**Jurnal Komunikasi, Sains dan Teknologi**
Vol. 1, No. 1, Juni 2022, pp. 10-18
E-ISSN: 2964 - 5077 

# *Faster Video Processing* Menggunakan Teknik *Parallel Processing* Dengan *Library OpenCV*

**Kresna Wira Widjanarko[1](✉), Krisna Aditya Herlambang[1] Muhamad Abdul Karim[1]**
[1)]Teknik Informatika, Fakultas Teknik Universitas Pancasila, Jakarta Selatan, Indonesia
*4519210052@univpancasila.ac.id*

**Informasi Artikel**

***Sejarah Artikel:***

Disubmit 13 April 2022
Direvisi 22 April 2022
Diterima 10 Mei 2022

***Kata Kunci:***

*Video Processing*,
proses parallel,
webcam,
python,
*OpenCV*

**Abstrak**

Video yang diputar seringkali terlalu lama diproses, terutama aplikasi yang membutuhkan pemrosesan real-time, seperti pemrosesan aplikasi video webcam, oleh karena itu dilakukan pemrosesan paralel untuk mempercepat proses komputasi video. Penelitian ini membahas tentang pemrosesan paralel pada video agar proses komputasi video berjalan lebih cepat dibandingkan tanpa pemrosesan paralel. Pengujian dilakukan dengan dua jenis data: aliran video dari webcam laptop dan file video .mp4. Bahasa pemrograman yang digunakan dalam pengujian ini adalah Python dengan bantuan library OpenCV. Penelitian ini menghasilkan perbedaan yang signifikan dalam pemrosesan video baik dengan sumber webcam maupun File Video dalam format .mp4 tanpa pemrosesan paralel (multithreading) dengan Video Show dan Video Read serta penggabungan keduanya (Multithreading).



***Corresponding Author:***

Kresna Wira Widjanarko
Teknik Informatika, Fakultas Teknik Universitas Pancasila, Jakarta Selatan, Indonesia
Email: *4519210052@univpancasila.ac.id*

## 1. Pendahuluan

Seiring perkembangan zaman, masyarakat semakin dimudahkan untuk berkomunikasi maupun berkirim informasi. Awal mula perkembangan film di dunia sejak ditemukan cara merekam gambar menjadi sebuah video foto pertama dibuat pada tahun 1826 selama 8 jam Louis Jacques mande Daquerre merupakan bapak fotografi dunia (1837), sehingga pada zaman tersebut semua orang berlomba-lomba untuk mengembangkan fotografi menjadi tayangan video[1]. Semakin dimudahkannya pembuatan video, masyarakat semakin tertarik akan video. Dulu video hanya bisa dibuat menggunakan kamera namun seiring perkembangan zaman, handphone pun sudah bisa merekam video bahkan hingga resolusi 4K sekalipun. Video bisa dijadikan untuk merekam kenangan yang terjadi di sekitar kita, lalu berubah untuk dijadikan komersil karena dalam satu video bisa memuat banyak pesan di dalamnya. Ketika pandemi covid-19 menyerang hampir atau bahkan seluruh dunia, semua aktifitas berubah. Dari yang seharusnyadilakukan tatap muka diubah menjadi video conference. Namun masih ada beberapa sektor yang memang diharuskan untuk bertatap muka masih bisa dilakukan. Banyak sektor yang terkena dampak diubahnya peraturan ini mulai dari sekolah, kantor, hiburan, dan lainnya. Dengan diberlakukannya aturan baru ini banyak sekali masyarakat yang diharuskan untuk bertatap muka secara online alias melalui gadget atau komputernya. Salah satu teknologi yang digunakan untuk menunjang aturan baru tersebut yaitu menggunakan video *conference* atau menggunakan video stream webcam. Video stream webcam digunakan hampir di seluruh gadget, mulai dari *smartphone* hingga komputer. Namun bukan tidak mungkin ada terjadinya halangan yaitu ketika pemrosesan komputasi video ini. Kemungkinan halangannya yaitu bisa berupa terjadinya kelambatan ketika proses. Pada penelitian ini dilakukan proses paralelisasi dengan cara. Mendapatkan frame dari sumber data berupa webcam atau file video berformat .mp4 lalu memproses frame tersebut kemudian frame yang sudah diproses ditampilkan. Dengan memindahkan proses operasi membaca dan menampilkan ke dalam thread yang berbeda, setiap perulangan pada while loop akan memerlukan waktu yang lebih sedikit untuk mengeksekusi. Alasan pemilihan judul yaitu ukuran video yang semakin besar dan semakin berkembangnya layanan *streaming video* mengharuskan proses video menjadi

lebih cepat agar experience pengguna juga tetap nyaman. Oleh karena itu, kami mendapat ide untuk menjadikan Video *Processing* sebagai topik kami. Penelitian dan eksperimen ini telah dilakukan oleh Najam R. Syed. Sehingga, kami hanya menjelaskan dan melakukan eksperimen ulang menggunakan perangkat kami mengenai penelitian yang telah dilakukan olehnya. Dengan eksperimen ini, diharapkan dapat diketahui bahwa video processing bisa dipercepat dengan menggunakan teknik parallel processing khususnya menggunakan bahasa pemrograman Python.

