

Instytut Elektrotechniki Teoretycznej i Systemów Informacyjno-Pomiarowych.

Projektowanie systemów wizyjnych- Projekt

**Temat: Odczyt godziny ze zdjęcia zegara analogowego przy pomocy narzędzi OpenCV.**

Kierunek: Elektrotechnika

Specjalność: Systemy wbudowane

Prowadzący: dr hab. inż. Marcin Kołodziej

**Wykonała: Monika Marciniuk**

21.01.2021r.

1

***1. Cel projektu***

Celem projektu było zbudowanie oprogramowania w języku Python w oparciu o bibliotekę OpenCV, które pozwoli na odczyt godziny ze zdjęcia zegara analogowego wykorzystując analizę obrazu. Program wymaga wyodrębnienia tarczy zegara na zdjęciu, rozpoznanie wskazówek i następnie odczyt godziny poprzez analizę kątów wskazówek zegara.

1. ***Założenia projektowe***

Założenia projektowe przedstawiają się następująco:

* + Projekt zakłada poddaniu testom bazę 30 zdjęć zegarów analogowych,
  + Wczytanie analizowanych zdjęć,
  + Wykrycie tarczy zegara na zdjęciu (okręgu) i jego środka,
  + Wyodrębnienie tarczy zegara,
  + Wykorzystanie metody transformacji Hougha w celu wyodrębnienia linii wskazówek zegara,
  + Odczyt kątów między liniami w celu określenia godziny na zegarze,
  + Wyświetlenie otrzymanej godziny.

1. ***Baza danych***

W projekcie zostały wykorzystane zdjęcia ze stron internetowych, które udostępniają je darmowo: [1,2] i przedstawiają rzeczywiste zegary analogowe. Baza danych zostanie stworzona na podstawie znalezionych pojedynczych zdjęć. Wybierając zdjęcia kierowano się tym, aby tarcza zegara była okrągła, zegar nie posiadał więcej niż 3 wskazówki (budziki posiadają dodatkową wskazówkę dla ustalenia alarmu), zdjęcie było zrobione na wprost zegara i aby zegar nie był w żaden sposób obrócony. Przykładowe zdjęcie:

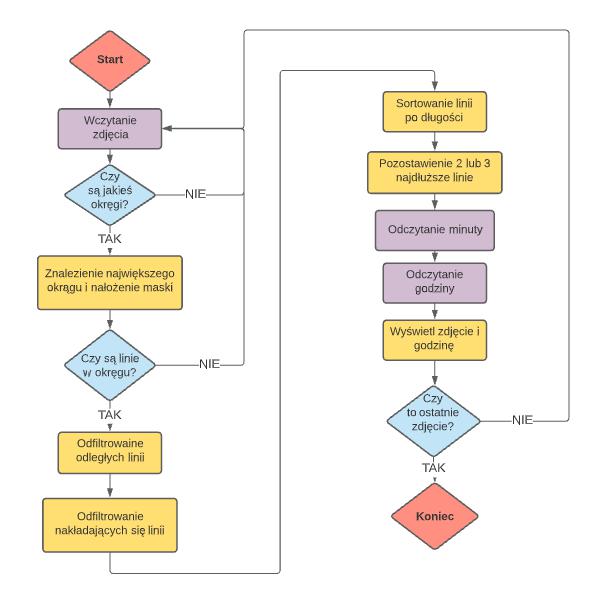


*Rys. 1 Przykładowe zdjęcia zegara analogowego pobranego do testów.*

2

***4. Algorytm program***

Dla prostego zobrazowania procesu, który musi się wykonać w programie narysowano następujący schemat:



*Rys. 2 Algorytm programu.*

3

1. ***Użyte narzędzia***

Program napisany został w języku Python 2.7 wykorzystując bibliotekę OpenCV na licencji BSD w środowisku programistycznym Pycharm 2019.3.4. OpenCV jest biblioteką funkcji, która pozwala na obróbkę obrazu. Jest ona wieloplatformowa i daje możliwość przetwarzania obrazu w czasie rzeczywistym. Ponadto korzystano z biblioteki Numpy do operacji na macierzach i obliczeń oraz Math do funkcji trygonometrycznych.

W programie wykorzystano takie funkcje z biblioteki OpenCV jak:

* + - imread() – wcztanie zdjęcia,
    - resize() – skalowanie obrazu,
    - medianBlur() – rozmycie obrazu,
    - cvtColor(img, cv2.COLOR\_GRAY2BGR) – odczyt obrazu w szarości,
    - threshold() – przełożenie obrazu w odcieniach szarości w reprezentacji jedynie kolorów bieli i czerni,
    - HoughCircles() - funkcja umożliwiająca wykrycie okręgów na zdjęciu metodą Hough,
    - circle() - wyrysowanie okręgu na zdjęciu,
    - bitwise\_or()- nałożenie maski
    - Canny() – filtr Canny wykrywający krawędzie,
    - HoughLinesP()- wykrycie linii probabilistyczną metodą Hough,
    - Line()- narysowanie linii,
    - putText() - wyświetlenie tekstu na zdjęciu,
    - imshow() – wyświetlenie zdjęcia,
    - waitKey(0)- odczytanie znaku z klawiatury,
    - destroyAllWindows()- zamknięcie otwartych okien.

1. ***Najważniejsze funkcje***
   * **Main()**

Funkcja main() jest główną funkcją programu, zawiera m. in. pętlę iteracyjnie wczytującą zdjęcia do programu, zdjęcie te jest początkowo skalowane i przygotowywane do wykrycia okręgu (zastosowano rozmycie, transformację na odcienie szarości i threshold), następnie wywoływane są kolejno funkcje stworzone w ramach projektu:

* ***linesP,circles,cdst = kola(cimg2, img)*** - funkcja kola() zwracająca wartości: linesP zawiera informacje o wykrytych liniach, circles informacje o wykrytych okręgach, a cdst obraz zegara,
* ***tab\_l, tab\_a, tab\_d = linie(linesP, circles, cdst)*** - funkcja linie() zwraca tablice zawierające kolejno: wektor linii, tablicę wykrytych kątów linii i długości tych linii,
* ***cdst,x,y,z = sort\_function(tab\_l, tab\_a, tab\_d, cdst)*** – funkcja wykonuje sortowanie linii tak, aby wyeliminować nakładające się linie oraz pozostawić jedynie 2 lub 3 najdłuższe.

Funkcja zwraca 4 parametry, kolejno: obraz i tablice: x- kat, y- wektor linii, z -długość linii.

* ***min = minuta(x[0], c1, str1,j1,j2,clock\_min)*** – funkcja minuta() zwraca minuty odczytane z zegara na podstawie wykrytych linii,
* ***h = minuta(x[1], c2, str2,j3,j4,clock\_h)*** – korzystając z tej samej funkcji minuta (), tym razem zwracając godzinę odczytaną z zegara na podstawie wykrytych linii.

4

Dokładnie funkcje te zostaną omówione w kolejnych punktach. Na koniec odczytana godzina jest wyświetlana na zdjęciu przy pomocy funkcji putText().

Kod funkcji main() prezentuje się następująco:

**def** main():

*#deklaracja zmiennych pomocniczych*

str1= **"minuta: "**

str2= **"godzina: "**

c1=6; c2=30 *# 1 minuta= 6 stopni, 1h= 30 stopni*

j1=15; j2=60; j3=2; j4=11 *# j1,j2 minuty do iteracji, j3,j4 godziny do iteracji*

clock\_min = 0; clock\_h =1;

* *deklaracja zmiennych przechowujacych nazwe i format zdjecia* name = [str(i) **for** i **in** range(1, 31 )]

format = **'.jpg'**

**for** i **in** name: *#glowna petla programu iterujaca zdjecia* **print**(i) *#numer zdjecia*

c = i + format *#nazwa pliku*

img = cv2.imread(c, 0) *#wczytanie zdjecia*

img = cv2.resize(img, (0, 0), fx=0.1, fy=0.1) *#zmiana wielksci obrazu* img\_blur = cv2.medianBlur(img, 7) *# rozmycie zdjecia*

cimg = cv2.cvtColor(img\_blur, cv2.COLOR\_GRAY2BGR) *# zdjecie w odcieniach*

* + *szarosci*

cimg2 = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR)

\_, thersh1 = cv2.threshold(cimg, 150, 255, cv2.THRESH\_BINARY\_INV) linesP,circles,cdst = kola(cimg2, img) *#wywolanie funkcji wykrywajacej* **if** linesP **is not** None:

*# kolo i wskazowki zegara*

tab\_l, tab\_a, tab\_d = linie(linesP, circles,cdst) *#wywolanie funkcji*

*# wyliczajacej katy wskazowek*

cdst,x,y,z = sort\_function(tab\_l,tab\_a,tab\_d,cdst) *#wywolanie funkcji*

*# sortujacej linie*

min = minuta(x[0], c1, str1,j1,j2,clock\_min) *# wywolanie funkcji*

*liczacej*

*# minuty*

1. = minuta(x[1], c2, str2,j3,j4,clock\_h) *# wywolanie funkcji liczacej*

*# godzine*

czas= str(h) +**":"** + str(min) *# string przechowujacy godzine ze zdjecia* **print** (czas) *# wyswietlenie w teminalu godziny*

font = cv2.FONT\_HERSHEY\_SIMPLEX *#font wyswietlanego tekstu na zdjeciu* org = (50, 50) *# wielkosc fontu*

fontScale = 1 *# skala fontu*

color = (255, 0, 0) *# kolor niebieski w BGR* thickness = 2 *# grubosc linii rowna 2 px*

