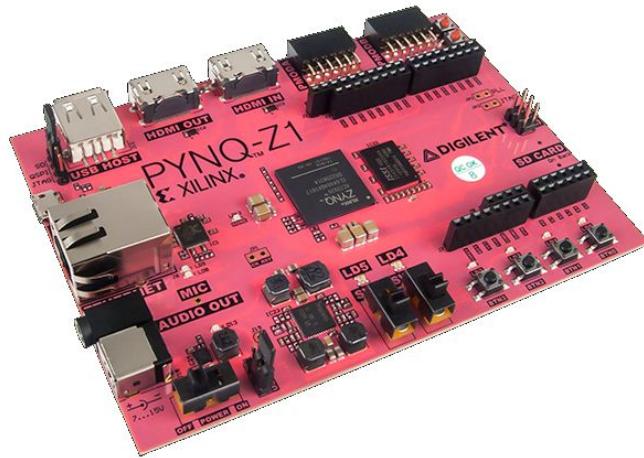
Część PL stanowi logika z rodziny urządzeń Artix-7, z następującymi parametrami:

- 13300 elementów *slice*, każdy wyposażony w 4 sześcioportowe tablice LUT (ang. Look-Up Table) oraz 8 przerzutników,
- 630KB szybkiej pamięci Block RAM,
- 4 obszary zarządzania zegarami, każdy z układem PLL (ang. Phase Locked Loop) i MMCM (ang. Mixed-Mode Clock Manager),
- 220 elementów DSP (ang. digital signal processing – cyfrowe przetwarzanie sygnałów),
- konwerter analogowo-cyfrowy (XADC – ang. Xilinx Analog-to-Digital Converter).

Poniżej przedstawiono pozostałe najważniejsze parametry platformy PYNQ:

- sygnał zegarowy o częstotliwości 125MHz
- 512 MB pamięci DDR3 z 16-bitową magistralą o przepustowości 1050Mbps,
- 16MB pamięci flash,

- slot na kartę MicroSD,
- złącza: MicroUSB z interfejsem JTAG, USB, Ethernet, 2x PMOD (ang. Peripheral Module), 3.5mm audio, we/wy HDMI,
- 4 przyciski, 2 przełączniki, 4 jednokolorowe diody LED oraz 2 RGB LED.
- możliwość zasilania baterijnego, poprzez USB lub dołączony zasilacz 12V.



Rysunek 6.1. Platforma PYNQ z układem Zynq SoC XC7Z020 w centrum

## 6.2. Tor wizyjny w części PL

Podstawowym komponentem toru wizyjnego jest moduł odbierający sygnał wideo z zewnętrznego źródła – w tym wypadku poprzez port HDMI – i zapewniający jego poprawne zdekodowanie do podstawowej, użytecznej przestrzeni barw RGB. Taki zestaw sygnałów może być w prosty sposób wykorzystany w tworzonych algorytmach. Opcjonalne i zalecane jest również wyprowadzenie przetworzonego sygnału RGB na monitor, co pozwala zweryfikować poprawność tworzonej architektury. Wyżej opisany szkielet został stworzony w tzw. *Block Designie*, a poza modułami firmy Xilinx wykorzystano konwertery DVI→RGB oraz RGB→DVI dostarczone przez firmę Digilent – producenta zestawu PYNQ. Ponadto, podstawowym zegarem systemu jest dostarczany z karty PYNQ zegar o częstotliwości 125MHz, na bazie którego powstaje szereg pozostałych zegarów.

W zadaniu śledzenia i detekcji zdecydowanie najważniejszym parametrem jest częstotliwość próbkowania sygnału wideo ze względu na potrzebę analizy jak najmniejszych ruchów

obiektu. Kolejnym parametrem jest rozdzielczość – obiekt na obrazie o wyższej rozdzielczości będzie opisywany większą liczbą pikseli, jednak niekorzystnie wpływa to na zużycie zasobów układu – wymagane jest osiągnięcie pewnego kompromisu. Z tego względu założono, że transmisja z kamery i przetwarzanie wideo będzie się odbywać w oparciu o standard  $1280 \times 720/60fps$ .

Sygnały wideo, które można wyodrębnić po zdekodowaniu, to:

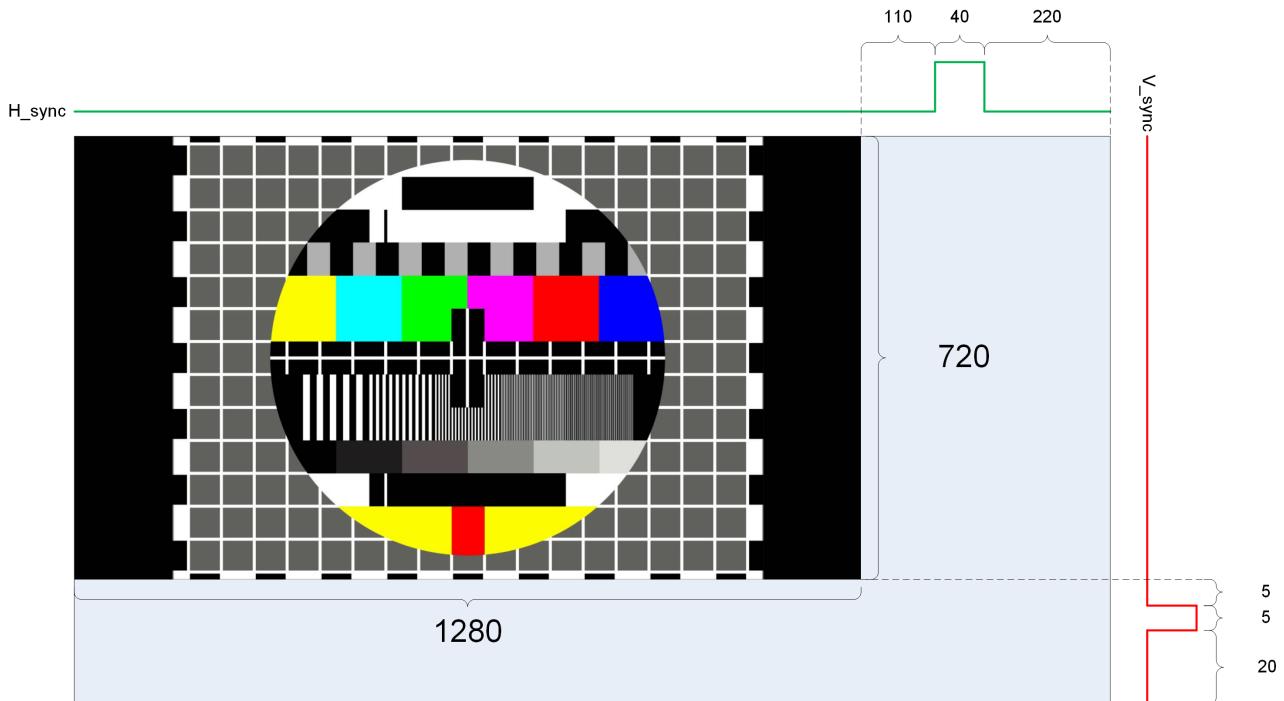
- zegar taktujący pikseli (74.25MHz),
- kolor czerwony R (8 bitów),
- kolor zielony G (8 bitów),
- kolor niebieski B (8 bitów),
- synchronizacja pozioma – poziom wysoki sygnalizuje koniec poziomej linii,
- synchronizacja pionowa – poziom wysoki sygnalizuje koniec klatki obrazu,
- sygnał aktywny – poziom wysoki sygnalizuje obecność poprawnego piksela.

Uwzględniając narzut związany z polami wygaszania pionowego i poziomego, ostateczna częstotliwość taktowania piksela wynosi w tym przypadku 74.25MHz. To właśnie ten zegar, nazywany dalej w pracy *pixel\_clk*, steruje pracą modułów odpowiadających za potokowe przetwarzanie obrazów w części PL. Ilustracja 6.2 przedstawia zapis klatki z sygnałami sterującymi, gdzie jednostką jest cykl zegara taktującego piksel.

Algorytmy opisane w dalszej części pracy będą często wymagać informacji o aktualnym położeniu piksela na właściwym obrazie. Stworzono w tym celu licznik obliczający tę pozycję dla osi pionowej i poziomej w oparciu o długości trwania sygnałów kontrolnych z rysunku 6.2.

### 6.3. Implementacja algorytmu MeanShift

Działanie algorytmu MeanShift rozpoczyna się po otrzymaniu sygnału *meanshift\_en* i polega na zdefiniowaniu wzorca. Uwzględniając parametry optyczne kamery, jak i rozdzielczość wejściową materiału wideo, podjęto decyzję o śledzeniu obszaru o wymiarach  $100 \times 100$  pikseli. Na jego bazie obliczana i zapisywana jest funkcja gęstości prawdopodobieństwa, tworzona w oparciu o jądro i jego gradient, które wygenerowano w procesie inicjalizacji. Następnie porównuje się ją z funkcjami gęstości prawdopodobieństwa kandydatów uzyskanych w kolejnych ramkach obrazu. By uwzględnić ruch obiektu pomiędzy klatkami, zapisywany jest wówczas fragment poszerzony o otoczenie 15 pikseli z każdej strony – o wymiarach  $130 \times 130$  pikseli. Na tej podstawie obliczany jest wektor MeanShift, który określa przesunięcie kandydata



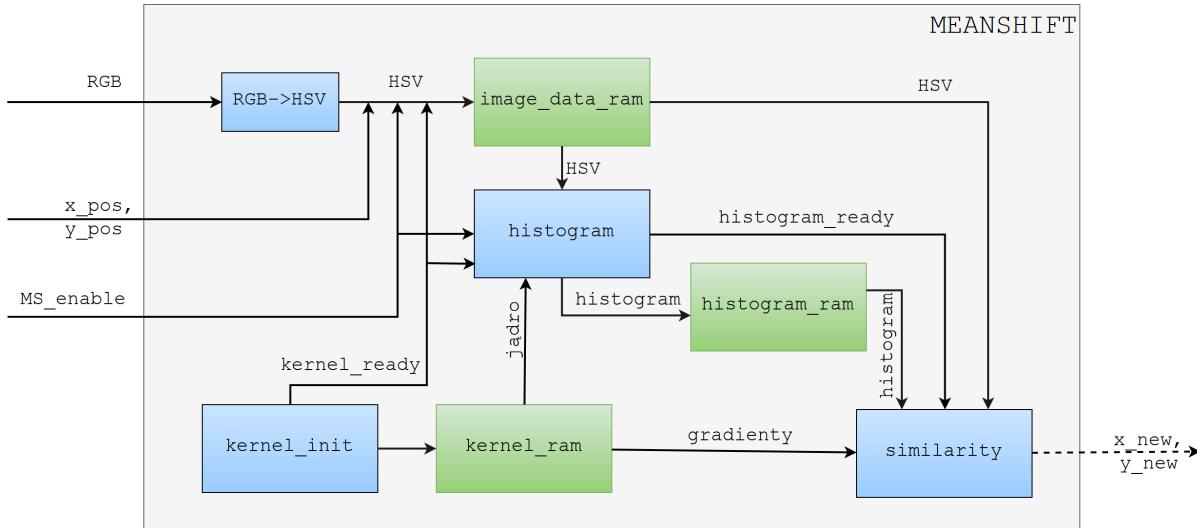
Rysunek 6.2. Schemat zapisu klatki w rozdzielcości 720p

i jednocześnie definiuje obszar następnych poszukiwań. Ten etap jest wykonywany w sposób iteracyjny, tj. powiększa precyzję przesunięcia śledzonego obszaru, wykorzystując zapisane sąsiedztwo obszaru w celu obliczenia aktualnej funkcji gęstości prawdopodobieństwa. Po określonej liczbie iteracji podejmowana jest akwizycja obszaru kandydata z kolejnej klatki.

Analizując możliwości pracy układu w oparciu o testy symulacyjne okazało się, że algorytm jest w stanie pracować z częstotliwością 60Hz. Może bowiem przetworzyć rozpatrywany fragment z bieżącej klatki obrazu w czasie ok. 4ms, co przy okresie trwania ramki wynoszącym 16.(6)ms pozwala zakończyć analizę przed zapisem obszaru z kolejnej ramki. Ogólny schemat blokowy jest przedstawiony na rysunku 6.3.

### 6.3.1. Konwersja przestrzeni barw RGB->HSV

Wstępny etapem toru wizyjnego wewnątrz algorytmu MeanShift jest konwersja przestrzeni barw, realizowana zgodnie ze wzorami (4.1)–(4.3) w module *rgb2hsv*. Moduł działa w trybie potokowym pracując z zegarem *pixel\_clk* i odpowiednio opóźnia wszystkie sygnały sterujące, co jest istotnym warunkiem poprawnego rozpoznawania odpowiednich pikseli w dalszych etapach przetwarzania. Dane te są na bieżąco dostarczane do modułu *Meanshift*, realizującego główną część zadań.



Rysunek 6.3. Schemat blokowy przedstawiający zależności pomiędzy modułami algorytmu MeanShift

### 6.3.2. Inicjalizacja

Mimo, że właściwe działanie algorytmu ma miejsce po otrzymaniu sygnału zewnętrznego (*algorithm\_en*), wymaga on wcześniejszej inicjalizacji – odbywa się ona tuż po zaprogramowaniu części PL układu Zynq. W głównej mierze jest ona związana z obliczeniem jądra i jego gradientów. Dane te, wykorzystywane potem w charakterze informacji tylko do odczytu, muszą tu być zapisane w dość uporządkowany sposób. Najlepiej nadaje się do tego konfigurowalna pamięć BRAM. Powinna ona przechowywać informacje dla wszystkich elementów obszaru, to jest 10000 pól. Jej ostateczną organizację przedstawia tabela 6.1. W kolumnie „Format” litera *U* oznacza liczbę bez znaku, natomiast *S* uwzględnia znak. Kropka oddziela dwie wartości, które określają liczbę bitów przyporządkowanych odpowiednio części całkowitej i ułamkowej (zapis stałoprzecinkowy).

Tablica 6.1. Organizacja pamięci BRAM *kernel\_ram*

Informacja	Adres rejestru	Format
Jądro: $K(\ P - P'(x, y)\ )$	0:9999	U3.15
Gradienty: $g_x, g_y$	10000:19999	S0.11, S0.11
Norma: $\sqrt{g_x^2 + g_y^2}$	20000:29999	U0.11

Warto nadmienić, że zestaw informacji związanych z konkretnym pikselem w obszarze 100x100 opisują rejestrury pod adresami:

$$\{100y + x, 10000 + 100y + x, 20000 + 100y + x\}, x, y = 0..99, \quad (6.1)$$

Ułatwia to zaprogramowanie dostępu do tych danych. By umożliwić odczyt obydwu gradien-tów w jednym cyklu zegara, zagregowano je w wektor o długości rejestru, tj. 18 bitów. Obser-wacje poczynione na zapisie gradientów w symulacji dowiodły jednak, że ich wartości są na tyle małe, że 3 najstarsze bity ułamkowe są zawsze równe bitowi znaku. Przykładowo, wartość 0.01 w formacie S0.11 ma postać  $12'b111111011000$ . Dysponując  $18/2 = 9$  bitami na gradient można było rozszerzyć precyzję informacji pomijając te 3 nieistotne bity i zapisując ją w pa-mięci w notacji S0.11, a nie S0.8 – co pozytywnie wpływa do dokładność obliczeń. Należy pamiętać jednak o tym, by odczyt z rejestru przechowującego gradienty dołączył te 3 bity do wartości przed rozpoczęciem jakichkolwiek obliczeń.

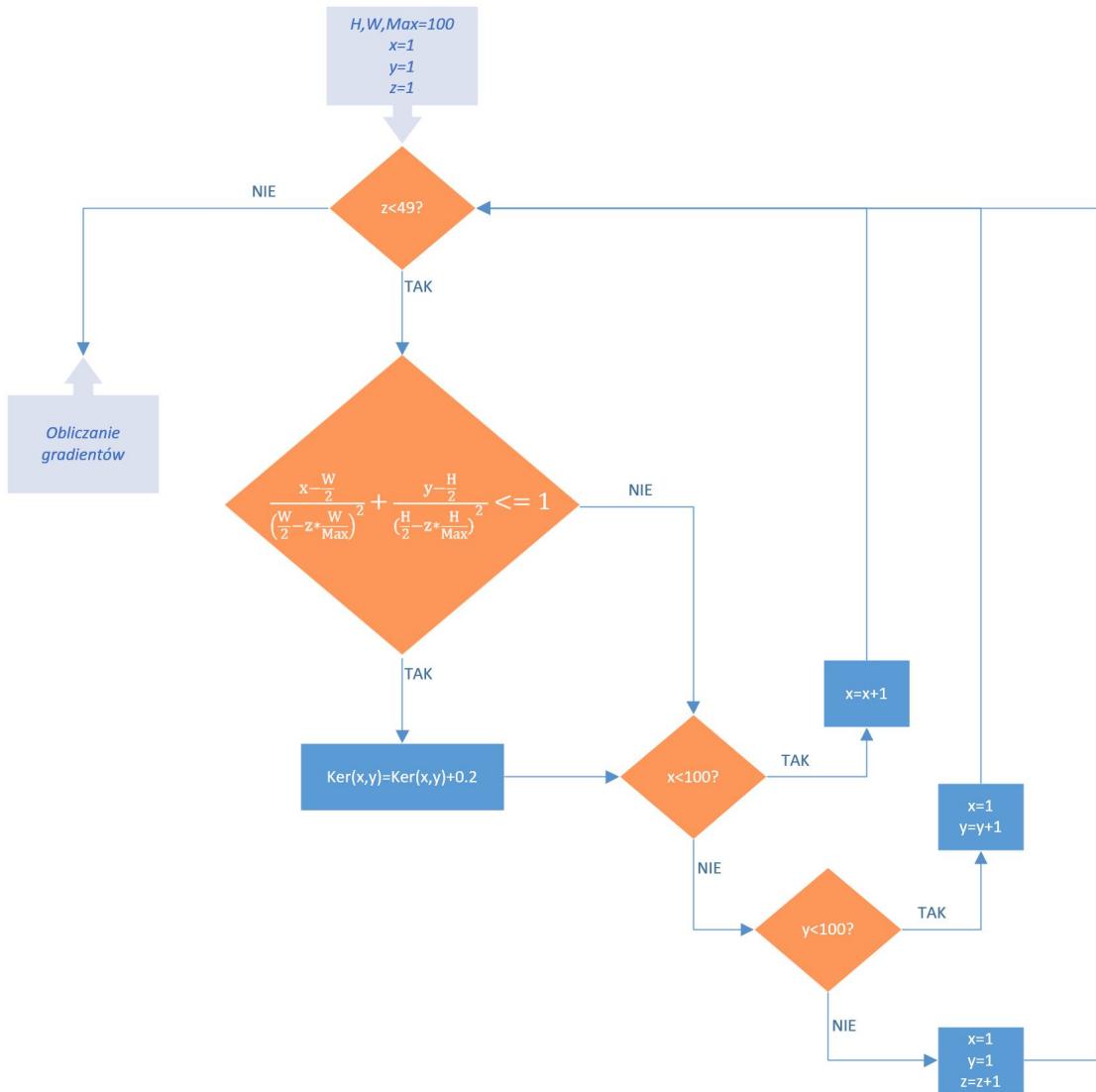
Pierwszym etapem inicjalizacji jest obliczenie jądra o wymiarach  $100 \times 100$ . Proces ten przebiega iteracyjne, budując kolejne warstwy jądra 4.3 w 49 powtórzeniach (długość boku obszaru/2 - 1). Wartością dodawaną do odpowiednich elementów jądra jest  $1/100/2 = 0.02$ , zapisywana w formacie U3.15 jako 0.019989013671875. Algorytm przechodzi przez 3 zagnież-dżone pętle, przemieszczając się po komórkach jądra  $\{x,y\}$  i kolejnych warstwach  $z$ . Schemat 6.4 przedstawia przebieg algorytmu. Główny warunek może wydawać się wyzwaniem, jed-nak wyrażenia typu  $W/2, H/2$  wymagają jedynie przesunięcia wektora o 1 bit w prawo. Nie zmienia to faktu, że implementacja obliczeń wymaga uwzględnienia odpowiednich opóźnień, a następnie synchronizacji z odczytem i zapisem danego rejestru jądra.