## 2. Metodolologi

Pada penelitian ini, penulis menggunakan data yang bersumber dari streaming video webcam dan/atau video berformat .mp4. Metode yang digunakan yaitu mendapatkan frame dari video yang ada. Proses ini berfokus pada tercapainya peningkatan performa dari video processing, di mana proses mendapatkan frame dari video dan proses menampilkan framedipisahkan dalam thread yang berbeda. Dengan memindahkan operasi pembacaan frame dan penampilan frame ke thread lain, setiap iterasi dari loop while akan membutuhkan lebih sedikit waktu untuk dieksekusi.

### *2.1 Computer Vision*

*Computer Vision* (CV), didefinisikan sebagai bidang studi yang berupaya mengembangkan teknik untuk membantu komputer "melihat" dan memahami konten gambar digital seperti foto dan video. Sebagai suatu disiplin ilmu, visi komputer berkaitan dengan teori di balik sistem buatan bahwa ekstrak informasi dari gambar. Data gambar dapat mengambil banyak bentuk, seperti urutan video, pandangan dari beberapa kamera, atau data multi-dimensi dari scanner medis [2]. *Computer Vision* juga dapat diartikan dengan pengolahan citra yang berkaitan dengan akuisisi citra, klasifikasi, pemrosesan, dan pencakupan keseluruhan[3]. Computer Vision memiliki tiga proses antara lain Mendapatkan citra digital, Memodifikasi data citra dengan teknik komputasi, Menganalisis dan menginterpretasi citra. Tujuan dari computer vision adalah untuk memahami isi dari gambar digital.

### *2.2 Parallel Processing*

*Parallel Processing* adalah metode dalam komputasi yang menjalankan dua atau lebih prosesor (CPU) untuk menangani bagian-bagian terpisah dari tugas keseluruhan. Memecah bagian tugas yang berbeda di antara beberapa prosesor akan membantu mengurangi jumlah waktu untuk menjalankan program. Prosesor multi-core adalah chip IC yang berisi dua atau lebih prosesor untuk kinerja yang lebih baik, konsumsi daya yang lebih rendah, dan pemrosesan banyak tugas yang lebih efisien. Cara kerja dari teknik ini yaitu dengan membagi tugas yang kompleks menjadi beberapa bagian dengan alat perangkat lunak dan menetapkan setiap bagian ke prosesor, kemudian setiap prosesor akan menyelesaikan bagiannya, dan data dirakit kembali oleh alat perangkat lunak untuk membaca solusi atau menjalankan tugas. Setiap prosesor akan beroperasi secara normal dan akan melakukan operasi secara paralel seperti yang diinstruksikan, menarik data dari memori komputer. Prosesor juga akan mengandalkan perangkat lunak untuk berkomunikasi satu sama lain sehingga mereka dapat tetap sinkron terkait perubahan nilai data. Dengan asumsi semua prosesor tetap sinkron satu sama lain, pada akhir tugas, perangkat lunak akan menyatukan semua bagian data. Banyak hal yang bisa dilakukan dengan menggunakan proses paralel salah satunya yaitu pemrosesan citra satelit optik yang menyatu melalui teknik pemrosesan paralel dalam multi GPU.

### *2.3 OpenCV*

OpenCV (*Open Source Computer Vision Library*) merupakan sebuah open source library untuk *computer vision* dan machine learning. OpenCV dibangun untuk menyediakan infrastruktur umum untuk aplikasi computer vision dan untuk mempercepat penggunaan persepsi mesin dalam produk komersial. Library ini memiliki lebih dari 2500 algoritma yang dioptimalkan, yang mencakup serangkaian lengkap computer vision klasik dan canggih serta algoritma machine learning. Algoritma - algoritma tersebut dapat digunakan untuk mendeteksi dan mengenali wajah, mengidentifikasi objek, mengklasifikasikan tindakan manusia dalam video, melacak pergerakan kamera, melacak objek bergerak, mengekstrak model objek 3D, hingga menggabungkan gambar untuk menghasilkan resolusi tinggi. OpenCV memiliki antarmuka C++, Python, Java dan MATLAB serta mendukung Windows, Linux, Android dan Mac OS. OpenCV sebagian besar condong ke real-time vision applications dan memanfaatkan instruksi MMX dan SSE bila tersedia. Antarmuka CUDA dan OpenCL berfitur lengkap sedang dikembangkan secara aktif saat ini. Ada lebih dari 500 algoritma dan sekitar 10 kali lebih banyak fungsi yang menyusun atau mendukung algoritma tersebut. OpenCV juga bisa digunakan untuk membuat program barcode, dengan menggunakan OpenCV, barcode bisa dibaca dalam waktu sekitar 0.3 detik sedangkan menggunakan aplikasi android membutuhkan waktu sekitar 0.4 detik.

