

BAB IV

HASIL DAN PEMBAHASAN

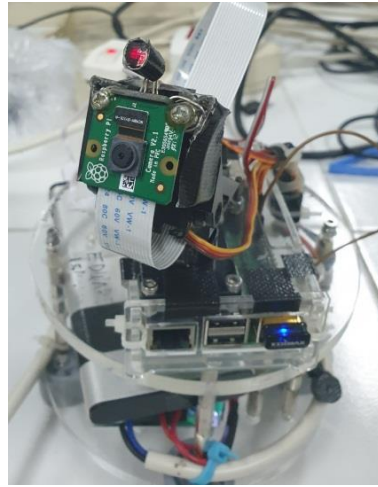
4.1. Pendahuluan

Pada Bab IV ini berisi tentang hasil dan pembahasan mengenai pengujian Penerapan Sistem *Pan-Tilt Camera* pada Objek berdasarkan Warna menggunakan *Raspberry Pi* yang sudah dibuat. Hasil yang ditampilkan pada alat ini yaitu tampilan *display* apakah objek dapat terdeteksi ataupun tidak dan mengamati pergerakan dari sistem *Pan-Tilt* dari beberapa teori dan rancangan yang sudah disajikan pada bab-bab sebelumnya.

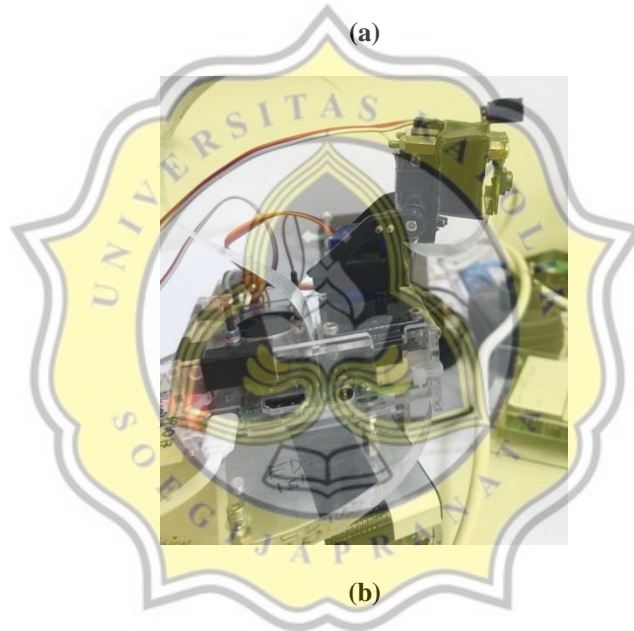
Pada bab ini juga ditampilkan *prototype* alat yang sudah dibuat sesuai dengan perancangan yang dimuat pada laporan ini.

4.2. Gambar *Prototype* Alat

Prototype alat yang telah dibuat dari beberapa komponen utama dan pendukung yang pengujiannya dilakukan di rumah dan laboratorium Tugas Akhir Program Studi Teknik Elektro Universitas Katolik Soegijapranata Semarang. Pada (Gambar-4.1) merupakan konstruksi *prototype* alat dari hasil rancangan yang sudah dibahas pada bab sebelumnya.



(a)

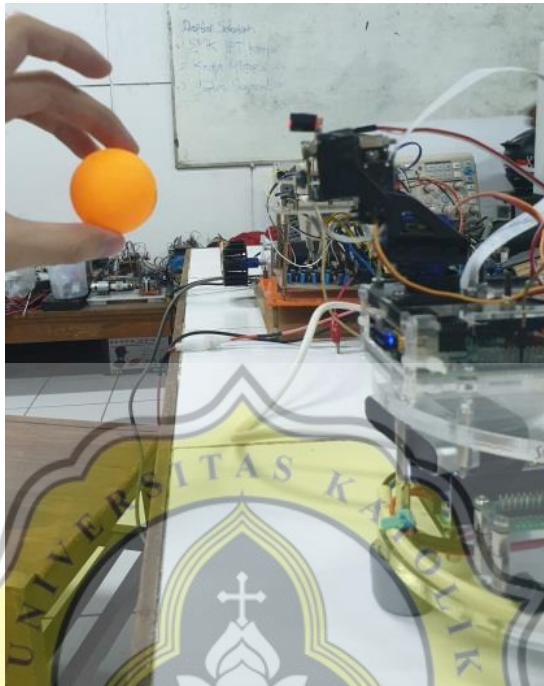


(b)

Gambar-4.1 *Prototype* bentuk alat (a) bentuk keseluruhan (b) *Pan-Tilt*

Pada pengujian kali ini peneliti akan menggunakan bola tenis meja berwarna *orange* seperti pada (Gambar-4.2) yang akan dideteksi oleh kamera. Alasan menggunakan warna *orange* karena kamera menangkap gambar yang lebih baik objek berwarna terang seperti merah, hijau, *orange*, dan sebagainya. Berbeda dengan ketika menangkap warna yang gelap seperti hitam, coklat, dan sebagainya,

maka kualitasnya yang didapat tidak sebaik ketika menangkap objek yang berwarna terang.



Gambar-4.2 Objek berupa bola yang akan dideteksi

4.3. Hasil Pengujian

Hasil data pengujian awal dengan *output* berupa gambar *display* yang ditampilkan oleh kamera secara *realtime*. Pengujian pertama menentukan data warna pada objek menggunakan program konversi RGB ke HSV. Pengujian kedua kamera akan mendeteksi objek dengan program *color tracking* kemudian menampilkan *display* pada monitor ketika dalam intensitas cahaya yang terang dan gelap. Pengujian terakhir kamera akan mendeteksi objek seperti pada pengujian kedua, namun ditambah dengan program tracking pada servo sistem *Pan-Tilt*.

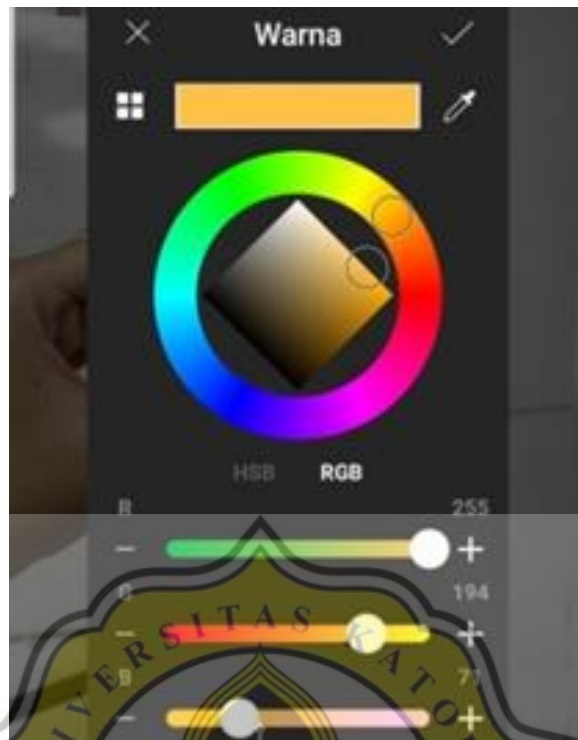
4.3.1 Pengujian awal pencarian data warna pada objek

Pengujian awal diperlukan karena sebelum pembuatan program *color tracking*, data warna harus ditentukan terlebih dahulu baru bisa mendeteksi objek yang akan ditarget. Pencarian data warna harus menggunakan format *RGB* (*Red, Green, Blue*) yang kemudian dikonversikan kembali menjadi format *HSV* (*Hue, Saturation, Value*) yang sudah ditentukan dalam *library OpenCV*. Sebelum itu, objek yang akan kita deteksi harus difoto terlebih dahulu dengan intensitas cahaya yang sesuai dalam ruangan pengujian seperti pada (Gambar-4.3).