Kolejnym etapem jest obliczenie gradientów i ich normy na podstawie gotowego jądra. Dla każdego elementu odczytywane są wartości 4 sąsiadów i wyznaczane gradienty przy użyciu masek:  $[1, 0, -1]$  oraz  $[1, 0, -1]^T$ . W przypadku sytuacji brzegowych niedostępny sąsiad zostaje zastąpiony rozpatrywanym elementem, a wartość gradientu jest przesuwana o jeden bit w lewo (dwukrotnie zmniejszona odległość pomiędzy obiektami skutkuje dwukrotnym powiększeniem wyniku ilorazu różnicowego). Na tej podstawie obliczana jest norma gradientu, która wymaga zastosowania dwóch mnożarek i pierwiastkującego bloku CORDIC. Jest to proces na tyle długi, że po zapisie gradientów w pamięci zdecydowano się przejść do kolejnych elementów jądra, jedynie zapamiętując adres rejestru, w którym gotowa wartość normy powinna być zapisana.

Po zebraniu kompletu danych ustawiana jest flaga *kernel\_rom\_ready*, której obecność sy-gnalizuje gotowość uruchomienia właściwej części algorytmu. Jak stwierdzono w oparciu o sy-mulacje, w rzeczywistości inicjalizacja pamięci *kernel\_rom* trwa około 5ms. Jest to zatem po-mijalnie krótki czas, niewpływający na funkcjonalność (<1 klatka obrazu).

### 6.3.3. Zapis obszaru wideo

Na szerszy opis zasługuje sposób zapisu informacji z obrazu, należy bowiem tymczasowo zapamiętać wartości pikseli  $H$ , na których będą kilkukrotnie wykonywane obliczenia. Dużym



Rysunek 6.4. Proces tworzenia jądra obszaru

obciążeniem dla zasobów układu byłaby próba zapisania całych klatek – pojedyncza wymagałaby:  $1280 \cdot 720 \cdot 9\text{b} = 1.037\text{MB}$  dostępnego miejsca. Z tego względu postanowiono zapisywać jedynie obszar w aktualnym położeniu obiektu, z dodatkowym sąsiedztwem 15 pikseli z każdej strony. Szerokość sąsiedztwa dobrano metodą doświadczalną, uwzględniając maksymalne przemieszczenia obiektu pomiędzy kolejnymi klatkami i ograniczenia związane z zasobami. W procesie zapisu uwzględniono zabezpieczenie na wypadek próby wyjścia poza przestrzeń obrazu. Opiera się ono na sprawdzeniu pozycji środka obszaru względem ramki obrazu – minimalna odległość od każdej krawędzi to 65. Zabezpieczenie to funkcjonuje również w samym algorytmie, utrzymując ostateczną pozycję obszaru wewnątrz dozwolonego wycinka ramki.

Proces śledzenia musi poprawnie określić położenie aktualnego, „użytecznego” piksela, a także rozpoznać początek kolejnej klatki. Jest to możliwe poprzez stworzenie logiki opierającej się na licznikach: horyzontalnym i wertykalnym, która wykrywa zbocza sygnałów sterujących. Liczniki wykorzystują informacje przedstawione na rysunku 6.2; zliczają odpowiednio do wartości 1280 oraz 720.

Otrzymanie sygnału *algorithm\_en* inicjuje realizację algorytmu. Istotne jest, aby w momencie pojawiения się tego sygnału, w obszarze zdefiniowanym jako startowy (domyślnie środek obrazu) znajdował się śledzony obiekt. Obszar ten, będący od teraz wzorcem, jest zatrzaskiwany na najbliższej pełnej klatce obrazu. Następnie obliczana jest funkcja gęstości prawdopodobieństwa, opisana w podrozdziale 6.3.4.

Wartości pikseli – linia po linii – są przechowywane w module BRAM działającym w trybie True Dual Port, *image\_data\_ram*. Dzięki temu możliwy jest jednoczesny zapis/odczyt pod warunkiem, że porty nie pracują jednocześnie na tym samym adresie. Ze względu na konieczność posiadania kompletnego fragmentu obszaru i możliwość zapisu kolejnego, pamięć zdolna jest pomieścić 2 obszary z ostatnich klatek, każdy o wymiarach  $130 \times 130$ : łącznie 33800 adresów. Zamiennie, co klatkę, wykonywany jest zapis najnowszych danych do jednej połowy przestrzeni, i przetwarzanie (odczyt) drugiej 6.2.

**Tablica 6.2.** Organizacja pamięci BRAM *image\_data\_ram*

Informacja	Adres rejestru	Format
Klatka $t \% 2 == 0$	0:16899	U9
Klatka $t \% 2 == 1$	16900:33799	U9

### 6.3.4. Funkcja gęstości prawdopodobieństwa

Wartość funkcji gęstości prawdopodobieństwa jest inaczej wartością histogramu dla danej barwy piksela ( $H$  z zakresu 0-359). W tym celu utworzono pamięć BRAM, która przechowuje dwa histogramy dla wzorca oraz kandydata – w sumie 720 rejestrów 6.3.

**Tablica 6.3.** Organizacja pamięci BRAM *histogram\_ram*

Informacja	Adres rejestru	Format
Histogram wzorca	0:359	U10.15
Histogram kandydata	360:719	U10.15

Histogram wzorca jest tworzony raz, w oparciu o pierwszy obraz; kolejne histogramy są związane z kandydatem i zostają zapisane w górnej połowie adresowej ( $H + 360$ ). Wartość

piksela, umożliwiając dostęp do odpowiedniego rejestru pamięci, pozwala na odczyt dotychczasowej wartości przedziału histogramu i przygotowanie go do aktualizacji. Z kolei współrzędne piksela w odniesieniu do obszaru  $100 \times 100$  są wykorzystywane do odczytu elementu jądra. Przedział histogramu jest aktualizowany zapisem sumy obu wartości. Wielkość rejestrów pamięci *histogram\_ram* pozwala na zapis wartości w formacie U10.15, czyli w zakresie [0:1023.99997]. Zdarzają się jednak przypadki, gdy większość pikseli na obszarze (zwłaszcza w centrum – miejscu największych wartości jądra) jest jednokolorowa, co może skutkować przekroczeniem dopuszczalnych wartości dla rejestru. Chroni przed tym dodatkowy fragment logiki, który wykrywając takie zdarzenie, pozostawia rejestr z maksymalną wartością.

Wartość piksela jest odczytywana bezpośrednio z pamięci przechowującej analizowany obszar (opisanej w podrozdziale 6.3.3). Do obliczenia histogramu wymaganych jest jedynie  $100 \times 100$  pikseli, pozostałe (sąsiedztwo) należy zignorować. Wartości *offset\_X* oraz *offset\_Y* określają pozycję pierwszego analizowanego piksela (w lewym górnym rogu). Startując z tego miejsca, praca liczników *H\_count* oraz *W\_count* pomaga w zebraniu jedynie 100 pikseli ze 100 linii. Przykładowo, obliczając po raz pierwszy histogram dla ostatnio zapisanej klatki zmienne *offset\_X* i *offset\_Y* mają wartości domyślne (15, 15). Oznacza to, że z zapisanego obszaru o wymiarach  $130 \times 130$  należy odczytać jego środek, a współrzędne pierwszego piksela będą wynosić (15, 15). W kolejnych iteracjach algorytmu, dla tej samej klatki obrazu, zmienne mogą przyjmować inne wartości, tworząc w efekcie funkcję gęstości prawdopodobieństwa odpowiadającą nowemu fragmentowi obszaru  $100 \times 100$ . Zakres dopuszczalnych wartości dla każdej z tych zmiennych to  $<0, 29>$ .

Po przejściu przez wszystkie piksele obszaru następuje ustawienie flagi *histogram\_ready*, gdzie w przypadku przetwarzania klatki-wzorca kolejnym zadaniem jest akwizycja pierwszego kandydata, natomiast dla każdej kolejnej rozpoczęcie obliczanie funkcji podobieństwa. Podobnie jak *image\_data\_ram*, *histogram\_ram* działa w trybie True Dual Port. Po obliczeniu funkcji gęstości dla aktualnego obszaru, z pamięci jednocześnie są odczytywane wartości funkcji dla wzorca (port A) i kandydata (port B pamięci) – umożliwiając wykonywanie części dalszych obliczeń w sposób równoległy.

### 6.3.5. Funkcja podobieństwa

Moduł implementujący obliczanie funkcji podobieństwa (i ostatecznie przesunięcia) obszaru, jest najbardziej złożony w tej części systemu. Funkcja podobieństwa, opisana wzorem (4.20), jest wyznaczana dla każdego piksela w pamięci, należącym do właściwego obszaru (bez sąsiedztwa).

Dla każdego piksela z obszaru  $100 \times 100$ , którego wartość *H* jest adresem dla pamięci *histogram\_ram*, wczytywana jest wartość funkcji prawdopodobieństwa wzorca i kandydata.

Moduł oblicza ich iloraz (wzorzec/kandydat) używając bloku Divider Generator (w wersji 5.1) dostarczonego przez firmę Xilinx. Ze względów optymalizacyjnych obcięto 9 najmłodszych bitów obu wartości, gdyż nie wpływały w większym stopniu na dokładność, a ich pominięcie pozwoliło zmniejszyć latencję modułu. Podczas takiego dzielenia logika musi ponadto sprawdzić obecność zera w mianowniku – wówczas iloraz powinien być zerowany. Brak obsługi takiego zdarzenia doprowadziłby do otrzymywania ilorazu o trudnej do przewidzenia wartości – mogłoby to powodować poważne błędy w działaniu algorytmu.

Gotowy iloraz należy poddać pierwiastkowaniu. Moduł obliczający pierwiastek wymaga, by wejście z danymi było liczbą całkowitą lub też ułamkową – ale z jednym bitem dla części całkowitej: U1.X. W tym celu wynik dzielenia – liczba w formacie: U16.16 – została obcięta do U13.11., a następnie „wirtualnie” przesunięta o 12 bitów w prawo, do postaci: U1.23. Wynik pierwiastkowania wymaga przesunięcia w lewo już tylko o 6 bitów ( $\sqrt{2^{12}} = 2^6$ ), po czym jest obcinany do 16 najbardziej znaczących bitów: U7.9. Operacje te, mimo że zawiłe, nie wpływają na wynik, lecz gwarantują poprawne działanie modułu *CORDIC* realizującego pierwiastkowanie.

Wartości pierwiastka współtworzyć będą zarówno licznik, jak i mianownik ilorazu wektora MeanShift – oddzielnie dla przesunięcia pionowego i poziomego. Obliczenie iloczynów wymaga pobrania z pamięci *kernel\_ram* obu gradientów oraz ich normy. Logika realizująca odczyt jest zoptymalizowana pod zminimalizowanie liczby cykli zegara potrzebnych na dostęp do obu wartości i została oparta o maszynę stanu. Po zakończeniu przetwarzania wszystkich pikseli obszaru  $100 \times 100$  (sumowania obliczonych na ich podstawie wartości) ustawiana jest flaga *div\_enable\_de*, która jest podłączona do modułów realizujących dzielenie w wektorze MeanShift. Otrzymane wyniki dzielenia odpowiadają przesunięciu obszaru śledzenia, należy je jednak znormalizować do postaci całkowitej i upewnić się, że obszar po przesunięciu nadal będzie w całości znajdował się w zapisanym fragmencie o wymiarach  $130 \times 130$ . Pozwoli to wykonać kolejne iteracje, które mogą poprawić wynik śledzenia. O ile pierwsza iteracja rozpoczyna analizę w środku obszaru, to w trakcie kolejnych informacja o położeniu obszaru wewnętrz  $130 \times 130$  jest zapamiętywana przez zmienne *offset\_X* oraz *offset\_Y*.

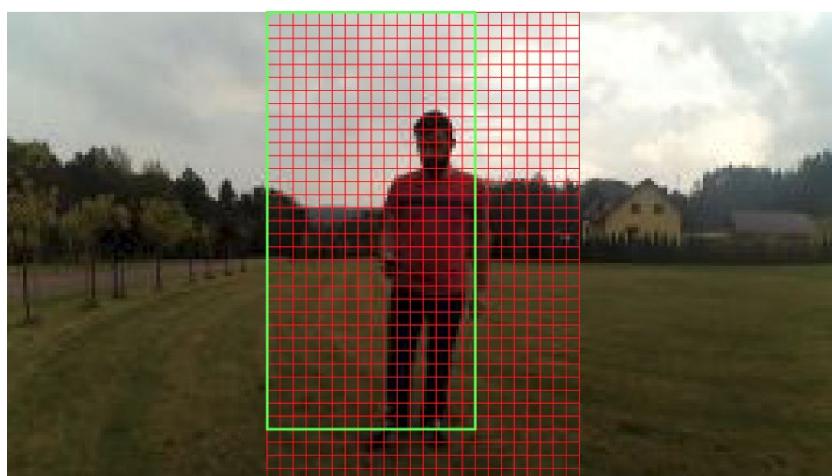
## 6.4. Realizacja algorytmu HOG+SVM

Wymagania stawiane przez system mówią o konieczności rozpoznania osoby wraz z dodatkowym określeniem jej wielkości na ekranie (skala ta pozwoli dość zgrubnie wyznaczyć odległość drona od obiektu, co wpłynie na sterowanie maszyny). Jednym z celów jest bowiem utrzymanie stałej odległości od postaci. Z tego względu implementacja musi wykorzystać tzw. piramidę skal, czyli równoległą detekcję HOG+SVM na serii obrazów o różnych wymiarach.

Dysponując obrazem o rozdzielczości  $1280 \times 720$ , należy przeskalać go do kilku mniejszych, a następnie przeprowadzić na nich opisane wcześniej operacje wyliczenia wektora cech i sklasyfikować go przy użyciu SVM. Odpowiednia lokalizacja postaci (i jej odległość od kamery) zostanie wybrana na podstawie najlepszego **pozytywnego wyniku klasyfikacji**, w oparciu o wartość  $r$  (4.30) (pełny opis algorytmu znajduje się w podrozdziale 4.3).

Algorytm ten może rozpoczęć działanie jedynie na pełnej klatce obrazu, zatem stworzono mechanizm, który niezależnie od momentu pojawienia się sygnału wyzwalającego pracę algorytmu będzie oczekiwał na zbocze opadające sygnału synchronizacji pionowej, czyli nową, pełną ramkę. Tylko w takim przypadku gwarantowane jest poprawne działanie metody. Drugim warunkiem jest to, by w momencieadejścia nowej klatki algorytm nie był w trakcie przetwarzania poprzedniej – w przeciwnym wypadku histogramy mogłyby być nadpisywane nowymi wartościami. Oznacza to jednak, że co druga klatka obrazu będzie pomijana w przetwarzaniu, ograniczając częstotliwość pracy algorytmu do  $30Hz$ .

W pierwszej kolejności obraz wejściowy poddawany jest konwersji do skali odcieni szarości. Następna w kolejności operacja skalowania do 5 mniejszych obrazów metodą najbliższego sąsiedztwa, jest realizowana potokowo, dzięki czemu można zachować sygnały kontrolne VGA (synchronizację poziomą oraz pionową). Na pomniejszonych obrazach obliczane są wektory cech, bazując na komórkach o wielkości  $4 \times 4$  i blokach o rozmiarze  $2 \times 2$ . Dysponując określoną liczbą zasobów układu XC7Z020, nie jest możliwa analiza całego obrazu, a jedynie oto- czenia aktualnie śledzonego fragmentu. Musi on być jednak wystarczająco duży, by możliwe było osiągnięcie jak najlepszego wyniku detekcji osoby w jego wnętrzu. Jak wspomniano wcześniej, wektor cech tworzony jest na wycinku o rozmiarze  $128 \times 64$  pikseli. Przedstawia to obraz 6.5, gdzie taki wycinek zaznaczono zieloną linią.

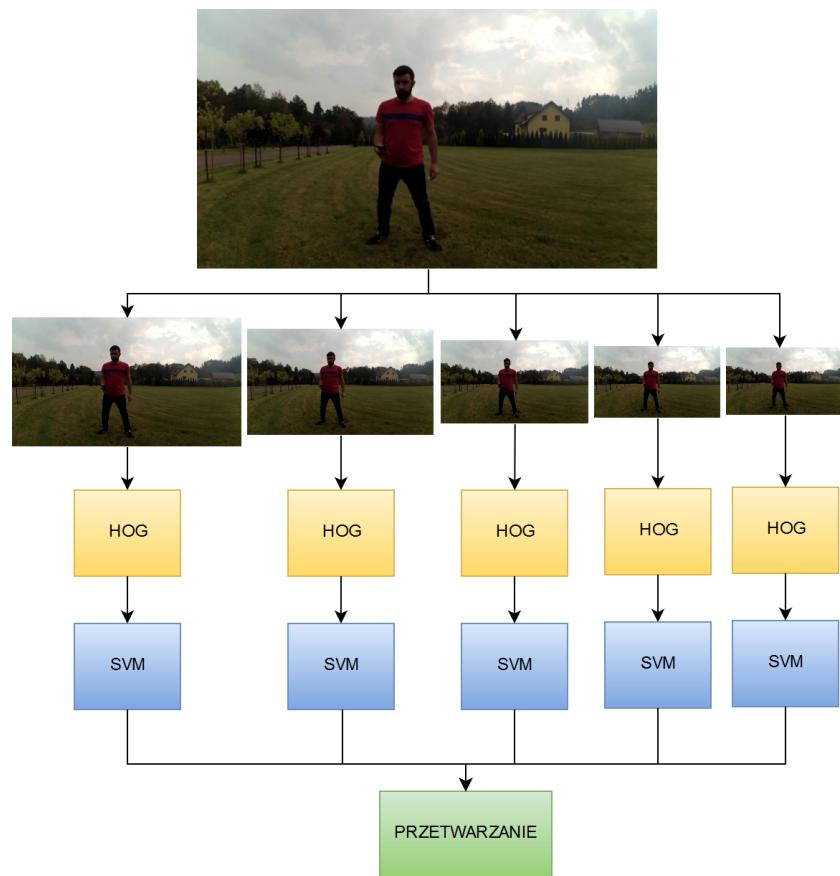


Rysunek 6.5. Analiza obrazu o rozmiarach  $144 \times 256$

Jeśli algorytm będzie pracował na obszarze  $144 \times 96$ , to zakładając stałe położenie komórek na obrazie (są to kwadraty  $4 \times 4$  wydzielone czerwonymi liniami), powstanie łącznie  $5 \cdot 9 = 45$

wektorów cech. Reszta obrazu zostaje zignorowana. Współrzędne centrum obszaru detekcji są określone na początku działania pojedynczej iteracji algorytmu i są natychmiastowo konwertowane do odpowiednich skal obrazu. Po zakończeniu klasyfikacji współrzędne odpowiadające najlepszej detekcji są konwertowane do oryginalnej skali i wykorzystywane podczas analizy kolejnej dostępnej klatki obrazu. Rysunek 6.6 opisuje metodę z perspektywy działania na kilku skalach.

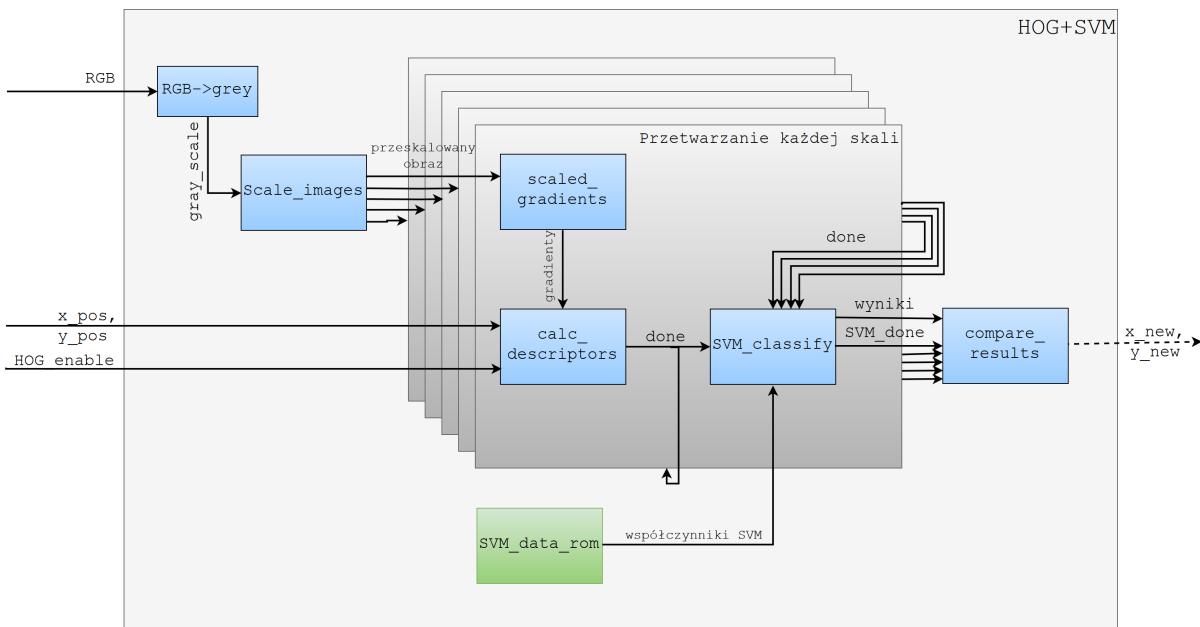
Ogólny schemat blokowy jest przedstawiony na rysunku 6.7. Opis poszczególnych modułów znajduje się w kolejnych podrozdziałach.