### *2.4 Video Processing*

*Video processing* adalah suatu bidang pengolahan video yang digunakan untuk memperbaiki suatu citra atau mendapatkan suatu informasi pada sebuah video, dan video processing bisa diaplikasikan pada perbaikan citra, karena sejatinya video adalah kumpulan citra yang dijalankan secara sekuensial [7].

### *2.5 Thread dan Multithreading*

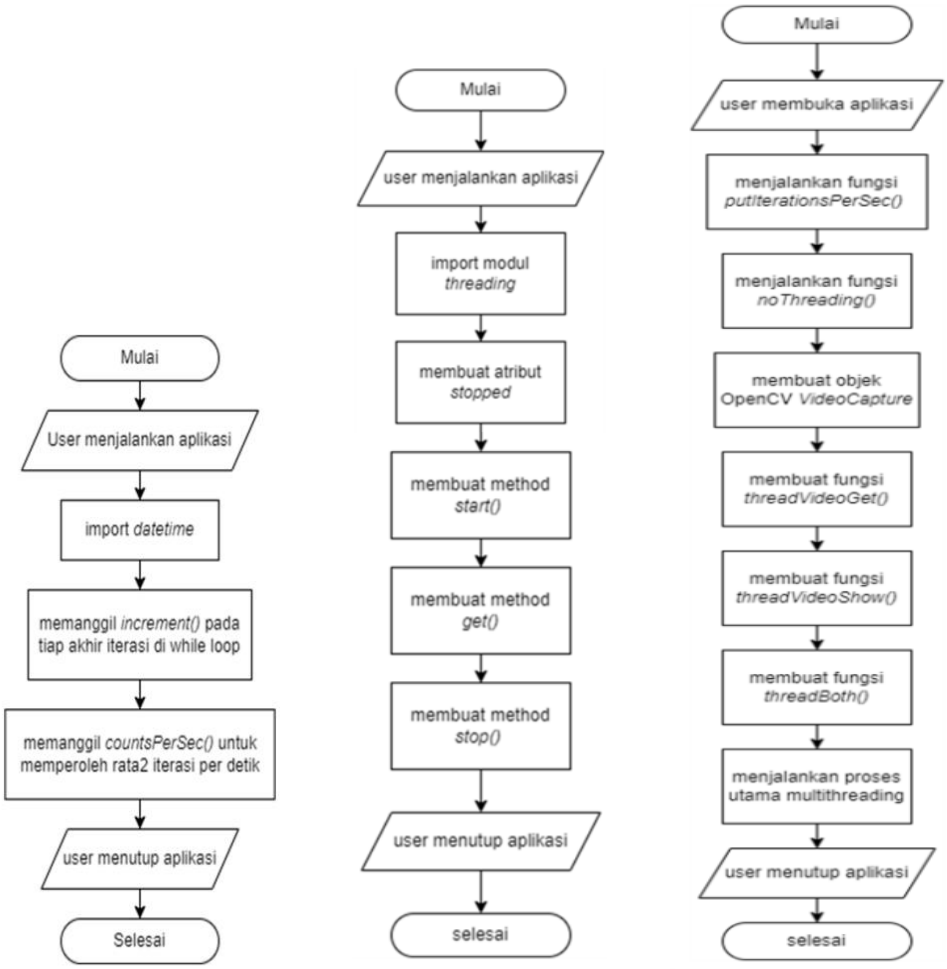
Thread adalah sebuah eksekusi dari kernel dengan sebuah indeks yang diberikan/ditentukan. Setiap thread akan menggunakan index tersebut untuk mengakses element di dalam array seperti koleksi-koleksi dari semua thread yang bekerja sama pada semua data set [8]. *Multithreading* adalah kumpulan beberapa thread yang dijalankan bersamaan pada sebuah processor. Pada multithreading ini merupakan proses eksekusi dari kumpulan *thread* dimana kumpulan thread tersebut diproses secara berulang – ulang dengan perpindahan dalam waktu *nanosecond* [9].