Gambar-4.3 Warna pada bola yang akan dicari data RGB nya

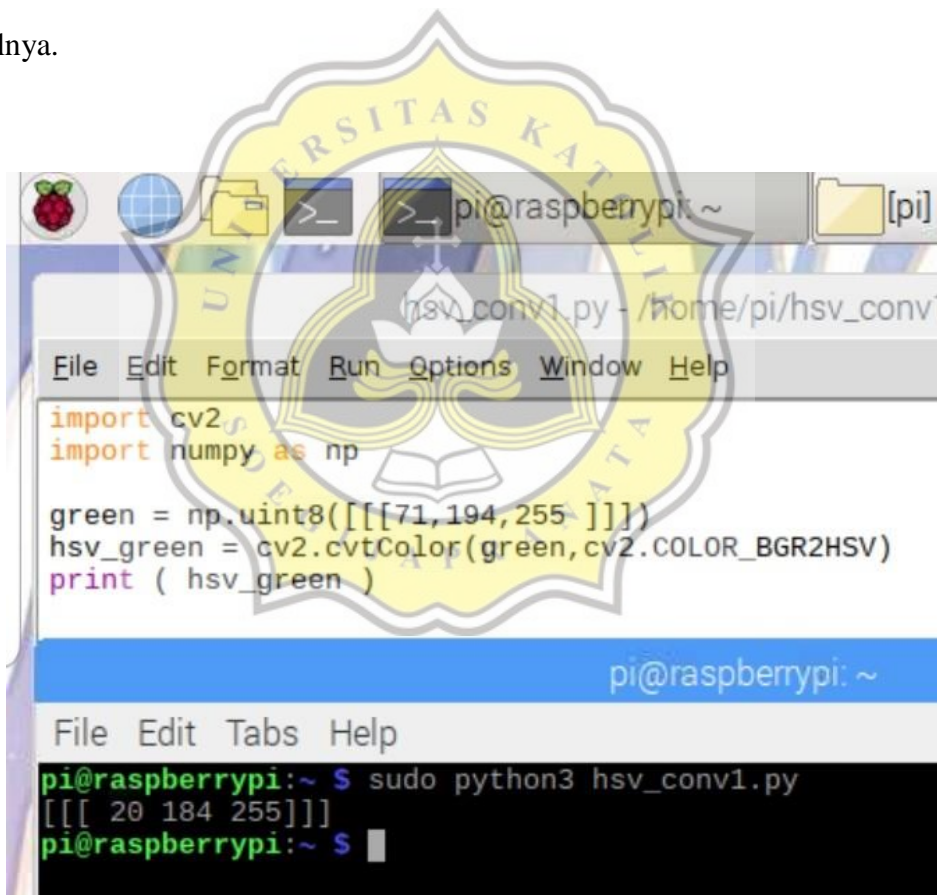
Pencarian data warna RGB pada objek dicari menggunakan aplikasi editor foto seperti *Pics Art* yang akan penulis gunakan pada penelitian ini, dan hasilnya seperti pada (Gambar-4.4) berikut ini.



Gambar-4.4 Data warna RGB pada bola

Setelah mendapat data warna RGB nya kemudian dilanjutkan dengan mencari data titik warna terendah dan titik warna teratas untuk program warna yang akan dideteksi dengan program *OpenCV* yang sudah dibuat seperti pada (Gambar-4.5). Titik terendah dan titik teratas tersebut dicari menggunakan program konverter *BGR to HSV* yang terdapat pada website resmi dari *OpenCV*. Pada program konverter tersebut memerlukan dua *library* yaitu, `import cv2` untuk mengaktifkan *library OpenCV* pada program dalam bentuk variabel `cv2` dan `import numpy as np` untuk mengaktifkan *library Numpy* sebagai *library* perhitungan angka dan `as np` sebagai bentuk kode variabel yang akan diinput pada program yang dibuat. Pada baris ketiga berisi `green = np.uint8([[71, 194, 255]])` merupakan penentuan angka RGB

warna yang diinginkan. Selanjutnya, pada baris keempat berisi `hsv_green = cv2.cvtColor(green, cv2.COLOR_BGR2HSV)` merupakan program kerja dari *OpenCV* untuk merubah format angka RGB menjadi format angka HSV yang dapat diproses oleh *library OpenCV*. Setelah itu `print (hsv_green)` untuk menampilkan hasil program pada *LXTerminal* kemudian didapat hasil angka (20,184,255) yang dikonversi menjadi titik bawah yaitu (20-10,100,100) dan (20+10,255,255). Berikut pada (Gambar-4.5) merupakan tampilan program dan hasilnya.