**Rysunek 6.6.** Schemat przedstawiający ideę działania algorytmu HOG+SVM w wielu skalach

#### 6.4.1. Konwersja RGB do skali odcieni szarości

Obraz wejściowy jest poddawany konwersji zgodnie ze wzorem (4.21). Używane są tu trzy równoległe mnożarki, a suma iloczynów jest zaokrąglana do 8 bitów (do postaci liczby całkowitej z zakresu 0-255).



**Rysunek 6.7.** Schemat blokowy przedstawiający zależności pomiędzy modułami algorytmu HOG+SVM

## 6.4.2. Skalowanie

Kolejny etap to przeskalowanie obrazu wejściowego. Sama idea okazuje się być tym bardziej na miejscu, jeśli wziąć pod uwagę parametry kamery zamontowanej na dronie – w przypadku tego projektu jest to Xiaomi Yi, urządzenie do zastosowań sportowych i charakteryzujące się dużym kątem widzenia –  $155^\circ$ . W efekcie osoba oddalająca się od kamery bardzo szybko zmniejszy swoje wymiary na obrazie. Materiał  $1280 \times 720$  pikseli przeskalowano do 5 obrazów przy użyciu następujących skali:

- **1:2** :  $720 \times 1280 \rightarrow 360 \times 640$  pikseli
- **1:2.5** :  $720 \times 1280 \rightarrow 288 \times 512$  pikseli
- **1:3** :  $720 \times 1280 \rightarrow 240 \times 426$  pikseli
- **1:3.5** :  $720 \times 1280 \rightarrow 205 \times 365$  pikseli
- **1:4** :  $720 \times 1280 \rightarrow 180 \times 320$  pikseli

Powyższe wartości pozwalają jednocześnie zachować prostotę implementacji (skale są reprezentowane w formacie U3.1) i z akceptowalnym marginesem błędu określić odległość drona od postaci. Skalowanie przebiega w dość prosty sposób i polega na pomijaniu odpowiednich wierszy lub/i kolumn oryginalnego obrazu. Jeśli założyć, że:

- $x_i, y_i$  – współrzędne obrazu wejściowego,

- $x_o, y_o$  – współrzędne obrazu wyjściowego (przeskalowanego),
- $s_c$  – skala do zastosowania w pionie oraz w poziomie,

to przypisanie wartości obrazu wejściowego nastąpi przy jednoczesnym spełnieniu obu poniższych warunków:

$$\begin{aligned} x_i &== \lfloor s_c x_o \rfloor \\ y_i &== \lfloor s_c y_o \rfloor \end{aligned} \quad (6.2)$$

Po natrafieniu na odpowiedni piksel, oprócz przypisania jego wartości do wyjścia, wystawiony zostanie sygnał sterujący *valid*, bardzo ważny dla dalszej części algorytmu. Przykład dla kilku pierwszych wartości  $x_o$  jest widoczny w tabeli 6.4.

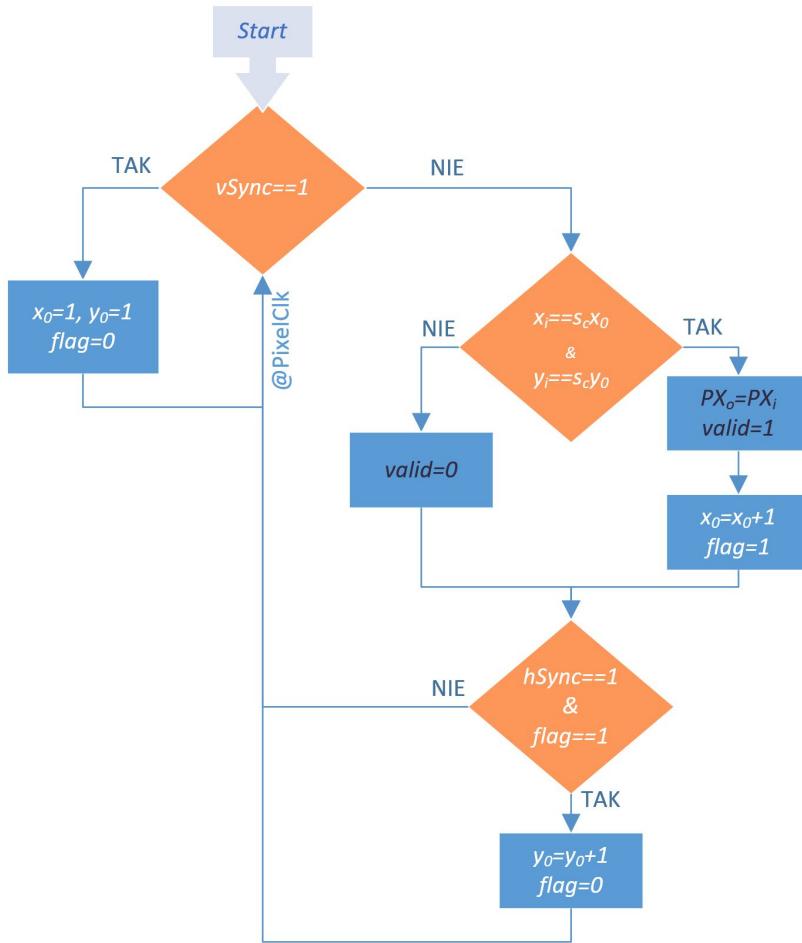
**Tablica 6.4.** Przykładowy przebieg skalowania dla  $s_c = 2.5$  wraz z przypisywanymi pikselami wejściowymi

$x_o$	$x_i s_c$	$x_i$
1	2.5	2
2	5	5
3	7.5	7
4	10	10

Działanie modułu opiera się na stworzeniu dwóch zestawów liczników dla każdej skali. Pierwszy zestaw pozwala na określenie indeksów piksela wejściowego ( $x_i, y_i$ ) i został opisany w rozdziale 6.2. Drugi zestaw liczników działa zgodnie ze schematem 6.8. Oznaczeniem *@PixelClk* opisano proces oczekiwania na kolejne zbocze narastające zegara piksela. W momencie pojawiienia się sygnału synchronizacji pionowej następuje inicjalizacja liczników współrzędnymi piksela zlokalizowanego w lewym górnym rogu docelowej klatki skalowanego obrazu. W trakcie otrzymywania kolejnych pikseli na wejściu porównywane są wartości obu liczników – spełniony warunek oznacza wystawienie wartości piksela na wyjście modułu wraz z sygnałem aktywnym. W przeciwnym wypadku sygnał aktywny jest wystawiany w stan nieaktywny. Obecna na schemacie zmienna *flag* służy do jednorazowych inkrementacji licznika  $y_0$  z uwagi na dwie kwestie:

- sygnał synchronizacji poziomej jest ustawiany na 40 cykli zegara pikselowego,
- sygnał synchronizacji poziomej jest obecny po sygnale synchronizacji pionowej, a przed nadaniem pierwszego piksela.

Gwarancją poprawnie przeprowadzonego procesu skalowania jest obecność sygnałów sterujących VGA – tylko wtedy następuje poprawny przyrost wartości liczników. Z kolei inicjalizacja (po lewej stronie diagramu) ma miejsce po otrzymaniu sygnału synchronizacji pionowej, zatem podłączony do układu sygnał video będzie skalowany już od pierwszej pełnej klatki.



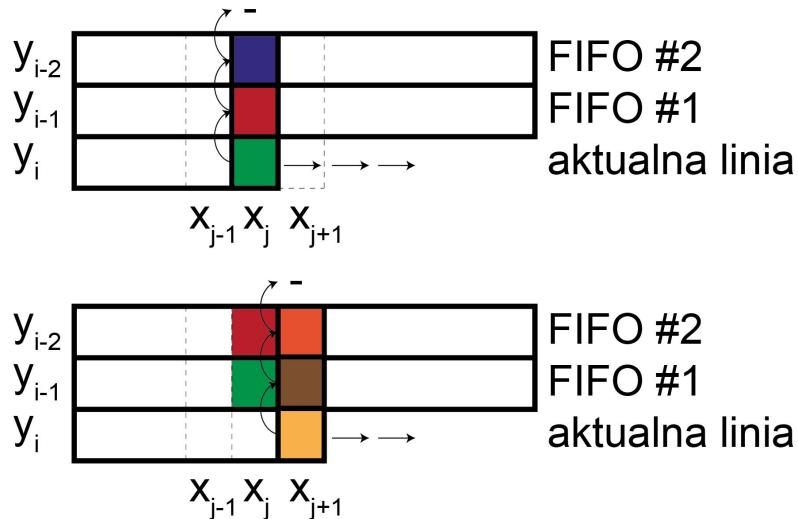
Rysunek 6.8. Schemat działania licznika skalującego

### 6.4.3. Obliczanie gradientów

Odpowiednio przeskalowane obrazy są zbyt duże, by przechowywać informację o ich gradientach w wewnętrznych zasobach układu XC7Z020. Do ich obliczenia zastosowano moduł działający potokowo i opierający działanie o sygnał aktywnego piksela *valid* z modułu skalowania.

Implementacja gradientu pionowego jest nieco złożona, gdyż wymagane w pojedynczej operacji piksele leżą w kilku kolejnych liniach obrazu. Konieczne jest zapamiętanie dwóch ostatnich linii – zrealizowano to przy użyciu dwóch kolejek FIFO 6.9. Do jednej z nich (oznaczonej numerem #1) wpisywane są wartości bieżących pikseli. Przejście do każdej kolejnej linii obrazu powoduje systematyczną wymianę pikseli na najnowsze - wówczas do kolejki #2 są zapisywane wartości bezpośrednio z #1. Logika została zaprojektowana w sposób pozwalający uzyskać jednoczesny dostęp do 3 kolejnych pikseli leżących w linii pionowej. Umożliwia to specjalny tryb modułu FIFO – First Word Fall Through (FWFT), dzięki któremu pierwsze dostępne słowo jest natychmiastowo wystawiane na wyjście, i tylko zdejmowane (zastępowane

kolejnym) w odpowiedzi na wysoki stan sygnału odczytu [16]. Ostatecznie logika, będąc w linii  $i$  ( $i > 1$ ), obliczy gradient pionowy dla piksela z linii  $i - 1$ .



**Rysunek 6.9.** Schemat działania kolejek FIFO w procesie obliczania gradientu pionowego

Obliczanie gradientu poziomego nie nastręcza już tak wielu trudności – sąsiadujące ze sobą piksele pojawiają się tuż po sobie, jednak w tym wypadku zamiast aktualnych wykorzystywane są piksele wychodzące z FIFO #1 i zapamiętywane w rejestrze przesuwnym. Oznacza to, że w chwili pojawienia się na wejściu do modułu nowego piksela  $(i, j)$ , obliczony zostanie gradient poziomy piksela  $(i - 1, j - 1)$ . Przez tę latencję potrzebne jest również nieznaczne opóźnienie gradientu pionowego, by obie wartości były zsynchronizowane i ustawione na wyjściu w tym samym momencie.

Sytuacje opisane powyżej dotyczą gradientów dla pikseli wewnętrz obrazu. Dla piksela znajdującego się na „początku” obrazu (lewa oraz góra krawędź), gradientem będzie dwukrotność różnicy pomiędzy nim a jedynym jego sąsiadem – w odpowiedniej osi.

Poprzednie obliczenia były przeprowadzane w oparciu o sygnał aktywnego piksela (*valid*), którego zbocza były wykorzystywane do określenia gradientów aż do przedostatniego wiersza i kolumny obrazu  $(i - 1, j - 1)$ . Piksele znajdujące się przy prawej i dolnej krawędzi ekranu wymagają innego podejścia. W tym przypadku, wymagane było stworzenie logiki kontynuującej obliczenia i generującej wyjściowe sygnały aktywne pomimo braku sygnału (*valid*). Opisywana sytuacja to również brak nowych wartości pikseli dla kolejek FIFO, co skutkuje ich spodziewanym opróżnieniem krótko po odebraniu pełnej ramki obrazu. O ile poprzednio tempo obliczania kolejnych gradientów było podkutowane obecnością nowych pikseli, to w tym przypadku logika korzysta wyłącznie z opróżnianych kolejek FIFO, redukując odstęp pomiędzy wynikami

do minimum (1 cykl zegara), co jest w zasadzie bez znaczenia dla dalszych obliczeń, które korzystają z poprawnie wygenerowanego sygnału aktywnego sygnalizującego obecność wyniku na wyjściu.

#### 6.4.4. Histogram gradientów

Wyznaczone gradienty służą następnie do obliczenia modułu i kąta.

Fragment logiki odpowiedzialny za obliczenie modułu został zrealizowany przy użyciu dwóch mnożarek dla obu gradientów, a sumę ich kwadratów następnie poddano pierwiastkowaniu w bloku CORDIC [15]. Moduł będzie rozpoczynać obliczenia dla danych wejściowych tylko w przypadku, gdy policzone zostały oba gradienty (dwa niezależne sygnały *valid\_x/y*) oraz gdy przynajmniej jeden z nich jest różny od zera ( $\frac{0}{0}$  jest elementem nieoznaczonym, z którego nie sposób policzyć implementowaną funkcję). Opisane warunki *valid\_x/y*, połączone odpowiednimi operatorami logicznymi, wyprowadzono jako sygnał aktywny modułu.

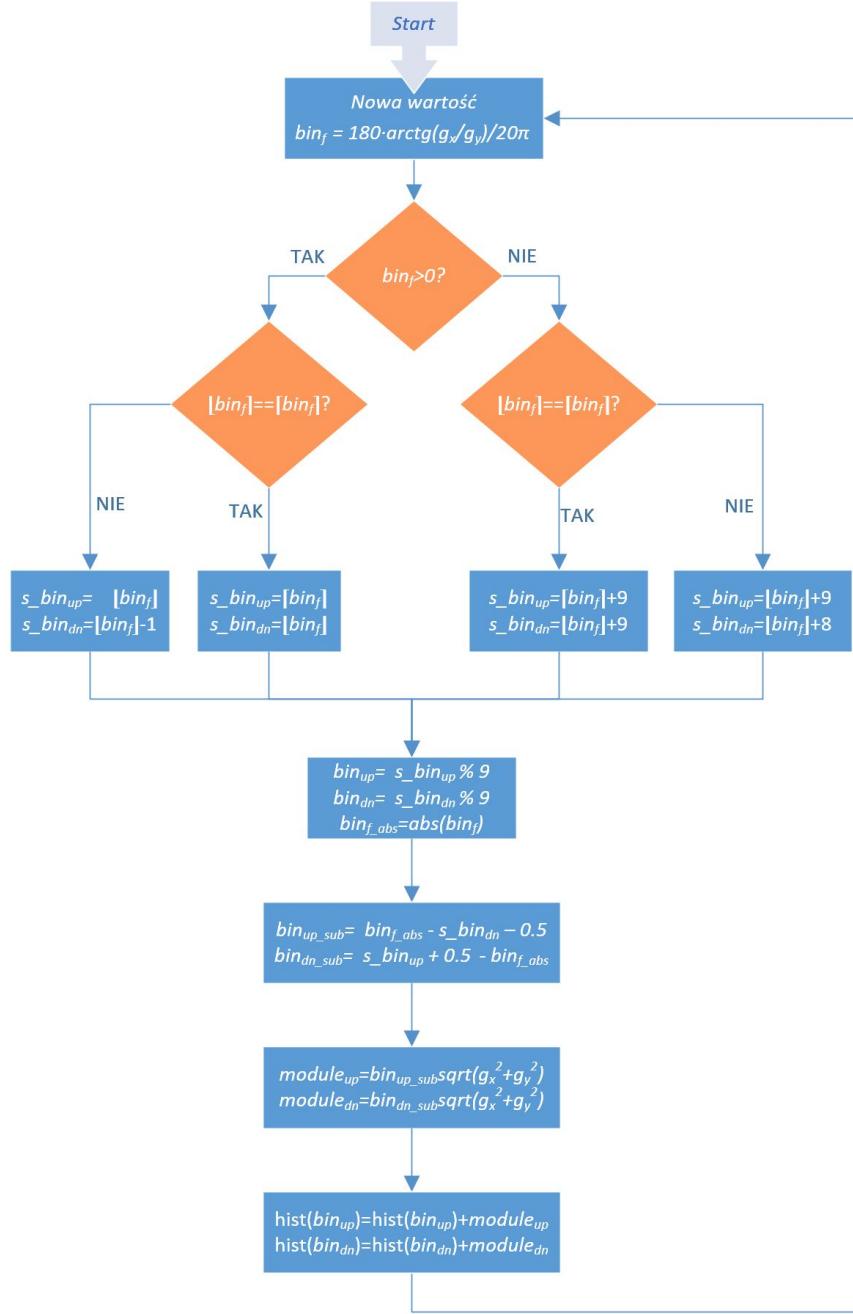
Z kolei wartość kąta jest obliczana w oparciu o wyrażenie  $\text{arctg}(\frac{g_y}{g_x})$ . Wykorzystano tutaj również blok IP CORDIC, który na wejściu otrzymuje wektor złożony z licznika oraz mianownika o tych samych długościach, przy czym jego całkowita długość jest zaokrąglana do wielokrotności liczby 16. Gradienty uzyskane z poprzedniego modułu są zapisane w notacji S9.1, zatem wektor wejściowy musi mieć długość 32 – po 16 bitów na oba gradienty. Bardziej znaczące bity tych połówek zostały wypełnione zerami i nie mają znaczenia dla obliczeń.

Otrzymane wartości należy następnie umieścić w dziewięciu 20-stopniowych przedziałach, co opisuje schemat 6.10.

Pierwszym krokiem jest operacja mnożenia kątów podanych w radianach przez  $\frac{180}{20\pi}$ , która konwertuje je do liczb ułamkowych stanowiących wstępny przydział (niech będzie to  $bin_f$ ).

Następnie, w zależności od położenia względem środka danego przedziału, wybierane są przedziały: górny i dolny w postaci liczb całkowitych:  $s\_bin_{up}$  oraz  $s\_bin_{dn}$ . Ostateczne będą one jednak wymagać normalizacji do postaci liczb z zakresu 0-8 i dopiero wówczas dwa przedziały histogramu zostaną powiększone o interpolowane wartości modułu gradientów.

W procesie obliczania modułu, suma mnożeń podnoszących gradienty do kwadratu jest wektorem U21.2, który poprzez dopisanie bitu '0' rozszerzono do U21.3. Zgodnie z dokumentacją bloku CORDIC, wektor o tej długości jest traktowany jako U1.23 – zatem jest wirtualnie przemnożony przez  $2^{20}$ . Wartość wyjściową należy później interpretować jako U11.13 (wirtualnie podzieloną przez  $\sqrt{2^{20}} = 2^{10}$ ). Podstawową informacją wykorzystywaną w interpolacji jest odległość  $abs(bin_f)$  od środków przedziałów  $s\_bin_{up}$  oraz  $s\_bin_{dn}$ . Na tej podstawie obliczane są  $module_{up}$  oraz  $module_{dn}$ , których suma jest równa pełnemu modułowi gradientów. Ostateczne informacje – to jest dane o przedziałach i odpowiadające im części modułu zostały



Rysunek 6.10. Schemat obliczeń histogramu gradientów

przekazane dalej, wraz z wygenerowanymi sygnałami aktywnymi, które oznaczają obecność wyniku.

Cały powyższy fragment podrozdziału skupiał się na operacjach związanych z pojedynczym pikselem. Teraz należy spojrzeć jednak z innej perspektywy, mianowicie na grupowanie pikseli w komórki, bloki i tworzenie wektorów cech na podstawie histogramu.