* *Uzycie metody puttext w celu wyswietlenia tekstu na zdjeciu* image = cv2.putText(img, czas, org, font,

fontScale, color, thickness, cv2.LINE\_AA) cv2.imshow(**'detected circles'**, image) *#wyswietlenie zdjecia* cv2.waitKey(0) *# czekanie na przycisk* cv2.destroyAllWindows() *# wylaczenie zdjecia*

**else**:

**print**(**"Pominieto"**)

5

* ***kola(cimg2, img)***

Funkcja kola() została napisana w celu wyodrębnienia tarczy zegara na zdjęciu, zatem konieczne było wykrycie koła, czyli tarczy zegara. Założono, że tarcza zegara jest największym okręgiem na zdjęciu. Początkowo skorzystano z filtru **Canny**, który wyodrębnia krawędzie na zdjęciu. Następnie użyto funkcji **HoughCircles** do wykrycia okręgów. Jej parametry ustawiono tak, aby odległość od okręgów wynosiła 300 oraz promień wynosił minimalnie 30 i makrymalnie 200px. Pozostałe zostały ustawione na domyślne. Nastawy te wyznaczone zostały na podstawie testów, by na jak największej liczbie zdjęć wykryte były okręgi. Kolejno zmienna circles jest ograniczana jedynie do przechowywania parametrów największego okręgu, zakładając że to w nim znajduje się zegar. Następnie koło te zostaje wyrysowane na zdjęciu wraz z jego środkiem. By wyodrębnić samą tarczę zegara nałożona zostaje maska (zmienna mask) w postaci czarnego tła z wyciętym okręgiem w miejscu zegara. Nałożenie tych 2 obrazów daje postać zegara na czarnym tle (zmienna fg), co umożliwia dalszą analizę.

Dalszym krokiem było wykrycie wskazówek zegara. Użyto do tego funkcji **HoughLinesP(edges, 1, np.pi / 180, 30, None, 30, 10)**, która pozwala na wykrycie odcinków na obrazie korzystając z probabilistycznej transformacij Hougha. Parametry tej funkcji to kolejno zdjęcie, rozdzielczość odległości, rozdzielczość kątowa, threshold, minimalna długość linii i maksymalna dopuszczalna przerwa między punktami w tej samej linii, aby je połączyć. Wektor zawierający dane o liniach przechowuje zmienna linesP. Na koniec zdjęcie to w odcieniach szarości jest zwracane poprzez funkcję, a razem z nią zmienna circles i cdst (czyli wektor okręgu i zdjęcie). Jeśli okrąg nie jest wykryty, pojawia się odpowiedni komunikat.

Kod:

**def** kola(cimg2,img):

edges = cv2.Canny(cimg2, 100, 200) *# wykrycie krawedzi*

* *cv2.imshow('canny\_filter', edges)*
* *cv2.waitKey(0)*
* *cv2.destroyAllWindows()*
* *wykrycie okregow na zdjeciu*

circles = cv2.HoughCircles(edges, cv2.HOUGH\_GRADIENT, 1, 300,

param1=50, param2=30, minRadius=30, maxRadius=200)

**print**(circles)

**if** circles **is not** None:

circles = np.uint16(np.around(circles)) *# zmiana typu zmiennych tablicy*

*okregu*

diameter = [] *# deklaracja tablicy do przechowywania srednicy okregu* **for** k **in** circles[0, :]:

diameter.append(k[2]) *# wypelnienie tablicy srednica*

**if** max(diameter) == k[2]: *# wykrycie najwiekszego okregu*

circles = k *# ograniczenie sie do jednego okregu*

*# wyrysowanie okregu na zdjeciu*

cv2.circle(cimg2, (circles[0], circles[1]), circles[2], (0, 255, 0), 2)

*# zaznaczenie na zdjeciu srodka okregu*

cv2.circle(cimg2, (circles[0], circles[1]), 2, (0, 0, 255), 3)

mask = np.full((img.shape[0], img.shape[1]), 0, dtype=np.uint8) *# czarne*

*tlo*

*# wyciecie w* *tle okregu*

cv2.circle(mask, (circles[0], circles[1]), circles[2], (255, 255, 255),-1)

*# nalozenie 2 zdjec*

fg = cv2.bitwise\_or(img, img, mask=mask)

*# cv2.imshow('fg', fg)*

6

* *cv2.waitKey(0)*
* *cv2.destroyAllWindows()*

edges = cv2.Canny(fg, 100, 200) *# wykrycie krawedzi*

*# wykrycie linii w okregu*

linesP = cv2.HoughLinesP(edges, 1, np.pi / 180, 30, None, 30, 10) cdst = cv2.cvtColor(edges, cv2.COLOR\_GRAY2BGR) *# zdjecie w odcieniach # szarosci*

**if** linesP **is not** None:

**return** (linesP,circles,cdst) *# zwrocenie parametrow linii, okregu i*

*zdjecia*

**else**:

**print**(**"None lines in circle"**) *# komunikat o braku okregow na zdjeciu*

linesP = None

circles = None

cdst = None

**return** (linesP, circles, cdst)

**else**:

**print**(**"None objects in circles"**) *# komunikat o braku okregow na zdjeciu*

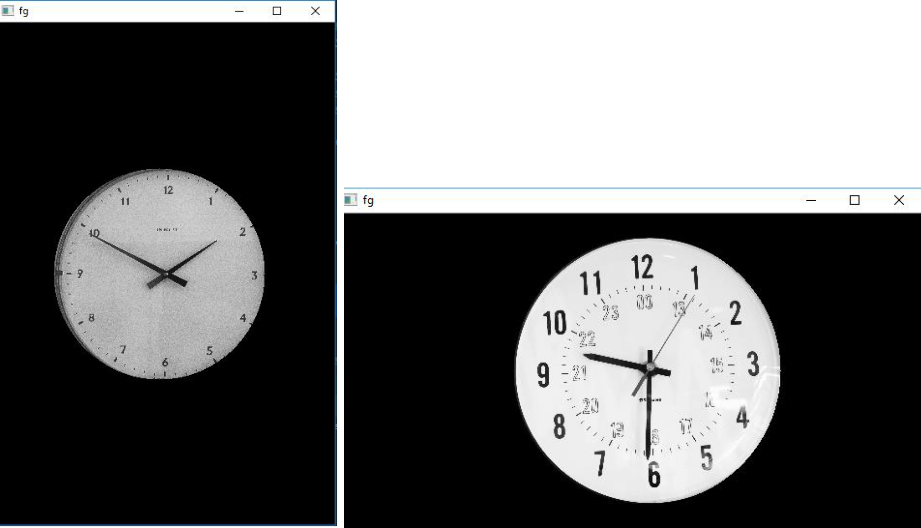
linesP = None

circles = None

cdst=None

**return**(linesP,circles,cdst)

Przykładowe wyniki wykrycia największego okręgu i nałożenia maski:



*Rys. 3 Przykładowy wynik działania funkcji koła().*

* ***linie(linesP, circles, cdst)***

Funkcja linie() ma za zadanie zwrócić trzy tablice, które przechowują parametry wskazówek zegara jak wektor linii, kąt nachylenia i jego długość. Aby wyznaczyć te parametry najpierw sprawdzany jest warunek, czy zostały wykryte linie. Następnie pętla iteruje po parametrach zmiennej linesP, by następnie przypisać parametry linii do zmiennej l, oraz wyznaczyć x, czyli odległość danej linii od środka okręgu oraz y, czyli to samo ale względem drugiego końca wskazówki. Potem zmienna d określa długość odcinka a jej parametr jest zapisywany w tablicy tab\_d.

Następnie wyznaczany jest kąt tych wskazówek, których odległość od środka okręgu jest większa niż 40 px. Do wyznaczenia kątów wykorzystana została funkcja atan2, która zwraca kąty w zakresie od -

7

do + . Dla uproszczenia dalszych obliczeń kąty są podawane w stopniach, a do ujemnych dodano 360˚ by otrzymać na tarczy zegara zakres kątów od 0 do 360 ˚. Na koniec kąt linii i jego parametry są dodawane kolejno do tablic tab\_a i tab\_l.

Kod:

**def** linie(linesP,circles,cdst):

tab\_d =[]

tab\_l =[]

tab\_a =[]

**if** linesP **is not** None:

**for** j **in** range(0, len(linesP)):

1. = linesP[j][0] *#parametry linii*
2. = np.sqrt((circles[0] - l[0]) \*\* 2 + (circles[1] - l[1]) \*\* 2) *# odlegosc jednego konca wskazowki od srodka okregu*
3. = np.sqrt((circles[0] - l[2]) \*\* 2 + (circles[1] - l[3]) \*\* 2) *# odlegosc drugiego konca wskazowki od srodka okregu*
4. = np.sqrt((l[0] - l[2]) \*\* 2 + (l[1] - l[3]) \*\* 2) *#dlugosc odcinka* **if** (x <= 40) **or** (y <= 40): *#pominiecie linii od srodka okregu o*

*odleglosci wiekszej niz podana*

tab\_d.append(d) *#dodanie parametru dlugosci wskazowki do tablicy* **if** x < y:

*# obliczenie kata nachylenia wskazowki zegara w zaleznosci od*

*tego*

* *ktora strona wskazowki jest blizej srodka okregu* angle = int(math.atan2(-(l[3] - circles[1]), l[2] -

circles[0]) \* 180 / math.pi)

**else**:

angle = int(math.atan2(-(l[1] - circles[1]), l[0] - circles[0]) \* 180 / math.pi)

**if** angle < 0: *# gdy kat ujemny dodaj 360 stopni* angle += 360

tab\_l.append(l) *# dodaj parametry wskazowki do tablicy* tab\_a.append(angle) *# dodaj kat do tablicy*

**else**:

**pass** *# jesli wskazowka oddalona od srodka okregu, pomin*

**else**:

**print**(**"no lines detected"**) *# gdy nie wykryto linii wypisz komunikat* **return** (tab\_l,tab\_a,tab\_d) *# zwroc utworzone tablice*

* ***sort\_function(tab\_l,tab\_a,tab\_d,cdst)***

Ze względu na to że wykrywane linie często nakładały się i było ich dużo zastosowano filtrację pozwalającą na pozbycie się linii o bardzo zbliżonym kącie oraz pozostawienie jedynie 2, z czego ta dłuższa określa minuty a krótsza godziny.