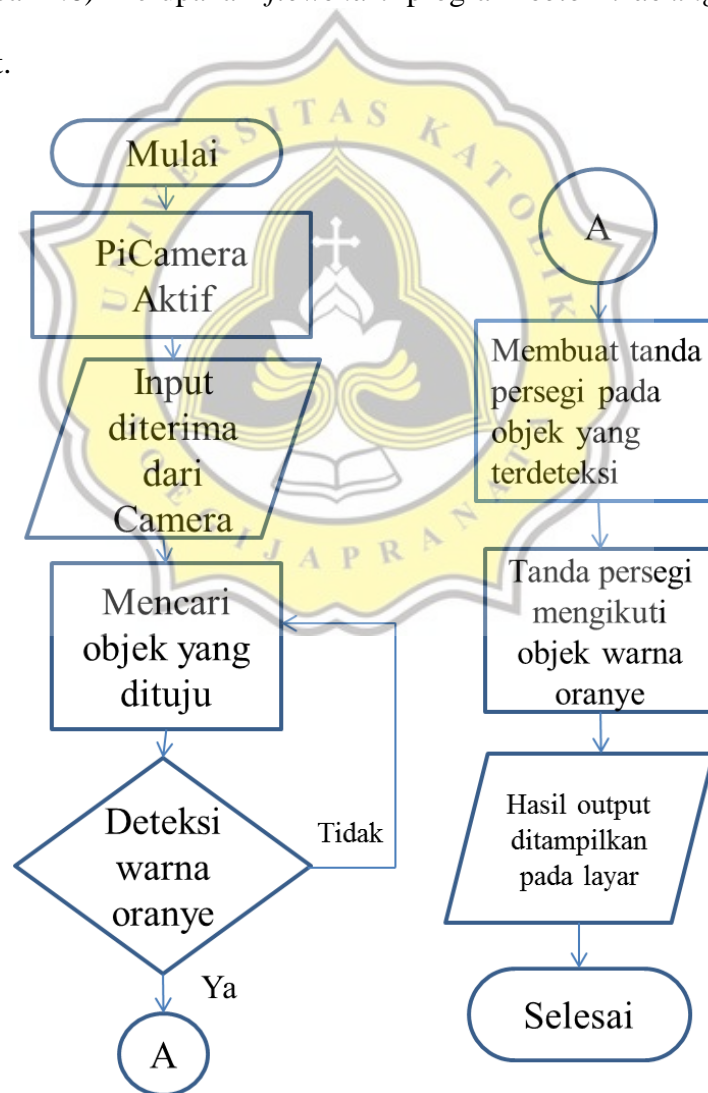


```
pi@raspberrypi: ~  
hsv_conv1.py - /home/pi/hsv_conv1  
File Edit Format Run Options Window Help  
import cv2  
import numpy as np  
  
green = np.uint8([[ [71, 194, 255 ] ]])  
hsv_green = cv2.cvtColor(green, cv2.COLOR_BGR2HSV)  
print ( hsv_green )  
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ sudo python3 hsv_conv1.py  
[[ [ 20 184 255 ] ]]  
pi@raspberrypi:~ $
```

Gambar-4.5 Program konversi data RGB ke HSV

4.3.2 Pengujian pemrograman *color tracking* pada objek bola

Pengujian kedua ini merupakan cara kerja program yang dibuat agar dapat mengenali data warna bola tenis meja yang sudah diinput pada program dan selalu mengikuti objek yang ditangkap kamera lalu diberi penanda berupa persegi di sekeliling objek kemudian ditampilkan pada *display* monitor. Setelah itu dilakukan pengujian program di intensitas cahaya yang terang dan gelap. Berikut pada (Gambar-4.6) merupakan *flowchart* program *color tracking* yang sudah peneliti buat.



Gambar-4.6 Flowchart cara kerja program *color tracking*

Pada *flowchart* di atas, program ini bekerja dengan resolusi gambar 640x480p di 50 FPS (*Frame Rate per Second*) dengan alasan agar program bekerja dengan optimal menyesuaikan dengan spesifikasi dari *Raspberry Pi* dibandingkan dengan resolusi yang besar namun dengan konsekuensi FPS lebih rendah dan objek sulit dilacak oleh program karena performa yang lambat. Pada pemrograman *color tracking* memerlukan beberapa *library* yang dibutuhkan seperti pada beberapa baris awal program yang sudah dibuat meliputi,

```
# import the necessary packages
from __future__ import print_function
from imutils.video import VideoStream
import argparse
import imutils
import time
import cv2
```

Pada beberapa baris yang tertera di atas dijelaskan bahwa untuk menjalankan program *color tracking* dibutuhkan *library* untuk mengaktifkan video kamera secara *real time* menggunakan `from imutils.video import VideoStream` yang berasal dari *software imutils* yang sudah menjadi satu paket dengan *library* utamanya yaitu *OpenCV*. Namun program tidak dapat berjalan tanpa `import cv2` atau input utama *library OpenCV*. Kemudian untuk fungsi `from __future__ import print_function` dibuat untuk penggunaan tanda kurung dalam isi program seperti menampilkan suatu tulisan atau angka ketika program dijalankan. Dalam program ini juga terdapat `import argparse` yang berfungsi untuk menjalankan program dalam bentuk argument seperti *x* dan *y*. Dan yang terakhir `import time` digunakan untuk pemanggilan fungsi waktu seperti *delay* ataupun *timer* yang dibutuhkan dalam suatu program.


```

# initialize the video stream and allow the Camera sensor to
    warm up
vs = VideoStream(0).start()
time.sleep(2.0)

# define the lower and upper boundaries of the object
# to be tracked in the HSV color space
colorLower = (10, 100, 100)
colorUpper = (30, 255, 255)

```

Setelah *library* yang dibutuhkan dalam program sudah di *input*, selanjutnya pada kutipan program diatas menjelaskan bagaimana cara mengaktifkan kamera ketika program dijalankan menggunakan `vs = VideoStream(0).start()` dan `time.sleep(2.0)` dan jeda waktu ketika kamera diaktifkan. Setelah itu menentukan variabel jarak warna HSV yang akan dideteksi oleh kamera menggunakan `colorLower = (10, 100, 100)` sebagai gradiasi warna terendah dan `colorUpper = (30, 255, 255)` sebagai gradiasi tertinggi. Untuk mencari jarak warna ini menggunakan program konversi RGB ke HSV seperti pada pengujian pertama.

```

while True:
    # grab the next frame from the video stream, Invert 0o,
    # resize the
    # frame, and convert it to the HSV color space
    frame = vs.read()
    frame = imutils.resize(frame, width=640)
    frame = imutils.rotate(frame, angle=0)
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

```

Berberapa kutipan program diatas menjelaskan penentuan resolusi tampilan kamera ketika program dijalankan. Pada program ini ketika resolusi yang diinginkan sebesar 640x480p maka menggunakan `frame = imutils.resize(frame, width=640)` dan untuk penentuan *angle* tampilan seperti 0 yang berarti tegak dan 180 yang berarti terbalik dapat diatur menggunakan `frame = imutils.rotate(frame, angle=0)`. Setelah itu

library OpenCV akan memproses warna yang akan dideteksi menggunakan `hsv`

```
= cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) .  
  
# construct a mask for the object color, then perform  
# a series of dilations and erosions to remove any small  
# blobs left in the mask  
mask = cv2.inRange(hsv, colorLower, colorUpper)  
mask = cv2.erode(mask, None, iterations=2)  
mask = cv2.dilate(mask, None, iterations=2)  
# find contours in the mask and initialize the current  
# (x, y) center of the object  
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,  
cv2.CHAIN_APPROX_SIMPLE)  
cnts = cnts[0] if imutils.is_cv2() else cnts[1]
```