## 3. Pembahasan

Hipotesis awal dari penelitian ini adalah parallel processing bisa mempercepat processing video. Lalu, hasil akhir yang akan didapatkan yaitu berupa perbandingan mengenai perbedaan saat digunakannya teknik parallel (dalam hal ini multithreading) dengan tidak menggunakan teknik parallel. Penelitian yang kami lakukan memerlukan dua jenis data yaitu video stream dari webcam laptop dan file video berjenis .mp4. Kami menggunakan python dalam proses penelitian ini dan juga program yang ada dijalankan langsung pada laptop bersistem operasi windows 10 dengan prosesor AMD FX dan RAM 8GB.

### 3.1. Algoritma

Algoritman dari count data perdetik untuk melakukan proses. Algoritma yang digunakan pada file CountsPerSec.py dan VideoGet.py dapat dilihat pada gambar 1 dan 2.




(a) Algoritma file CountsPerSec.py (b) Algoritma file VideoShow.py (c) Algoritma file thread_demo.py

Gambar 1. Algoritma

Dalam single-threaded video processing application, kami meminta thread utama menjalankan beberapa *task* dalam perulangan tanpa batas saat mengulang:

- Mendapatkan frame dari stream webcam atau file video dengan cv2.VideoCapture.read().
- Memproses frame sesuai dengan yang kami inginkan.

- Menampilkan proses frame ke layar dengan memanggil cv2.imshow().

Dengan memindahkan proses membaca dan menampilkan frame ke dalam thread yang berbeda, tiap iterasi dari while loop memerlukan waktu yang lebih sedikit untuk mengeksekusi. Oleh karena itu, kami mendefinisikan metrik kinerja sebagai jumlah iterasi dari while loop di thread utama yang dieksekusi per detik.

### 3.2. Pemaparan Analisis Program

#### 3.2.1. Mengukur perubahan kinerja

Untuk mengukur iterasi dari while loop yang dieksekusi per detik, kami membuat class yang bernama CountsPerSec. Kami meng-*import datetime* untuk melacak waktu yang telah berlalu. Tiap akhir dari pengulangan while loop, kami memanggil increment() untuk meng-*increment* jumlahnya. Selama setiap iterasi, kami mendapatkan iterasi-iterasi rata-rata per detik untuk video dengan memanggil *method* CountsPerSec.

```python
from datetime import datetime

class CountsPerSec:
    """
    Class that tracks the number of occurrences ("counts") of an
    arbitrary event and returns the frequency in occurrences
    (counts) per second. The caller must increment the count.
    """

    def __init__(self):
        self._start_time = None
        self._num_occurrences = 0

    def start(self):
        self._start_time = datetime.now()
        return self

    def increment(self):
        self._num_occurrences += 1

    def countsPerSec(self):
        elapsed_time = (datetime.now() - self._start_time).total_seconds()
        return self._num_occurrences / elapsed_time if elapsed_time > 0 else 0
```

Gambar 2. Potongan source code dari file CountsPerSec.py

**3.2.2.** Kinerja tanpa multithreading

Dimulai dengan meng-import beberapa hal, termasuk *CountsPerSec* yang tadi dibuat, walaupun *VideoGet* dan *VideoShow* belum dibuat tetapi nantinya akan digunakan untuk mengeksekusi *tasks* dalam mendapatkan frame video dan menampilkan frame tersebut dalam threadnya masing-masing. Fungsi *putIterationPerSec()* menampilkan teks yang mengindikasikan frekuensi dari while loop utama dalam *iterations per second* ke sebuah frame sebelum menampilkan framenya. Menerimanya sebagai (numpy array) dan iterations per second (float) menampilkan nila sebagai text dengan cv2.putText() dan mengembalikan *frame modified.*

```python
import argparse
import os
import cv2
from CountsPerSec import CountsPerSec
from VideoGet import VideoGet
from VideoShow import VideoShow

def putIterationsPerSec(frame, iterations_per_sec):
    """
    Add iterations per second text to lower-left corner of a frame.
    """

    cv2.putText(frame, "{:.0f} iterations/sec".format(iterations_per_sec),
        (10, 450), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255))
    return frame
```

