Strumieniowanie wideo z kamery USB RaspberryPi oraz kamery natywnej laptopa

Rafał Ileczko

Semestr zimowy 2018/19

Streszczenie

Tematem projektu było stworzenie oprogramowania pozwalającego w łatwy sposób pobierać obraz z kamer z zarówno kamery zamontowanej w laptopie, jak i kamery USB połączonej z RaspberryPi. Obraz miał być przesyłany w czasie rzeczywistym z możliwością jego modyfikacji przez bibliotekę OpenCV zarówno przed wysłaniem obrazu, jak i po jego odbiorze. Transmisja odbywać się miała za pomocą Socketów TCP po sieci lokalnej WiFi z uwzględnieniem możliwości korzystania z sieci WAN.

Projekt został wykonany na potrzeby przedmiotu *Systemy wbudowane i rekonfigurowalne*. Jego efekt zostanie później wykorzystany przy tworzeniu pracy dyplomowej magisterskiej przez autora.

Spis treści

1 O projekcie

Celem projektu było stworzenie oprogramowania wraz z infrastrukturą pozwalającą na obróbkę obrazu pochodzącego ze źródła nie połączonego fizycznie z komputerem. Jest to sposób na ułatwienie testowania aplikacji AR. Konieczność wykorzystywania kamery natywnej laptopa, bądź podłączonej do niego bezpośrednio przewodem powoduje szereg problemów, głównie związanych z koniecznością obecności w pobliżu kamery. Dzięki wykorzystaniu RaspberryPi z podłączoną doń kamerą, oraz komunikacją WiFi ograniczenia te znikają i pozwalają na przyspieszenie prac. Stworzono zabezpieczenie w postaci *fallbacku* do kamery połączonej bezpośrednio z komputerem, przez co nie są tworzone inne niedogodności. Do przyspieszenia prac stworzono również skrypty pomocnicze.

1.1 Wykorzystany sprzęt i technologie

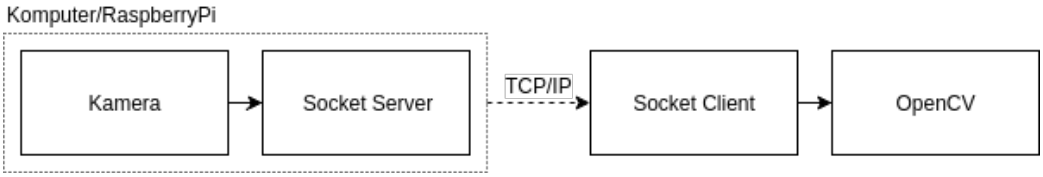
Do wykonania założonych zadań wykorzystano:

- Hardware
  - RaspberryPi 3B+
  - kamera USB
- Software
  - język Python3
  - język Bash
  - technologię TCP Socket
  - bibliotekę Zero MQ
  - bibliotekę OpenCV
  - bibliotekę NumPy

1.2 Architektura

Projekt można podzielić na 2 sekcje: serwerową oraz kliencką. Zadaniem pierwszej jest stworzenie serwera Socket, a po połączeniu się klienta - przesyłanie sygnału wideo z kamery, oraz ewentualnie wstępna obróbka obrazu przed wysłaniem. Każdy kadr przed wysłaniem jest konwertowany do kodowania base64, a po odebraniu dekodowany z pomocą biblioteki NumPy.

Klient ma połączyć się z serwerem, odebrać strumień wideo, obróbka obrazu z pomocą OpenCV i wyświetlenie go użytkownikowi.



Rysunek 1: Architektura

Do komunikacji wykorzystano bibliotekę ZeroMQ, która poprzez sockety przesyła obraz.

Skrypty pomocnicze zostały napisane w języku Bash, natomiast komunikacyjne i związane z obróbką obrazu w języku Python.

2 Realizacja

2.1 Drzewo projektu

```
.
|-- connect_rpi.sh
|-- dist.txt
|-- docs
|   |-- architecture.png
|   |-- missfont.log
|   |-- sprawozdanie.md
|   `-- sprawozdanie.pdf
|-- find_rpi.sh
|-- mtx.txt
|-- process_stream.py
|-- read_stream.py
`-- start_stream.py
```

2.2 Opis plików

Opis pomniejszych plików i katalogów:

- `docs` - folder zawierający pliki związane z dokumentacją
- `mtx.txt` - plik tekstowy z macierzą kamery USB użytej w projekcie
- `dist.txt` - plik tesktowy z macierzą zniekształceń kamery USB użytej w projekcie

2.2.1 find_rpi.sh

```
#!/bin/bash

arp -a | awk '/b8:27:eb/{ print substr($2, 2, length($2)-2); exit; }'
```

Listing 1: Poszukiwanie RaspberryPi w podsieci

Skrypt zwraca pierwsze znalezione IP urządzenia RaspberryPi. Na początku pobierane są wszystkie urządzenia w podsieci za pomocą komendy `arp-a`. Następnie każda zwrotka (jedna zwrotka = 1 urządzenie w podsieci) jest przeszukiwana na obecność ciągu znaków `b8:27:eb`, który to znajduje się w adresie MAC każdego RPI. Następnie z lini zawierającej wspomniany ciąg znaków "wyciągany" jest adres IP, a następnie wysyłany do strumienia wyjścia.