Berberapa baris program diatas menjelaskan bagaimana ketika pendeteksian objek berwarna agar dapat sesuai dengan jarak warna yang sudah ditentukan sebelumnya menggunakan `mask = cv2.inRange(hsv, colorLower, colorUpper)` dan untuk menentukan ukuran objek yang terdeteksi oleh kamera agar tidak mengalami *noise* yang besar menggunakan `mask = cv2.erode(mask, None, iterations=2)` dan penentuan lebarnya menggunakan `mask = cv2.dilate(mask, None, iterations=2)`. Setelah itu dilakukan pendeteksian tepi pada objek agar lebih akurat menggunakan `cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`. Kemudian agar program dapat mengenali bagaimana tepi objek dapat terdeteksi oleh kamera ataupun tidak menggunakan `cnts = cnts[0] if imutils.is_cv2() else cnts[1]`.

```

max_area = 0
best_cnt = 1
if len(cnts) > 0:
    for Pic, contour in enumerate(cnts):
        area = cv2.contourArea(contour)
        if (area > max_area) :
            max_area = area
            best_cnt = contour
            x,y,w,h = cv2.boundingRect(best_cnt)
            cv2.rectangle(frame, (x, y),
(x+w,y+h), (0, 255, 0), 2)

```

Kutipan program diatas merupakan pembuatan tanda pada *display* apabila objek dapat terdeteksi ataupun tidak. Sebelum itu, agar program dapat memfokuskan pada satu objek berwarna saja, maka warna pada objek yang terdeteksi oleh kamera harus yang paling sesuai dengan format yang ditentukan menggunakan `max_area = 0` dan `best_cnt = 1`. Untuk mengetahui apakah objek sudah terdeteksi atau tidak, maka diperlukan penanda dalam bentuk persegi berwarna menggunakan fitur dari *OpenCV* yaitu `contourArea` dan `boundingRect` menggunakan vektor `x, y, w, h` seperti pada kutipan program diatas. Setelah itu penanda persegi yang dibuat juga diberi warna dalam format RGB seperti `(0, 255, 0)` yaitu warna hijau.

```

cv2.imshow('frame', frame)
key = cv2.waitKey(1) & 0xFF
if key == ord("q"):
    break

cv2.destroyAllWindows()
vs.stop()

```

Kutipan program diatas merupakan beberapa baris terakhir dari program color tracking yang berfungsi untuk menghentikan atau mematikan program ketika sudah berjalan. Pada program diatas cara menghentikan kerja programnya menggunakan tombol “q” pada keyboard, sehingga ketika tombol “q” ditekan

maka kamera dan tampilan displaynya juga akan mati menggunakan `cv2.destroyAllWindows()` dan `vs.stop()` pada program yang dibuat.

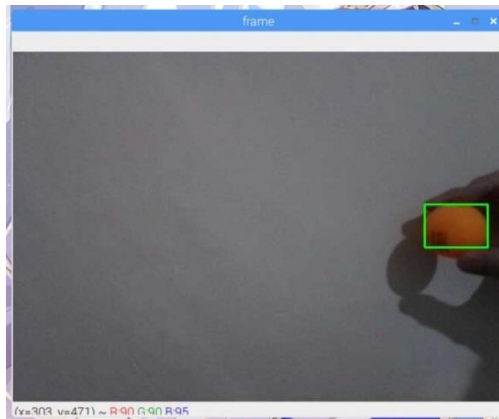
Berikut tampilan ketika program sudah dijalankan dan objek dapat terdeteksi ataupun tidak seperti pada (Gambar-4.7).



Gambar-4.7 Objek yang (a) terdeteksi dan (b) tidak terdeteksi

Setelah melihat gambar diatas maka diambil kesimpulan bahwa cahaya sangat mempengaruhi warna pada objek yang akan dilacak. Ketika kekurangan cahaya maka program akan sulit menangkap warna pada bola yang sudah ditentukan pada sebelumnya. Namun, jika bola tetap ingin terdeteksi dalam kondisi cahaya yang kurang, maka data warna pada program harus dirubah kembali menyesuaikan dengan foto yang juga diambil dalam kondisi cahaya tersebut. Pelacakan objek yang ditangkap kamera dapat dilihat di (Gambar-4.8).

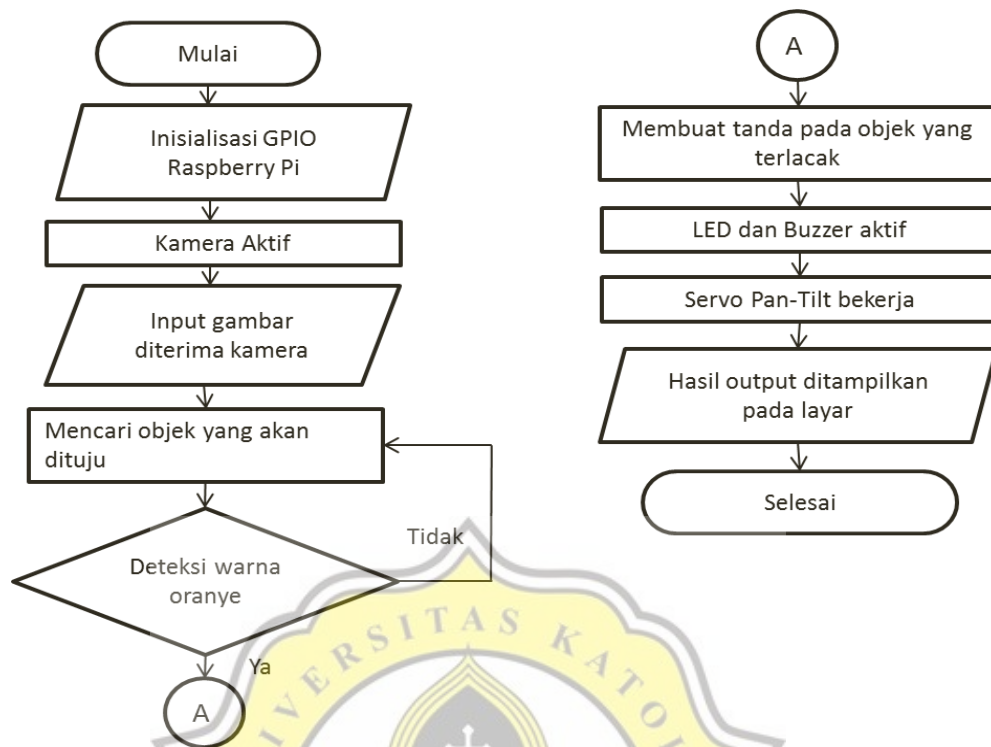




Gambar-4.8 Hasil pengujian program *color tracking*

4.3.3 Pengujian *color tracking* pada sistem *Pan-Tilt Camera*

Pengujian terakhir ini merupakan pengujian pemrograman *color tracking* yang diterapkan pada sistem *Pan-Tilt Camera*. Untuk cara kerja pemrograman *color tracking* sama seperti pengujian sebelumnya. Namun pada pengujian ini ditambahkan program keluarannya yaitu dua servo pada *Pan-Tilt* yang terdiri dari servo pertama bergerak ke kanan atau kekiri sebanyak 180 derajat dan servo kedua bergerak ke atas dan kebawah sebanyak 90 derajat menyesuaikan dengan rangka konstruksi sistem *Pan-Tilt*. Pada *output* juga diberi program untuk Lampu LED warna merah dan *buzzer* ketika ada objek yang terdeteksi ataupun tidak. Berikut pada (Gambar-4.9) merupakan *flowchart* cara kerja program yang peneliti buat.



