**Practical Assignment**

**Objective: - Hand Gesture Recognition**

In this project, you are going to determine the gesture of the hand in real-time using a webcam.

First, the background is separated from the hand region and then the fingers are segmented to predict hand gestures.

With different hand gestures, you can perform different actions.

**Dataset Link: -**

Use anyone of your choice.

**Task: -** Create a Desktop App for real time detection where the user can show hand gestures and identify them.

**Deployment: -** Any Free Platform(Try to look out for free options.)

**Assignment Submission: -** Only submit the hosted app link. OR GitHub Link

**Python Hand Gesture Recognition (Checkpoint 1)**

This checkpoint is at the end of Objective 1.

At this point, your code should be able to get input from the camera and display it on the screen as if it were a mirror. It should also draw a rectangle where the user should put their hand.

**Header: Importing libraries and creating global variables**

**import** numpy **as** np

**import** cv2

*# Hold the background frame for background subtraction.*

background **=** **None**

*# Hold the hand's data so all its details are in one place.*

hand **=** **None**

*# Variables to count how many frames have passed and to set the size of the window.*

frames\_elapsed **=** 0

FRAME\_HEIGHT **=** 200

FRAME\_WIDTH **=** 300

*# Humans come in a ton of beautiful shades and colors.*

*# Try editing these if your program has trouble recognizing your skin tone.*

CALIBRATION\_TIME **=** 30

BG\_WEIGHT **=** 0.5

OBJ\_THRESHOLD **=** 18

**Main function: Get input from camera and call functions to understand it**

*# Our region of interest will be the top right part of the frame.*

region\_top **=** 0

region\_bottom **=** int(2 **\*** FRAME\_HEIGHT **/** 3)

region\_left **=** int(FRAME\_WIDTH **/** 2)

region\_right **=** FRAME\_WIDTH

frames\_elapsed **=** 0

capture **=** cv2**.**VideoCapture(0)

**while** (**True**):

*# Store the frame from the video capture and resize it to the window size.*

ret, frame **=** capture**.**read()

frame **=** cv2**.**resize(frame, (FRAME\_WIDTH, FRAME\_HEIGHT))

*# Flip the frame over the vertical axis so that it works like a mirror, which is more intuitive to the user.*

frame **=** cv2**.**flip(frame, 1)

*# Show the previously captured frame.*

cv2**.**imshow("Camera Input", frame)

frames\_elapsed **+=** 1

*# Check if user wants to exit.*

**if** (cv2**.**waitKey(1) **&** 0xFF **==** ord('x')):

**break**

*# When we exit the loop, we have to stop the capture too.*

capture**.**release()

cv2**.**destroyAllWindows()

**Python Hand Gesture Recognition (Checkpoint 2)**

This checkpoint is at the end of Objective 2.

At this point, your code should be able to get input from the camera and display it on the screen as if it were a mirror. It should also draw a rectangle where the user should put their hand.

**Header: Importing libraries and creating global variables**

**import** numpy **as** np

**import** cv2

*# Hold the background frame for background subtraction.*

background **=** **None**

*# Hold the hand's data so all its details are in one place.*

hand **=** **None**

*# Variables to count how many frames have passed and to set the size of the window.*

frames\_elapsed **=** 0

FRAME\_HEIGHT **=** 200

FRAME\_WIDTH **=** 300

*# Humans come in a ton of beautiful shades and colors.*

*# Try editing these if your program has trouble recognizing your skin tone.*

CALIBRATION\_TIME **=** 30

BG\_WEIGHT **=** 0.5

OBJ\_THRESHOLD **=** 18

**HandData: A class to hold all the hand's details and flags**

**class** HandData:

top **=** (0,0)

bottom **=** (0,0)

left **=** (0,0)

right **=** (0,0)

centerX **=** 0

prevCenterX **=** 0

isInFrame **=** **False**

isWaving **=** **False**

fingers **=** **None**

**def** \_\_init\_\_(self, top, bottom, left, right, centerX):

self**.**top **=** top

self**.**bottom **=** bottom

self**.**left **=** left

self**.**right **=** right

self**.**centerX **=** centerX

self**.**prevCenterX **=** 0

isInFrame **=** **False**

isWaving **=** **False**

**def** update(self, top, bottom, left, right):

self**.**top **=** top

self**.**bottom **=** bottom

self**.**left **=** left

self**.**right **=** right

**write\_on\_image(): Write info related to the hand gesture and outline the region of interest**

*# Here we take the current frame, the number of frames elapsed, and how many fingers we've detected*

*# so we can print on the screen which gesture is happening (or if the camera is calibrating).*

**def** write\_on\_image(frame):

text **=** "Searching..."