2.2.2 connect_rpi.sh

```
#!/bin/bash
# ssh_rpi:
#   Connect to first found RaspberryPi device


PROGNAME=$(basename $0)
error_exit() {
    echo -e "\e[1m\e[31mError␣(${PROGNAME}):␣${1:-"Unknown Error"}" 1>&2
    exit 1
}

echo -e "\e[33mSearching␣for␣RaspberryPi␣Devices...\e[0m"
ip=$(./find_rpi.sh)

if [ ! -z "$ip" ]
    then
    if ping -c1 -W1 -q $ip &>/dev/null
        then
        echo -e "\e[32mFound␣$ip\e[0m"
        echo -e "\e[32mConnecting␣to␣RPI\n\e[0m"
        ssh pi@$ip
        else
            error_exit "RaspberryPi␣cannot␣be␣connected"
    fi
```

```
    else
        error_exit "RaspberryPi cannot be found on local network"
fi
```

Listing 2: Poszukiwanie i połączenie z RaspberryPi przez ssh

Skrypt wykorzystuje komendy z poprzedniego podrozdziału do znalezienia urządzenia. Ponieważ jednak jest możliwość, że znalezione IP jest cachowane, tj. może zostać znalezione urządzenie, które było dostępne kilka minut wcześniej, ale już nie jest. Z tego powodu przeprowadzany jest test pingowania urządzenia, a po jego zdaniu popełniana jest próba połączenia poprzez protokół SSH jako użytkownik `pi`.

2.2.3 `start_stream.py`

```
#!/usr/bin/env python3

import base64
import cv2
import socket
import zmq

cap = cv2.VideoCapture(0)  # init the camera
context = zmq.Context()
footage_socket = context.socket(zmq.PUB)
footage_socket.bind('tcp://0.0.0.0:5555')

print("Starting stream")
while True:
    try:
        grabbed, frame = cap.read()  # grab the current frame
        frame = cv2.resize(frame, (320, 240))  # resize the frame
        encoded, buffer = cv2.imencode('.jpg', frame)
        jpg_as_text = base64.b64encode(buffer)
        footage_socket.send(jpg_as_text)

    except KeyboardInterrupt:
        print("Exception raised! Exiting...")
        cap.release()
        cv2.destroyAllWindows()
```

```
26          break
```

Listing 3: Poszukiwanie i połączenie z RaspberryPi przez ssh

Stream uruchamiany jest na porcie `5555` dla wszystkich interfejsów sieciowych urządzenia. Tworzony jest obiekt `VideoCapture` biblioteki OpenCV, który następnie, w nieskończonej pętli, pobiera kadr z kamery, enkoduje do standardu base64 i wysyła poprzez API biblioteki ZeroMQ. W razie błędów obiekt kamery jest zwalniany, przez możliwe jest ponowne uruchomienie skryptu bez konieczności dodatkowego zwalniania urządzenia. Kadr jest zmniejszany do rozmiaru 320x240px, aby zapewnić płynność transmisji.

2.2.4 read_stream.py

```
1  #!/usr/bin/env python3
2
3  import cv2
4  import zmq
5  import socket
6  import base64
7  import sys
8  import getopt
9  import numpy as np
10
11
12 class Stream(object):
13     def __init__(self, host="127.0.0.1", port=5555):
14         addr = "tcp://{}:{}".format(host, port)
15         print("Connecting to socket: {}".format(addr))
16         try:
17             self.context = zmq.Context()
18             self.footage_socket = self.context.socket(zmq.SUB)
19             self.footage_socket.connect(addr)
20             self.footage_socket.setsockopt_string(zmq.SUBSCRIBE,
                   np.unicode(''))
21         except Exception:
22             print("Cannot connect to socket!")
23
24     def read(self):
25         try:
26             frame = self.footage_socket.recv_string()
```

```
27              img = base64.b64decode(frame)
28              npimg = np.fromstring(img, dtype=np.uint8)
29              source = cv2.imdecode(npimg, 1)
30              return True, source
31          except Exception:
32              return False, None
33
34  def help_print():
35      print('./read_stream.py␣-i␣<ip>␣-p␣<port>')
36      sys.exit()
37
38  if __name__ == "__main__":
39      kwargs = {}
40
41      try:
42          opts, args = getopt.getopt(sys.argv[1:], "hi:p:", ["ip=",
                "port="])
43      except getopt.GetoptError:
44          help_print()
45
46      for opt, arg in opts:
47          if opt == '-h':
48              help_print()
49          elif opt in ("-i", "--ip"):
50              kwargs["host"] = arg
51          elif opt in ("-p", "--port"):
52              kwargs["port"] = arg
53
54
55      s = Stream(**kwargs)
56      while True:
57          try:
58              cv2.imshow("Stream", s.read()[1])
59              cv2.waitKey(1)
60
61          except KeyboardInterrupt:
62              cv2.destroyAllWindows()
63              break
```