Gambar-4.9 Flowchart cara kerja program *Pan-Tilt color tracking*

Pada program diatas, untuk menentukan pergerakan servo ditentukan dari koordinat x dan y pada gambar yang ditangkap oleh kamera. Pada pengujian program ini resolusi gambar yang ditangkap juga diatur pada 640x480p dengan 50FPS(*Frame Rate per Second*). Seperti pada pengujian sebelumnya, untuk pengujian kali ini dibutuhkan beberapa software library yang akan diinput ke pemrograman pan-tilt color tracking seperti di bawah ini.

```

import cv2
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import RPi.GPIO as GPIO
import pigpio
import argparse
from time import sleep
from numpy import interp
  
```

Library yang dibutuhkan pada pemrograman kali ini tidak jauh berbeda dengan pemrograman pada pengujian sebelumnya namun pada kali ini dilakukan penambahan `import RPi.GPIO as GPIO` untuk mengaktifkan GPIO pada Raspberry dan `import pigpio` sebagai *library* bantu untuk program pergerakan servo. Setelah itu, dilakukan pendeklarasian *input* dan *output* pada GPIO seperti pada kutipan program dibawah ini.

```
#define Servos GPIOs
panServo = 22
tiltServo = 27

# initialize LED GPIO
redLed = 17
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(redLed, GPIO.OUT)

# initialize buzzer GPIO
buzzer = 18
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(buzzer, GPIO.OUT)
```

Untuk mendeklarasikan beberapa hardware seperti servo, lampu LED dan *buzzer*, maka variabel setiap *hardware* harus ditentukan terlebih dahulu begitu juga dengan port GPIO yang akan digunakan seperti `panServo = 22` , `tiltServo = 27` untuk dua buah servo, kemudian `redLed = 17` sebagai lampu LED dan `buzzer = 18` sebagai *Buzzer* atau alarm. Setelah deklarasi untuk *port* GPIO, selanjutnya dilakukan pengaktifan library *pigpio* untuk menggerakkan servo.

```
MIN_PW = 500
MID_PW = 1500
MAX_PW = 2500

panPos = 1500
tiltPos = 1000

servo = pigpio.Pi()
servo.set_servo_pulsewidth(panServo, panPos)
servo.set_servo_pulsewidth(tiltServo, tiltPos)
```



```
minMov = 10
maxMov = 60
```

Sebelum mengaktifkan *pigpio* maka harus ditentukan terlebih dahulu minimal dan maksimal sinyal PWM untuk pergerakan servo yang diatur menggunakan *MIN_PW* yang merupakan titik minimal, *MID_PW* merupakan titik tengah, dan *MAX_PW* merupakan titik maksimal. Lebar pulsa servo biasanya berkisar antara 500 hingga 2500 titik maksimalnya. Untuk pergerakan awal servo ketika program diaktifkan diatur pada variabel *panPos = 1500* untuk servo kanan-kiri dan *tiltPos = 1000* untuk servo atas-bawah. Untuk menggerakkan servo menggunakan *pigpio*, maka harus diawali dengan perintah *servo = pigpio.Pi()* kemudian dilanjutkan dengan perintah bawaan *pigpio* yaitu *set_servo_pulsewidth* untuk mengirimkan sinyal PWM untuk setiap variabel dari servo yang sudah ditentukan. Setelah itu dilakukan penentuan variabel dan lebar pulsa PWM untuk menentukan derajat pergerakan servo. Pada program ini menggunakan variabel *minMov = 10* sebagai pergerakan minimal dan *maxMov = 60* sebagai pergerakan maksimal. Semakin kecil lebar pulsa maka pergerakan servo semakin lambat namun lebih halus namun sebaliknya dengan semakin besar maka pergerakan lebih cepat namun lebih kasar. Penentuan kecepatan minimal dan maksimal ini bergantung pada mekanisme *pan-tilt* dan servo yang digunakan. Kutipan baris program selanjutnya merupakan penentuan warna objek dan pengaktifan kamera seperti di bawah ini.

```
# define the lower and upper boundaries of the object
# to be tracked in the HSV color space
colorLower = (10, 100, 100)
colorUpper = (30, 255, 255)
```

```

# initialize the video stream and allow the Camera sensor to
  warmup
vs = VideoStream(0).start()
time.sleep(2.0)
cod = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('detec1.avi', cod, 100.0, (640,480))

```

Untuk perintah diatas merupakan penentuan format warna RGB pada objek yang akan dideteksi dan pengaktifan kamera yang sama seperti pada program *color tracking* pada pengujian sebelumnya. Pada pemrograman kali ini ditambahkan fitur untuk menyimpan rekaman video menggunakan `cv2.VideoWriter` dari *OpenCV* yang disertai dengan nama file, lama waktu perekaman dan ukuran resolusi yang diinginkan. Setelah ini akan dijelaskan beberapa kutipan program untuk mematikan lampu LED dan *Buzzer* menggunakan *GPIO Raspberry Pi*.

```

# Start with LED off
GPIO.output(redLed, GPIO.LOW)
ledOn = False
# Start with buzzer off
GPIO.output(buzzer, GPIO.LOW)
buzzerOn = False

```

Ketika program dijalankan, maka lampu LED dan *buzzer* dipastikan tidak menyala terlebih dahulu. Untuk memastikan *hardware* semisal Lampu LED pada program ini yang terhubung dengan GPIO tidak aktif, maka menggunakan perintah `GPIO.output(redLed, GPIO.LOW)` dilanjutkan dengan variabel `ledOn = False`. Setelah ini akan dijelaskan beberapa baris program pergerakan servo yang disesuaikan ketika mengikuti objek.

```

def movePanTilt(x, y, w, h):
    global panPos
    global tiltPos
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0),
        2)
    if int(x+(w/2)) > 360:
        panPos = int(panPos - interp(int(x+(w/2)), (360,
            640), (minMov, maxMov)))