Najlepszy możliwy rezultat detekcji osiąga się analizując i klasyfikując jak największą liczbę okien detekcji w danym obszarze zainteresowań. Te powinny być wygenerowane dla

fragmentów obrazu, których przesunięcie względem siebie jest jak najmniejsze – zilustrowano to na rysunku 6.5. Wektory cech są zapisywane w pamięci BRAM, jednak szybki przyrost zużycia zasobów układu ogranicza implementację do przetwarzania jedynie określonej liczby obszarów w sąsiedztwie miejsca estymowanej lokalizacji postaci – dane te muszą być przechowywane jednocześnie do momentu zakończenia klasyfikacji.

By nie zużywać cennego miejsca w blokach BRAM, zdecydowano się zapisywać histogramy a nie znormalizowane w blokach wektory cech – wiedząc, że dalsza logika dokonując odczytu z tej pamięci w odpowiedni sposób przekaże te informacje do klasyfikatora. Ostatecznie, pojedyncze okno detekcji  $128 \times 64$  to  $32 \cdot 16 = 512$  histogramów, czyli 4608 wartości. Wektor cech to aż  $31 \cdot 15 \cdot 4 \cdot 9 = 16740$  wartości. Oszczędność wynikająca z zapisu pojedynczych histogramów pozwoli utworzyć znacznie więcej wektorów cech. Przykładowo, dla okna o wielkości  $144 \times 96$  należy zapisać 7776 wartości. Pozwala to jednak wygenerować  $9 \cdot 5 = 45$  wektorów cech. Gdyby zaś wpisywać je do pamięci w gotowej formie, wymagałoby to aż  $16740 \cdot 45 = 753300$  elementów.

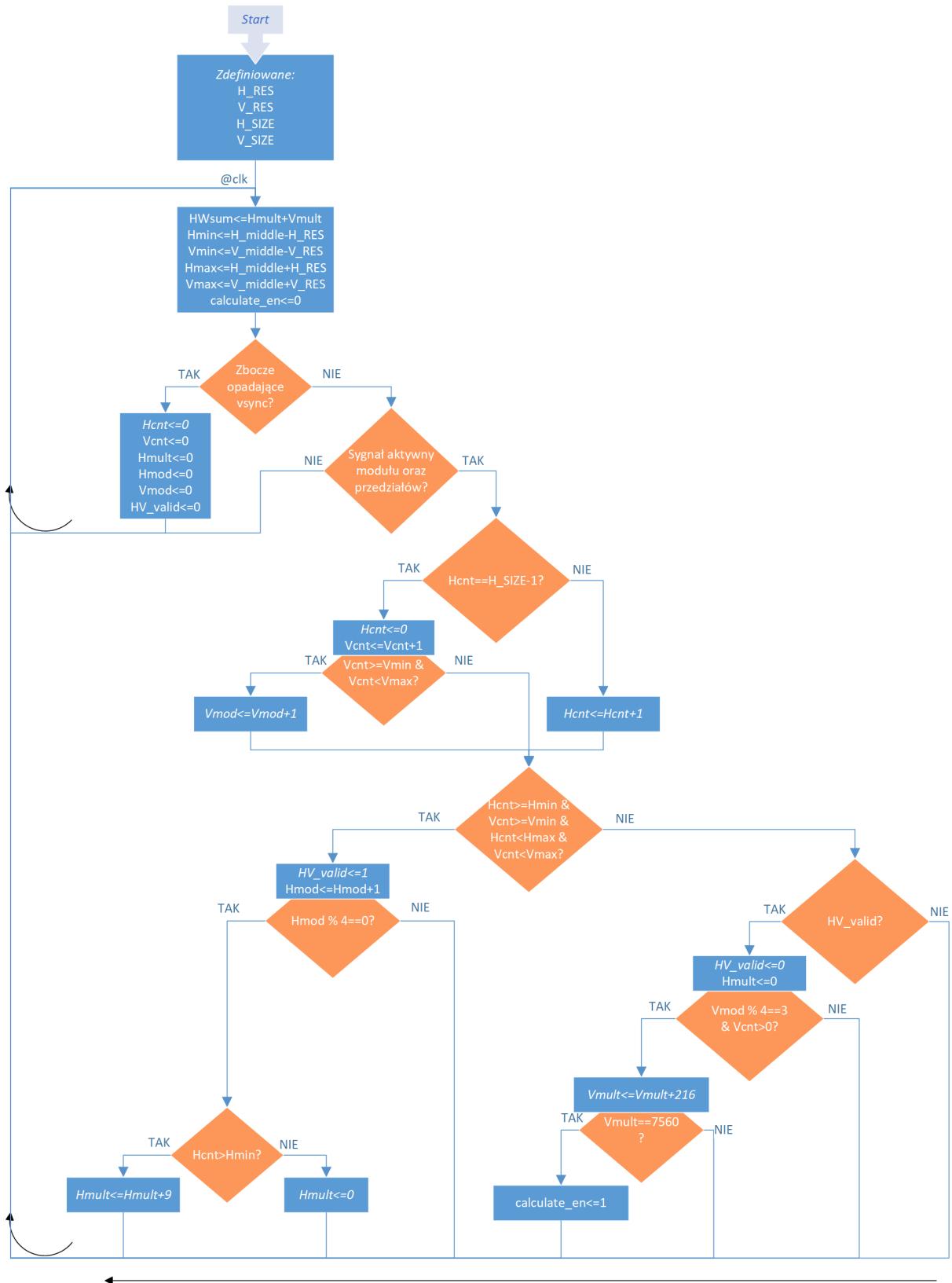
Pamięć RAM należy potraktować jako zbiór 9-elementowych histogramów ułożonych obok siebie. Przetwarzanie obrazu, rozumianego jako obiekt dwuwymiarowy, wymaga odpowiedniego mapowania tworzonych wartości do postaci jednowymiarowej, adresowej. Schemat 6.11 przedstawia działanie logiki na ramce obrazu w kontekście zapisu histogramów do pamięci. Symbolem „ $<=$ ” określa się przypisanie nieblokujące, które rzeczywisty efekt będzie miało dopiero na następnym zboczu narastającym zegara (i może być zastąpione kolejnym przypisaniem w obrębie jednego cyklu zegara).

Określenie aktualnego położenia na obrazie jest możliwe dzięki zastosowaniu liczników, opierających swoje działanie na obecności sygnału aktywnego sygnalizującego gotowe dane wejściowe. Wykorzystano następujące parametry:

- $H\_SIZE$ ,  $V\_SIZE$  – rozdzielcość obrazu przeskalowanego (indywidualnie dla każdej skali),
- $H\_middle$ ,  $V\_middle$  – współrzędne piksela środkowego, będącego w centrum analizowanego obszaru (w odniesieniu do odpowiedniej skali, dostarczone przed rozpoczęciem analizy pełnej klatki),
- $H\_RES$ ,  $V\_RES$  – wartości określające zasięg analizowanego obszaru – w odległości od piksela środkowego - dla tego projektu są to odpowiednio:  $96/2 = 48$  oraz  $144/2 = 72$ ,

oraz zmienne:

- $Hcnt$ ,  $Vcnt$  – zmienne inkrementowane odpowiednio do wartości maksymalnych  $H\_SIZE$ ,  $V\_SIZE$  – pozwalają określić aktualne położenie na obrazie (i względem analizowanego obszaru),



**Rysunek 6.11.** Procedura wyboru adresu pamięci RAM w oparciu o pozycję aktualnego piksela

- $HV\_valid$  – sygnał aktywny, który wysokim stanem informuje o aktualnym położeniu wewnątrz analizowanego obszaru,
- $Hmod$ ,  $Vmod$  – liczniki modulo służące do rozdzielenia pikseli wchodzących w skład różnych histogramów (kwadratów o boku  $4 \times 4$ ),
- $Hmult$ ,  $Vmult$  – zmienne będące bazą adresową do zapisu aktualnego histogramu (horizontalna zmienna powiększana o 9, wertykalna o  $9 \cdot 24$  – liczba histogramów w linii poziomej),
- $calculate\_en$  – sygnalizacja zakończonego procesu obliczania i zapisywania histogramów – gotowość do rozpoczęcia normalizacji i klasyfikacji dla danej skali.

Logika, działająca niezależnie dla każdej skali, jest reinitializowana po odebraniu sygnału nowej ramki (*vsync*). W momencie otrzymania informacji o kolejnym zestawie przedziałów i modułu, liczniki  $Hcnt$  oraz  $Vcnt$  określają jego obecność względem oczekiwanej obszaru detekcji. Jeśli analizowany zestaw danych przynależy do obszaru, liczniki modulo  $Hmod$  i  $Vmod$  są inkrementowane – odpowiednio co każdy piksel w obszarze, oraz co kolejną linię w obszarze. Podzielność któregokolwiek z nich przez 4 oznacza zmianę histogramu dla kolejnych danych – wymaga to powiększenia rejestru  $Hmult$  o 9, lub  $Vmult$  o 216. Ostatecznie, dostęp do odpowiednich adresów pamięci są przedstawione równaniem:

$$\begin{aligned} addr_{up} &= Hmult + Vmult + s\_bin_{up} \\ addr_{dn} &= Hmult + Vmult + s\_bin_{dn} \end{aligned} \quad (6.3)$$

Pamięć histogramu pracuje w trybie True Dual Port, umożliwiając jednoczesny dostęp do dwóch interpolowanych przedziałów aktualnego histogramu,  $s\_bin_{up}$  oraz  $s\_bin_{dn}$ . Zapis danych do pamięci histogramu jest realizowany po odczycie aktualnych wartości komórek i powiększeniu ich o odpowiednie części modułu:  $module_{up}$  oraz  $module_{dn}$ .

#### 6.4.5. Uczenie

Założeniem jest, by podczas pracy systemu wbudowanego nie ingerować we współczynniki, a opierać się na pierwotnych wynikach uczenia. Dane te muszą być przechowane w odpowiedni sposób, by możliwy był do nich prosty i szybki dostęp. Postanowiono zapisać wektor w pamięci ROM inicjalizowanej plikami typu *\*.mem*, utworzonymi podczas wykonywania skryptu uczenia w MATLABie. Ręcznie dostosowany moduł pamięci posiada trzy niezależne sektory (w zakresie adresowania i długości danych), inicjalizowane następującymi informacjami:

- składniki skalujące  $shifts$  – 16740 elementów wymaganych do przesunięcia każdego elementu wektora cech. Wartości w przedziale:  $< -0.2616, -0.0527 >$ ; precyzja zapisu: S0.11.

- współczynniki maszyny wektorów nośnych *vectors*. Wartości w przedziale:  $< -0.0076, 0.0063 >$ ; precyza zapisu: S0.27 (w formacie S0.23, lecz 4 najstarsze bity mają zawsze postać bitu znaku).

Dodatkowym współczynnikiem jest wartość przesunięcia gotowego wyniku o precyzyji S0.40, jednak jest ona przechowywana w logice. Powyższa pamięć zajmuje aż 36 z wszystkich 140 bloków BRAM dostępnych w rozważanym układzie. Należy zauważyć, iż wymusza to współdzielenie pojedynczej instancji modułu we wszystkich procesach klasyfikacji. Z tego względu istotne jest stworzenie logiki synchronizującej początek przetwarzania wektorów cech ze wszystkich skal – opisane jest to w kolejnym podrozdziale.

#### 6.4.6. Klasyfikacja

Po obliczeniu wektorów cech następuje proces klasyfikacji. Poprzedza ją normalizacja w blokach, która jest częścią tego modułu – ze względu na obecność każdego histogramu w kilku różnych blokach. Odczytywane z pamięci ROM wartości współczynników klasyfikatora muszą być współdzielone pomiędzy obliczeniami przeprowadzanymi dla każdej ze skal obrazu, jednak w każdym przypadku tempo generowania histogramów nie jest jednakowe – proces ten przebiega szybciej dla większych obrazów (tam analizowany fragment obrazu pojawi się na wejściu wcześniej). O gotowości histogramów z odpowiedniej skali informuje indywidualny dla niej sygnał *calculate\_en*. Dopiero w momencie otrzymania wszystkich sygnałów *calculate\_en* (stan wysoki na wyjściu iloczynu logicznego) rozpoczęty jest właściwy proces klasyfikacji.

Moduł odpowiadający za sklasyfikowanie informacji pochodzących z pojedynczej skali zrealizowano w formie krótkiej maszyny stanu, na którą składają się następujące etapy:

- inicjalizacja – oczekiwanie na sygnał *full\_frame*, informujący o rozpoczęciu algorytmu na pełnej klatce obrazu
- czyszczenie pamięci przechowującej wektory cech z poprzednich uruchomień algorytmu
  - etap ten ma miejsce tuż po otrzymaniu sygnału *full\_frame*, który pojawia się podczas stanu wysokiego synchronizacji pionowej; jest wykonywany na tyle szybko, by pamięć mogła być zapisana wartościami histogramów z nowej ramki obrazu
- oczekiwanie na iloczyn sygnałów *calculate\_en*; inicjalizacja zmiennych algorytmu
- właściwa normalizacja i klasyfikacja

O ile zapisane w pamięci ROM współczynniki mają postać wektora cech, tak pamięć RAM przechowuje nieuporządkowane fragmenty histogramów. Wymagało to stworzenia logiki, która

łączy ze sobą dane ze ścisłe określonych adresów pamięci, interpretując je w postać deskryptora. Opisuje to diagram 6.12.

Kolejnym etapem jest normalizacja w blokach, która ze względu na prostszą realizację została umieszczona wewnątrz procesu klasyfikacji (tym bardziej, że idea bloków właściwie nie funkcjonowała we wcześniejszych modułach). Blokiem jest struktura 4 histogramów, czyli łączanie 36 wartości. Odczytane z pamięci RAM dane są podnoszone do kwadratu przez mnożarkę, a następnie sumowane z kolejnymi wynikami. Suma 36 takich wartości, dodatkowo zinkrementowana, jest poddawana pierwiastkowaniu i będzie stanowić mianownik w procesie normalizacji powiązanego bloku (zgodnie z równaniem (4.28)). Moduł pierwiastkujący działa potokowo, więc zwraca również wartość pierwiastkową z niepełnych sum – dlatego ważne jest wygenerowanie sygnału aktywnego w momencie zsumowania 36 elementów, a także zatrzaśnięcie poprawnej wartości pierwiastka na czas 36 dzieleń.

Normalizacja wymusza ponowne odczytanie danych z pamięci RAM. Zastosowanie dwuportowego modułu pozwala na uzyskanie dostępu do uprzednio przetworzonych danych na drugim porcie, podczas gdy pierwszy kontynuuje zwracanie informacji potrzebnych do obliczenia współczynników normalizacji dla kolejnych bloków. Poprawną kolejność danych na drugim porcie osiągnięto przez odpowiednie opóźnienie sygnału adresowego z portu pierwszego.

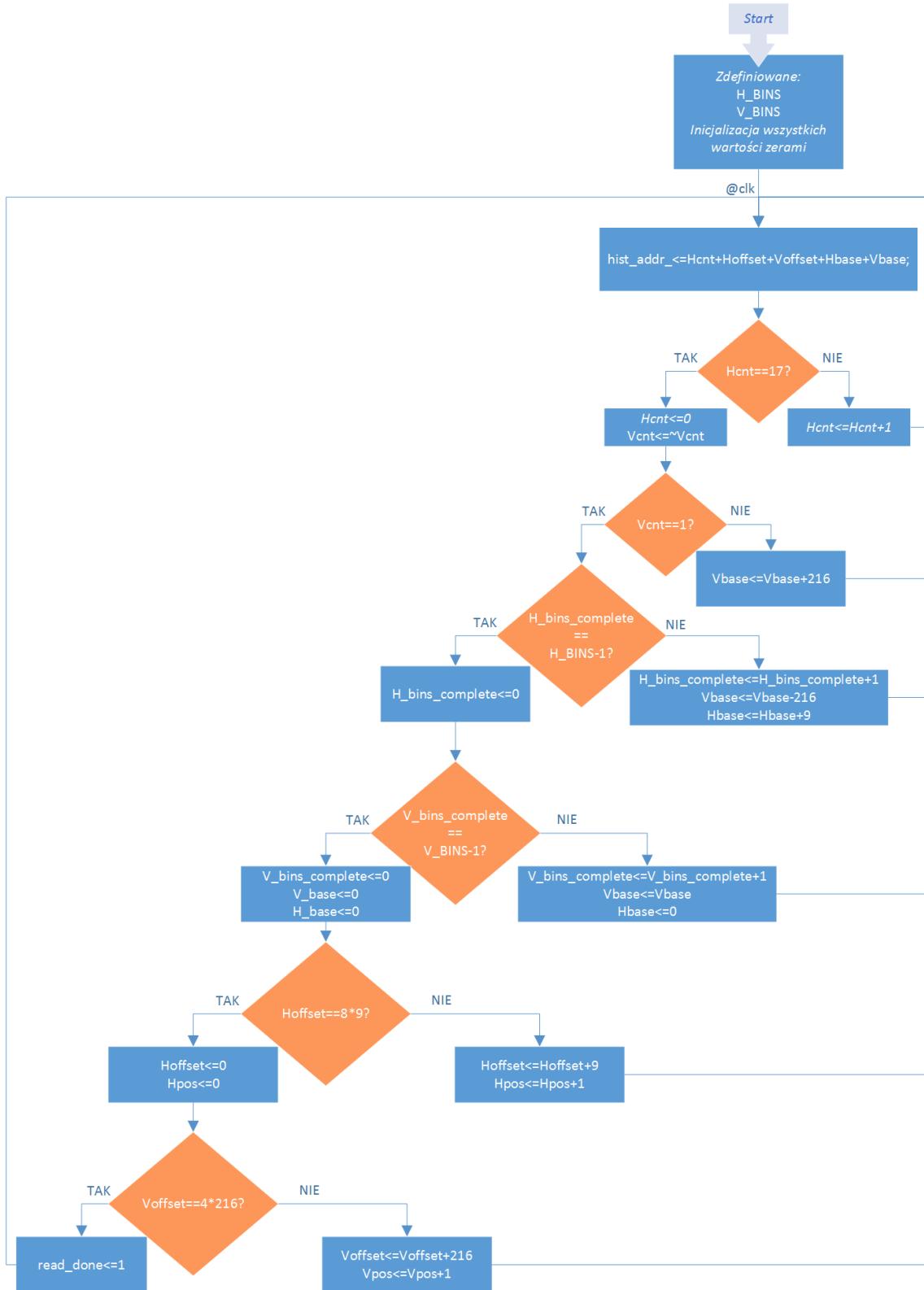
Napływające potokowo dane z portu drugiego, podzielone przez odpowiedni współczynnik normalizacji, mają ustawioną flagę *normalized\_valid*. Jej stan wysoki rozpoczyna inkrementację adresu pamięci ROM przechowującej współczynniki przesunięć (*shifts*) pochodzące z procesu uczenia. Każda z 16740 par zostanie do siebie dodana. Znormalizowane elementy wektora cech mają postać U11.13, zatem należało uprzednio rozszerzyć wektor *shifts* do tego formatu.

Kolejnym krokiem jest wymnożenie każdej z sum przez właściwy współczynnik maszyny wektorów nośnych. Aby to osiągnąć, należało ponownie i odpowiednio opóźnić linie adresowe uzyskujące dostęp do przestrzeni adresowej danych oraz *vectors*. Ostateczna szerokość pojedynczego elementu wynosi S7.40.

Docelowa wartość definiująca detekcję jest sumą wszystkich 16740 przetworzonych elementów oraz jeszcze jednej stałej wyznaczonej na etapie uczenia, *offset* równej  $-0.6327$ . *Offset* stanowi wartość początkową sumy przy rozpoczęciu obliczeń dla kolejnego wektora cech. Ze względu na konieczność zapewnienia dobrej dokładności w procesie sumowania, wynik jest zapisywany w notacji S7.40.

#### 6.4.7. Przetwarzanie wyników

Niezależnie od liczby przetwarzanych skal obrazu, etap klasyfikacji jest dla nich realizowany równolegle, przetwarzając po  $5 \times 9$  wektorów cech i zakończy się w tym samym momencie. Zarówno tworzenie deskryptorów, jak i klasyfikacja są zaimplementowane w modułach



Rysunek 6.12. Procedura wyboru adresu pamięci RAM w procesie odczytu kolejnych histogramów

działających w oparciu o zegar 148.5MHz. Pozwala to zrealizować sam etap klasyfikacji w czasie ok. 5ms.