**if** frames\_elapsed **<** CALIBRATION\_TIME:

text **=** "Calibrating..."

**elif** hand **==** **None** **or** hand**.**isInFrame **==** **False**:

text **=** "No hand detected"

**else**:

**if** hand**.**isWaving:

text **=** "Waving"

**elif** hand**.**fingers **==** 0:

text **=** "Rock"

**elif** hand**.**fingers **==** 1:

text **=** "Pointing"

**elif** hand**.**fingers **==** 2:

text **=** "Scissors"

cv2**.**putText(frame, text, (10,20), cv2**.**FONT\_HERSHEY\_COMPLEX, 0.4,( 0 , 0 , 0 ),2,cv2**.**LINE\_AA)

cv2**.**putText(frame, text, (10,20), cv2**.**FONT\_HERSHEY\_COMPLEX, 0.4,(255,255,255),1,cv2**.**LINE\_AA)

*# Highlight the region of interest.*

cv2**.**rectangle(frame, (region\_left, region\_top), (region\_right, region\_bottom), (255,255,255), 2)

**Main function: Get input from camera and call functions to understand it**

*# Our region of interest will be the top right part of the frame.*

region\_top **=** 0

region\_bottom **=** int(2 **\*** FRAME\_HEIGHT **/** 3)

region\_left **=** int(FRAME\_WIDTH **/** 2)

region\_right **=** FRAME\_WIDTH

frames\_elapsed **=** 0

capture **=** cv2**.**VideoCapture(0)

**while** (**True**):

*# Store the frame from the video capture and resize it to the window size.*

ret, frame **=** capture**.**read()

frame **=** cv2**.**resize(frame, (FRAME\_WIDTH, FRAME\_HEIGHT))

*# Flip the frame over the vertical axis so that it works like a mirror, which is more intuitive to the user.*

frame **=** cv2**.**flip(frame, 1)

*# Write the action the hand is doing on the screen, and draw the region of interest.*

write\_on\_image(frame)

*# Show the previously captured frame.*

cv2**.**imshow("Camera Input", frame)

frames\_elapsed **+=** 1

*# Check if user wants to exit.*

**if** (cv2**.**waitKey(1) **&** 0xFF **==** ord('x')):

**break**

*# When we exit the loop, we have to stop the capture too.*

capture**.**release()

cv2**.**destroyAllWindows()

**Python Hand Gesture Recognition (Checkpoint 3)**

This checkpoint is at the end of Objective 3.

At this point, your code should be able to average the backgrounds, segmented the current frame to separate the object from the region of interest, and display it for viewing.

**Header: Importing libraries and creating global variables**

**import** numpy **as** np

**import** cv2

*# Hold the background frame for background subtraction.*

background **=** **None**

*# Hold the hand's data so all its details are in one place.*

hand **=** **None**

*# Variables to count how many frames have passed and to set the size of the window.*

frames\_elapsed **=** 0

FRAME\_HEIGHT **=** 200

FRAME\_WIDTH **=** 300

*# Humans come in a ton of beautiful shades and colors.*

*# Try editing these if your program has trouble recognizing your skin tone.*

CALIBRATION\_TIME **=** 30

BG\_WEIGHT **=** 0.5

OBJ\_THRESHOLD **=** 18

**HandData: A class to hold all the hand's details and flags**

**class** HandData:

top **=** (0,0)

bottom **=** (0,0)

left **=** (0,0)

right **=** (0,0)

centerX **=** 0

prevCenterX **=** 0

isInFrame **=** **False**

isWaving **=** **False**

fingers **=** **None**

**def** \_\_init\_\_(self, top, bottom, left, right, centerX):

self**.**top **=** top

self**.**bottom **=** bottom

self**.**left **=** left

self**.**right **=** right

self**.**centerX **=** centerX

self**.**prevCenterX **=** 0

isInFrame **=** **False**

isWaving **=** **False**

**def** update(self, top, bottom, left, right):

self**.**top **=** top

self**.**bottom **=** bottom

self**.**left **=** left

self**.**right **=** right

**write\_on\_image(): Write info related to the hand gesture and outline the region of interest**

*# Here we take the current frame, the number of frames elapsed, and how many fingers we've detected*

*# so we can print on the screen which gesture is happening (or if the camera is calibrating).*

**def** write\_on\_image(frame):

text **=** "Searching..."