```

```

elif int(x+(w/2)) < 280:
    panPos = int(panPos + interp(int(x+(w/2)), (280,
        0), (minMov, maxMov)))
if int(y+(h/2)) > 280:
    tiltPos = int(tiltPos + interp(int(y+(h/2)),
        (280, 480), (minMov, maxMov)))
elif int(y+(h/2)) < 200:
    tiltPos = int(tiltPos - interp(int(y+(h/2)),
        (200, 0), (minMov, maxMov)))
if not panPos < 2500 or panPos > 500:
    servo.set_servo_pulsewidth(panServo, panPos)
if not tiltPos < 2500 or tiltPos > 500:
    servo.set_servo_pulsewidth(tiltServo, tiltPos)

```

Agar pergerakan servo dibuat supaya dapat mengikuti objek sesuai dengan koordinat x dan y pada kamera maka harus dibuat terlebih dahulu penanda berupa persegi ketika objek yang terdeteksi. Untuk membuat persegi berwarna hijau sebagai penanda menggunakan format x, y, w, h pada *OpenCV* yang perintahnya yaitu, `cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)`. Agar servo dapat bergerak sesuai dengan titik koordinat kamera, maka prinsip kerja yang digunakan yaitu ketika titik koordinat objek terfokus di tengah maka kedua servo mati. Sedangkan, ketika titik koordinat x melebihi 360, maka servo bergerak ke kanan perintahnya menggunakan `if int(x+(w/2)) > 360: panPos = int(panPos - interp(int(x+(w/2)), (360, 640), (minMov, maxMov)))` sebaliknya ketika koordinat kurang dari 280, maka servo bergerak ke kiri perintahnya menggunakan `elif int(x+(w/2)) < 280: panPos = int(panPos + interp(int(x+(w/2)), (280, 0), (minMov, maxMov)))`. Untuk Kecepatan pergerakan servo sudah diatur dari variabel `minMov` dan `maxMov` yang sudah dibahas sebelumnya. Setelah program untuk pergerakan servo horizontal atau koordinat x sudah diterapkan, begitu pula dengan prinsip kerja program pergerakan servo ke atas dan ke bawah atau koordinat y . Ketika titik koordinat y pada objek melebihi 280 maka servo akan bergerak ke bawah dan menggunakan

perintah `if int(y+(h/2)) > 280: tiltPos = int(tiltPos + interp(int(y+(h/2)), (280, 480), (minMov, maxMov)))`. Sebaliknya ketika titik koordinat y kurang dari 200 maka servo akan bergerak ke atas dan menggunakan perintah `elif int(y+(h/2)) < 200: tiltPos = int(tiltPos - interp(int(y+(h/2)), (200, 0), (minMov, maxMov)))`. Dengan menggunakan prinsip kerja program x dan y diatas maka servo beserta mekanisme pan-tiltnya akan terus bergerak mengikuti objek yang terdeteksi oleh kamera. Program servo diatas hanya sesuai untuk resolusi 640x480p, sedangkan ketika resolusinya dirubah maka titik koordinatnya juga harus dirubah kembali. Setelah itu pergerakan kedua servo juga dapat dibatasi derajatnya menggunakan perintah `if not panPos < 2500 or panPos > 500: servo.set_servo_pulsewidth(panServo, panPos)` untuk servo gerakan kanan-kiri, kemudian `if not tiltPos < 2500 or tiltPos > 500: servo.set_servo_pulsewidth(tiltServo, tiltPos)` untuk servo gerakan atas-bawah. Setelah, program pergerakan servo, dilanjutkan dengan program untuk mengaktifkan kamera dan pengaturan resolusi kameranya.

```
while True:
    # grab the next frame from the video stream, Invert 0o,
    # resize the
    # frame, and convert it to the HSV color space
    frame = vs.read()
    frame = imutils.resize(frame, width=640)
    frame = imutils.rotate(frame, angle=0)
```

Untuk beberapa baris program diatas tidak jauh berbeda dengan pengujian sebelumnya, diawali dengan pengaktifan kamera dengan resolusi 640x480p dengan perintah `frame = vs.read()` sebagai pembacaan kameranya kemudian perintah `frame = imutils.resize(frame, width=640)` sebagai

pengaturan resolusinya dan perintah `frame = imutils.rotate(frame, angle=0)` sebagai pengaturan derajat tampilannya. Setelah ini akan dibahas beberapa baris program mengenai pengenalan warna pada objek.

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
# construct a mask for the object color, then perform
# a series of dilations and erosions to remove any small
# blobs left in the mask
mask = cv2.inRange(hsv, colorLower, colorUpper)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)
# find contours in the mask and initialize the current
# (x, y) center of the object
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                        cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if imutils.is_cv2() else cnts[1]
```

Berberapa baris program diatas menjelaskan bagaimana ketika pengenalan warna pada objek agar dapat sesuai dengan jarak warna yang sudah ditentukan sebelumnya menggunakan `mask = cv2.inRange(hsv, colorLower, colorUpper)` dan untuk menentukan ukuran objek yang terdeteksi oleh kamera agar tidak mengalami *noise* yang besar menggunakan `mask = cv2.erode(mask, None, iterations=2)` dan penentuan lebarnya menggunakan `mask = cv2.dilate(mask, None, iterations=2)`. Setelah itu dilakukan pendeteksian tepi pada objek agar lebih akurat menggunakan `cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`. Kemudian agar program dapat mengenali bagaimana tepi objek dapat terdeteksi oleh kamera ataupun tidak menggunakan `cnts = cnts[0] if imutils.is_cv2() else cnts[1]`. Setelah ini, akan dibahas mengenai program

```
max_area = 0
best_cnt = 1
if len(cnts) > 0:
    for Pic, contour in enumerate(cnts):
        area = cv2.contourArea(contour)
```