Listing 4: Skrypt odbierający obraz ze strumienia i wyświetalający go

Skrypt zawiera klasę `Stream`, która pozwala w łatwy sposób pobrać obraz ze strumienia. Do konstrukora podać można 2 opcjonalne argumenty - `host` oraz `port`, które definiują adres serweru socket. Ustawienie tychże parametrów jest możliwe z linii poleceń za pomocą opcji `-i` oraz `p`, bądź `ip=`, `port=`.

Obraz pobierany jest poprzez methodę `read`. Dla autora ważne było, aby nazwa metod była tożsama z tymi obiektu `VideoCapture` z OpenCV, przez co możliwe jest łatwe zmienianie źródła obrazu.

Jeśli skrypt wywoływany jest z linii poleceń, uruchamiana jest pętla pobierają obraz ze streamu i wyświetlająca go bez żadnych modyfikacji. Służy to przetestowaniu poprawności transmisji.

2.2.5 process_stream.py

```
1  #!/usr/bin/env python3
2
3  import sys
4  import getopt
5  import requests
6  import cv2
7  import numpy as np
8  import read_stream
9
10
11 # termination criteria
12 criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
       0.001)
13
14 # Load camera matrix and distortions matrix
15 mtx = np.loadtxt('mtx.txt')
16 dist = np.loadtxt('dist.txt')
17
18 # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(8,6,0)
19 objp = np.zeros((7*9, 3), np.float32)
20 objp[:, :2] = np.mgrid[0:9, 0:7].T.reshape(-1, 2)
21
22 # prepare axis
23 axis = np.float32([[3, 0, 0], [0, 3, 0], [0, 0, -3]]).reshape(-1, 3)
24
25 FPS = int(1000/60)
```