**if** frames\_elapsed **<** CALIBRATION\_TIME:

text **=** "Calibrating..."

**elif** hand **==** **None** **or** hand**.**isInFrame **==** **False**:

text **=** "No hand detected"

**else**:

**if** hand**.**isWaving:

text **=** "Waving"

**elif** hand**.**fingers **==** 0:

text **=** "Rock"

**elif** hand**.**fingers **==** 1:

text **=** "Pointing"

**elif** hand**.**fingers **==** 2:

text **=** "Scissors"

cv2**.**putText(frame, text, (10,20), cv2**.**FONT\_HERSHEY\_COMPLEX, 0.4,( 0 , 0 , 0 ),2,cv2**.**LINE\_AA)

cv2**.**putText(frame, text, (10,20), cv2**.**FONT\_HERSHEY\_COMPLEX, 0.4,(255,255,255),1,cv2**.**LINE\_AA)

*# Highlight the region of interest.*

cv2**.**rectangle(frame, (region\_left, region\_top), (region\_right, region\_bottom), (255,255,255), 2)

**get\_region(): Separate the region of interest and preps it for edge detection**

**def** get\_region(frame):

*# Separate the region of interest from the rest of the frame.*

region **=** frame[region\_top:region\_bottom, region\_left:region\_right]

*# Make it grayscale so we can detect the edges more easily.*

region **=** cv2**.**cvtColor(region, cv2**.**COLOR\_BGR2GRAY)

*# Use a Gaussian blur to prevent frame noise from being labeled as an edge.*

region **=** cv2**.**GaussianBlur(region, (5,5), 0)

**return** region

**get\_average(): Create a weighted average of the background for image differencing**

**def** get\_average(region):

*# We have to use the global keyword because we want to edit the global variable.*

**global** background

*# If we haven't captured the background yet, make the current region the background.*

**if** background **is** **None**:

background **=** region**.**copy()**.**astype("float")

**return**

*# Otherwise, add this captured frame to the average of the backgrounds.*

cv2**.**accumulateWeighted(region, background, BG\_WEIGHT)

**segment(): Use image differencing to separate the hand from the background**

*# Here we use differencing to separate the background from the object of interest.*

**def** segment(region):

**global** hand

*# Find the absolute difference between the background and the current frame.*

diff **=** cv2**.**absdiff(background**.**astype(np**.**uint8), region)

*# Threshold that region with a strict 0 or 1 ruling so only the foreground remains.*

thresholded\_region **=** cv2**.**threshold(diff, OBJ\_THRESHOLD, 255, cv2**.**THRESH\_BINARY)[1]

*# Get the contours of the region, which will return an outline of the hand.*

(\_, contours, \_) **=** cv2**.**findContours(thresholded\_region**.**copy(), cv2**.**RETR\_EXTERNAL, cv2**.**CHAIN\_APPROX\_SIMPLE)

*# If we didn't get anything, there's no hand.*

**if** len(contours) **==** 0:

**if** hand **is** **not** **None**:

hand**.**isInFrame **=** **False**

**return**

*# Otherwise return a tuple of the filled hand (thresholded\_region), along with the outline (segmented\_region).*

**else**:

**if** hand **is** **not** **None**:

hand**.**isInFrame **=** **True**

segmented\_region **=** max(contours, key **=** cv2**.**contourArea)

**return** (thresholded\_region, segmented\_region)

**Main function: Get input from camera and call functions to understand it**

*# Our region of interest will be the top right part of the frame.*

region\_top **=** 0

region\_bottom **=** int(2 **\*** FRAME\_HEIGHT **/** 3)

region\_left **=** int(FRAME\_WIDTH **/** 2)

region\_right **=** FRAME\_WIDTH

frames\_elapsed **=** 0

capture **=** cv2**.**VideoCapture(0)