```

if (area > max_area) :
    max_area = area
    best_cnt = contour
    (x, y, w, h) = cv2.boundingRect(best_cnt)
    movePanTilt(x, y, w, h)
    # if the buzzer is not already on, turn the
    buzzer on
    if not buzzerOn:
        GPIO.output(buzzer, GPIO.HIGH)
        buzzerOn = True
    # if the led is not already on, turn the LED
    on
    if not ledOn:
        GPIO.output(redLed, GPIO.HIGH)
        ledOn = True
# if the ball is not detected, turn the buzzer off
elif buzzerOn:
    GPIO.output(buzzer, GPIO.LOW)
    buzzerOn = False
# if the ball is not detected, turn the LED off
elif ledOn:
    GPIO.output(redLed, GPIO.LOW)
    ledOn = False

```

Kutipan program di atas merupakan pembuatan tanda pada display dan mengatur pergerakan servo ketika objek dapat terdeteksi ataupun tidak. Sebelum itu, agar program dapat memfokuskan pada satu objek berwarna saja, maka warna pada objek yang terdeteksi oleh kamera harus yang paling sesuai dengan format yang ditentukan menggunakan `max_area = 0` dan `best_cnt = 1`. Untuk mengetahui apakah objek sudah terdeteksi atau tidak, maka diperlukan penanda dalam bentuk persegi berwarna menggunakan fitur dari OpenCV yaitu `contourArea` dan `boundingRect` menggunakan vektor `x, y, w, h` seperti pada kutipan program di atas. Setelah itu penanda persegi yang dibuat juga diberi warna dalam format RGB seperti `(0, 255, 0)` yaitu warna hijau. Ketika penanda dalam bentuk persegi sudah terbentuk, maka dilanjutkan perintah `movePanTilt(x, y, w, h)` untuk memulai program pergerakan servo ketika penanda objek sudah terbentuk atau objek sudah terdeteksi. Selain, menjalankan program pergerakan servo, Lampu LED dan *Buzzer* juga akan bekerja ketika

objek terdeteksi menggunakan perintah `if not buzzerOn`
`GPIO.output(buzzer, GPIO.HIGH) buzzerOn = True` begitu juga dengan
lampu led menggunakan perintah `if not ledOn: GPIO.output(redLed,`
`GPIO.HIGH) ledOn = True` . Selanjutnya ketika objek tidak terdeteksi, maka
perintah untuk mematikan lampu led dan buzzer menggunakan `elif buzzerOn:`
`GPIO.output(buzzer, GPIO.LOW) buzzerOn = False` . Begitu juga dengan
perintah lampu LED yaitu `elif ledOn: GPIO.output(redLed, GPIO.LOW)`
`ledOn = False`. Berikut beberapa baris program selanjutnya untuk mematikan
program yang sedang berjalan.

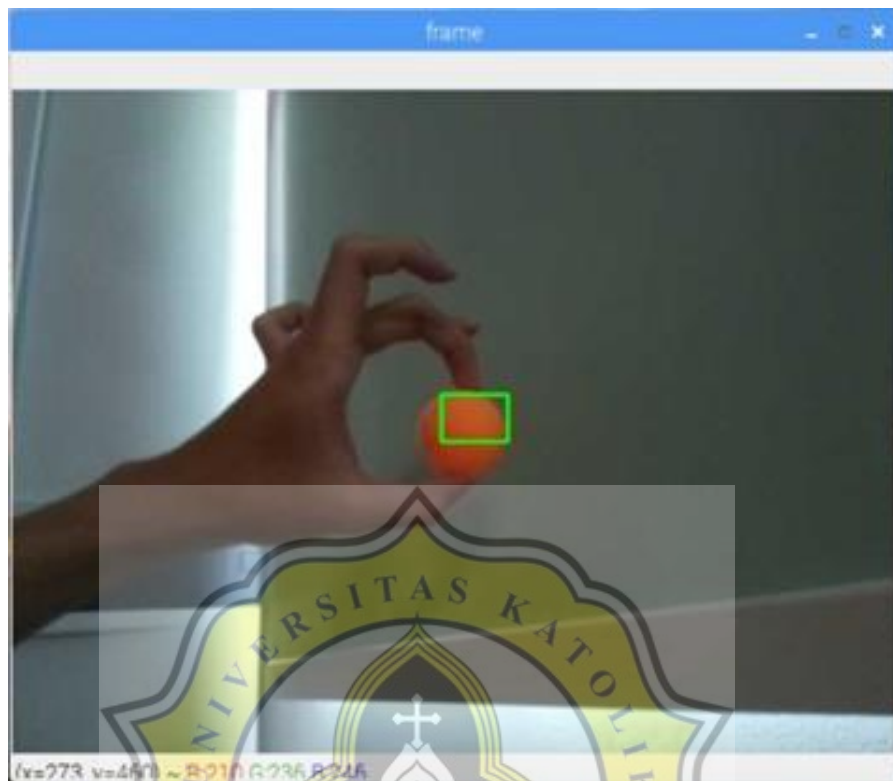
```

cv2.imshow('frame', frame)
out.write(frame)
key = cv2.waitKey(1) & 0xFF
if key == ord("q"):
    break

servo.set_servo_pulsewidth(22, 0)
servo.set_servo_pulsewidth(27, 0)
servo.stop()
GPIO.cleanup()
out.release()
cv2.destroyAllWindows()