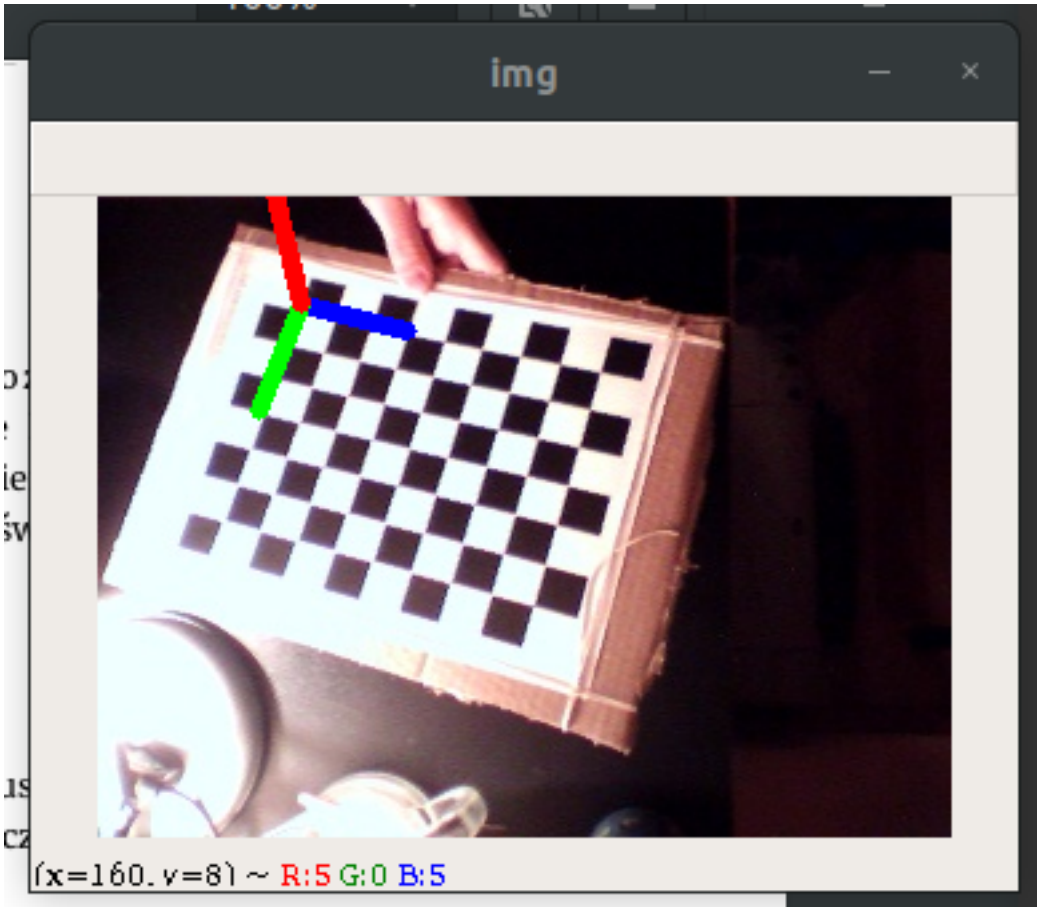
```
26
27
28
29  ### Functions
30
31  def draw(img, corners, imgpts):
32      corner = tuple(corners[0].ravel())
33      img = cv2.line(img, corner, tuple(imgpts[0].ravel()), (255, 0, 0),
            5)
34      img = cv2.line(img, corner, tuple(imgpts[1].ravel()), (0, 255, 0),
            5)
35      img = cv2.line(img, corner, tuple(imgpts[2].ravel()), (0, 0, 255),
            5)
36      return img
37
38  def read_video(cap):
39      while(True):
40          # Capture frame-by-frame
41          ret, img = cap.read()
42          if img is not None:
43              # Display the resulting frame
44              # cv2.imshow('img', img)
45              gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
46              # cv2.imshow('img', gray)
47              ret, corners = cv2.findChessboardCorners(gray, (9, 7),
                    flags=cv2.CALIB_CB_FAST_CHECK)
48
49              if ret is True:
50                  corners2 = cv2.cornerSubPix(gray, corners, (11, 11),
                        (-1, -1), criteria)
51
52                  # Find the rotation and translation vectors.
53                  ret, rvecs, tvecs, _ = cv2.solvePnPRansac(objp,
                        corners2, mtx, dist)
54
55                  # project 3D points to image plane
56                  imgpts, jac = cv2.projectPoints(axis, rvecs, tvecs,
                        mtx, dist)
57                  img = draw(img, corners2, imgpts)
58                  cv2.imshow('img', img)
```

```
59              else:
60                  cv2.imshow("img", img)
61
62              # Press q to close the video windows before it ends if you
                    want
63              cv2.waitKey(FPS)
64          else:
65              print("Frame is None")
66              break
67
68  def help_print():
69      print('./process_stream.py -i <ip> -p <port>')
70      sys.exit()
71
72
73  if __name__ == "__main__":
74      kwargs = {}
75
76      try:
77          opts, args = getopt.getopt(sys.argv[1:], "hi:p:", ["ip=",
                "port="])
78      except getopt.GetoptError:
79          help_print()
80
81      for opt, arg in opts:
82          if opt == '-h':
83              help_print()
84          elif opt in ("-i", "--ip"):
85              kwargs["host"] = arg
86          elif opt in ("-p", "--port"):
87              kwargs["port"] = arg
88
89
90      s = read_stream.Stream(**kwargs)
91      read_video(s)
```

Listing 5: Skrypt odbierający obraz ze strumienia i transformujący go

Podobnie jak w poprzednim podrozdziale, możliwe jest opcjonalne podanie hosta oraz portu serwera socketu. Skrypt pobiera obraz ze strumienia i modyfikuje go wedle wskazówek.

Zawartość skryptu zmienia się i związana jest bezpośrednio ze stanem zaawansowania pracy magisterskiej autora. Powyższy skrypt wykorzystuje kadry do znalezienia na nim szachownicy o rozmiarze `9x11`, wyliczenie jej orientacji w przestrzeni, narysowanie w rogu szachownicy układu współrzędnych i wyświetlenie ostatecznego efektu użytkownikowi. Proces wykrywania i wyznaczania orientowacji i położenia szachownicy, oraz rysowanie linii na kadrze realizowane jest za pomocą funkcji biblioteki OpenCV.



Rysunek 2: Przykład narysowanego układu współrzędnych

2.3 Raspberry Pi

Skrypt `start_stream.py` za pomocą `crontab` został ustawiony jako skrypt uruchamiany od razu po starcie systemu, dlatego też do rozpoczęcia strumieniowania konieczne jest jedynie podłączenie zasilania do urządzenia.

Jedyną sytuacją w której wymagana jest manualna ingerencja w urządzenie jest połączenie z nową sieciową WiFi.

Skrypt w systemie urządzenia umieszczony jest w `/var/stream_video/main.py`. Strumień wyjściowy skryptu zapisywany jest w pliku `logs.log`, dzięki czemu możliwe jest debuggowanie w razie problemów.