Danymi wejściowymi są tablice przechowujące informacje o wszystkich liniach i zdjęcie zegara. Aby sortując od najdłuższego odcinka nie utracić informacji o pozostałych parametrach dla danej wskazówki wykorzystano zmienne typu słownik (list, list2). Posortowane parametry dodano do nowo utworzonych tablic, następnie nadpisano nimi tablice: tab\_a,tab\_d,tab\_l.

Kolejnym krokiem było pozbycie się nakładających się odcinków. Początkowo brane są pierwsze parametry linii i dodawane do nowych tablic, następnie obliczana jest delta kąta kolejnej i poprzedniej linii, a gdy wyniesie ona mniej niż 20˚ lub więcej niż 350˚ parametry tej linii są dodawane do tablic x,y,z. Parametry tablicy tab uzupełniane 0 lub 1 pełnią rolę niesienia informacji czy dany kąt się powtarza czy nie (zależność od delty).

8

Dzięki temu, że linie były posortowane po długości w tablicach x,y,z znajdą się informacje o najdłuższych odcinkach i różnych kątach. Program pozwala dodać jedynie 3 parametry do tych tablic (wskazówkę godziny, minuty i sekundy).Zakłada się, że sekundnik jest najdłuższą linią spośród tych trzech dlatego, gdy występują 3 linie w tablicy, usuwana jest pierwsza pozycja z tablic.

Na koniec przy pomocy funkcji line() można nałożyć odcinki w kolorze czerwonym na zdjęcie. Funkcja zwraca parametry x,y,z oraz obraz z nałożonymi liniami.

Kod:

**def** sort\_function(tab\_l,tab\_a,tab\_d,cdst): *#fukcja filtrujaca linie po dlugosci i kacie*

x=[]

tab\_L=[] *#tablice do przechowywania kolejno,parametrow wskazowki, kata i dlugosci wskazowki*

tab\_A=[]

tab\_D=[]

y = []

z= []

list={};list2={} *#slowniki by moc sortowac po dlugosci linie i nie stracic jej pozostalych parametrow*

**for** i **in** range(len(tab\_a)):

list[round(tab\_d[i],2)]= tab\_a[i] *#uzaleznienie dl. wskazowki od jej kata* list2[round(tab\_d[i],2)]= tab\_l[i] *#uzaleznienie dl. wskazowki od jej*

*wektora parametrow*

**for** i **in** sorted(list, reverse = True):

tab\_D.append(i) *#posortowane dlugosci linii*

tab\_A.append(list[i]) *# dodanie do tablicy katow posortowanych linii* **for** i **in** sorted(list2, reverse = True):

tab\_L.append(list2[i]) *# analogicznie dodane do tablicy wektor parametrow posortowanych linii*

tab\_a = tab\_A

tab\_d = tab\_D

tab\_l = tab\_L

*#pominiecie linii o zblizonym kacie*

iter = 0

**for** i **in** tab\_a:

tab = []

**if** len(x) == 0: *# jesli tablica jest pusta uzupelnij pierwsza wartosciami [kat, wektor linii,dlugosc]*

x.append(tab\_a[0]); y.append(tab\_l[iter]),z.append(tab\_d[iter])

**else**:

**for** j **in** range(len(x)):

* *zbadaj roznice katow gdy mniejsze niz 20 stopni lub wieksze jak 350, dodaj do tab jedynke*

**if** (abs(i-x[j])< 20) **or** (abs(i-x[j])>350) :

tab.append(1)

* + *inaczej dodaj do tab 0* **else**:

tab.append(0)

* *gdy 1 reprezentujaca zblizony kat, pomin linie* **if** 1 **in** tab:

**pass**

* *gdy 0 i w tablicy x mniej niz jedna wskazowka dodaj parametry linii*

*do tablic*

**else**: *#*

**if** len(x) <= 2:

x.append(i); y.append(tab\_l[iter]) *#tab x- kat, tab y wektor*

9

*linii, tab z -dl. linii*

z.append(tab\_d[iter])

iter += 1

**for** i **in** range(len(x)):

**print**(x[i],y[i]) *# wyswietl parametry linii* l = y[i]

*#narysuj na zdjeciu linie, nastepnie wyswietl zdjecie*

cv2.line(cdst, (l[0], l[1]), (l[2], l[3]), (0, 0, 255), 3, cv2.LINE\_AA)

cv2.imshow(**'sorted** **'**, cdst)

cv2.waitKey(0)

cv2.destroyAllWindows()

**if** len(x) > 2: *# gdy wiecej niz 2 wskazowki, pozbadz sie parametrow tej najdluzszej (sek)*

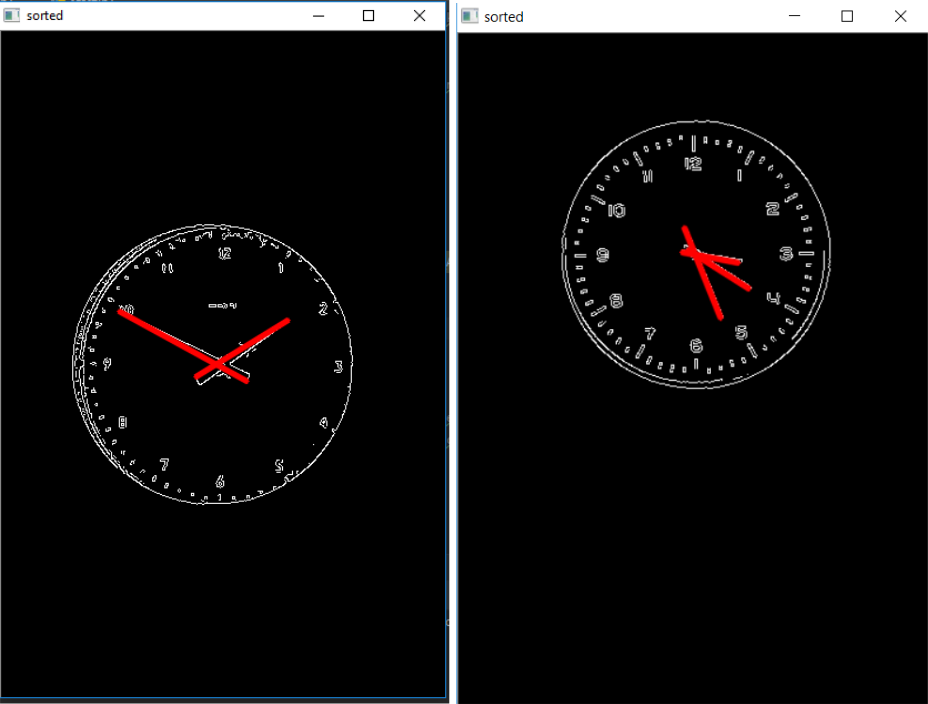
**del** x[0]

**del** y[0]

**del** z[0]

**return** cdst,x,y,z *#zwroc parametry wskazowek zegara i zdjecie zegara*

Wynik wyświetlenia zdjęcia (cdst), gdzie na zdjęciu po lewej stronie można zobaczyć odczyt w przypadku dwóch wskazówek, a po prawej w przypadku trzech:



*Rys. 4 Wynik funkcji sort\_function(), czyli wykryte i odfiltrowane linie na tarczy zegara.*

10

* ***minuta(angle, c, str,t1,t2,x)***

Aby otrzymać określić godzinę na zegarze została napisana funkcja minuta(). W zależności od zadanych parametrów może ona zwrócić godzinę lub minutę wskazywaną przez odpowiednią wskazówkę.

Głównym problemem określenia godziny na zegarze jest fakt, że kąt 0 stopni wskazuje godzinę 3 i minutę 15 i zwiększając kąt, wskazówka zegara porusza się w przeciwną stronę. Zatem aby prawidłowo wyznaczyć godziny i minuty kąt wskazówki jest badany na dwóch łukach od 90 do 0 stopni oraz od 90 do 360 stopni dekrementując kąt o 1 minutę (6 stopni) lub 1 godzinę (30 stopni). Dekrementując kąty zliczane są kolejne iteracje, które określają wskazywaną minutę/godzinę przez zegar.

Z uwagi na to, że kąt jest dekrementowany z pewnym krokiem, należałoby sprawdzać, czy kąt wyznaczony metodą Hough znajduje się w danym zakresie między poprzednim i kolejnym kątem iterowanym. Dodatkowo zostały dodane funkcje warunkowe zerujące minuty, gdy te osiągną 60 lub gdy godzina mniejsza równa zero przypisz 12. Funkcja minuta() zwraca zmienną minuta przechowującą parametr godziny lub minuty w zależności od zadanych parametrów : angle, c, str, t1,t2 i x.