**while** (**True**):

*# Store the frame from the video capture and resize it to the window size.*

ret, frame **=** capture**.**read()

frame **=** cv2**.**resize(frame, (FRAME\_WIDTH, FRAME\_HEIGHT))

*# Flip the frame over the vertical axis so that it works like a mirror, which is more intuitive to the user.*

frame **=** cv2**.**flip(frame, 1)

*# Separate the region of interest and prep it for edge detection.*

region **=** get\_region(frame)

**if** frames\_elapsed **<** CALIBRATION\_TIME:

get\_average(region)

**else**:

region\_pair **=** segment(region)

**if** region\_pair **is** **not** **None**:

*# If we have the regions segmented successfully, show them in another window for the user.*

(thresholded\_region, segmented\_region) **=** region\_pair

cv2**.**drawContours(region, [segmented\_region], **-**1, (255, 255, 255))

cv2**.**imshow("Segmented Image", region)

*# Write the action the hand is doing on the screen, and draw the region of interest.*

write\_on\_image(frame)

*# Show the previously captured frame.*

cv2**.**imshow("Camera Input", frame)

frames\_elapsed **+=** 1

*# Check if user wants to exit.*

**if** (cv2**.**waitKey(1) **&** 0xFF **==** ord('x')):

**break**

*# When we exit the loop, we have to stop the capture too.*

capture**.**release()

cv2**.**destroyAllWindows()

**Python Hand Gesture Recognition (Checkpoint 4)**

This checkpoint is at the end of Objective 4.

At this point, your code should be able to detect waving by using a function get\_hand\_data() along with the HandData class's functions.

**Header: Importing libraries and creating global variables**

**import** numpy **as** np

**import** cv2

*# Hold the background frame for background subtraction.*

background **=** **None**

*# Hold the hand's data so all its details are in one place.*

hand **=** **None**

*# Variables to count how many frames have passed and to set the size of the window.*

frames\_elapsed **=** 0

FRAME\_HEIGHT **=** 200

FRAME\_WIDTH **=** 300

*# Humans come in a ton of beautiful shades and colors.*

*# Try editing these if your program has trouble recognizing your skin tone.*

CALIBRATION\_TIME **=** 30

BG\_WEIGHT **=** 0.5

OBJ\_THRESHOLD **=** 18

**HandData: A class to hold all the hand's details and flags**

**class** HandData:

top **=** (0,0)

bottom **=** (0,0)

left **=** (0,0)

right **=** (0,0)

centerX **=** 0

prevCenterX **=** 0

isInFrame **=** **False**

isWaving **=** **False**

fingers **=** **None**

**def** \_\_init\_\_(self, top, bottom, left, right, centerX):

self**.**top **=** top

self**.**bottom **=** bottom

self**.**left **=** left

self**.**right **=** right

self**.**centerX **=** centerX

self**.**prevCenterX **=** 0

isInFrame **=** **False**

isWaving **=** **False**

**def** update(self, top, bottom, left, right):

self**.**top **=** top

self**.**bottom **=** bottom

self**.**left **=** left

self**.**right **=** right

**def** check\_for\_waving(self, centerX):

self**.**prevCenterX **=** self**.**centerX

self**.**centerX **=** centerX

**if** abs(self**.**centerX **-** self**.**prevCenterX **>** 3):

self**.**isWaving **=** **True**

**else**:

self**.**isWaving **=** **False**

**write\_on\_image(): Write info related to the hand gesture and outline the region of interest**

*# Here we take the current frame, the number of frames elapsed, and how many fingers we've detected*

*# so we can print on the screen which gesture is happening (or if the camera is calibrating).*

**def** write\_on\_image(frame):

text **=** "Searching..."

**if** frames\_elapsed **<** CALIBRATION\_TIME:

text **=** "Calibrating..."

