1. Program Pengolahan Citra pada Raspberry

```
# import library
from collections import deque
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
import numpy as np
import argparse
import imutils
import cv2
import serial

#mengaktifkan komunikasi serial ke arduino
arduino = serial.Serial('/dev/ttyS0',baudrate=9600,timeout=3.0)

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video",
	help="path to the (optional) video file")
ap.add_argument("-b", "--buffer", type=int, default=32,
	help="max buffer size")
args = vars(ap.parse_args())

# define the lower and upper boundaries of the "green"
# ball in the HSV color space
greenLower = (25, 74, 6)
```

```
greenUpper = (64, 255, 255)
pts = deque(maxlen=args["buffer"])
counter = 0
(dX, dY) = (0, 0)
direction = ""

# define the lower and upper boundaries of the "green"
# ball in the HSV color space
redLower = (170, 50, 50)
redUpper = (180, 255, 255)
pts = deque(maxlen=args["buffer"])
counter = 0
(dX1, dY1) = (0, 0)
direction1 = ""


# if a video path was not supplied, grab the reference
# to the webcam
if not args.get("video", False):
	camera = PiCamera()
	camera.resolution = (320, 240)
	camera.framerate = 35
	rawCapture = PiRGBArray(camera, size=(320, 240))
	# allow the camera to warmup
	time.sleep(0.1)


# otherwise, grab a reference to the video file
```

```
else:
    camera = cv2.VideoCapture(args["video"])


# keep looping
#while True:
# capture frames from the camera
for image in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
    #grab the raw NumPy array representing the image, then initialize the timestamp
and occupied/unoccupied text
    frame = image.array


    # resize the frame, blur it, and convert it to the HSV
    # color space
    # frame = imutils.resize(frame, width=640)
    # blurred = cv2.GaussianBlur(frame, (11, 11), 0)
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # show the frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)
    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break


        # construct a mask for the color "green", then perform
        # a series of dilations and erosions to remove any small
        # blobs left in the mask
```

```
	red = cv2.inRange(hsv, redLower, redUpper)
	red = cv2.erode(red, None, iterations=2)
	red = cv2.dilate(red, None, iterations=2)

	# construct a mask for the color "green", then perform
		# a series of dilations and erosions to remove any small
		# blobs left in the mask
	green = cv2.inRange(hsv, greenLower, greenUpper)
	green = cv2.erode(green, None, iterations=2)
	green = cv2.dilate(green, None, iterations=2)

		# find contours in the mask and initialize the current
		# (x, y) center of the ball
	cnts = cv2.findContours(red.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]
	center = None

	cnts1 = cv2.findContours(green.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]
	center = None

		# only proceed if at least one contour was found
	if len(cnts) > 0:
			# find the largest contour in the mask, then use
			# it to compute the minimum enclosing circle and
			# centroid
		c = max(cnts, key=cv2.contourArea)
		((x, y), radius) = cv2.minEnclosingCircle(c)
		M = cv2.moments(c)
```

```
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))


                    # only proceed if the radius meets a minimum size
            if radius > 10:
                            # draw the circle and centroid on the frame,
                            # then update the list of tracked points
                cv2.circle(frame, (int(x), int(y)), int(radius),
                        (0, 255, 255), 2)
                cv2.circle(frame, center, 5, (0, 0, 255), -1)
                pts.appendleft(center)
                print(int(x), int(y))


                if int(x) >= 5 and int(y) >= 60 :
                    arduino.write('L'.encode('ascii'))
                    print('belok ke kanan')
                else :
                    arduino.write('F'.encode('ascii'))
                    print('lurus')


            # show the movement deltas and the direction of movement on
            # the frame
                    cv2.putText(frame, direction, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.65,
(0, 0, 255), 3)
                    cv2.putText(frame, "X, Y: {}".format(center), (10, frame.shape[0]- 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)


        # only proceed if at least one contour was found
        if len(cnts1) > 0:
                    # find the largest contour in the mask, then use
```

```
            # it to compute the minimum enclosing circle and
            # centroid
        c = max(cnts1, key=cv2.contourArea)
        ((x1, y1), radius1) = cv2.minEnclosingCircle(c)
        M1 = cv2.moments(c)
        center1 = (int(M1["m10"] / M1["m00"]), int(M1["m01"] / M1["m00"]))

            # only proceed if the radius meets a minimum size
        if radius1 > 10:
                    # draw the circle and centroid on the frame,
                    # then update the list of tracked points
            cv2.circle(frame, (int(x1), int(y1)), int(radius1),
                (0, 255, 255), 2)
            cv2.circle(frame, center1, 5, (0, 0, 255), -1)
            pts.appendleft(center1)
            print(int(x1), int(y1))

            if int(x1) <= 200 and int(y1) >= 60:
                arduino.write('R'.encode('ascii'))
                print('belok ke kiri')
            else :
                arduino.write('F'.encode('ascii'))
                print('lurus')

        # show the movement deltas and the direction of movement on
        # the frame
            cv2.putText(frame, direction1, (130, 30), cv2.FONT_HERSHEY_SIMPLEX,
0.65, (0, 0, 255), 3)
```

```
                cv2.putText(frame, "X1, Y1: {}".format(center1), (130, frame.shape[0]- 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)



        #menampilkan frame pada layar dan increment the frame counter
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    counter += 1

    #menampilkan image threshold bola hijau
    cv2.imshow("threshold hijau",green)
    key = cv2.waitKey(1) & 0xFF
    counter += 1

    #menampilkan image threshold bola merah
    cv2.imshow("threshold merah", red)
    key = cv2.waitKey(1) & 0xFF
    counter += 1

    #jika tombol q pada keyboard ditekan maka proses berhenti
    if key == ord("q"):
        break



# menutup kamera dan beberapa jendela yang masih terbuka
camera.close()
cv2.destroyAllWindows()
```

2. Program untuk Menggerakan Motor dan Servo pada Mikrokontroler

```
#include <Servo.h>
Servo finsservo, brushless; //devinisi servo & devinisi motor brushless
int start = 2; //tombol untuk start
int awalstart = 0; //kondisi awal start

void setup()
{
  finsservo.attach(8); //servo pada pin 8
  brushless.attach(9); //brushless pada pin 9
  pinMode(start, INPUT); //tombol start masukan
  Serial.begin(9600); //komunikasi serial
}

void loop()
{
  awalstart = digitalRead(start);

  if (awalstart == HIGH)
  {
  brushless.write(40);
  delay(1000);
  finsservo.write(90);
  }

  else
```

```
 {
  if (Serial.available()>0)
  {
   int baca=Serial.read();

   if(baca=='R')
   {
    brushless.write(65);
    finsservo.write(60);
    Serial.println("Motor Servo Bergerak ke kiri");
   }

   else if(baca=='L')
   {
    brushless.write(65);
    finsservo.write(135);
    Serial.println("Motor Servo Bergerak ke Kanan");
   }

   else
   {
    finsservo.write(90);
    brushless.write(65);
    Serial.println("Motor Servo Lurus");
   }
  }
 }
}
```