Kod:

**def** minuta(angle, c, str,t1,t2,x): *#odczyt wskazowek zegara*

**if** angle >= 0 **and** angle <=90: *# gdy kat pomiedzy podanymi przypisz parametry do iteracji*

1. =0; i\_old=0; j=t1 *# i obecnie badany kat, i\_old poprzedni kat, j- czas (min lub h) przy kacie = 0*

**while** i < 90: *# gdy kat mniejszy od 90*

i+= c *# inkrementuj co krok c (6-minuty lub 30-godziny) stopni* **if** angle >= i\_old **and** angle < i: *# jesli kat mniejszy miesci sie*

*miedzy iterowanym przedzialem*

minuta = j *# czas rowny j*

**print**(str,minuta) *#wyswietl czas w terminalu*

**if** (x <= 0) & (minuta >= 60): *# jesli liczone sa minuty (x = 0) i minuty wieksze jak 60*

minuta = 0; *# minuty = 0*

**elif** (x>=0) & (minuta<=0): *# gdy wywolana funkcja godziny i zmienna minuty mniejsza od zera*

minuta = 12; *#godzina jest rowna 12*

**break**

j -= 1 *# dekrementacja czasu*

i\_old = i *#przypisanie ostatniego kata*

**else**:

* 1. =90; i\_old=90; j=t2; *# gdy kat wiekszy niz 90 zacznij iteracje kata od*

1. *= 90, pozostale analogicznie*

**while** i < 360: *# gdy kat mniejszy niz 360*

i+= c *# inkrementuj kat co krok c*

**if** angle >= i\_old **and** angle < i: *# analogicznie odczytaj czas* minuta = j

**if** minuta >= 60: *# gdy minuty = 60, przypisz 0* minuta = 0;

**print**(str,minuta)

**break**

j -= 1

i\_old = i

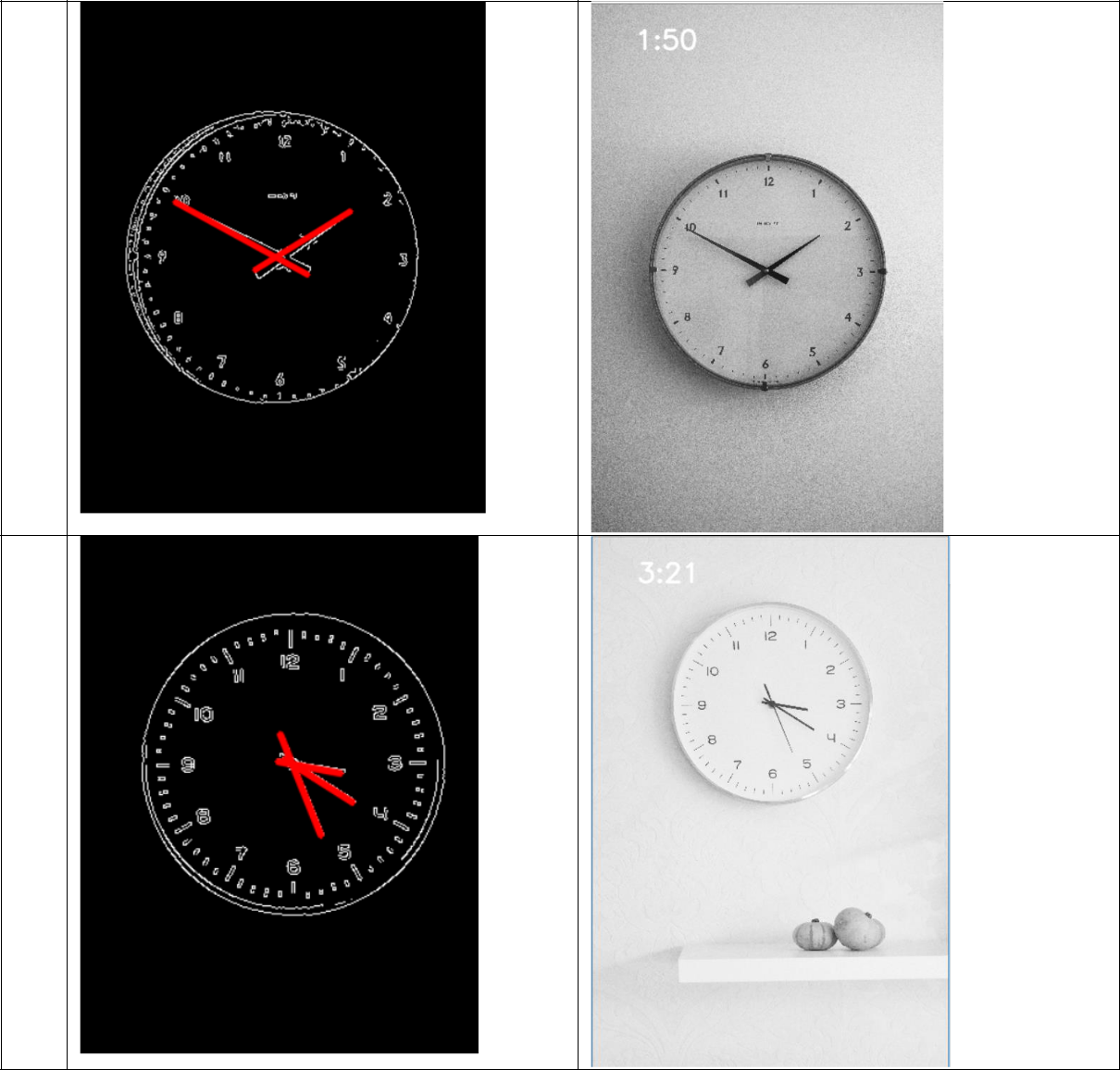
**return** minuta *# zwroc parametr przechowujacy czas zegara*

11

***7. Uzyskane wyniki***

Poniżej przedstawiono wyniki przeprowadzonego testu dla 30 zdjęć zegarów, gdzie po lewej stronie obrazek przedstawia rezultat wykrycia tarczy zegara i jego wskazówek, a po prawej odczytaną godzinę z danego zdjęcia.

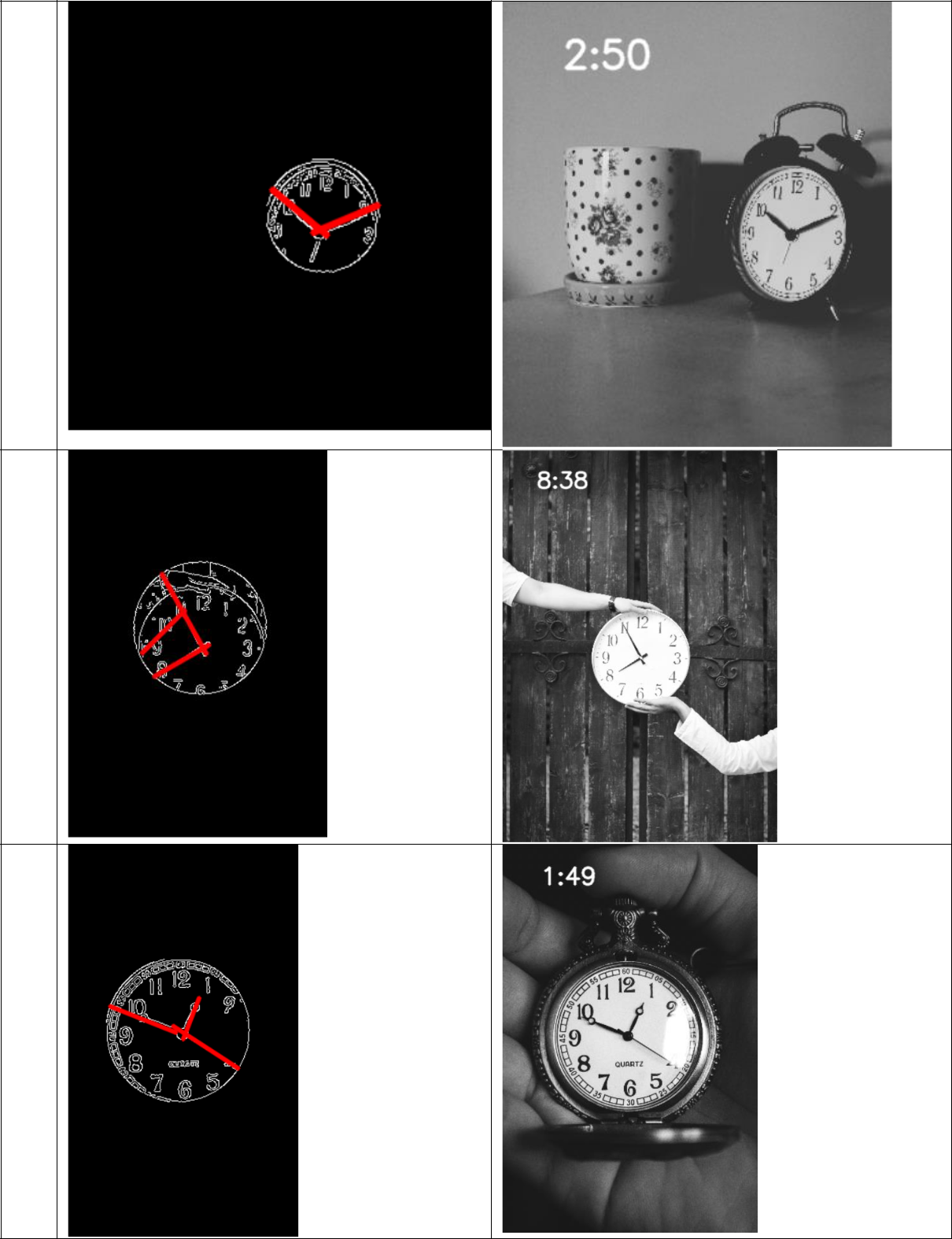
*Tabela 1 Wyniki testów dla 30 zdjęć zegarów.*



1.