Kolejnym etapem jest analiza wyników ze wszystkich skali. Jest on dość prosty, gdyż zakłada jedynie porównanie najlepszych rezultatów – w modułach klasyfikacji każdej ze skali zaimplementowano logikę, która zapamiętuje swoje lokalne minima i parametry obszarów (w odpowiednich skalach), na podstawie których je osiągnięto. Skutkiem porównań jest wyłonienie skali z najlepszym wynikiem, a koordynaty tego obszaru zostają dostosowane do rozdzielczości  $1280 \times 720$  i mogą być wykorzystane przez warstwę wyższą.

## 6.5. Integracja układu FPGA z dronem

Opisana w poprzednim podrozdziałach architektura sprzętowa dotyczyła dwóch algorytmów, które mogą funkcjonować niezależnie od siebie. Istnieje jednak jeszcze najwyższa warstwa logiczna, która łączy ich działanie, zwiększając niezawodność działania systemu. Na podstawie ostatecznych wyników śledzenia podejmowane są decyzje związane z ruchem drona. Muszą być one przekazane niezależnej jednostce odpowiedzialnej za stabilizację drona – autopilotowi. Wymagało to m.in. określenia sposobu komunikacji pomiędzy nim a układem Zynq.

### 6.5.1. Zdalne uruchomienie pracy systemu

Początkowo stworzono (i ostatecznie pozostawiono) możliwość rozpoczęcia pracy algorytmów z wykorzystaniem jednego z przełączników na płycie PYNQ, gdyż prace nad projektem wymagały dość częstego uruchamiania algorytmu jeszcze bez udziału drona. Jednak lot maszyny lub nawet jego start praktycznie wyklucza fizyczny dostęp do układu FPGA, dlatego koniecznością stało się zdalne wyzwolenie pracy algorytmów. Sygnał wyzwalający nosi nazwę *trigger\_algorithm*.

Podczas lotu maszyny użytkownik ma pod ręką jedynie aparaturę radiową, z poziomu której może jednak w dość prosty sposób wysyłać sygnały. Aparatura jest bowiem wyposażona w szereg konfigurowalnych przełączników analogowych i cyfrowych. Ponadto, dane są wysyłane poprzez 16 kanałów – przy czym do kontroli podstawowych funkcji drona wykorzystuje się zaledwie 4. Wybrano zatem jeden z pozostałych dostępnych kanałów, któremu przypisano funkcję trzypoziomowego przełącznika obecnego na panelu urządzenia radiowego.

Sparowany z aparaturą odbiornik, który jest zamocowany na dronie, wysyła wszystkie dane po jednym zestawie przewodów w formie PPM (ang. *Pulse Position Modulation*). Taki sygnał musiałby być zdekodowany w części PL układu Zynq w celu uzyskania użytecznej wartości. Proces dekodowania rozwiązuje jednak autopilot, który konwertuje 16-kanałowy PPM na znacznie prostsze sygnały PWM (ang. *Pulse Width Modulation*) m.in. do kontroli silników. Co

więcej, urządzenie Pixhawk umożliwia przypisanie reszty kanałów transmisyjnych do pomocniczych wyjść PWM. Rozwiążanie to znalazło zastosowanie chociażby w ustawieniu wartości zadanych serwomechanizmom odpowiedzialnym za pracę gimbala kamery.

Z pomocą oprogramowania Mission Planner, służącego do konfiguracji autopilota, ustalono zakres szerokości wysyłanego pulsu PWM na 1100 – 1900 us. Wybranie odpowiedniej wartości wyzwalającej pracę systemu wymagało dodatkowej weryfikacji, podczas której sprawdzono przypadek z wyłączoną aparaturą radiową (brak wejściowego sygnału PPM) i ewentualne błędy w transmisji PWM. Do pomiarów zastosowano analizator logiczny ILA wbudowany w środowisko Vivado i stworzono prosty licznik, który jest poddawany następującym operacjom:

- kasowanie na zboczu narastającym sygnału wejściowego,
- inkrementację podczas aktywnego stanu sygnału,
- przypisanie tej samej wartości w każdym innym przypadku.

Logika jest taktowana zegarem  $calc\_clk = 100\text{MHz}$  i dla poszczególnych pozycji przełącznika na aparaturze radiowej wartości licznika są przedstawione w tabeli 6.5. Pokazuje ona, że zakres długości pulsu nie odstaje od normy typowego sygnału PWM, mimo że jest nieco zawężony.

**Tablica 6.5.** Szerokości pulsu PWM dla przełącznika odpowiedzialnego za start algorytmu

Pozycja przełącznika	Wartość licznika	Długość pulsu [ms]
-1	109000	1.09
0	149000	1.49
1	189000	1.89

Na tej podstawie określono, że warunkiem koniecznym do rozpoczęcia algorytmu będzie wystąpienie dwóch kolejnych pulsów o szerokości przynajmniej 1.8 ms (górnny limit przezornie ustalono na 2 ms). Analogicznie, zakończenie pracy algorytmu i powrót do ustawień domyślnych będzie miało miejsce po zmianie pozycji przełącznika z „1”, czyli po otrzymaniu dwóch kolejnych pulsów o szerokościach spoza zakresu [1.8 ms, 2 ms].

### 6.5.2. Kontrola pracy algorytmów MeanShift oraz HOG+SVM

Najwyższa warstwa w części programowej Zynq jest odpowiedzialna za integrację działania algorytmów MeanShift i HOG+SVM i zarządza sygnałami używanymi do ich niezależnego uruchamiania.

### 6.5.2.1. Komunikacja pomiędzy PL i PS

Nie jest to jednak najwyższy poziom zarządzania w układzie PYNQ – warstwa ta komunikuje się bowiem z aplikacją uruchomioną w części PS, która jest odpowiedzialna za nadzór nad pracą zarówno autopilota, jak i opisywanej do tej pory części PL układu Zynq. Komunikacja pomiędzy PS a PL jest realizowana dwutorowo:

- PL odbiera ustawienia konfiguracyjne z przestrzeni adresowej 32-bitowych rejestrów. Używanych jest 5 rejestrów, które są inicjalizowane w PS wartościami domyślnymi. Ich opis przedstawia tabela 6.7
- PL wysyła informacje 32-bitowym sygnałem GPIO, który jest skonfigurowany w części PS do wywoływanego przerwań po każdej jego zmianie. Utworzone są również 2 dodatkowe rejesty, z których dane odczytywane są w PS podczas obsługi tego przerwania. Strukturę przedstawia tabela 6.6

**Tablica 6.6.** Informacje wysyłane do PS w postaci rejestrów

Adres rejestru	Sygnały	Format	Pozycja w rejestrze
– (GPIO)	<p><b>PL_to_PS_control</b> – wymuszenie ruchu drona w oparciu o dane rejestru 0x04</p> <p><b>PL_to_PS_status</b> – status pracy algorytmów MS/HOG+SVM</p> <p><b>trigger_algorithm</b> – rozpoczęcie misji (sygnał z aparatury radiowej)</p>	U1.0 U4.0 U1.0	[12:12] [7:4] [0:0]
0x04	<p><b>y<sub>pos</sub></b> – odległość do punktu zadanego na osi x obrazu</p> <p><b>x<sub>pos</sub></b> – odległość do punktu zadanego na osi y obrazu</p> <p><b>z<sub>scale</sub></b> – wartość skali dla ostatnich detekcji</p>	S10.0 S10.0 U3.2	[30:20] [18:8] [4:0]

Do rozpoczęcia pracy algorytmów wymagane jest otrzymanie informacji o gotowości drona, czyli ustabilizowaniu jego pozycji w powietrzu. Służy do tego rejestr *PS\_to\_PL\_run\_alg* opisany w tabeli 6.7.

### 6.5.2.2. Pierwsza detekcja osoby

Po otrzymaniu sygnału *PS\_to\_PL\_run\_alg* następuje etap skanowania obrazu w celu pierwszej detekcji postaci. Przy założeniu, że osoba nie pojawi się w górnej i dolnej części ramki obrazu, można ograniczyć obszar poszukiwań do jego środkowego pasa. Wykorzystywany w tym celu algorytm HOG+SVM uruchamiany jest cyklicznie na sąsiednich obszarach detekcji w jego

**Tablica 6.7.** Informacje odbierane z PS w postaci rejestrów

Adres rejestru	Sygnał	Format	Wartość domyślna
0x00	<b>PS_to_PL_run_alg</b> – kontrola pracy maszyny stanów w PL nadzorującej algorytmy MS/HOG+SVM: 0 - nieaktywna 1 - aktywna	U1.0	0
0x04	<b>SVM_threshold</b> – wartość progu	S7.24	-0.1
0x08	<b>SVM_strong_threshold</b> – wartość progu drugiego stopnia	S7.24	-1
0x0C	<b>SVM_lost_iter</b> – maksymalna liczba iteracji opartych wyłącznie na wyniku MeanShift	U15.0	100
0x10	<b>SVM_iter_between_control</b> – liczba iteracji algorytmu SVM pomiędzy aktualizacjami uchybu regulacji do PS	U6.0	15

obrębie. Przedstawia to schemat 6.13. Istotne jest jednak dobranie wielkości, o jaką obszar detekcji (punkt środkowy) powinien być przesuwany wewnątrz pasa. Chcąc zapewnić wysoką dokładność detekcji, należy upewnić się, że pas ten zostanie przeanalizowany w sposób wystarczający dla każdej ze skal. W trakcie pojedynczej iteracji algorytmu HOG+SVM każda ze skal dysponuje obszarem detekcji o wymiarze  $144 \times 96$ , który należy odnieść do oryginalnej wielkości obrazu. Takie porównanie skal przedstawia tabela 6.8.

Można zauważyć, że obszar detekcji dla skali o najniższym współczynniku jest najmniejszy, co w efekcie wymusza dobranie odpowiednio małych przesunięć. Ponadto, kolejnym aspektem jest sposób działania algorytmu dla pojedynczej skali. Z obszaru detekcji o rozmiarach  $144 \times 96$  wybierane są okna  $128 \times 64$  z krokiem wynoszącym 4 piksele. Oznacza to, że pełnej analizie poddawanych jest jednorazowo jedynie  $20 \times 36$  pikseli. Przeanalizowanie wszystkich okien detekcji  $128 \times 64$  w skanowanym pasie w danej skali wymaga współdzielenia pewnych fragmentów pikseli pomiędzy kolejnymi iteracjami algorytmu HOG+SVM.

Skanowanie rozpoczyna się od analizy obszaru zlokalizowanego w lewym górnym rogu ramki – z punktem środkowym w  $\{204, 96\}$  – jest on następnie przemieszczany horyzontalnie o zadaną w tabeli wartość 6.8. Po zakończonej analizie obszarów znajdujących się w linii poziomej, cały proces jest powtarzany z przesunięciem wertykalnym – i tak do osiągnięcia dolnej krawędzi skanowanego pasa. Skanowanie obrazu  $1280 \times 720$  wymaga wykonania  $7 \cdot 16 = 112$

**Tablica 6.8.** Wielkość analizowanego obszaru HOG+SVM dla poszczególnych skali, w odniesieniu do oryginalnej wielkości obrazu

Skala	Rozmiar obszaru $144 \times 96$ na oryginalnym obrazie	Uwagi
1	$144 \times 96$	Skala nieużywana (referencja do poniższych wartości)
2	<b><math>288 \times 192</math></b>	Wymagane przesunięcia obszaru w procesie skanowania: <b><math>40 \times 72</math></b>
2.5	$360 \times 240$	
3	$432 \times 288$	
3.5	$504 \times 336$	
4	$576 \times 384$	

iteracji algorytmu dla kolejnych nieparzystych klatek, co trwa nieco mniej niż 4 sekundy. Proces ten może trwać krócej, jeśli w jego trakcie osiągnie się bardzo dobry wynik klasyfikacji, poniżej zadanego poziomu  $SVM\_strong\_threshold = -1$ ; w standardowej procedurze wystarczy zapamiętany najlepszy wynik poniżej  $SVM\_threshold = -0.1$ .

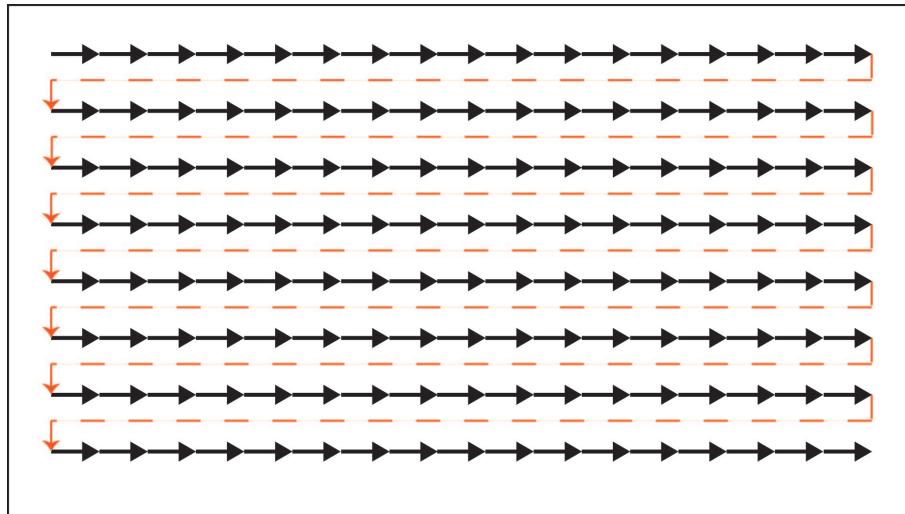
#### 6.5.2.3. Weryfikacja pierwszej detekcji

Kolejny etap, weryfikacyjny, jest uruchamiany bezpośrednio po zakończeniu skanowania i polega na przeprowadzeniu 3 kolejnych iteracji SVM – w trybie śledzenia (czyli na ograniczonym oknie, analizując obszar dla ostatniego wyniku SVM). Warunkiem przejścia przez ten etap jest, by wynik każdej iteracji był mniejszy od założonego maksimum  $SVM\_threshold$ . W przeciwnym wypadku część PL przechodzi w stan bezczynności, ustawiana jest odpowiednia wartość statusu  $PL\_to\_PS\_status$ , w odpowiedzi na którą część PS wyśle sygnał  $PS\_to\_PL\_run\_alg$  rozpoczynający ponownie etap skanowania. Mechanizm ten stworzono, by kwestie decyzyjne związaną z dalszym postępowaniem przenieść do PS.

#### 6.5.2.4. Śledzenie

Standardowe śledzenie opiera się na zweryfikowaniu wyniku ostatniej iteracji HOG+SVM, jednak proces uwzględnia jeszcze kilka elementów:

- start algorytmu MeanShift – wcześniej nieużywany algorytm MS ma możliwość rozpoczęcia pracy po zarejestrowaniu poprawnego wyniku klasyfikacji SVM (wartość mniejsza niż  $SVM\_strong\_threshold$ ). Wzorcem dla MS zostanie obszar z kolejnej klatki obrazu – kwadrat o środku w  $\{x, y - 30\}$  – gdzie  $\{x, y\}$  to aktualny środek obszaru wyliczonego



**Rysunek 6.13.** Proces skanowania – schemat przemieszczania okna detekcji na obrazie

przez algorytm SVM. Podane przesunięcie w pionie pozwala rozpoczęć śledzenie MS na obszarze w obrębie klatki piersiowej, wartość dobrano empirycznie.

- wyłączenie algorytmu MeanShift – MS może przesunąć obszar śledzenia poza śledzony obiekt, na przykład w związku ze zmianą oświetlenia. By się przed tym ustrzec, konieczne jest wyłączenie algorytmu, by w stosownym momencie ponownie go uruchomić. Odpowiedni warunek sprawdza, czy punkt MeanShift nie jest zbyt oddalony od punktu zwartego przez poprawnie działający algorytm SVM.
- utrata śledzonego obiektu – jeśli wynik ostatniej klasyfikacji nie spełnia *SVM\_threshold*, może to oznaczać potencjalną utratę śledzonej postaci z pola widzenia. Wówczas położenie osoby jest określone przez wynik działania algorytmu MeanShift, który stanowić będzie również wejście dla kolejnych iteracji HOG+SVM (poprawione o wspomniane wyżej przesunięcie 30). Warunkiem wyjścia z tego stanu (powrotu do standardowego śledzenia) jest osiągnięcie wyniku klasyfikacji rzędu *SVM\_strong\_threshold*. Istotnym ograniczeniem jest wartość *SVM\_lost\_iter*, która określa liczbę iteracji, w których ten tryb może pracować – domyślnie są to ok. 3 sekundy (100 iteracji), jednak użytkownik może tę wartość modyfikować poprzez zapis do rejestru 0xC z poziomu konsoli. Skala w tym trybie nie jest aktualizowana – ze względu na brak pozytywnych detekcji algorytmu HOG+SVM w śledzeniu wykorzystywana zostanie wartość skali z ostatniego pozytywnego wyniku detekcji.

Celem śledzenia jest utrzymanie obiektu w środku kadru. Sprowadza się to do ustalenia określonych wartości zadanych:

- położenie punktu środkowego SVM:  $\{x_{set}, y_{set}\} = \{640, 450\}$

- odległość kamery od osoby: 4m, co umożliwia pozytywną detekcję dla określonej skali obrazu:  $z_{set} = 3$

Wartość  $x_{set}$  jest dość oczywista i wyznacza dokładnie połowę szerokości kadru. Wartość  $y_{set}$  została dobrana eksperymentalnie i jest związana potrzebą utrzymania odpowiedniej wysokości przez drona – ok 1.5m (zależnie od skali). Ostatecznie, dla skal użytych w projekcie: 2/2.5/3/3.5/4, dobór  $z_{set} = 3$  jest najbardziej rozsądny ze względu szacowaną odległość postaci od drona – 4m – oraz możliwość detekcji postaci w bliższej oraz dalszej odległości.

Uchyb regulacji jest wyznaczany po każdej iteracji algorytmu SVM. Opiera się na obliczeniu różnicy pomiędzy obecnymi koordynatami SVM, a ustalonymi wartościami zadanimi. Ponadto, w przypadku skali zdecydowano, by obliczana była średnia krocząca 4 ostatnich wyników poprzez ich zsumowanie, a następnie przesunięcie bitowe o 2 w prawo, równe dzieleniu przez 4. Taki zestaw danych jest co określoną liczbę iteracji (domyślnie 15, czyli 0.5s) przepisywany do rejestru 0x04 do którego dostęp ma część PS. Flaga *PL\_to\_PS\_control* ustawiana wówczas w sygnale GPIO wywołuje obsługę przerwania, skutkującego ruchem drona. Częstotliwość wysyłania takiego sygnału można konfigurować rejestrem *SVM\_iter\_between\_control*.

### 6.5.3. Komunikacja Zynq <-> autopilot

Wymiana informacji pomiędzy autopilotem, a układem SoC bazuje na transmisji UART o standardowej prędkości równej 115200 bodów. Za transmisję po stronie układu SoC jest odpowiedzialna część PS, jednak istnieje możliwość lokalizacji sygnałów RxD oraz TxD po stronie części PL (wybór spośród większej liczby pinów) [39]. Z kolei w autopilotie skonfigurowano port TELE 2 [35]. Warstwą transportową jest protokół MAVLink, który opisano w dodatku B.1.2

Został on zaprojektowany w postaci plików nagłówkowych – taka forma umożliwia łatwe wykorzystanie w aplikacji tworzonej na jeden z rdzeni procesora ARM dostępnego na SoC Zynq. Ponadto, każda z dostępnych wiadomości ma swój podzbiór funkcji umożliwiających proste pakowanie w zestaw bajtów lub poprawne sparsowanie odebranych informacji.

#### 6.5.3.1. Aplikacja uruchomiona na procesorze ARM układu SoC Zynq

O ile część PL układu Zynq jest tworzona w środowisku Xilinx Vivado, to narzędziem deweloperskim obsługującym PS jest inny program będący częścią pakietu firmy Xilinx – SDK (Software Development Kit). By jednak możliwe było rozpoczęcie pracy, konieczne jest zaimportowanie z Vivado tzw. konfiguracji sprzętowej, czyli informacji o dostępnych w PS peryferyach, oraz połączeniach pomiędzy PS a PL.

Jednym z nich jest GPIO, do którego podłączony został sygnał *trigger\_algorithm*, status algorytmów oraz wartości uchybu regulacji. Odpowiednio skonfigurowane GPIO może generować przerwania, co wykorzystano jako reakcję na zmianę stanu któregokolwiek z opisanych wyżej sygnałów.

Pozostałymi peryferiami są dwa interfejsy UART. Lokalizacją dla pierwszego interfejsu UART są dwa piny, które połączono z odpowiednim portem autopilota. Aplikacja, korzystając z biblioteki MAVLink, jest w stanie zdekodować i sparsować ramki przychodzące oraz odpowiednio spakować komendy wysyłane do autopilota.