**elif** hand **==** **None** **or** hand**.**isInFrame **==** **False**:

text **=** "No hand detected"

**else**:

**if** hand**.**isWaving:

text **=** "Waving"

**elif** hand**.**fingers **==** 0:

text **=** "Rock"

**elif** hand**.**fingers **==** 1:

text **=** "Pointing"

**elif** hand**.**fingers **==** 2:

text **=** "Scissors"

cv2**.**putText(frame, text, (10,20), cv2**.**FONT\_HERSHEY\_COMPLEX, 0.4,( 0 , 0 , 0 ),2,cv2**.**LINE\_AA)

cv2**.**putText(frame, text, (10,20), cv2**.**FONT\_HERSHEY\_COMPLEX, 0.4,(255,255,255),1,cv2**.**LINE\_AA)

*# Highlight the region of interest.*

cv2**.**rectangle(frame, (region\_left, region\_top), (region\_right, region\_bottom), (255,255,255), 2)

**get\_region(): Separate the region of interest and preps it for edge detection**

**def** get\_region(frame):

*# Separate the region of interest from the rest of the frame.*

region **=** frame[region\_top:region\_bottom, region\_left:region\_right]

*# Make it grayscale so we can detect the edges more easily.*

region **=** cv2**.**cvtColor(region, cv2**.**COLOR\_BGR2GRAY)

*# Use a Gaussian blur to prevent frame noise from being labeled as an edge.*

region **=** cv2**.**GaussianBlur(region, (5,5), 0)

**return** region

**get\_average(): Create a weighted average of the background for image differencing**

**def** get\_average(region):

*# We have to use the global keyword because we want to edit the global variable.*

**global** background

*# If we haven't captured the background yet, make the current region the background.*

**if** background **is** **None**:

background **=** region**.**copy()**.**astype("float")

**return**

*# Otherwise, add this captured frame to the average of the backgrounds.*

cv2**.**accumulateWeighted(region, background, BG\_WEIGHT)

**segment(): Use image differencing to separate the hand from the background**

*# Here we use differencing to separate the background from the object of interest.*

**def** segment(region):

**global** hand

*# Find the absolute difference between the background and the current frame.*

diff **=** cv2**.**absdiff(background**.**astype(np**.**uint8), region)

*# Threshold that region with a strict 0 or 1 ruling so only the foreground remains.*

thresholded\_region **=** cv2**.**threshold(diff, OBJ\_THRESHOLD, 255, cv2**.**THRESH\_BINARY)[1]

*# Get the contours of the region, which will return an outline of the hand.*

(\_, contours, \_) **=** cv2**.**findContours(thresholded\_region**.**copy(), cv2**.**RETR\_EXTERNAL, cv2**.**CHAIN\_APPROX\_SIMPLE)

*# If we didn't get anything, there's no hand.*

**if** len(contours) **==** 0:

**if** hand **is** **not** **None**:

hand**.**isInFrame **=** **False**

**return**

*# Otherwise return a tuple of the filled hand (thresholded\_region), along with the outline (segmented\_region).*

**else**:

**if** hand **is** **not** **None**:

hand**.**isInFrame **=** **True**

segmented\_region **=** max(contours, key **=** cv2**.**contourArea)

**return** (thresholded\_region, segmented\_region)

**get\_hand\_data(): Find the extremities of the hand and put them in the global hand object**

**def** get\_hand\_data(thresholded\_image, segmented\_image):

**global** hand

*# Enclose the area around the extremities in a convex hull to connect all outcroppings.*

convexHull **=** cv2**.**convexHull(segmented\_image)

*# Find the extremities for the convex hull and store them as points.*

top **=** tuple(convexHull[convexHull[:, :, 1]**.**argmin()][0])

bottom **=** tuple(convexHull[convexHull[:, :, 1]**.**argmax()][0])

left **=** tuple(convexHull[convexHull[:, :, 0]**.**argmin()][0])

right **=** tuple(convexHull[convexHull[:, :, 0]**.**argmax()][0])

*# Get the center of the palm, so we can check for waving and find the fingers.*

centerX **=** int((left[0] **+** right[0]) **/** 2)

*# We put all the info into an object for handy extraction (get it? HANDy?)*

**if** hand **==** **None**:

hand **=** HandData(top, bottom, left, right, centerX)

**else**:

hand**.**update(top, bottom, left, right)

*# Only check for waving every 6 frames.*

**if** frames\_elapsed **%** 6 **==** 0:

hand**.**check\_for\_waving(centerX)

**Main function: Get input from camera and call functions to understand it**

*# Our region of interest will be the top right part of the frame.*

region\_top **=** 0

region\_bottom **=** int(2 **\*** FRAME\_HEIGHT **/** 3)

region\_left **=** int(FRAME\_WIDTH **/** 2)

region\_right **=** FRAME\_WIDTH

frames\_elapsed **=** 0

capture **=** cv2**.**VideoCapture(0)