2.

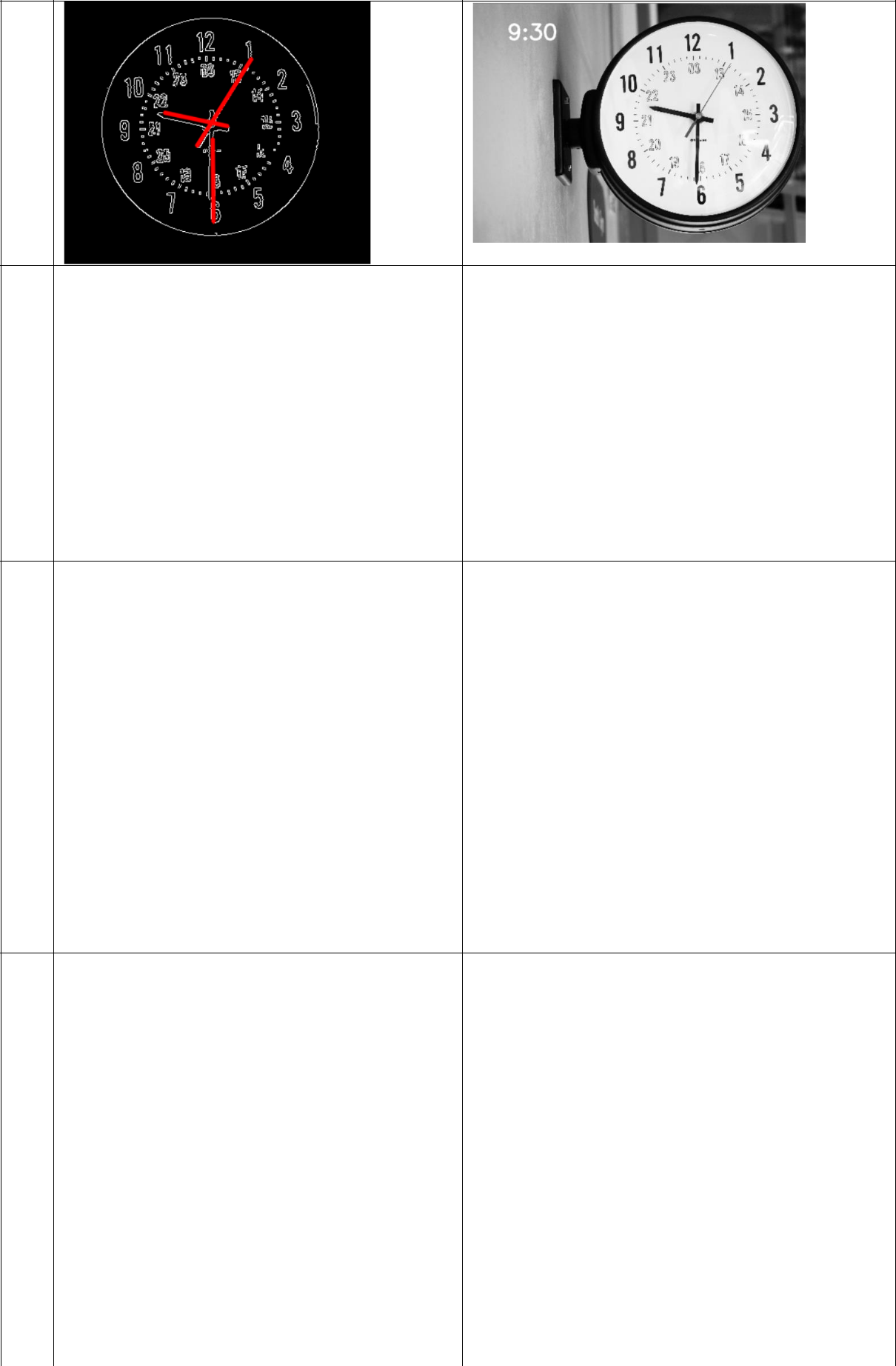
12

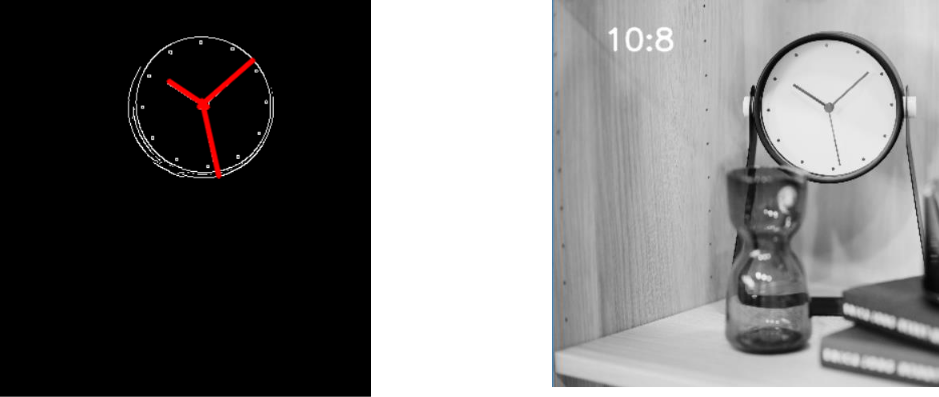
3.

4.

5.

13

6.



7.



8.



9.

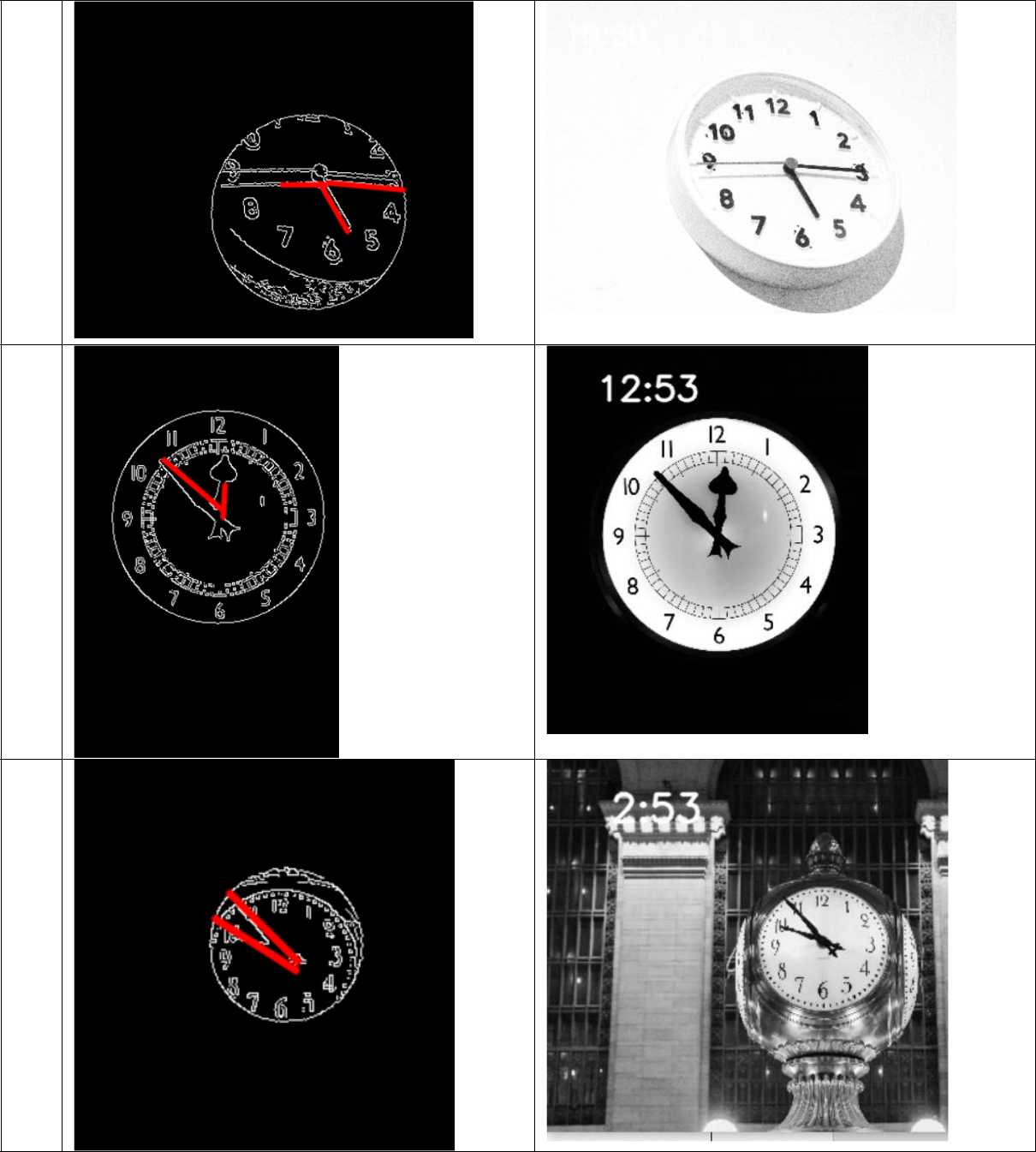
14

10.