Gambar 3. Potongan satu source code dari file thread_demo.py

Selanjutnya, kami mendefinisikan fungsi noThreading() untuk mendapatkan frame, mengomputasi, dan menampilkan nilai *iterations per second* dalam tiap frame dan menampilkan frame nya. Fungsi tersebut mengambil sumber video hanya argumennya, jika dimasukkan integer, parameter *source* mengindikasi sumber video berasal dari webcam. Angka 0 dari webcam pertama, angka 1 webcam kedua yang terhubung dan seterusnya. Jika sebuah string yang dimasukkan berarti diinterpretasikan sebagai *path* dari file video.

```python
def noThreading(source=0):
    """Grab and show video frames without multithreading."""

    cap = cv2.VideoCapture(source)
    cps = CountsPerSec().start()

    while True:
        grabbed, frame = cap.read()
        if not grabbed or cv2.waitKey(1) == ord("q"):
            break

        frame = putIterationsPerSec(frame, cps.countsPerSec())
        cv2.imshow("Video", frame)
        cps.increment()
```

Gambar 4. Potongan dua source code dari file thread_demo.py

Baris 20-21, kami membuat sebuah OpenCV objek yang bernama VideoCapture untuk mengambil dan men-decode frame dari webcam atau file video dan juga objek CountsPerSec untuk melacak performa while loop utama. Baris 23 memulai while loop utama. Baris 24 kami memanfaatkan method read() dari objek VideoCapture untuk mendapatkan dan men-decode frame selanjutnya dari video stream dan mengembalikan nilai boolean. Grabbed mengindikasikan apakah frame berhasil diambil dan di-decode dan juga frame dalam bentuk numpy array frame. Baris 25, mengecek jika frame tidak berhasil diambil atau jika pengguna menekan tombol ‘q’ untuk keluar dari program . Dalam kedua kasus, kami menghentikan eksekusi while loop dengan break. Kecuali kondisi mana pun, kami melanjutkan dengan secara bersamaan memperoleh dan melapisi “kecepatan” loop saat ini (dalam iterations per second) di sudut kiri bawah bingkai pada baris 28. Akhirnya, frame ditampilkan dalam baris 29 dengan memanggil cv2. imshow() dan perhitungan iterasi di increment pada baris 30.

**3.2.3.** Sebuah thread terpisah untuk mendapatkan frame video

Pertama, kami buat class bernama VideoGet dalam sebuah file VideoGet.py Lalu, kami import threading modul yang bisa mengizinkan kami untuk memunculkan thread baru. Dalam method ___init___(), kami menginisialisasi sebuah object OpenCV yang bernama VideoCapture dan membaca frame pertama. Kami juga membuat attribut stopped untuk bertindak sebagai flag dan mengindikasikan sebuah thread harus berhenti mengambil frame baru.

```python
from threading import Thread
import cv2

class VideoGet:
    """
    Class that continuously gets frames from a VideoCapture object
    with a dedicated thread.
    """

    def __init__(self, src='0'):
        self.stream = cv2.VideoCapture(src)
        (self.grabbed, self.frame) = self.stream.read()
        self.stopped = False
```

Gambar 5. Potongan satu source code dari file VideoGet.py

Method start() membuat dan memulai thread pada baris 16. *Thread* mengeksekusi fungsi get() yang didefinisikan pada baris 19. Fungsi ini secara berlanjut menjalankan while loop yang membaca sebuah frame dari video stream dan menyimpannya dalam sebuah atribut class instance frame selama flag stopped tidak diatur. Jika sebuah frame tidak berhasil dibaca (mungkin terjadi jika webcam tidak terhubung atau video telah berakhir), flag stopped akan diatur True dengan memanggil fungsi stop() yang didefinisikan pada baris 26.

```python
    def start(self):
        Thread(target=self.get, args=()).start()
        return self

    def get(self):
        while not self.stopped:
            if not self.grabbed:
                self.stop()
            else:
                (self.grabbed, self.frame) = self.stream.read()

    def stop(self):
        self.stopped = True
```

Gambar 6. Potongan dua source code dari file VideoGet.py

Kembali pada file thread_demo.py, kami definisikan sebuah fungsi threadVideoGet() yang akan menggunakan objek VideoGet tadi untuk membaca frame video dalam yang terpisah selama thread utama menampilkan frame. Fungsi nya hampir mirip dengan fungsi noThreading() sebelumnya, hanya saja di sini kami menginisialisasi objek VideoGet dan memulai thread kedua pada baris 38. Pada baris 42-44 dari *while loop* utama, kami mengecek untuk melihat apakah pengguna menekan tombol 'q' atau jika atribut objek VideoGet stopped telah diatur menjadi true, dalam hal ini kami menghentikan *while loop.* Jika tidak, *loop* mendapatkan *frame* yang saat disimpan di objek VideoGet pada baris 46, kemudian melanjutkan memproses dan menampilkannya seperti pada fungsi noThreading().

```
32  def threadVideoGet(source=0):
33      """
34      Dedicated thread for grabbing video frames with VideoGet object.
35      Main thread shows video frames.
36      """
37
38      video_getter = VideoGet(source).start()
39      cps = CountsPerSec().start()
40
41      while True:
42          if (cv2.waitKey(1) == ord("q")) or video_getter.stopped:
43              video_getter.stop()
44              break
45
46          frame = video_getter.frame
47          frame = putIterationsPerSec(frame, cps.countsPerSec())
48          cv2.imshow("Video", frame)
49          cps.increment()
```