**while** (**True**):

*# Store the frame from the video capture and resize it to the window size.*

ret, frame **=** capture**.**read()

frame **=** cv2**.**resize(frame, (FRAME\_WIDTH, FRAME\_HEIGHT))

*# Flip the frame over the vertical axis so that it works like a mirror, which is more intuitive to the user.*

frame **=** cv2**.**flip(frame, 1)

*# Separate the region of interest and prep it for edge detection.*

region **=** get\_region(frame)

**if** frames\_elapsed **<** CALIBRATION\_TIME:

get\_average(region)

**else**:

region\_pair **=** segment(region)

**if** region\_pair **is** **not** **None**:

*# If we have the regions segmented successfully, show them in another window for the user.*

(thresholded\_region, segmented\_region) **=** region\_pair

cv2**.**drawContours(region, [segmented\_region], **-**1, (255, 255, 255))

cv2**.**imshow("Segmented Image", region)

get\_hand\_data(thresholded\_region, segmented\_region)

*# Write the action the hand is doing on the screen, and draw the region of interest.*

write\_on\_image(frame)

*# Show the previously captured frame.*

cv2**.**imshow("Camera Input", frame)

frames\_elapsed **+=** 1

*# Check if user wants to exit.*

**if** (cv2**.**waitKey(1) **&** 0xFF **==** ord('x')):

**break**

**if** (cv2**.**waitKey(1) **&** 0xFF **==** ord('r')):

frames\_elapsed **=** 0

*# When we exit the loop, we have to stop the capture too.*

capture**.**release()

cv2**.**destroyAllWindows()

**Python Hand Gesture Recognition (Checkpoint 5)**

This checkpoint is at the end of Objective 5.

At this point, your code should be fully functional -- it'll recognize waving, pointing, a peace sign (scissors), and a fist (rock). Congrats on making it here!

**Header: Importing libraries and creating global variables**

**import** numpy **as** np

**import** cv2

*# Hold the background frame for background subtraction.*

background **=** **None**

*# Hold the hand's data so all its details are in one place.*

hand **=** **None**

*# Variables to count how many frames have passed and to set the size of the window.*

frames\_elapsed **=** 0

FRAME\_HEIGHT **=** 200

FRAME\_WIDTH **=** 300

*# Humans come in a ton of beautiful shades and colors.*

*# Try editing these if your program has trouble recognizing your skin tone.*

CALIBRATION\_TIME **=** 30

BG\_WEIGHT **=** 0.5

OBJ\_THRESHOLD **=** 18

**HandData: A class to hold all the hand's details and flags**

**class** HandData:

top **=** (0,0)

bottom **=** (0,0)

left **=** (0,0)

right **=** (0,0)

centerX **=** 0

prevCenterX **=** 0

isInFrame **=** **False**

isWaving **=** **False**

fingers **=** **None**

gestureList **=** []

**def** \_\_init\_\_(self, top, bottom, left, right, centerX):

self**.**top **=** top

self**.**bottom **=** bottom

self**.**left **=** left

self**.**right **=** right

self**.**centerX **=** centerX

self**.**prevCenterX **=** 0

isInFrame **=** **False**

isWaving **=** **False**

**def** update(self, top, bottom, left, right):

self**.**top **=** top

self**.**bottom **=** bottom

self**.**left **=** left

self**.**right **=** right

**def** check\_for\_waving(self, centerX):

self**.**prevCenterX **=** self**.**centerX

self**.**centerX **=** centerX

**if** abs(self**.**centerX **-** self**.**prevCenterX **>** 3):

self**.**isWaving **=** **True**

**else**:

self**.**isWaving **=** **False**

**write\_on\_image(): Write info related to the hand gesture and outline the region of interest**

*# Here we take the current frame, the number of frames elapsed, and how many fingers we've detected*

*# so we can print on the screen which gesture is happening (or if the camera is calibrating).*

**def** write\_on\_image(frame):

text **=** "Searching..."

**if** frames\_elapsed **<** CALIBRATION\_TIME:

text **=** "Calibrating..."