11.

12.

10:20



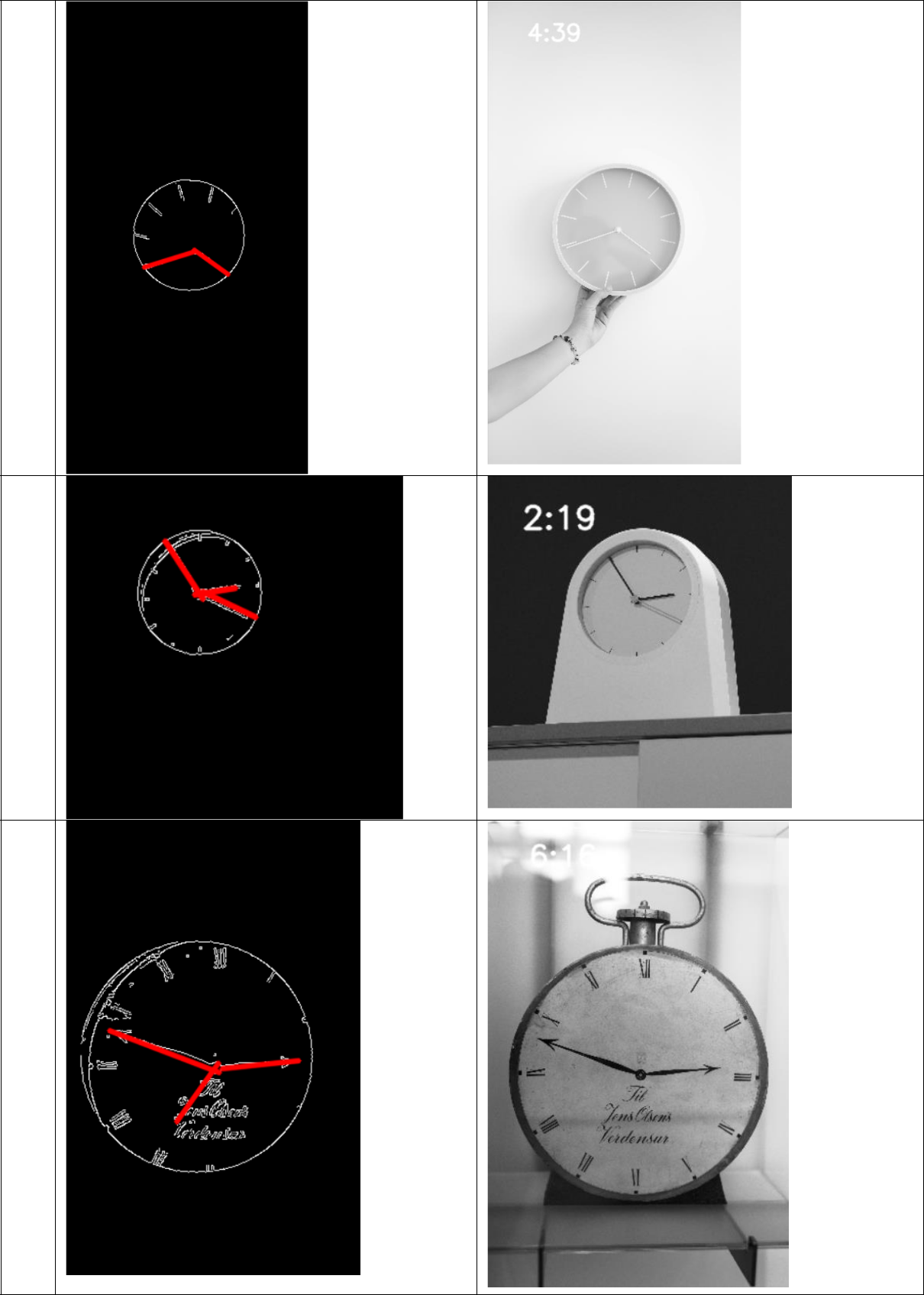
15

13.

14.

15.

4:39



6:16

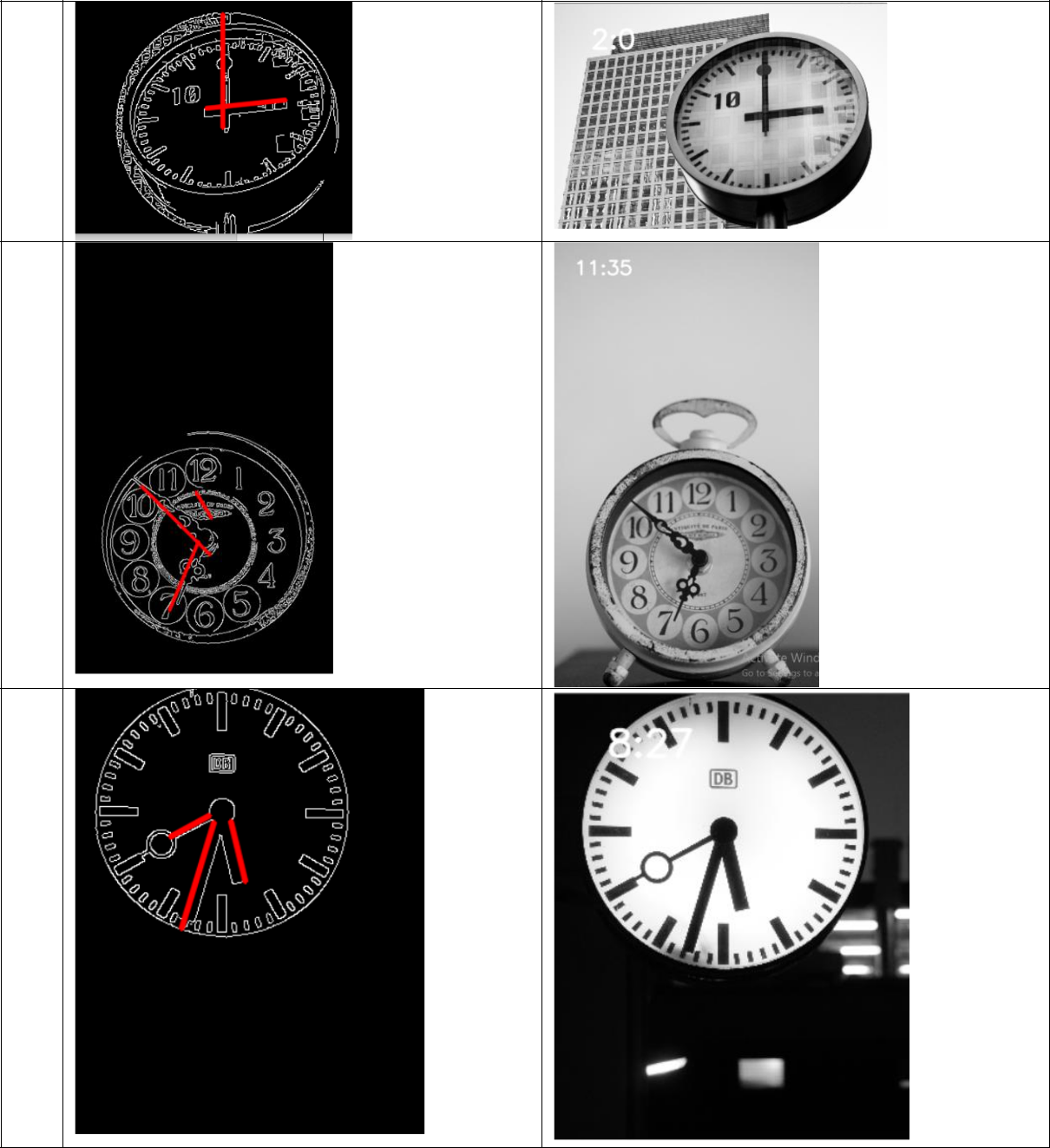
16

16.

17.

18.