Gambar 6. Potongan tiga source code dari file VideoGet.py

**3.2.4.** Sebuah thread terpisah untuk menampilkan *frame* video

Untuk memindahkan tugas dalam menampilkan *frame* video ke *thread* yang terpisah, kami mengikuti prosedur yang mirip dengan sebelumnya dan kami definisikan ke dalam *class* bernama Videoshow dalam sebuah file *VideoShow.py*.

```
1   from threading import Thread
2   import cv2
3
4   class VideoShow:
5       """
6       Class that continuously shows a frame using a dedicated thread.
7       """
8
9       def __init__(self, frame=None):
10          self.frame = frame
11          self.stopped = False
```

Gambar 7. Potongan source code dari file VideoShow.py

Kali ini, thread baru memanggil method show() yang didefinisikan pada baris 17. Perhatikan bahwa mengecek input pengguna pada baris 20 dicapai di thread terpisah daripada di thread utama, karena fungsi OpenCV waitKey() tidak selalu berfungsi dengan baik di aplikasi multithread.

```
13      def start(self):
14          Thread(target=self.show, args=()).start()
15          return self
16
17      def show(self):
18          while not self.stopped:
19              cv2.imshow("Video", self.frame)
20              if cv2.waitKey(1) == ord("q"):
21                  self.stopped = True
22
23      def stop(self):
24          self.stopped = True
```

Gambar 8. Potongan source code dari file VideoShow.py

Kembali pada file thread_demo.py kita definisikan fungsi yang bernama threadVideoShow(). Seperti yang sebelumnya, fungsi ini mirip dengan fungsi noThreading(), hanya saja kita menginisialisasi objek VideoShow yang bernama video_shower dan memulai thread baru pada baris 59. Baris 64 mengecek secara tidak langsung jika pengguna menekan tombol 'q' untuk keluar dari program, selama objek VideoShow

sebenarnya memeriksa *input* pengguna dan mengatur atributnya yang dihentikan ke *true* jika pengguna menekan ‘q’.Baris frame saat ini.

```
def threadVideoShow(source=0):
    """
    Dedicated thread for showing video frames with VideoShow object.
    Main thread grabs video frames.
    """

    cap = cv2.VideoCapture(source)
    (grabbed, frame) = cap.read()
    video_shower = VideoShow(frame).start()
    cps = CountsPerSec().start()

    while True:
        (grabbed, frame) = cap.read()
        if not grabbed or video_shower.stopped:
            video_shower.stop()
            break

        frame = putIterationsPerSec(frame, cps.countsPerSec())
        video_shower.frame = frame
        cps.increment()
```

Gambar 9. Potongan tiga source code dari file thread_demo.py

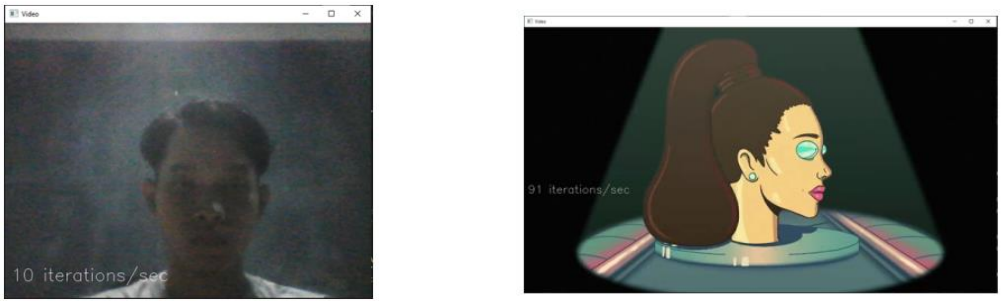
**3.2.4.** Thread terpisah untuk mendapatkan dan menampilkan frame video

Terakhir, kami mengimplementasikan sebuah fungsi yang bernama threadBoth() dalam thread_demo.py yang membuat sebuah *thread* untuk mendapatkan frame video melalui *class* VideoGet dan *thread* lain untuk menampilkan *frame* melalui *class* VideoShow dengan *thread* utama hadir untuk memproses dan melewati frame antara dua objek. Fungsi ini campuran dari fungsi threadVideoGet() dan threadVideoShow() yang ternyata memiliki hasil yang sangat menarik

**3.3.** Eksperimen

Eksperimen ini dilakukan dengan menjalankan program yang bernama thread_demo.py pada laptop bersistem operasi Windows 10 dengan prosesor AMD FX dan RAM 8GB. Untuk versi python yang digunakan yaitu python 3.9.7 64-bit.

