4.1 SAMPLE CODE

```
class Control:
def init (self, media_path):
self.mp_drawing = mp.solutions.drawing_utils
self.mp_hands = mp.solutions.hands
with open('gesture_recognition_model.pkl', 'rb') as f:
self.model = pickle.load(f)
self.media_player = vlc.MediaPlayer()
self.media = vlc.Media(media_path)
self.media_player.set_media(self.media)
self.media_player.play()
time.sleep(2)
self.value = self.media_player.is_playing()
self.launch_cam()
# Pause video
def pause_gesture(self):
self.media_player.set_pause(1)
# Play video
def play_gesture(self):
self.media_player.play()
# Fast forward
def skip_forwards(self):
current_time = self.media_player.get_time()
self.media_player.set_time(current_time + 1000)
# Rewind
def skip_backwards(self):
current_time = self.media_player.get_time()
self.media_player.set_time(current_time - 1000)# Increase volume
def vol_up(self):
value = self.media_player.audio_get_volume()
self.media_player.audio_set_volume(value + 1)
# Decrease volume
def vol_down(self):
value = self.media_player.audio_get_volume()
```

```
self.media_player.audio_set_volume(value - 1)
def launch_cam(self):
# start webcam with opencv
cap = cv2.VideoCapture(0)
# Use mediapipe Hands model
with self.mp_hands.Hands(
max_num_hands=2,
min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
while cap.isOpened():
success, image = cap.read()
if not success:
print("Ignoring empty camera frame.")
continue
# Convert to RGB and flip image so that camera output is like mirror.
image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
image.flags.writeable = False
results = hands.process(image)
# Draw annotations
image.flags.writeable = True
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
if results.multi_hand_landmarks17
for hand_landmarks in results.multi_hand_landmarks:
self.mp_drawing.draw_landmarks(
image, hand_landmarks, self.mp_hands.HAND_CONNECTIONS)
try:
# Get landmarks for hand 1
hand_1 = results.multi_hand_landmarks[0].landmark
landmarks_1 = list(
np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark
in hand_1]).flatten())
# get landmarks for hand 2
hand_2 = results.multi_hand_landmarks[1].landmark
landmarks_2 = list(
```

```
np.array([[landmark.x, landmark.y, landmark.z, landmark.visibility] for landmark
in hand_2]).flatten())
# Concatenate hand 1 and hand 2 data
data = landmarks_1 + landmarks_2
# Create pandas dataframe from data
X = pd.DataFrame([data])
# Make prediction
direction_gest = self.model.predict(X)[0]
# Annotate image with prediction text
cv2.putText(image, direction_gest.split(' ')[0]
, (90, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2, cv2.LINE_AA)
# Trigger the different functions based on the predicted gestures
if direction_gest.split(' ')[0] == "Play":
self.play_gesture()
elif direction_gest.split(' ')[0] == "Stop":
self.pause_gesture()
elif direction_gest.split(' ')[0] == "Up"self.vol_up()
elif direction_gest.split(' ')[0] == "Down":
self.vol_down()
elif direction_gest.split(' ')[0] == "Forwards":
self.skip_forwards()
elif direction_gest.split(' ')[0] == "Back":
self.skip_backwards()
except:
pass
# Show image
cv2.imshow('Gesture Controller', image)
if cv2.waitKey(10) & 0xFF == ord('q'):
break
cap.release()
cv2.destroyAllWindows()
```