Drugi interfejs UART, wyprowadzony domyślnie na wyjście microUSB płyty PYNQ, stanowi połączenie z komputerem jako element różnorakich testów i debugowania w trakcie implementacji. Utworzono dla niego prostą formę terminala, który w odpowiedzi na komendy wpisywane z klawiatury komputera umożliwia zapis i odczyt z rejestrów testowych, wysyłanie określonych wiadomości do autopilota, ale przede wszystkim wyświetla odpowiednio sparsowane informacje wysyłane przez autopilot.

W aplikacji zaimplementowano nadzorską maszynę stanu, której częstotliwość pracy jest nadana przez odbierane co sekundę wiadomości z autopilota o ID=0, które noszą nazwę „HEARTBEAT”. Odpowiada ona za przygotowanie maszyny do śledzenia: uzbrojenie (komenda ARM), start (komenda TAKEOFF) i oczekanie określonego czasu przed rozpoczęciem skanowania. Wysłanie nowej komendy ruchu w trakcie etapu śledzenia jest realizowane niezależnie podczas obsługi przerwania związanego z GPIO, po otrzymaniu sygnału *PL\_to\_PS\_control*. Wykorzystywana jest w tym celu komenda SET\_POSITION\_TARGET\_LOCAL\_NED, której opis znajduje się w dodatku B.1.2.1. Warto zaznaczyć, że o ile oprogramowanie autopilota ArduCopter wyróżnia 14 trybów lotu drona, to trybem, który realizuje komendy ruchu wysyłane protokołem MAVLink, jest tryb GUIDED. Zostało to szerzej opisane w dodatku B.1.1.

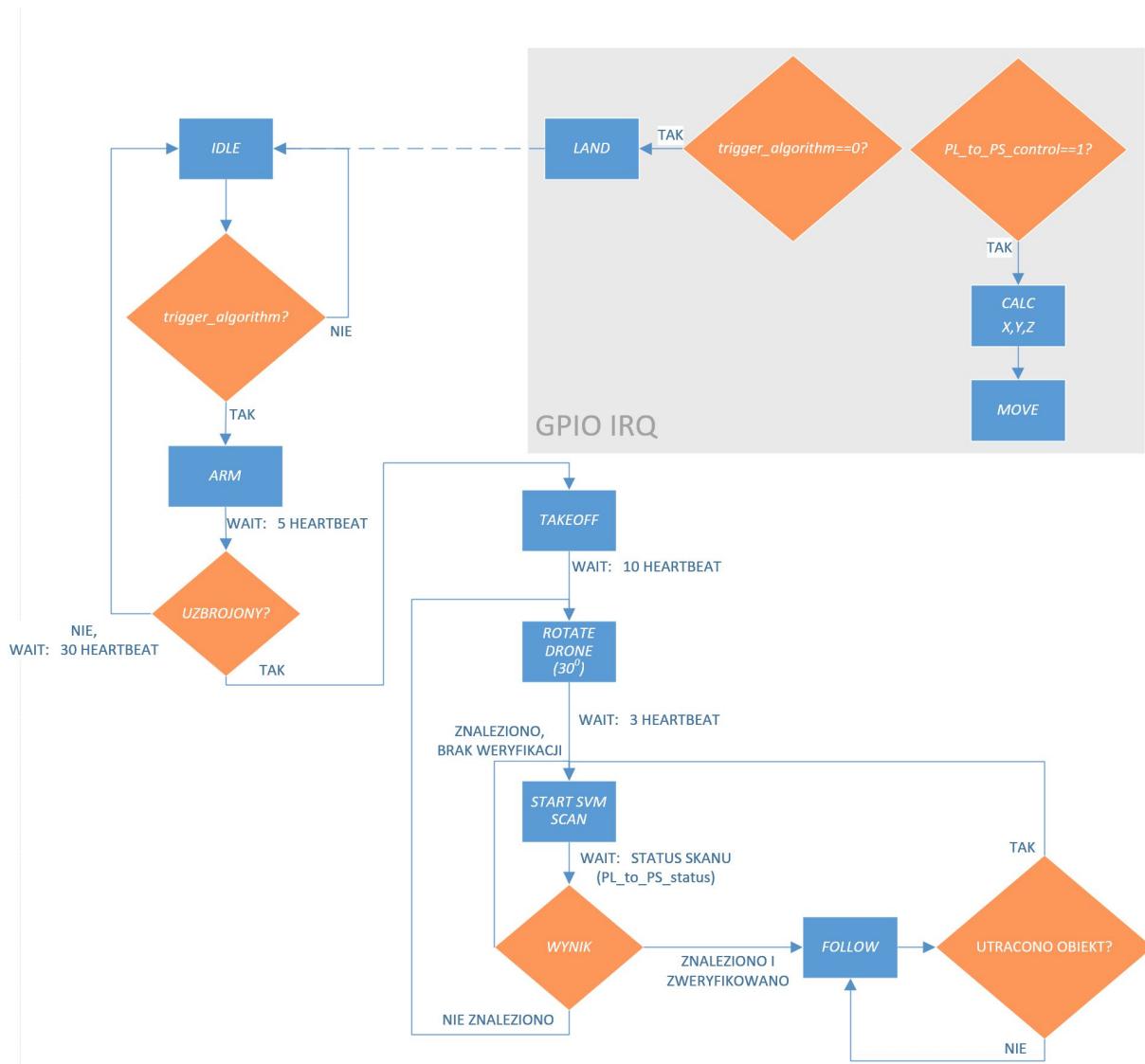
Podstawowym warunkiem pracy maszyny stanu jest obecność sygnału *trigger\_algorithm*. Po jego zboczu narastającym następuje próba uzbrojenia drona; po zmianie stanu sygnału na niski (zmiana stanu przełącznika aparatury radiowej) procesor wyśle komendę rozpoczynającą proces lądowania. Bardziej szczegółowy opis komunikacji z dronem przedstawia schemat 6.14. Ostatecznie, przesunięcia dla każdej osi są obliczane według wzorów realizujących regulator proporcjonalny:

$$\begin{aligned}x_{mov} &= \frac{z_{set} - z_{new}}{x_w} \\y_{mov} &= \frac{x_{new} - x_{set}}{z_{new} \cdot y_w} \\z_{mov} &= \frac{y_{new} - y_{set}}{z_{new} \cdot z_w},\end{aligned}\tag{6.4}$$

gdzie:  $y_{new}$ ,  $x_{new}$ ,  $z_{new}$  to informacje pochodzące z algorytmów MS/HOG+SVM, a  $x_w$ ,  $y_w$ ,  $z_w$  są empirycznie dobranymi współczynnikami dopasowującymi wartości do jednostek. Nie bez znaczenia jest także skojarzona z wynikiem detekcji skala obrazu – dla mniejszych skal (obiekt

dalej od kamery) potrzebny jest większy ruch drona. Warto zwrócić uwagę na zmianę osi układu współrzędnych drona:

- $x$  – oś pozioma odpowiadająca kierunkowi ruchu statku powietrznego (aspekt głębokości na obrazie),
- $y$  – oś pozioma prostopadła do osi  $x$ ,
- $z$  – oś pionowa, zwrócona w dół.



Rysunek 6.14. Schemat działania aplikacji w PS

Domyślnie, maszyna stanu znajduje się w trybie bezczynności i oczekuje na pojawienie się sygnału *trigger\_algorithm*. Wówczas podejmowana jest próba użbrojenia z czasem oczekiwania – 5 sekund. Autopilot wykonuje wtedy serię pomiarów weryfikacyjnych. Często ma jednak

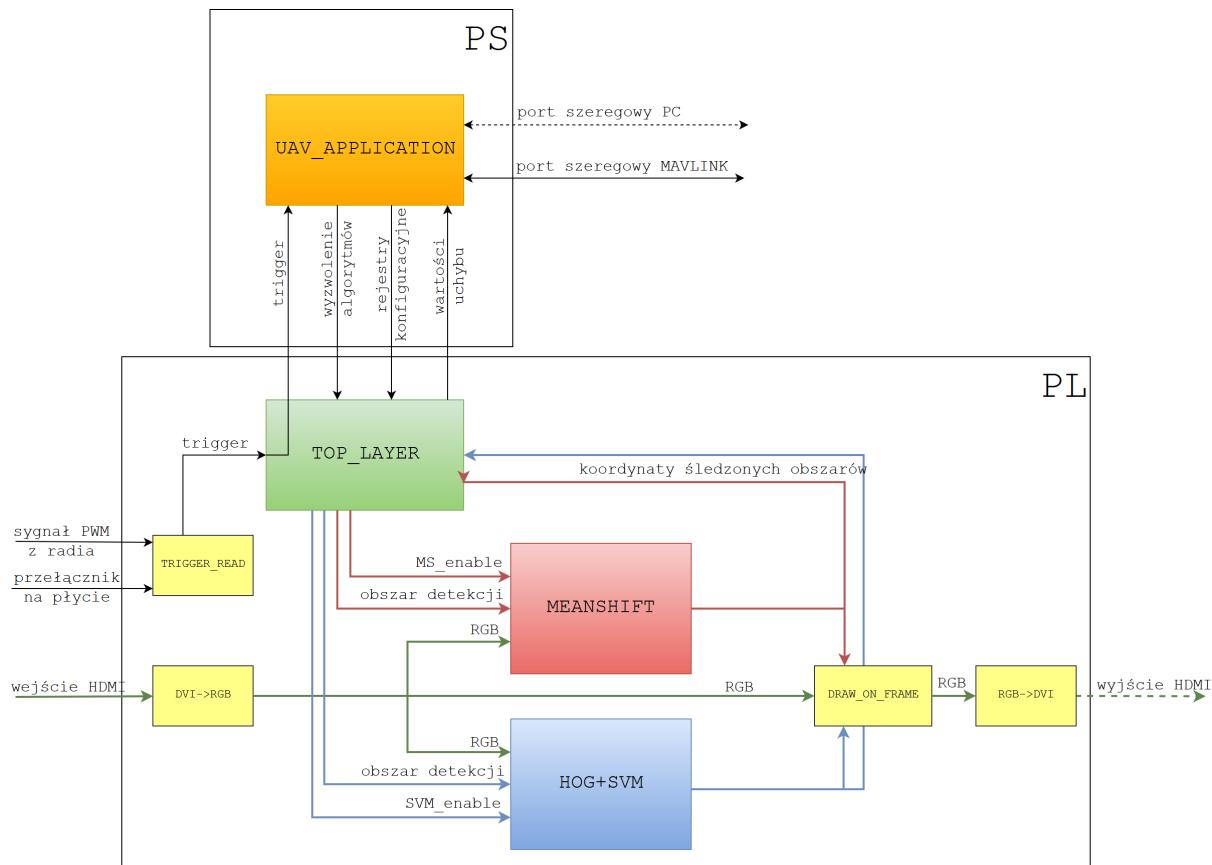
miejsce sytuacja, w której dron nie zostanie uzbijony np. w wyniku otrzymania niedokładnych pomiarów GPS. Jeśli proces ten się nie powódł (informacja o stanie drona jest przesyłana z wiadomością HEARTBEAT), mija kolejne 30 sekund, i ponawiana jest próba uzbijenia – do skutku. Pomyślny przebieg procedury uzbijenia pozwala wysłać komendę TAKEOFF z parametrem określającym docelową wysokość drona – 2m. Po kolejnych 10 sekundach dron potwierdza responsywność, obracając się o 30 stopni zgodnie z ruchem wskazówek zegara i po 3 sekundach rozpoczyna proces skanowania. PS jest informowane o stanie tego procesu poprzez sygnał *PL\_to\_PS\_status* – w oparciu o aktualizację jego wartości maszyna stanu wymusza obrót urządzenia i nowe skanowanie (brak detekcji) powtórzenie skanowania dla obecnej orientacji (brak weryfikacji) lub przechodzi do trybu śledzenia. W maszynie stanu ten etap został on zrealizowany dość pasywnie, bowiem analizuje jedynie stan detekcji i w przypadku permanentnej utraty obiektu z pola widzenia następuje powrót do procedury skanowania. Za wydawanie komend ruchu odpowiada funkcja obsługi przerwania GPIO, która jest uruchamiana po każdej zmianie tego sygnału.

## 6.6. Podsumowanie stworzonej architektury

Ostatecznie architekturę programowo-sprzętową może przedstawić schemat 6.15, opisujący najważniejsze zależności pomiędzy modułami. Zauważać można podobieństwo sygnałów związanych z implementacją poszczególnych algorytmów – upraszcza to ich integrację w module TOP\_LAYER.

Proces implementacji sprzętowej wymagał systematycznego nadzoru wykorzystania zasobów w układzie. Zbyt mała ilość dostępnych w logice elementów mogłaby mieć wpływ na rozmieszczenie (ang. *routing*) sygnałów i problemy związane z czasami ich ustalania lub podtrzymania (ang. *setup/hold time*). Tabele 6.9, 6.10 oraz 6.11 przedstawiają wykorzystanie zasobów w układzie Zynq 7Z020. Informacje te pozwalają wnioskować, że plany dalszego rozwoju systemu nie muszą wiązać się ze zmianą układu na większy. Jest to jednak aspekt, do którego należy podejść z rozwagą, mając na uwadze wymagania związane z odpowiednim rozmieszczeniem logiki w układzie. Z kolei bezpośrednie porównanie implementacji algorytmów MeanShift i HOG+SVM ukazuje znaczną dysproporcję w wykorzystaniu zasobów. Najbardziej trafnym uzasadnieniem może tu być fakt realizacji algorytmu HOG+SVM na pięciu skalach, które w uproszczeniu można potraktować jak 5 odrębnych instancji algorytmu.

Dodatkowo, postanowiono dokonać estymacji poboru mocy przez układ. W tym celu w środowisku Vivado wygenerowano raport w oparciu o określenie temperatury otoczenia równej 25°C i wybór najbardziej pesymistycznego poziomu estymacji. Z dokumentu wynika, że całkowity pobór mocy w układzie nie powinien przekraczać 3.5W, z czego jednak aż 1.3W to wartość



Rysunek 6.15. Schemat architektury programowo-sprzętowej

pobierana przez część PS. Fragment logiki realizujący metodę HOG+SVM potrzebuje 0.99W, natomiast MeanShift zaledwie 0.11W.

**Tablica 6.9.** Wykorzystanie zasobów dla implementacji algorytmu MeanShift

Rodzaj zasobu	Wykorzystane	Dostępne	Wykorzystanie [%]
LUT	5255	53200	9.8
FF	8562	106400	8.0
BRAM	25.5	140	18.2
DSP	28	220	12.7

**Tablica 6.10.** Wykorzystanie zasobów dla implementacji algorytmu HOG+SVM

Rodzaj zasobu	Wykorzystane	Dostępne	Wykorzystanie [%]
LUT	23894	53200	44.9
FF	32668	106400	30.7
BRAM	66	140	47.1
DSP	55	220	25

**Tablica 6.11.** Wykorzystanie zasobów - pełna architektura

Rodzaj zasobu	Wykorzystane	Dostępne	Wykorzystanie [%]
LUT	33729	53200	63.4
FF	46385	106400	43.6
BRAM	92	140	65.7
DSP	83	220	37.7



## 7. Weryfikacja działania systemu wizyjnego

Sprawdzenie poprawności działania obu algorytmów miało przebieg trzyetapowy:

- model programowy,
- symulacja modelu behawioralnego,
- uruchomienie algorytmu z warstwą sterującą na dronie.

### 7.1. Testy symulacyjne

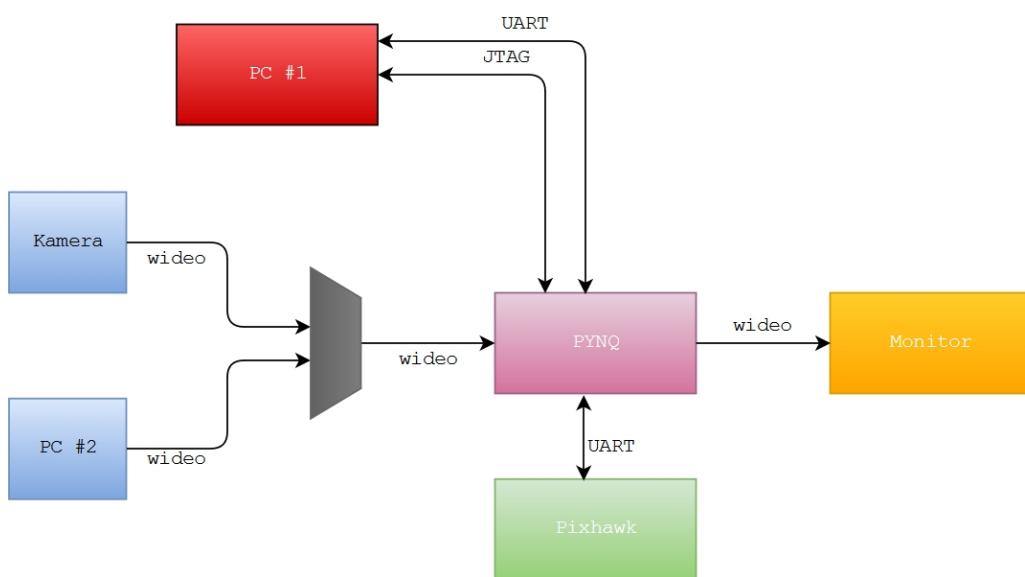
Implementacja sprzętowa jest procesem skomplikowanym, w którym łatwo o błąd utrudniający dalszą pracę. Z tego względu istotne jest równoległe przeprowadzanie testów symulacyjnych, które weryfikują poprawność tworzonej logiki. W projekcie testy te realizowano niezależnie dla obu algorytmów śledzących. Za referencję do symulacji obrano uprzednio stworzony model programowy w MATLABie, chociażby ze względu na wykorzystanie tych samych obrazów w celu porównania wyników. Moduł symulacyjny stworzony w środowisku Vivado i języku SystemVerilog nie uwzględniał jedynie części logiki zawartej w *Block Design* – a więc instancji procesora oraz wejściowego i wyjściowego fragmentu toru wizyjnego. Testy procesora nie mają związku z weryfikacją działania algorytmów, a mocno spowalniałyby pracę symulatora. Jako zmienną wejścia informacji wizyjnej, wystarczył zasymulowany komplet sygnałów RGB z informacją o wartości piksela, która została wczytana z pliku tekstopwego. Plik przechowujący jedną ramkę obrazu, wygenerowano w MATLABie umieszczając każdy kolejny piksel w nowej linii w formacie heksadecymalnym, po dwa znaki na każdą składową R, G i B.

Podstawowej zaletą symulacji jest możliwość podejrzenia propagowanych w układzie sygnałów w oknie wyświetlającym ich przebieg czasowy. Jednak ze względu na zwiększający się poziom skomplikowania projektu, z czasem postanowiono uprościć porównanie wybranych wyników symulacji z pracą modelu programowego. W kodzie architektury stworzono więc logikę zapisującą do plików określone informacje z pojedynczej iteracji algorytmu. By jednak zapis nie był realizowany na każdym zboczu narastającym zegara, potrzebne było określenie

sygnałów wyzwalających - zazwyczaj były to odpowiedniki sygnałów aktywnych powiązanych z określoną informacją. Do analizy stworzono w MATLABie dodatkowy skrypt, który parsował stworzone podczas symulacji pliki, uruchamiał pojedynczy przebieg modelu programowego i porównywał wyniki, określając liczbę błędów na danym etapie algorytmu dla całej ramki. Niektóre dane, z racji ograniczeń związanych ze stałoprzecinkową reprezentacją w architekturze, wymagały zdefiniowania akceptowalnego poziomu tolerancji błędu.

## 7.2. Testy w układzie Zynq

Symulacje są zbyt kosztownym czasowo narzędziem, dlatego w pewnym momencie należało przejść na testy na urządzeniu PYNQ. Proces budowy projektu sprowadza się do stworzenia konfiguracji sprzętowej w Vivado i skompilowania aplikacji uruchamianej na procesorze ARM. Docelowo układ SoC może być uruchomiony z poziomu karty SD, jednak ze względu na kwestię praktyczności, pozostało przy połączeniu JTAG. Stworzona konfiguracja testowa jest przedstawiona na schemacie 7.1. Zastosowaniem komputera (PC #1) jest nie tylko zaprogramowanie układu Zynq poprzez interfejs JTAG, ale i diagnostyczna komunikacja z układem, realizowana poprzez UART. Źródłem obrazu wideo może być dowolne urządzenie z możliwością wysłania sygnału 720p poprzez kabel HDMI. W tym wypadku jest to kamera, albo inny komputer – służący do odtwarzania wcześniej zapisanych materiałów wideo. Najważniejszym elementem podczas testów jest wyświetlenie obrazu wyjściowego, z nakreślonymi obszarami detekcji.