**3.3.1** Eksperimen dilakukan **t**anpa Multithreading




(a) Sumber data webcam (b) Sumber data file video mp4

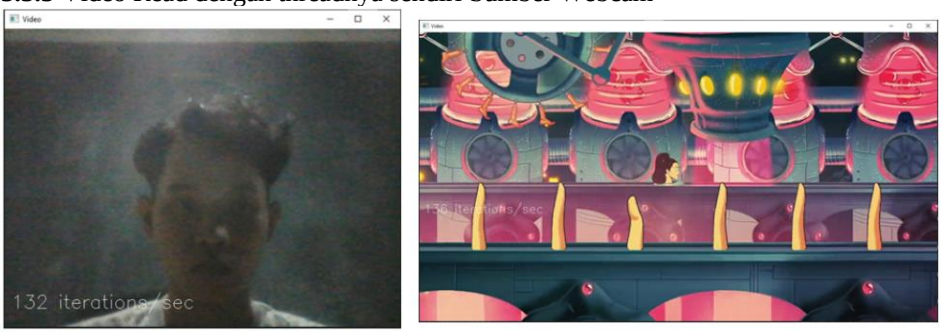
Gambar 10. Eksperimen dilakukan **t**anpa Multithreading

**3.3.2** Video Show dengan threadnya sendiri


(a) Sumber data webcam (b) Sumber data file video mp4

Gambar 11. Video Show dengan threadnya sendiri

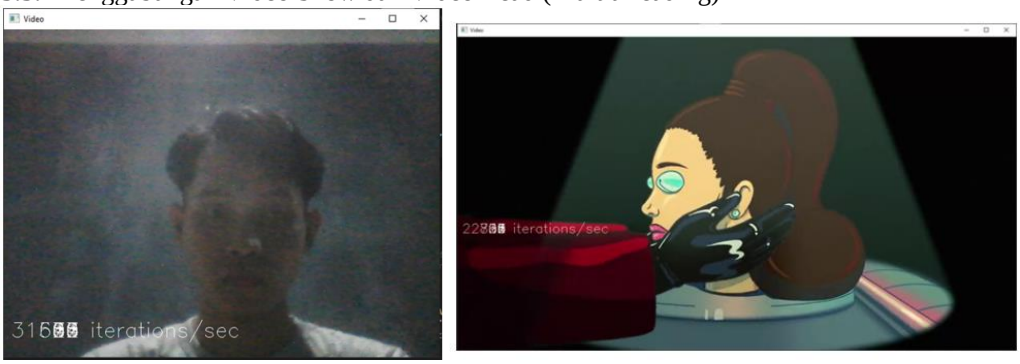
**3.3.3** Video Read dengan threadnya sendiri Sumber Webcam




(a) Sumber data webcam (b) Sumber data file video mp4

Gambar 12. Video Read dengan threadnya sendiri

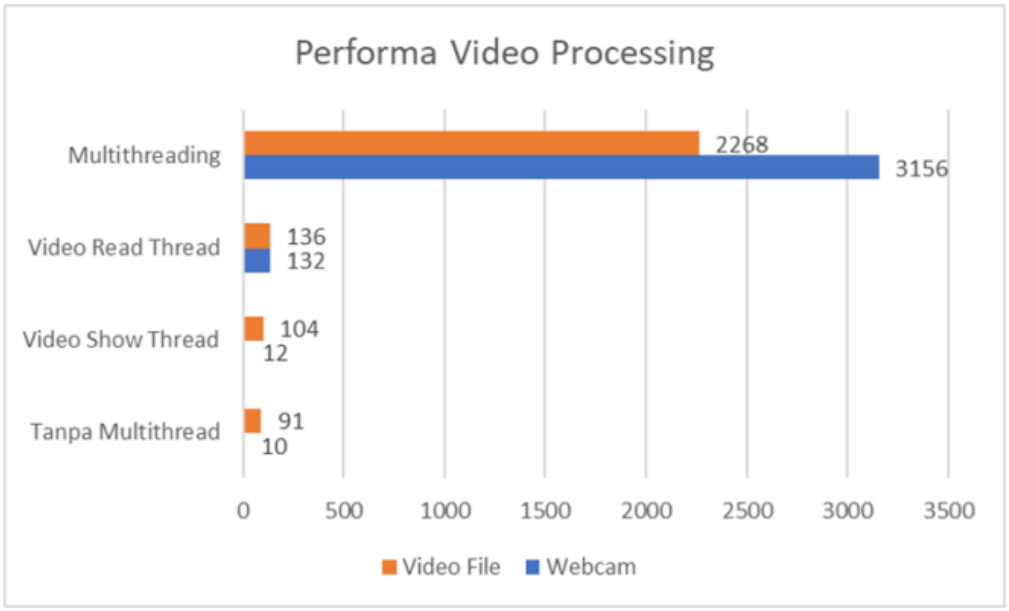
**3.3.4** Penggabungan Video Show dan Video Read (Multithreading)