**elif** hand **==** **None** **or** hand**.**isInFrame **==** **False**:

text **=** "No hand detected"

**else**:

**if** hand**.**isWaving:

text **=** "Waving"

**elif** hand**.**fingers **==** 0:

text **=** "Rock"

**elif** hand**.**fingers **==** 1:

text **=** "Pointing"

**elif** hand**.**fingers **==** 2:

text **=** "Scissors"

cv2**.**putText(frame, text, (10,20), cv2**.**FONT\_HERSHEY\_COMPLEX, 0.4,( 0 , 0 , 0 ),2,cv2**.**LINE\_AA)

cv2**.**putText(frame, text, (10,20), cv2**.**FONT\_HERSHEY\_COMPLEX, 0.4,(255,255,255),1,cv2**.**LINE\_AA)

*# Highlight the region of interest.*

cv2**.**rectangle(frame, (region\_left, region\_top), (region\_right, region\_bottom), (255,255,255), 2)

**get\_region(): Separate the region of interest and preps it for edge detection**

**def** get\_region(frame):

*# Separate the region of interest from the rest of the frame.*

region **=** frame[region\_top:region\_bottom, region\_left:region\_right]

*# Make it grayscale so we can detect the edges more easily.*

region **=** cv2**.**cvtColor(region, cv2**.**COLOR\_BGR2GRAY)

*# Use a Gaussian blur to prevent frame noise from being labeled as an edge.*

region **=** cv2**.**GaussianBlur(region, (5,5), 0)

**return** region

**get\_average(): Create a weighted average of the background for image differencing**

**def** get\_average(region):

*# We have to use the global keyword because we want to edit the global variable.*

**global** background

*# If we haven't captured the background yet, make the current region the background.*

**if** background **is** **None**:

background **=** region**.**copy()**.**astype("float")

**return**

*# Otherwise, add this captured frame to the average of the backgrounds.*

cv2**.**accumulateWeighted(region, background, BG\_WEIGHT)

**segment(): Use image differencing to separate the hand from the background**

*# Here we use differencing to separate the background from the object of interest.*

**def** segment(region):

**global** hand

*# Find the absolute difference between the background and the current frame.*

diff **=** cv2**.**absdiff(background**.**astype(np**.**uint8), region)

*# Threshold that region with a strict 0 or 1 ruling so only the foreground remains.*

thresholded\_region **=** cv2**.**threshold(diff, OBJ\_THRESHOLD, 255, cv2**.**THRESH\_BINARY)[1]

*# Get the contours of the region, which will return an outline of the hand.*

(\_, contours, \_) **=** cv2**.**findContours(thresholded\_region**.**copy(), cv2**.**RETR\_EXTERNAL, cv2**.**CHAIN\_APPROX\_SIMPLE)

*# If we didn't get anything, there's no hand.*

**if** len(contours) **==** 0:

**if** hand **is** **not** **None**:

hand**.**isInFrame **=** **False**

**return**

*# Otherwise return a tuple of the filled hand (thresholded\_region), along with the outline (segmented\_region).*

**else**:

**if** hand **is** **not** **None**:

hand**.**isInFrame **=** **True**

segmented\_region **=** max(contours, key **=** cv2**.**contourArea)

**return** (thresholded\_region, segmented\_region)

**get\_hand\_data(): Find the extremities of the hand and put them in the global hand object**

**def** get\_hand\_data(thresholded\_image, segmented\_image):

**global** hand

*# Enclose the area around the extremities in a convex hull to connect all outcroppings.*

convexHull **=** cv2**.**convexHull(segmented\_image)

*# Find the extremities for the convex hull and store them as points.*

top **=** tuple(convexHull[convexHull[:, :, 1]**.**argmin()][0])

bottom **=** tuple(convexHull[convexHull[:, :, 1]**.**argmax()][0])

left **=** tuple(convexHull[convexHull[:, :, 0]**.**argmin()][0])

right **=** tuple(convexHull[convexHull[:, :, 0]**.**argmax()][0])

*# Get the center of the palm, so we can check for waving and find the fingers.*

centerX **=** int((left[0] **+** right[0]) **/** 2)

*# We put all the info into an object for handy extraction (get it? HANDy?)*

**if** hand **==** **None**:

hand **=** HandData(top, bottom, left, right, centerX)

**else**:

hand**.**update(top, bottom, left, right)

*# Only check for waving every 6 frames.*

**if** frames\_elapsed **%** 6 **==** 0:

hand**.**check\_for\_waving(centerX)