Rysunek 7.1. Schemat stanowiska testowego

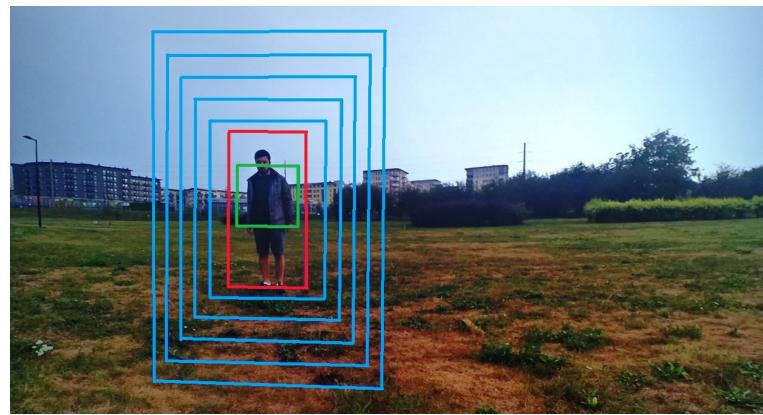
Test polega na skanowaniu ruchomego obrazu w poszukiwaniu postaci, a następnie śledzeniu znalezionej osoby. Analiza jest rozpoczętka w lewym górnym rogu, po czym sukcesywnie przechodzi w linii poziomej do prawej krawędzi, co jest następnie powtarzane dla niższych pozycji 6.13. Na przykładowych zrzutach z materiału wideo 7.2 zaprezentowano proces skanowania obrazu. Niebieskimi prostokątami oznaczone są aktualne okna detekcji dla poszczególnych skal. Zielone okno to obszar śledzenia algorytmem MeanShift, natomiast czerwonym kolorem oznaczono aktualnie najlepsze okno  $128 \times 64$  (przeskalowane ponownie do oryginalnej rozdzielczości).

Przebieg testu powinien być jak najbardziej zbliżony do pracy układu podczas lotu, z tego względu do procedury dodano komunikację UART z autopilotem. Nadzór nad nią jest sprawowany poprzez dodatkowe połączenie szeregowe łączące układ Zynq z komputerem, który wyświetla w oknie terminala wszystkie istotne komunikaty. Przykładowo, raport 7.1 przedstawia informacje uzyskane w ciągu pierwszych kilkunastu sekund jednego z testów. Po otrzymaniu sygnału startu (przełącznik na układzie PYNQ lub na aparaturze radiowej) do autopilota wysyłana jest komenda użbrojenia (ARM) i startu (TAKEOFF), jednak ze względu na obecność drona w budynku niemożliwe jest określenie pozycji poprzez GPS – odebrana z autopilota wiadomość COMMAND ACK informuje, że wykonanie komendy TAKEOFF (ID: 22) się nie powiodło (status: 4). Podczas testu naziemnego ten komunikat jest jednak ignorowany i rozpoczynany jest proces skanowania. Po dłuższej chwili układ wysyła informację o pozytywnym wyniku detekcji, po czym system rozpoczyna zadanie śledzenia. Od tego momentu większość komunikatów dotyczy informacji z PL o odległości śledzonego obiektu od wartości zadanej w pionie i poziomie. Trzecią wartością jest uśredniony wynik skali z 4 ostatnich detekcji – wartość przemnożona przez 10 z powodu braku funkcji wyświetlającej liczby zmiennoprzecinkowe. Niezależnie od przebiegu pracy systemu wizyjnego mogą pojawiać się wiadomości z autopilota o statusie (INFO). Test jest przerywany przez użytkownika w momencie zmiany pozycji przełącznika sygnału startu – oprócz zakończenia pracy algorytmu wysyłana jest wówczas komenda lądowania (LAND).

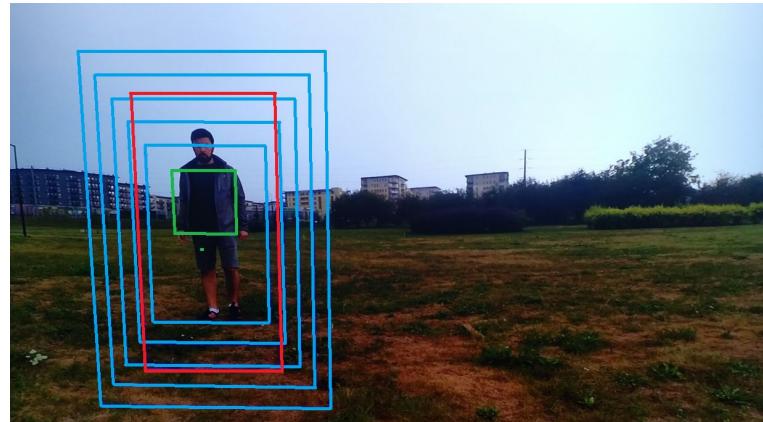
Ostatnim etapem testów było uruchomienie w pełni sprawnego systemu wizyjnego na platformie UAV. W ich trakcie nie tylko zweryfikowano poprawność działania systemu detekcji i śledzenia, ale również dostosowano parametry regulatora, który poprzez komunikację z autopilotem wpływał na ruch drona.



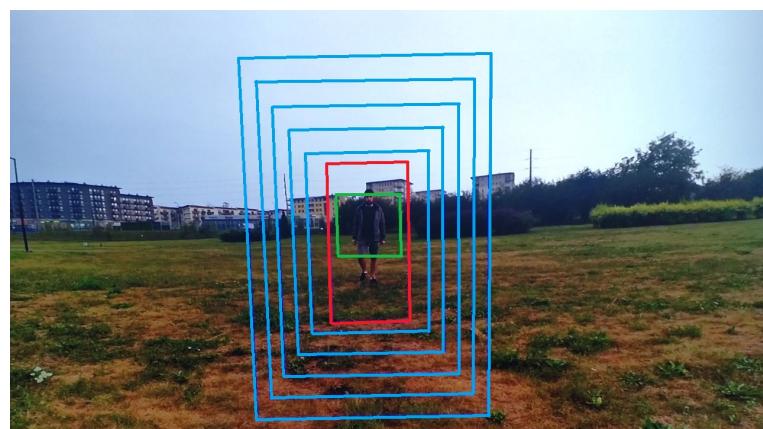
Rysunek 7.2. Proces skanowania



Rysunek 7.3. Śledzenie #1



Rysunek 7.4. Śledzenie #2



Rysunek 7.5. Śledzenie #3

```

prompt»ARMING...
ARMED!
TAKING OFF...
Sent TAKEOFF CMD.
COMMAND ACK: 22 4
STARTING SEARCH!
xDiff: 37      yDiff: 33      | scale: 27 ||
DATA OK, STARTING FOLLOW!
xDiff: 20      yDiff: 20      | scale: 25 ||
xDiff: 10      yDiff: 20      | scale: 25 ||
xDiff: 52      yDiff: 38      | scale: 22 ||
xDiff: 94      yDiff: 8       | scale: 22 ||
xDiff: 140     yDiff: 12      | scale: 20 ||
xDiff: 165     yDiff: 30      | scale: 25 ||
xDiff: 157     yDiff: 22      | scale: 22 ||
xDiff: 125     yDiff: 40      | scale: 25 ||
xDiff: 95      yDiff: 35      | scale: 22 ||
xDiff: 62      yDiff: 26      | scale: 20 ||
xDiff: 26      yDiff: 28      | scale: 20 ||
xDiff: 10      yDiff: 28      | scale: 20 ||
xDiff: -6      yDiff: 28      | scale: 20 ||
xDiff: -30     yDiff: 36      | scale: 20 ||
xDiff: -70     yDiff: 28      | scale: 20 ||
xDiff: -86     yDiff: 20      | scale: 20 ||
xDiff: -118    yDiff: 28      | scale: 20 ||
xDiff: -150    yDiff: 12      | scale: 20 ||
xDiff: -150    yDiff: 12      | scale: 20 ||
xDiff: -110    yDiff: 20      | scale: 20 ||
xDiff: -86     yDiff: 20      | scale: 20 ||
xDiff: -62     yDiff: 28      | scale: 20 ||
INFO: PreArm: Throttle below Failsafe
xDiff: 22      yDiff: 28      | scale: 20 ||
xDiff: 30      yDiff: 4       | scale: 22 ||
xDiff: 62      yDiff: 42      | scale: 27 ||
xDiff: 95      yDiff: 33      | scale: 27 ||
xDiff: 107     yDiff: 33      | scale: 30 ||
xDiff: 107     yDiff: 33      | scale: 30 ||
xDiff: 107     yDiff: 33      | scale: 30 ||
xDiff: 172     yDiff: 34      | scale: 20 ||
xDiff: 268     yDiff: 14      | scale: 20 ||
xDiff: 368     yDiff: 14      | scale: 22 ||
xDiff: 450     yDiff: 20      | scale: 25 ||
xDiff: 560     yDiff: 15      | scale: 22 ||
Sent LAND CMD.
COMMAND ACK: 21 0

```

**Tablica 7.1.** Przykładowa zawartość terminala ze śledzenia na podstawie gotowego materiału wideo

## **8. Podsumowanie i możliwości rozwoju pracy**

W zrealizowanym projekcie przedstawiono sprzętową-programową realizację detekcji i śledzenia osoby w heterogenicznym układzie Zynq SoC, na potrzeby kontroli bezzałogowego statku powietrznego. Osiągnięto przetwarzanie obrazu o rozdzielcości  $1280 \times 720$  dla 60 klatek na sekundę, z częstotliwością  $60Hz$  i  $30Hz$  odpowiednio dla algorytmów MeanShift oraz HOG+SVM.

Na szczególną uwagę zasługuje warstwa najwyższa, łącząca pracę obu algorytmów i komunikującą się z autopilotem Pixhawk. Zaimplementowano w niej mechanizmy realizujące ważną z punktu widzenia autonomizacji rolę decyzyjną, a wykorzystanie protokołu MAVLink zapewniło komunikację umożliwiającą ruch platformy UAV bez udziału pilota i ograniczyło zaangażowanie użytkownika dołączenia misji i jej zakończenia.

Zastosowany w pracy układ Zynq SoC okazał się być wydajną jednostką obliczeniową, która jest w stanie sprostać wymagającym założeniom projektu, a jej zróżnicowana architektura w postaci części PS oraz PL pozwala w najlepszy sposób wykorzystać korzyści wynikające z realizacji sprzętowej i programowej algorytmów.

Niemniej ważny okazał się dobór komponentów do budowy drona. Był on przedmiotem wielu analiz (również finansowych), a w efekcie otrzymano wartościową konstrukcję, stanowiącą platformę dla przyszłych badań w SKN AVADER.

Z projektem związanych jest kilka aspektów, których rozpatrzenie mogłyby wpłynąć na poprawienie niezawodności systemu. Jednym z nich mogłyby być próba zredukowania liczby fałszywych detekcji (HoG+SVM) poprzez zmianę wielkości bloków lub komórek. Innym pomysłem mogłyby być proste zwiększenie liczby analizowanych skal. Kolejnym, również ważnym pomysłem, mogłyby być udoskonalenie procedury uczenia klasyfikatora. Ponadto, obecny w projekcie zestaw współczynników SVM mógłby zostać przekonwertowany do klasycznej postaci (złożonej ze 16740 współczynników oraz przesunięcia), obniżając poziom wykorzystania pamięci BRAM.

Kolejnym usprawnieniem, związanym z algorymem MeanShift, byłoby zlikwidowanie rzadkich sytuacji utraty zbieżności ze śledzonym obszarem – co skutkuje „wędrowaniem okna”. Najczęściej ma na to wpływ nierównomierny i zmienny poziom oświetlenia sceny, co można byłoby korygować odpowiednim przetwarzaniem wstępny.

W warstwie najwyższej poprawie mógłby ulec zaimplementowany regulator – w tym wypadku nie chodzi tylko o lepszy sposób wyliczenia wartości sterujących, ale nawet o częstotliwość, z jaką są one wysyłane do autopilota. Biorąc pod uwagę specyfikę platformy UAV, wymaga to podejścia głównie empirycznego.

Barierą na drodze większości zmian jest liczba dostępnych zasobów w układzie – wymagana byłaby zmiana układu na dysponujący zwłaszcza większą liczbą bloków BRAM. Istnieją jednak zmiany niewymagające ingerencji w kod. Do podstawowych należałby lepszy dobór ustawień programowych kamery w celu poprawy rejestrowanych kolorów. Urządzenia sportowe tego typu są wyposażone w wiele usprawnień (tryb nocny, korekcja zniekształceń wprowadzanych przez obiektyw itp.), należy zatem zbadać wpływ ich zastosowania na działanie systemu wizyjnego. Kolejną mogłaby być lepsza stabilizacja obrazu – ze względu na podpięty do kamery dość sztywny przewód HDMI zmienia się jej środek ciężkości, co zaburza pracę gimbalu – silnik związany ze stabilizacją na jednej osi obrotu musiał być z tego powodu wyłączony. Ostatnią zmianą, mającą największy wpływ na śledzenie, byłoby poprawienie właściwości lotnych drona. Mowa tu głównie o problemach z płynnością ruchu oraz utrzymywaniem drona w zadanej pozycji – co może mieć związek z ustawieniami akceleratorów, dokładnością pomiaru sygnału GPS lub nawet zakłóceniami spowodowanymi niewystarczającą izolacją przewodów prądowych zasilających silniki.

Wykorzystanie układu rekonfigurowalnego na platformie UAV pozwala stworzyć ciekawe, a przy tym wartościowe naukowo projekty systemów wizyjnych lub systemów w inny sposób związanych z autonomizacją dronów. Podstawowym kierunkiem dla nowo tworzonych systemów mogłaby być zdolność omijania przeszkód (nadal kosztowna opcja w dronach komercyjnych). Detekcja mogłaby być realizowana przez specjalistyczne czujniki odległości, stereowizje lub nawet LIDAR.

## A. Spis zawartości płyty CD

Dołączona do pracy płyta CD zawiera następujące pliki i katalogi:

- praca.pdf – plik zawierający tekst pracy magisterskiej w formacie *pdf*,
- MATLAB – katalog zawierający pliki modelów programowych, a także pliki wideo służące do testów oraz zestaw treningowy SVM „INRIA Person dataset”,
- PYNQ – katalog zawierający projekt z plikami źródłowymi (PL oraz PS) do zbudowania i uruchomienia na układzie PYNQ,
- Praca\_TEX – katalog zawierający pliki L<sup>A</sup>T<sub>E</sub>Xz tekstem pracy magisterskiej oraz wykorzystanymi w niej rysunkami.



## B. Opis techniczny platformy

### B.1. Oprogramowanie ArduPilot

Oprogramowanie ArduPilot jest jednym z najbardziej popularnych systemów instalowanych na kontrolerach lotu. Jest to projekt *open source*, którego największą zaletą jest bogata możliwość konfiguracji. W zależności od modelu, którym użytkownik zamierza sterować, istnieją 3 wersje oprogramowania:

- ArduPlane – zarządzający samolotami i tzw. platformami FPV
- ArduCopter – obsługujący platformy multiotorowe i helikoptery jednowirnikowe
- ArduRover – przeznaczony do obsługi pojazdów naziemnych i nawodnych

Oprogramowanie można instalować na wspierających je kontrolerach, m.in urządzeniach serii APM lub Pixhawk. Proces taki przebiega z poziomu specjalnej aplikacji na PC, służącej głównie do zdalnego zarządzania ustawieniami autopilota: APM Planner lub MISSION Planner.

#### B.1.1. Tryby lotu autopilota

Oprogramowanie ArduCopter wyróżnia aż 14 trybów lotu, opisanych w dokumentacji [36]. Najważniejsze z nich to:

- *Stabilize* – podstawowy tryb umożliwiający manualne sterowanie dronem; autopilot stabilizuje osie obrotu Roll oraz Pitch. Wymaga nieustannego korygowania wychyleń drążków aparatury radiowej do utrzymania zadanej wysokości i pozycji.
- *Alt Hold* – wariant trybu *Stabilize* z automatycznym utrzymaniem wysokości dla sygnału Throttle o wartości zakresu 40 – 50%. Dla wychyleń drążka Throttle spoza tego zakresu dron wykona odpowiedni ruch pionowy, z maksymalną prędkością wznoszenia/opadania konfigurowaną przez parametr PILOT\_VELZ\_MAX (domyślnie 2.5m/s).
- *Loiter* – dalsze rozszerzenie trybu *Stabilize*, pozwalające utrzymać zadaną pozycję w przestrzeni. Użytkownik może dowolnie sterować dronem, jednak w momencie powrotu drążków do domyślnych pozycji platforma po krótkiej chwili się zatrzyma.

- *RTL - Return-to-Launch* – po zmianie trybu na RTL dron wznieśnie się na wysokość zdefiniowaną przez parametr RTL\_ALT – domyślnie 15 m (jeśli znajduje się wyżej, pomija ten proces). Następnie porusza się w linii prostej do lokalizacji, w której został uzbrojony – i opada na wysokość zdefiniowaną parametrem RTL\_FINAL.
- *Auto* – tryb realizujący predefiniowaną misję, która jest zapisywana w autopilocie z poziomu oprogramowania MISSION Planner.
- *Guided* (używany w projekcie) – tryb pozwalający akceptować komendy ruchu wydawane dynamicznie przez niezależne urządzenie, pełniące rolę tzw. stacji naziemnej. Komunikacja jest realizowana zazwyczaj poprzez bezprzewodowe łącze telemetryczne korzystające z portu szeregowego autopilota, jednak możliwa jest też transmisja przewodowa, z urządzenia umieszczonego na platformie. Protokołem wymiany danych jest MAVLink.
- *Land* – tryb rozpoczynający procedurę lądowania. Na wysokości większej niż 10 m opadanie następuje z prędkością zdefiniowaną przez parametr WPNAV\_SPEED\_DN (domyślnie 150cm/s), dla niższych wysokości jest to prędkość związana z parametrem LAND\_SPEED (domyślnie 50cm/s). Autopilot wykrywa kontakt z ziemią, jeśli różnica w zmierzonej przez barometr wysokości jest mniejsza niż 20cm przez minimum jedną sekundę.

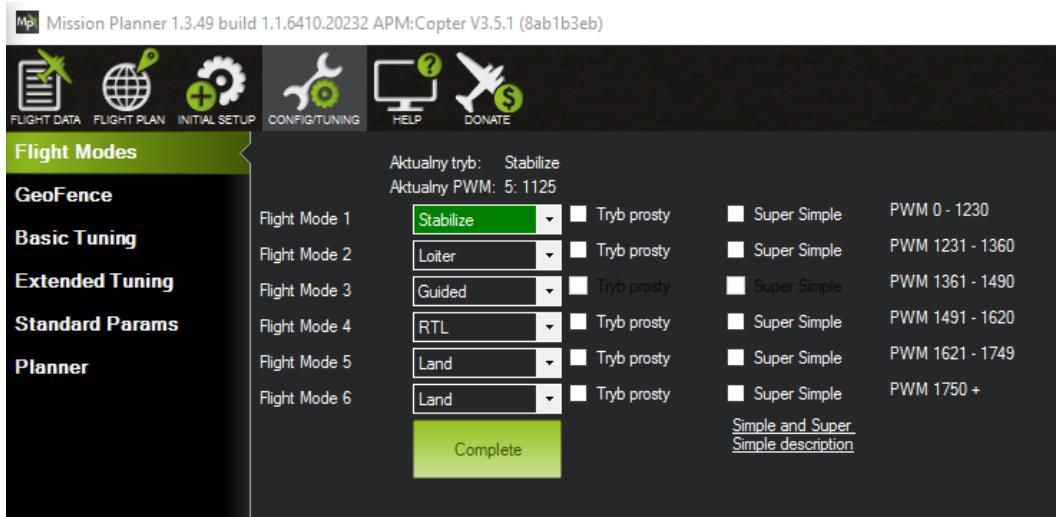
Aktualny tryb autopilota można zmienić, wysyłając odpowiednią komendę poprzez protokół MAVLink – jednak podstawowym sposobem jest wykorzystanie aparatury radiowej. ArduPilot rezerwuje bowiem piąty kanał transmisji radiowej do ustawienia jednego z 6 dostępnych trybów. Każdy z nich ma odgórnie przyporządkowany zakres długości pulsu PWM, dla którego może być aktywowany. Aplikacja MISSION Planner umożliwia konfigurację tej funkcjonalności z perspektywy autopilota – rysunek B.1 przedstawia tryby lotu zapisane w urządzeniu na potrzeby projektu.