(a) Sumber Webcam (b) Sumber data file video mp4

Gambar 13. Penggabungan Video Show dan Video Read (Multithreading)

Hasil pengujian performa video processing dari proses uji coba di bahwa terjadi perbedaan yang signifikan antara tanpa multi thread dan thread dengan multithreading yang menggunaka data secara real-time dari video menggunakan webcam serta terjadi juga pada video yang berasal dari mp4. Hasil dapat dilihat pada gambar dibawah ini.




Gambar 14. Grafik performa video processing

## 4. Kesimpulan

Penelitian ini melakukan eksperimen yang dijalankan untuk membuktikan perbedaan performa dari suatu perangkat komputer terhadap suatu proses tanpa parallel prosesing dengan menggunkannya. Pengujian dilakukan dengan dua objek data yaitu Video yang berasal dari webcam secara realtime dan video berupa file mp4. Penelitian ini menghasilkan perbedaan yang signifikan saat prosesing video baik dengan sumber webcam maupun Video File berformat .mp4 antara tanpa parallel processing (multithreading) dengan Video Show dan Video Read serta penggabungan keduanya (Multithreading).

## Daftar Pustaka


**[1]** Kabelen, N. W. (2021). Perjalanan Dan Perkembangan Videography Dari Ilmu Hingga Menjadi Sebuah Profesi. Jurnal Desain Komunikasi Visual Asia, 4(2), 79-86. (ISSN: 2580-8753 (print); 2597-4300 (online))

**[2]** Prabowo, D. A., & Abdullah, D. (2018). Deteksi dan perhitungan objek berdasarkan warna menggunakan Color Object Tracking. Pseudocode, 5(2), 85-91. (ISSN 2355-5920)

**[3]** Mauludy, A. T., Khrisne, D. C., & Saputra, K. O. (2020). Rancang Bangun Aplikasi Pencarian Slot Parkir Kosong Untuk Kendaraan Roda Empat Dengan Pendekatan Computer Vision. Jurnal SPEKTRUM Vol, 7(1).

**[4]** Puri, R., & Jain, V. (2019). Barcode detection using OpenCV-python. Science, 4(1), 97-99. (ISSN (Online): 2455-9024)

**[5]** Auccahuasi, W., Castro, P., Flores, E., Sernaque, F., Garzon, A., & Oré, E. (2020). Processing of fused optical satellite images through parallel processing techniques in multi GPU. Procedia Computer Science, 167, 2545-2553.

**[6]** TechTarget Contributor."Parallel Processing". https://searchdatacenter.techtarget.com/definition/ parallel-processing

**[7]** Wahyuni, E. S. (2020). Klasifikasi Usia Berdasarkan Kecepatan Berjalan Manusia Berbasis Video Processing.

**[8]** Rizaldi, H. I., Pramana, F. S., Bariq Najmi, R., Aditya Yudha, A. N., & Cholissodin, I. (2017). Optimasi Proses Rendering Objek Game 3D Menggunakan Pemrograman CUDA Pada Game Sandbox Craft. Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN, 2355, 207.

**[9]** Onggrono, K., & Tulus, E. B. N. (2017). Analisis Penggunaan Parallel Processing Multithreading Pada Resilient Backpropagation. InfoTekJar: Jurnal Nasional Informatika dan Teknologi Jaringan, 2(1), 33-40. (E-ISSN: 2540-7600, P-ISSN: 2540-7597)

**[10]** Jason Brownlee. 2019. "A Gentle Introduction to Computer Vision". https://machinelearningmastery.com/ what-is-computer-vision/

**[11]** OpenCV. https://opencv.org/

**[12]** Syed, Najam R. Multithreading with OpenCV-Python to improve video processing performance. 2018.[Online]. https://nrsyed.com/2018/07/05/multithreading-with opencv-python-to-improve-video-processing-performance/


## Biografi Penulis