*# We count the number of fingers up every frame, but only change hand.fingers if*

*# 12 frames have passed, to prevent erratic gesture counts.*

hand**.**gestureList**.**append(count\_fingers(thresholded\_image))

**if** frames\_elapsed **%** 12 **==** 0:

hand**.**fingers **=** most\_frequent(hand**.**gestureList)

hand**.**gestureList**.**clear()

**count\_fingers(): Count the number of fingers using a line intersecting fingertips**

**def** count\_fingers(thresholded\_image):

*# Find the height at which we will draw the line to count fingers.*

line\_height **=** int(hand**.**top[1] **+** (0.2 **\*** (hand**.**bottom[1] **-** hand**.**top[1])))

*# Get the linear region of interest along where the fingers would be.*

line **=** np**.**zeros(thresholded\_image**.**shape[:2], dtype**=**int)

*# Draw a line across this region of interest, where the fingers should be.*

cv2**.**line(line, (thresholded\_image**.**shape[1], line\_height), (0, line\_height), 255, 1)

*# Do a bitwise AND to find where the line intersected the hand -- this is where the fingers are.*

line **=** cv2**.**bitwise\_and(thresholded\_image, thresholded\_image, mask **=** line**.**astype(np**.**uint8))

*# Get the line's new contours. The contours are basically just little lines formed by gaps*

*# in the big line across the fingers, so each would be a finger unless it's very wide.*

(\_, contours, \_) **=** cv2**.**findContours(line**.**copy(), cv2**.**RETR\_EXTERNAL, cv2**.**CHAIN\_APPROX\_NONE)

fingers **=** 0

*# Count the fingers by making sure the contour lines are "finger-sized", i.e. not too wide.*

*# This prevents a "rock" gesture from being mistaken for a finger.*

**for** curr **in** contours:

width **=** len(curr)

**if** width **<** 3 **\*** abs(hand**.**right[0] **-** hand**.**left[0]) **/** 4 **and** width **>** 5:

fingers **+=** 1

**return** fingers

**most\_frequent(): Returns the value in a list that appears most frequently**

**def** most\_frequent(input\_list):

dict **=** {}

count **=** 0

most\_freq **=** 0

**for** item **in** reversed(input\_list):

dict[item] **=** dict**.**get(item, 0) **+** 1

**if** dict[item] **>=** count :

count, most\_freq **=** dict[item], item

**return** most\_freq

**Main function: Get input from camera and call functions to understand it**

*# Our region of interest will be the top right part of the frame.*

region\_top **=** 0

region\_bottom **=** int(2 **\*** FRAME\_HEIGHT **/** 3)

region\_left **=** int(FRAME\_WIDTH **/** 2)

region\_right **=** FRAME\_WIDTH

frames\_elapsed **=** 0

capture **=** cv2**.**VideoCapture(1)

**while** (**True**):

*# Store the frame from the video capture and resize it to the window size.*

ret, frame **=** capture**.**read()

frame **=** cv2**.**resize(frame, (FRAME\_WIDTH, FRAME\_HEIGHT))

*# Flip the frame over the vertical axis so that it works like a mirror, which is more intuitive to the user.*

frame **=** cv2**.**flip(frame, 1)

*# Separate the region of interest and prep it for edge detection.*

region **=** get\_region(frame)

**if** frames\_elapsed **<** CALIBRATION\_TIME:

get\_average(region)

**else**:

region\_pair **=** segment(region)

**if** region\_pair **is** **not** **None**:

*# If we have the regions segmented successfully, show them in another window for the user.*

(thresholded\_region, segmented\_region) **=** region\_pair

cv2**.**drawContours(region, [segmented\_region], **-**1, (255, 255, 255))

cv2**.**imshow("Segmented Image", region)

get\_hand\_data(thresholded\_region, segmented\_region)

*# Write the action the hand is doing on the screen, and draw the region of interest.*

write\_on\_image(frame)

*# Show the previously captured frame.*

cv2**.**imshow("Camera Input", frame)

frames\_elapsed **+=** 1

*# Check if user wants to exit.*

**if** (cv2**.**waitKey(1) **&** 0xFF **==** ord('x')):

**break**

*# When we exit the loop, we have to stop the capture too.*

capture**.**release()

cv2**.**destroyAllWindows()