Opis konfiguracji kanału piątego przedstawiony jest w podrozdziale poświęconym aparaturze radiowej.

### B.1.2. Protokół MAVLink

Oprogramowanie ArduPilot obsługuje komunikację z urządzeniami pełniącymi rolę stacji naziemnych. Może to być komputer, tablet, ale również obecna na platformie UAV inna jednostka. Niezależnie od wyboru, komunikacja z autopilotem bazuje na wykorzystaniu jednego z jego portów szeregowych [35].

Warstwą transportową takiej komunikacji jest protokół MAVLink, opracowany w 2009 roku na potrzeby małych pojazdów bezzałogowych. Dość duży podzbiór jego wiadomości i komend



Rysunek B.1. Tryby lotu w oknie konfiguracyjnym aplikacji MISSION Planner

jest wspierany przez oprogramowanie ArduCopter (pełna lista jest dostępna pod adresem: [37], dość poręczna jest również strona [38]). Ramka protokołu ma zmienną długość i jest opisana w tabeli B.1.

Tablica B.1. Ramka protokołu MAVLink

Bajt #	Oznaczenie	Uwagi
0	Początek ramki	Zawsze o wartości 254
1	Długość danych	Wartość $n$ (w bajtach )
2	Sekwencja pakietu	Wartość inkrementowana z każdą kolejną transmisją
3	ID systemu	Dla komputera pokładowego (SoC) równa 255
4	ID komponentu	Dla komputera pokładowego (SoC) równa 190
5	ID wiadomości	
6:n+6-1	Dane	Struktura zależna od rodzaju wiadomości
n+6:n+7	CRC	Suma kontrolna całego pakietu bez bajtu #0

Protokół został zaprojektowany w postaci plików nagłówkowych – taka forma umożliwia łatwe wykorzystanie w aplikacji tworzonej w języku C/C++.

### B.1.2.1. Implementacja komend MAVLink

Dokumentacja protokołu MAVLink jest dość obszerna, jednak doprowadzenie do pełnego zrozumienia działania poszczególnych komend i ich wpływu na zachowanie drona wymaga podjęcia testów praktycznych. Każda z komend autopilota ma swój własny plik nagłówkowy, którego najważniejszą częścią jest definicja struktury z danymi, funkcji pakującej (do wysłania

wiadomości) oraz dekodującej (do odebrania wiadomości). Mają one dość generyczne nazwy, przykładowo dla wiadomości HEARTBEAT są to odpowiednio:

- *mavlink\_heartbeat\_t*
- *mavlink\_msg\_heartbeat\_pack(...)*
- *mavlink\_msg\_heartbeat\_decode(...)*

Używanie tych funkcji ma dodatkową zaletę – uwzględniają liczenie sumy kontrolnej oraz sekwencji pakietu, a więc elementów, których błędne wartości powodowałyby nieprawidłowości w transmisji. Do komunikacji pomiędzy urządzeniami wykorzystywany jest port szeregowy, zatem najmniejszą jednostką wymiany informacji jest bajt. Dla procesu odbierania wiadomości należy ten zestaw bajtów odpowiednio zinterpretować – do formowania ich w ramki służy funkcja *mavlink\_frame\_char(...)*, która powinna być wywoływana w trakcie odebrania każdego bajtu danych. Zwracany przez nią status określa, czy ramka jest już gotowa do dalszej analizy. Pełna wiadomość zostanie przepisana do struktury typu *mavlink\_message\_t*. Jedno z pól gotowej struktury, *msgid*, określa typ wiadomości. Pozwala to zdefiniować użycie odpowiedniej funkcji dekodującej w zależności od ID – gotowe informacje będą się znajdować w odpowiedniej strukturze danych. Przykładowo, ID o numerze 253 oznacza wiadomość typu STATUSTEXT (informacja z autopilota w formacie tekstowym). W celu jej zdekodowania należy wywołać funkcję *mavlink\_msg\_statustext\_decode(...)*, która zwróci strukturę *mavlink\_statustext\_t* ze znakami gotowymi do wyświetlenia lub dalszej analizy.

Wysyłanie komend jest o wiele prostsze, wymagane jest bowiem spakowanie informacji w tablicę bajtów i wysyłanie ich w odpowiedniej kolejności. Najbardziej skomplikowaną w użyciu jest komenda ruchu, SET\_POSITION\_TARGET\_LOCAL\_NED. Funkcja, która jest odpowiedzialna za konwersję jej w tablicę bajtów nosi nazwę *mavlink\_msg\_set\_position\_target\_local\_ned\_pack*. Używa ona następujących parametrów:

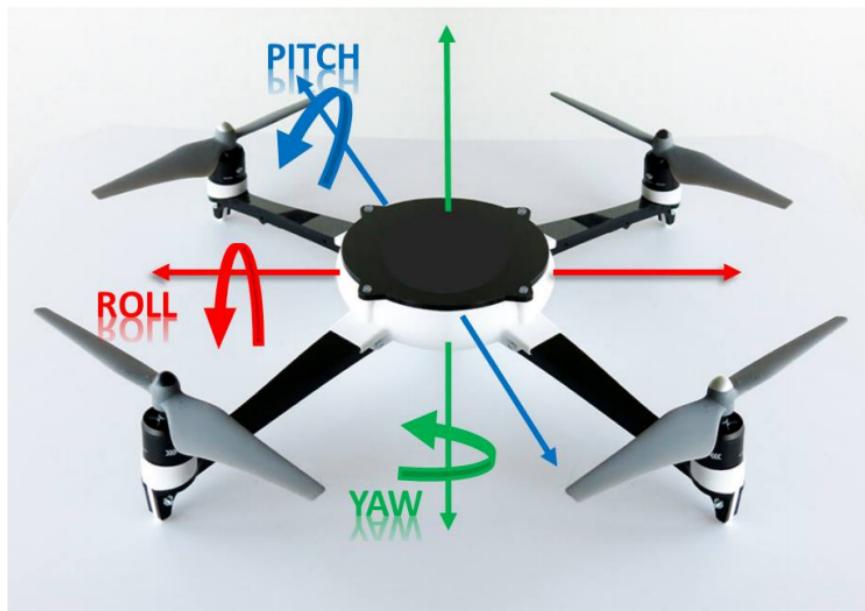
- *x, y, z* – przesunięcie w odpowiednim kierunku, w metrach
- *vx, vy, vz* – prędkość w odpowiednim kierunku, w *m/s*
- *afx, afy, afz* – przyśpieszenie w odpowiednim kierunku, w *m/s<sup>2</sup>* lub *N*
- *yaw* – obrót wokół pionowej osi, w radianach
- *yaw\_rate* – prędkość kątowa obrotu, w rad/s
- *type\_mask* – maska bitowa określająca wartości, które mają być zignorowane (wartość 1) przez autopilot. Mapowanie bitów 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7: ax, bit 8: ay, bit 9: az, bit 11: yaw, bit 12: yaw\_rate. Bit 10 jest używany do określenia jednostki przyśpieszenia

- *coordinate\_frame* – zdefiniowanie układu współrzędnych, w odniesieniu do którego odbywać się będzie ruch

Warto zaznaczyć, że do wykonania ruchu potrzebne jest odblokowanie wszystkich bitów związanych z przemieszczeniem LUB prędkością, kombinacja obu sposobów ruchu nie jest wspierana przez oprogramowanie ArduPilot. Parametry związane z przyśpieszeniem oraz obrotem w osi pionowej również nie są wspierane. Do przechowywania wartości wygodnie jest używać struktury *mavlink\_set\_position\_target\_local\_ned\_t*.

## B.2. Konfiguracja aparatury radiowej

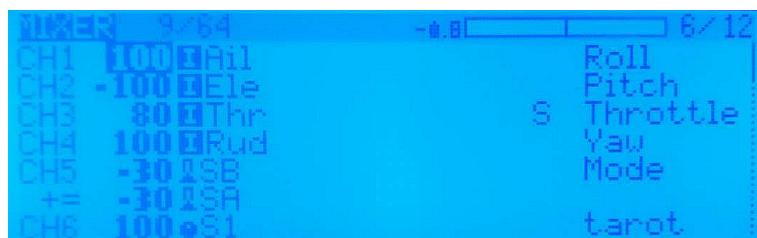
Aparatura radiowa FrSky Taranis X9D Plus pozwala przesyłać informacje zgrupowane w 16 kanałów. Do manualnej kontroli drona wymagane są jedynie 4 z nich, przyporządkowane dwóm głównym drążkom aparatury. Najważniejszym sygnałem jest Throttle, który odpowiada za ogólny poziom mocy wszystkich silników drona. Kolejnymi sygnałami są Roll, Pitch oraz Yaw, które są przyporządkowane ośmiom obrotu platformy B.2.



Rysunek B.2. Osi obrotu drona [12]

Kanał piąty jest używany przez autopilota do zmiany trybu lotu. Funkcja ta obsługuje wybór spośród maksymalnie 6 trybów, którym przyporządkowano zakresy szerokości pulsu PWM przedstawione w oknie konfiguracyjnym B.1. Elementem aparatury radiowej przypisanym do kanału musiał zostać zestaw kilku przełączników – zastosowanie potencjometrów nie byłoby ergonomicznym rozwiązaniem. Wybrano dwa przełączniki 3-pozycyjne oznaczone jako SA oraz

SB, które mikser aparatury dodaje do siebie – przedstawia to rysunek B.3. Zastosowane wagi: -30 dopasowano, by położenie dolne obu przełączników odpowiadało trybowi 1, a zmiana stanu któregokolwiek z nich skutkowała przejściem do innego zakresu szerokości pulsu PWM. Ostatecznie zestaw obu przełączników daje możliwość wyboru spośród 5 trybów lotu.



Rysunek B.3. Mikser sygnałów w aparaturze radiowej

Kanał szósty jest przypisany do potencjometru S1 i został skonfigurowany jako sygnał ciągły sterujący wartością zadaną nachylenia kamery. W trakcie lotu mechanizmy gimbala utrzymują tę wartość pomimo możliwych wychyleń platformy. Orientacja kamery z minimalną wartością nachylenia pozwala rejestrować obraz bezpośrednio przed dronem, z kolei maksymalna wartość nachylenia kieruje obiektyw ku ziemi. Ważne: ze względu na podłączony do kamery przewód HDMI regulacja nachylenia nie działa poprawnie – w projekcie zasilanie silnika odpowiedzialnego za nachylenie kamery nie jest podłączone.

Kanał siódmy przypisano do 3-pozycyjnego przełącznika SC. Sygnał PWM z informacją z tego kanału jest wysyłany przez autopilot do układu rekonfigurowalnego i służy do zdalnego rozpoczęcia misji śledzenia.

### B.3. Instrukcja powtórzenia eksperymentu śledzenia

- Upewnić się, że wszystkie przewody zostały poprawnie podłączone do płyty PYNQ (zasilanie 5V, sygnały z autopilota), a sama płyta jest dobrze unieruchomiona na platformie
- Podłączyć pakiet LiPo
- Włączyć zasilanie płyty PYNQ
  - jeśli układ Zynq nie jest konfigurowany z karty SD, należy skonfigurować go poprzez port USB-JTAG
- Włączyć kamerę i upewnić się, że jest w trybie rejestracji wideo
- Podłączyć autopilot do komputera (kabel USB) i w aplikacji MISSION Planner sprawdzić podstawowe informacje – obecność sygnału GPS i innych czujników, zwracane błędy. Zamknąć połączenie i odłączyć kabel

- Upewnić się, że nie ma przewodów lub elementów mogących stanowić zagrożenie podczas pracy silników
- Włączyć aparaturę radiową i za pomocą przełączników SA oraz SB wybrać tryb GUIDED
- Ustawić przełącznik SC w pozycji górnej – wysłać sygnał rozpoczynający misję
- Oczekiwając na osiągnięcie stałej wysokości przez drona przy zachowaniu pozycji stojącej i odległości kilku metrów od platformy
- Po ruchu drona wskazującym na rozpoczęcie śledzenia, można zmienić swoje położenie i obserwować działanie systemu
- Wyłączyć misję ustawieniem przełącznika SC w pozycji środkowej lub dolnej – rozpoczęcie się lądowanie



## Bibliografia

- [1] *THE DRONES REPORT*. Odwiedzono: 2017-08-15. URL: <http://www.businessinsider.com/uav-or-commercial-drone-market-forecast-2015-2?IR=T>.
- [2] Paul L. Rosin i Tim Ellis. *Image difference threshold strategies and shadow detection*. 1995, s. 347–356.
- [3] Aroh Barjatya. *Block Matching Algorithms For Motion Estimation*. 2004.
- [4] Afef Salhi. „Object tracking system using Camshift, Meanshift and Kalman filter”. W: 6 (kw. 2012), p674.
- [5] K. Fukunaga i L. Hostetler. „The estimation of the gradient of a density function, with applications in pattern recognition”. W: *IEEE Transactions on Information Theory* 21.1 (1975), s. 32–40. ISSN: 0018-9448.
- [6] Egorov Svetlana. *Vision Topics Seminar: Mean Shift*. Odwiedzono: 2017-08-19. URL: <http://slideplayer.com/slide/4973214/>.
- [7] Hitesh Patel i Ashish Singhadia. „Object Tracking System Using Mean Shift Algorithm and Implementation on FPGA”. W: *International Journal of Engineering Trends and Technology (IJETT)* (2014), s. 293–296.
- [8] Mateusz Komorkiewicz i Tomasz Kryjak. *Projektowanie struktury układów FPGA*. Skrypt do ćwiczeń laboratoryjnych. AGH, 2014.
- [9] Dorin Comaniciu i Visvanathan Ramesh. *Real-Time Tracking of Non-Rigid Objects using Mean Shift*. 2000.
- [10] Navneet Dalal i Bill Triggs. *Histograms of oriented gradients for human detection*. 2005.
- [11] Steve R. Gunn. *Support Vector Machines for Classification and Regression*. University of Southampton, 1998, s. 5–16.

- [12] Mustapha Bouhali i in. „FPGA Applications in Unmanned Aerial Vehicles - A Review”. W: *Applied Reconfigurable Computing: 13th International Symposium, ARC 2017, Delft, The Netherlands, April 3-7, 2017, Proceedings*. Wyd. Stephan Wong i in. Cham: Springer International Publishing, 2017, s. 217–228. ISBN: 978-3-319-56258-2. URL: [https://doi.org/10.1007/978-3-319-56258-2\\_19](https://doi.org/10.1007/978-3-319-56258-2_19).
- [13] Krzysztof Mazur. *Wbudowany system wizyjny do śledzenia obiektów dla potrzeb nawigacji robota autonomicznego*. AGH, 2016.
- [14] Michał Drożdż. *Implementacja sprzętowa modułu detekcji twarzy algorytmem HOG+SVM na potrzeby wbudowanego systemu wizyjnego do monitorowania stanu kierowcy*. AGH, 2016.
- [15] *CORDIC v6.0 - LogiCORE IP Product Guide*. Odwiedzono: 2017-08-19. URL: [https://www.xilinx.com/support/documentation/ip\\_documentation/cordic/v6\\_0/pg105-cordic.pdf](https://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf).
- [16] *FIFO Generator v13.1 - LogiCORE IP Product Guide*. Odwiedzono: 2017-08-19. URL: [https://www.xilinx.com/support/documentation/ip\\_documentation/fifo\\_generator/v13\\_1/pg057-fifo-generator.pdf](https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v13_1/pg057-fifo-generator.pdf).
- [17] *DJI SPARK*. Odwiedzono: 2017-08-15. URL: <http://www.dji.com/spark>.
- [18] R. Konomura i K. Hori. „Phenox: Zynq 7000 based quadcopter robot”. W: *2014 International Conference on ReConfigurable Computing and FPGAs (ReConFig14)*. 2014, s. 1–6.
- [19] Bazle Eizad, Ashray Doshi i Adam Postula. „FPGA Based Stability System for a Small-scale Quadrotor Unmanned Aerial Vehicle”. W: *Proceedings of the 8th FPGAWorld Conference*. FPGAWorld ’11. Copenhagen i dal.: ACM, 2011, 3:1–3:6. ISBN: 978-1-4503-1021-5. URL: <http://doi.acm.org/10.1145/2157871.2157874>.
- [20] Ashray A. Doshi i in. „Development of micro-UAV with integrated motion planning for open-cut mining surveillance”. W: *Microprocessors and Microsystems* 39.8 (2015), s. 829 –835. ISSN: 0141-9331. URL: <http://www.sciencedirect.com/science/article/pii/S0141933115001076>.
- [21] Henning Schlender i in. „Teaching Mixed-Criticality: Multi-Rotor Flight Control and Payload Processing on a Single Chip”. W: *Proceedings of the WESE’15: Workshop on Embedded and Cyber-Physical Systems Education*. WESE’15. Amsterdam, Netherlands: ACM, 2015, 9:1–9:8. ISBN: 978-1-4503-3897-4.
- [22] J. Soh i X. Wu. „An FPGA-Based Unscented Kalman Filter for System-On-Chip Applications”. W: *IEEE Transactions on Circuits and Systems II: Express Briefs* 64.4 (2017), s. 447–451. ISSN: 1549-7747.

- [23] Xiaolei Wang i in. „A prototype of MEMS gyroscope based on digital control”. W: *International Conference on Automatic Control and Artificial Intelligence (ACAI 2012)*. 2012, s. 275–278.
- [24] Benjamin Tefay i in. „Design of an Integrated Electronic Speed Controller for Compact Robotic Vehicles”. W: *Proceedings of Australasian Conference on Robotics and Automation*.
- [25] *HiSystems GmbH. MikroKopter-Boards*. Odwiedzono: 2017-08-19. URL: <http://wiki.mikrokopter.de/en/MK-Board>.
- [26] *Aerotenna. OcPoC*. Odwiedzono: 2017-08-19. URL: <http://aerotenna.com/ocpoc/>.
- [27] G. van der Wal i in. „FPGA acceleration for feature based processing applications”. W: *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2015, s. 42–47.
- [28] D. Honegger, H. Oleynikova i M. Pollefeys. „Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU”. W: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, s. 4930–4935.
- [29] H. Oleynikova, D. Honegger i M. Pollefeys. „Reactive avoidance using embedded stereo vision for MAV flight”. W: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, s. 50–56.
- [30] Xiao Zhou, Xiaoliang Zheng i Kejun Ou. „Power line detect system based on stereo vision and FPGA”. W: *2017 2nd International Conference on Image, Vision and Computing (ICIVC)*. 2017, s. 715–719.
- [31] T. Giitsidis i in. „Human and Fire Detection from High Altitude UAV Images”. W: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 2015, s. 309–315.
- [32] H. Chenini i in. „Embedded real-time localization of UAV based on an hybrid device”. W: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, s. 1543–1547.
- [33] *Model opisu przestrzeni barw HSV*. Odwiedzono: 2017-08-19. URL: [https://pl.wikipedia.org/wikiHSV\\_\(grafika\)](https://pl.wikipedia.org/wikiHSV_(grafika)).
- [34] *Multi-OS Support (AMP & Hypervisor)*. Odwiedzono: 2017-08-19. URL: <http://www.xilinx.com/Multi-OS+Support+%28AMP+%26+Hypervisor%29>.
- [35] *Telemetry / Serial Port Setup*. Odwiedzono: 2017-08-15. URL: <http://ardupilot.org/copter/docs/common-telemetry-port-setup-for-apm-px4-and-pixhawk.html>.

- [36] *Flight Modes*. Odwiedzono: 2017-08-15. URL: <http://ardupilot.org/copter/docs/flight-modes.html>.
- [37] *MAVLink Mission Command Messages (MAV\_CMD)*. Odwiedzono: 2017-08-15. URL: [http://ardupilot.org/copter/docs/common-mavlink-mission-command-messages-mav\\_cmd.html](http://ardupilot.org/copter/docs/common-mavlink-mission-command-messages-mav_cmd.html).
- [38] *MAVLINK ArduPilotMega Message Set*. Odwiedzono: 2017-08-15. URL: <http://mavlink.org/messages/ardupilotmega>.
- [39] *PYNQ-Z1 Reference Manual*. Odwiedzono: 2017-08-15. URL: <https://reference.digilentinc.com/reference/programmable-logic/pynq-z1/reference-manual>.