2:00



8:27

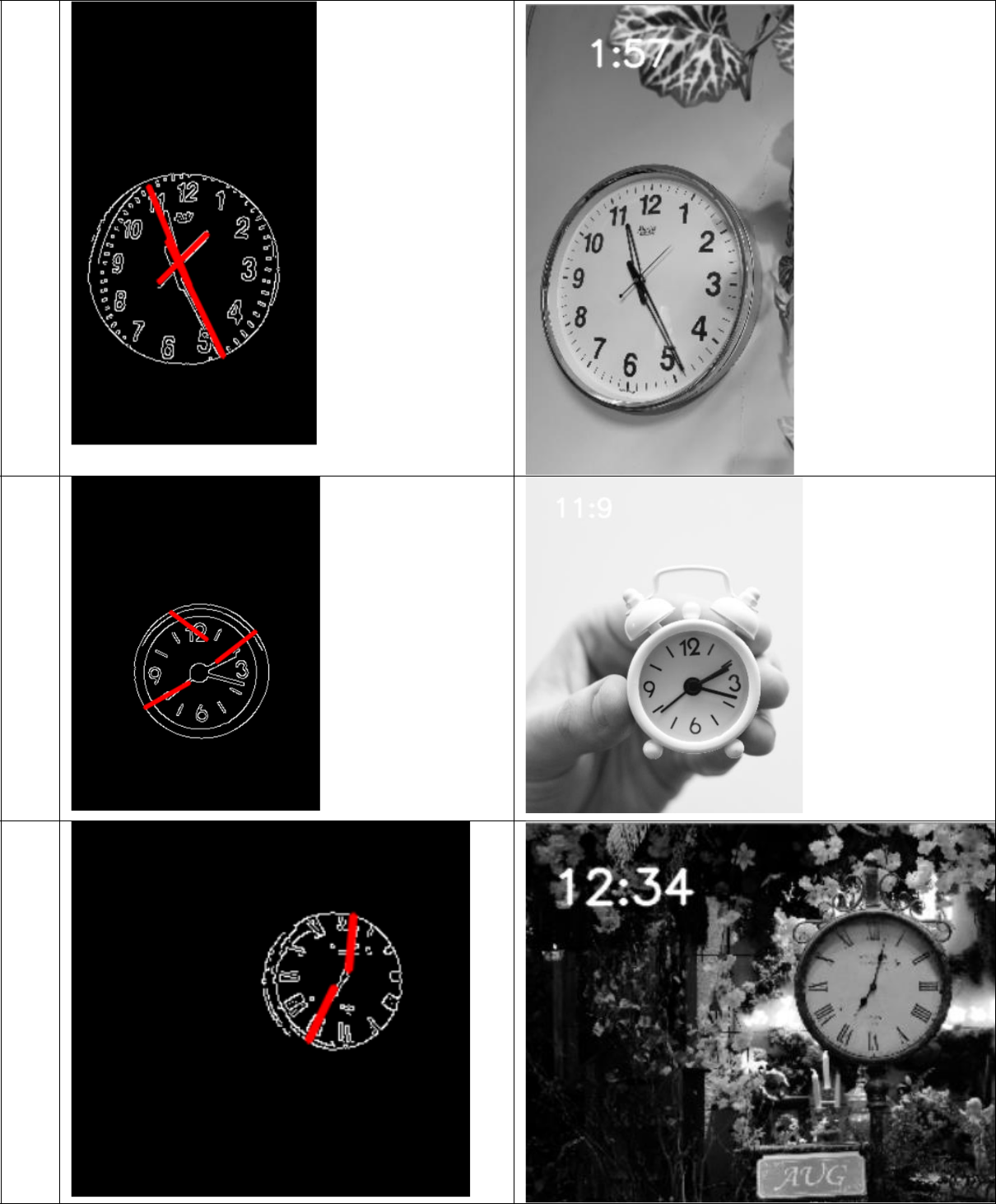
17

19.

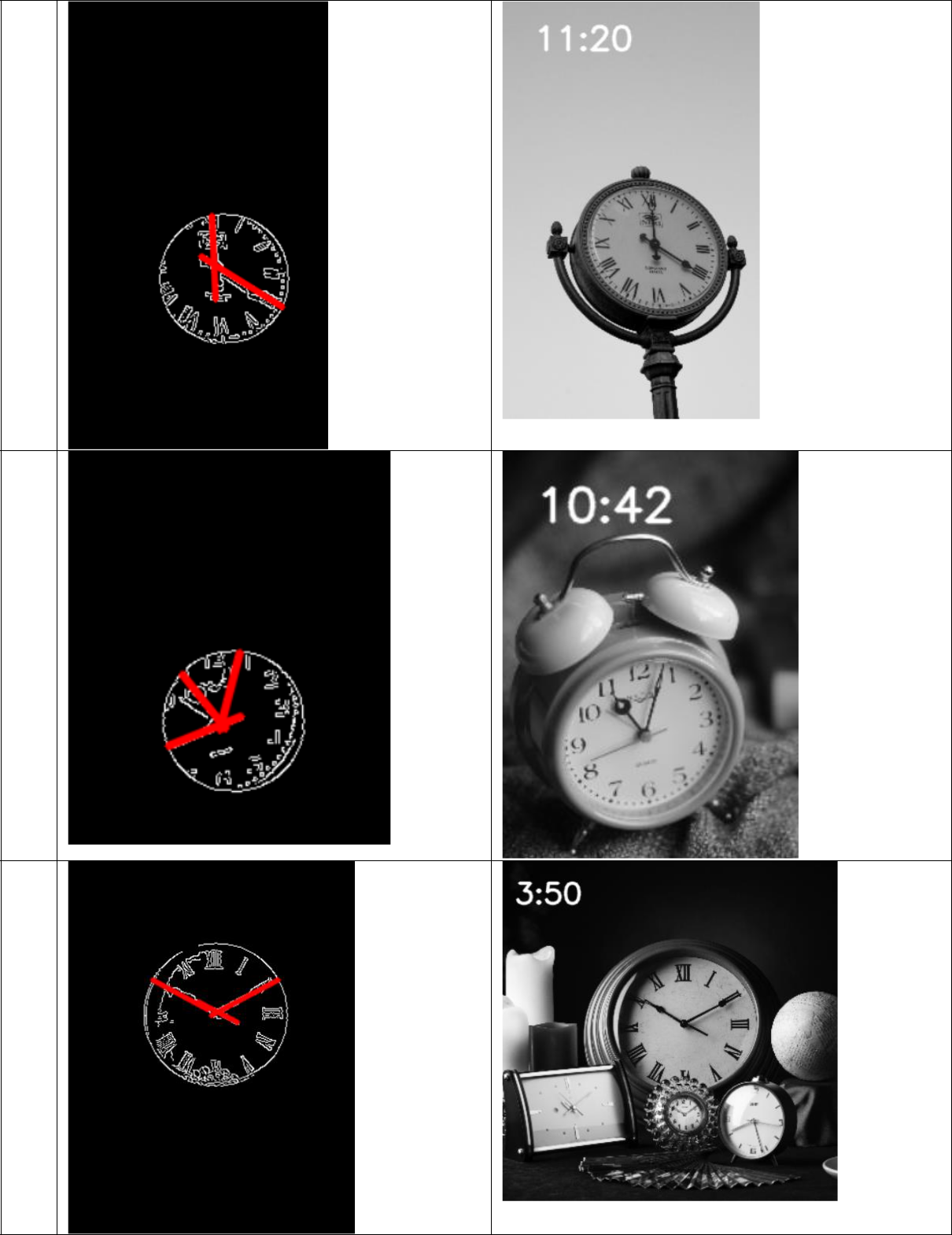
20.

21.

11:9



18

22.

23.

24.

19

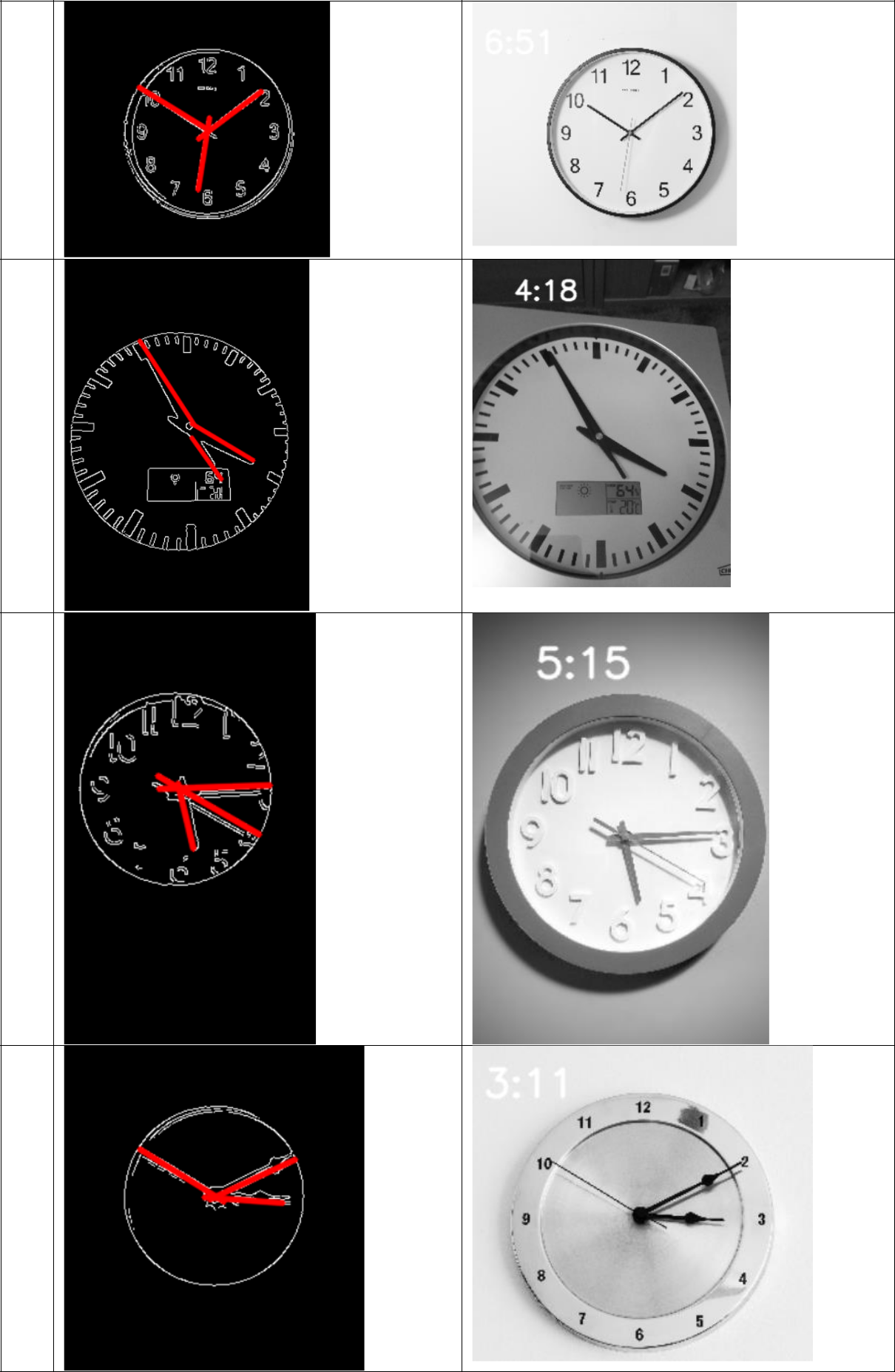
25.