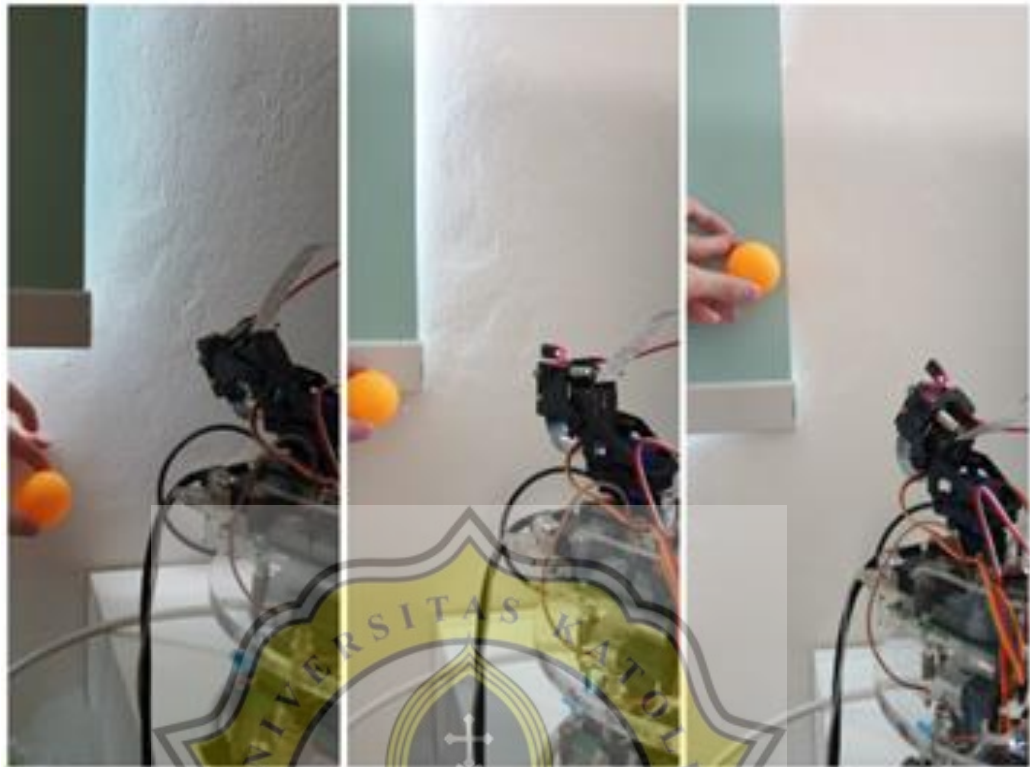
```

Kutipan program diatas merupakan beberapa baris terakhir dari program yang berfungsi untuk menghentikan atau mematikan program ketika sudah berjalan. Pada program diatas cara menghentikan kerja programnya menggunakan tombol “q” pada keyboard, sehingga ketika tombol “q” ditekan maka output yang terhubung pada GPIO beserta kamera dan tampilan displaynya juga akan mati menggunakan `cv2.destroyAllWindows()` dan `vs.stop()` pada program yang dibuat. Setelah program sudah dipastikan berjalan dengan baik, maka ketika program dijalankan akan muncul tampilan seperti pada (Gambar-4.10) di bawah ini.



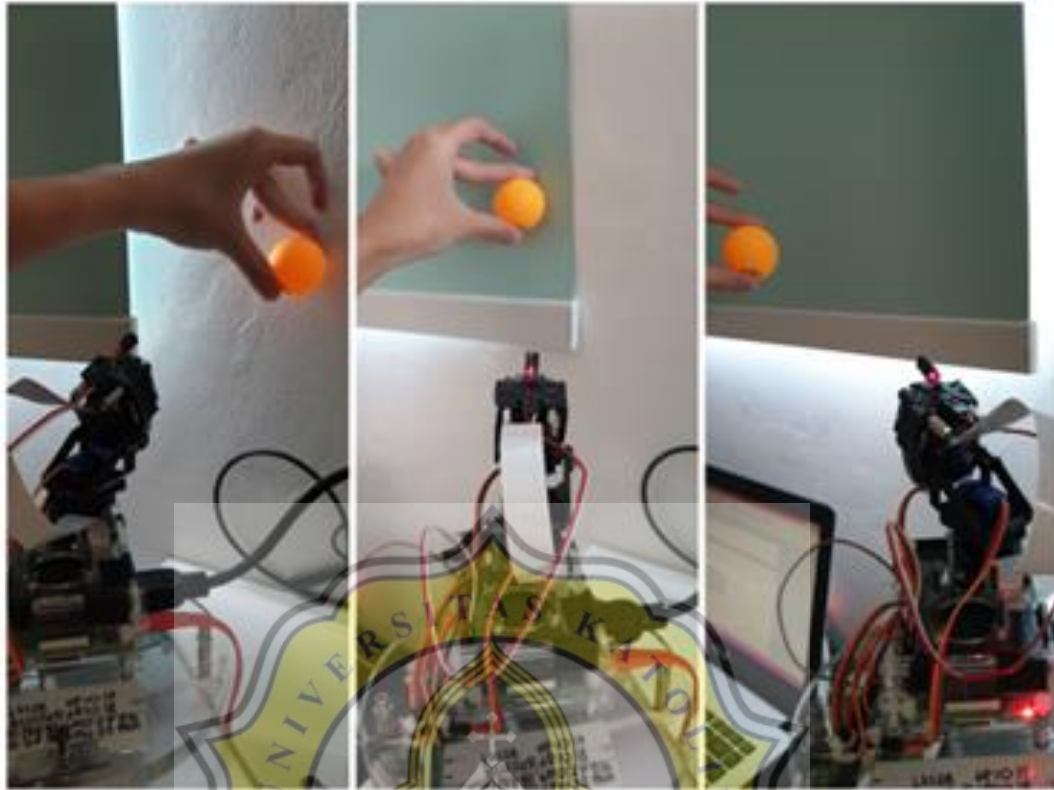
Gambar-4.10 *Display* yang ditangkap kamera dengan resolusi 640x480p

Seperti pada gambar diatas, objek tepat terdeteksi di tengah yaitu ($x=273$ dan $y=460$) seperti yang tertera pada layar. Dalam posisi di tengah kedua servo tidak bergerak. Ketika objek mulai bergerak kearah atas maupun ke bawah maka posisi y pada objek yang tertampil pada gambar akan berubah dan servo *tilt* akan terus bergerak memposisikan agar objek selalu di tengah seperti pada (Gambar-4.10) sebelumnya. Berikut pada (Gambar-4.11) merupakan pergerakan servo *tilt* ketika sedang melacak objek.



Gambar-4.11 Pergerakan pada servo *tilt*(atas-bawah)

Setelah melakukan pengujian pada servo *tilt*, dilanjutkan dengan pengujian servo *pan* yaitu servo yang bergerak kearah kanan dan kiri. Begitu juga dengan cara kerja servo *tilt* sebelumnya, ketika objek yang dilacak mulai bergerak kearah kanan ataupun ke kiri, maka servo *pan* juga akan mengikuti agar posisi x pada objek selalu di tengah seperti pada (Gambar-4.11) sebelumnya. Berikut pada (Gambar-4.12) merupakan pergerakan servo *pan* ketika sedang melacak objek.



Gambar-4.12 Pergerakan pada servo *pan*(kanan-kiri)

4.4. Pembahasan

Berdasarkan pada pengujian yang telah dilakukan, dapat dilihat bahwa pendeteksian suatu objek menggunakan format warna menggunakan *library OpenCV* memiliki kelebihan yaitu pendeteksian pada objek tertentu tidak bergantung pada ukuran objek tersebut, karena selama warna pada objek dapat terdeteksi, maka program akan memberi penanda pada objek . Pada jarak yang jauh sekalipun asalkan kamera masih menangkap warna pada objek tersebut walaupun kecil atau jauh, maka objek masih bisa terdeteksi. Sedangkan kekurangannya yaitu karena hanya mengandalkan warna untuk mendeteksi suatu objek, maka ketika ada objek lain yang berwarna mendekati atau sama maka

objek yang lain tersebut juga akan ikut terdeteksi. Yang kedua, pada *library OpenCV*, pendeteksian warna kurang akurat sehingga ketika peneliti ingin mendeteksi warna *orange*, maka warna yang mendekati warna *orange* seperti kuning ataupun coklat terang juga ikut terdeteksi seperti pada (Gambar-4.13).



(b)

Gambar-4.13 Hasil Pengujian objek warna lain (a) kuning dan (b) coklat

Intensitas cahaya juga sangat berpengaruh pada tingkat RGB pada warna seperti pada (Gambar-4.7) sehingga ketika kita ingin mencari data warna RGB suatu objek, maka diusahakan harus pada intensitas cahaya di ruangan yang sesuai di mana foto objek awal diambil seperti (Gambar-4.3).

Pengujian yang terakhir, yaitu pada sistem *Pan-Tilt* membuktikan bahwa pengendalian servo berdasarkan koordinat x dan y pada objek yang ditangkap kamera dapat bekerja dengan akurat karena servo terus bergerak hingga berhenti pada titik tengah dan y pada *display* yang ditangkap kamera seperti pada (Gambar-4.10). Namun pengujian ini memiliki kendala dimana kedua servo pada awalnya terdapat *jitter* sehingga membutuhkan *library* atau *software* bantu yang bernama *PigPi* pada (Gambar-3.8) untuk mengatasi error tersebut apabila Pin PWM pada servo dicolokkan langsung ke Pin GPIO pada *Raspberry Pi*.