26.

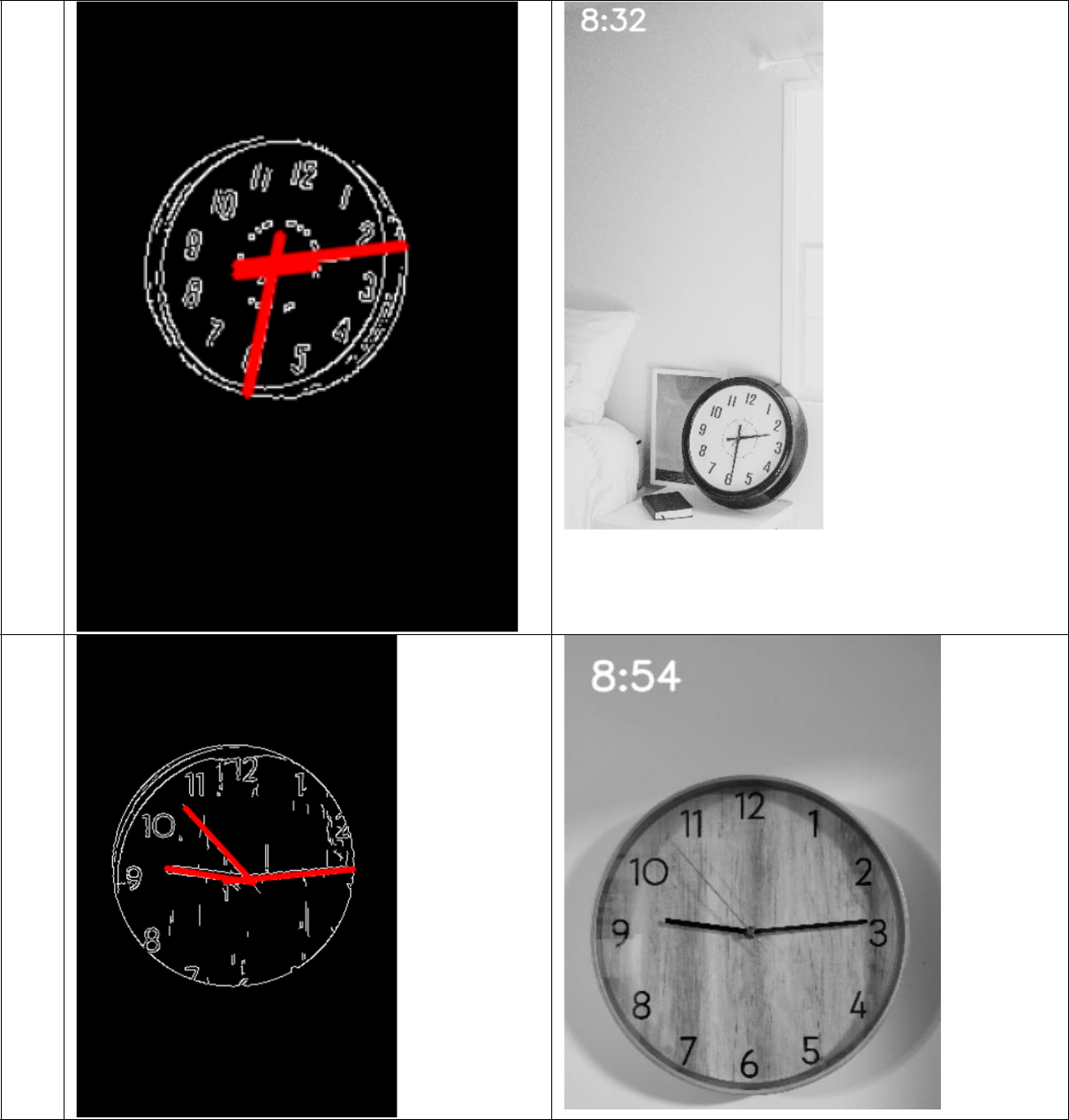
27.

28.

6:51



20

29.

30.

***8.*** ***Podsumowanie***

Spośród 30 zdjęć użytych do testu poprawną godzinę udało się uzyskać na 9 z nich. Odczyt godziny ze zdjęcia okazał się dużo trudniejszym zagadnieniem niż przypuszczano. Otrzymany wynik jest zadowalający, ponieważ istnieje wiele kwestii, które wpływają na taki odczyt.

Istnieją zegary o przeróżnym wyglądzie i kształcie, gdzie wskazówki także nie posiadają standardowego kształtu. Każde zdjęcie robione jest w innych warunkach o różnym oświetleniu. Dodatkowo amatorsko wykonywane zdjęcia najczęściej nie są robione trzymając aparat idealnie w poziomie.

Dużym problemem okazał się odczyt godziny w momentach, gdy minutnik wskazuje więcej jak 50 min np. 7:55. Kąt wskazówki godzinowej jest niemal równy już godziny 8 i algorytm wykrywa już następną godzinę. Likwidując ten problem dając niewielki zakres kąta, gdzie czas będzie odczytany

21

nadal jako poprzednia godzina, pojawia się kolejny, ponieważ minutnik może wskazywać 0, a godzina będzie odczytana jako poprzednia: przykład (test, zdjęcie nr.16).

Badanie bazy zdjęć zegarów dla jednakowych nastaw filtrów lub transformacji Hougha dla wszystkich zdjęć nie da wymiernych efektów, niż gdyby badać je osobno. Czasem pomimo wyraźnego zdjęcia program nie jest w stanie wykryć koła, a wykrycie linii transformacją Hougha wykrywa niekiedy więcej linii w pobliżu środka okręgu nie będącymi wskazówkami.

Podsumowując, biblioteka OpenCV jest przydatnym narzędziem do analizy obrazu zdjęć, jednak niekiedy wymaga analizy każdego zdjęcia z osobna lub konieczne są bardziej zaawansowane techniki filtracji szumów lub błędnych pomiarów.

Ze względu na to, że pobrane zdjęcia są wysokiej jakości, a system Isod jest ograniczony do plików 50MB, wszystkie zdjęcia użyte do testów można znaleźć pod linkiem :

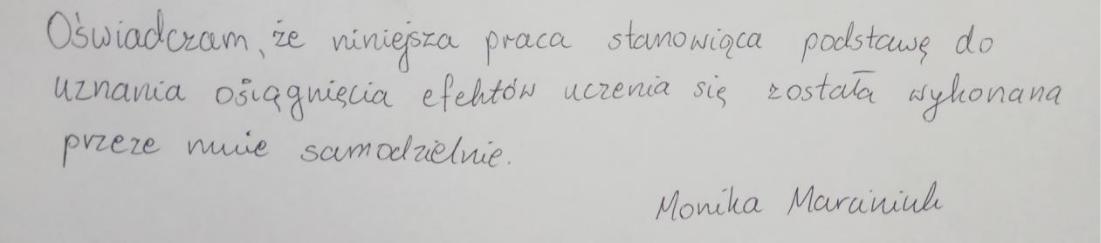
[https://wutwaw-](https://wutwaw-my.sharepoint.com/:f:/g/personal/01114689_pw_edu_pl/EppH5XJ45wpKlDyVRV8Meh4BSKolBvGqQ56hGCf6h1ImPg?e=z7o7hR)

[my.sharepoint.com/:f:/g/personal/01114689\_pw\_edu\_pl/EppH5XJ45wpKlDyVRV8Meh4BSKolBvGqQ](https://wutwaw-my.sharepoint.com/:f:/g/personal/01114689_pw_edu_pl/EppH5XJ45wpKlDyVRV8Meh4BSKolBvGqQ56hGCf6h1ImPg?e=z7o7hR) [56hGCf6h1ImPg?e=z7o7hR](https://wutwaw-my.sharepoint.com/:f:/g/personal/01114689_pw_edu_pl/EppH5XJ45wpKlDyVRV8Meh4BSKolBvGqQ56hGCf6h1ImPg?e=z7o7hR)

***9.*** ***Bibliografia***

1. [https://pixabay.com](https://pixabay.com/)
2. [https://unsplash.com](https://unsplash.com/)
3. [*https://docs.opencv.org/3.2.0/*](https://docs.opencv.org/3.2.0/)
4. [*https://wutwaw-*](https://wutwaw-my.sharepoint.com/:f:/g/personal/01114689_pw_edu_pl/EppH5XJ45wpKlDyVRV8Meh4BSKolBvGqQ56hGCf6h1ImPg?e=z7o7hR)

[*my.sharepoint.com/:f:/g/personal/01114689\_pw\_edu\_pl/EppH5XJ45wpKlDyVRV8Meh4BSKol*](https://wutwaw-my.sharepoint.com/:f:/g/personal/01114689_pw_edu_pl/EppH5XJ45wpKlDyVRV8Meh4BSKolBvGqQ56hGCf6h1ImPg?e=z7o7hR)[*BvGqQ56hGCf6h1ImPg?e=z7o7hR*](https://wutwaw-my.sharepoint.com/:f:/g/personal/01114689_pw_edu_pl/EppH5XJ45wpKlDyVRV8Meh4BSKolBvGqQ56hGCf6h1ImPg?e=z7o7hR)

**

22