Springer Link Search Log in

International Conference on Human-Computer Interaction

↳ HCII 2022: **Human-Computer Interaction. Technological Innovation** pp 195–208

A Paper-Based Keyboard Using ArUco Codes: ArUco Keyboard

Onur Toker, Bayazit Karaman & Doga Demirel

Conference paper | First Online: 16 June 2022

628 Accesses

Part of the Lecture Notes in Computer Science book series (LNCS,volume 13303)

Abstract

Object tracking in computer vision can be done either by using a marker-less or marker-based approach. Computer vision systems have been using Fiducial markers for pose estimation in different applications such as augmented reality [5] and robot navigation [4]. With the advancements in Augmented Reality (AR), new tools such as AugmentedReality uco (ArUco) [6] markers have been introduced to the literature. ArUco markers, are used to tackle the localization problem in AR, allowing camera pose estimation to be carried out by a binary matrix. Using a binary matrix not just simplifies the process but also increases the efficiency. As a part of our initiative to create a cost-efficient, 24/7 accessible, Virtual Reality (VR) based chemistry lab for underprivileged students, we wanted to create an alternative way of interacting with the virtual scene. In this study, we used ArUco markers to create a low-cost keyboard only using a piece of paper and an off-the-shelf webcam. We believe this method of keyboard will be more beneficial to the user as they can see the keys before they are typing in the corner of the screen instead of an insufficient on the screen VR keyboard or a regular keyboard where the user can't see what they are typing with a VR headset. As potential extensions of the base system, we have also designed and evaluated a stereo camera and an IMU sensor based system with various sensor fusion techniques. In summary, the stereo camera reduces occlusion related problems, and the IMU sensor detects vibrations which in turn simplifies the KeyPress detection problem. It has been observed that use of any of these additional sensors improves the overall system performance.

Keywords

ArUco codes **IMU sensors** **Sensor fusion**

- DOI: 10.1007/978-3-031-05409-9_15
- Chapter length: 14 pages

References

1. ArUco keyboard demo video: Base system. https://youtu.be/tnKc6zvXliY

2. ArUco keyboard demo video: IMU sensor based version. https://youtu.be/sIuhZQpu0AE

3. ArUco keyboard demo video: Stereo camera version (USB3 ZED camera). https://youtu.be/ssbv2NqfAJg

4. Bacik, J., Durovsky, F., Fedor, P., Perdukova, D.: Autonomous flying with quadrocopter using fuzzy control and ArUco markers. Intell. Serv. Robot.

10(3), 185–194 (2017). https://doi.org/10.1007/s11370-017-0219-8

5. Billinghurst, M., Clark, A., Lee, G.: A survey of augmented reality. Found. Trends Hum.-Comput. Interact. **8**(2–3), 73–272 (2015). http://dx.doi.org/10.1561/1100000049

6. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., Marín-Jiménez, M.: Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recogn. **47**(6), 2280–2292 (2014). https://doi.org/10.1016/j.patcog.2014.01.005

Acknowledgments

Funding is provided by NSF-1919855, Advanced Mobility Institute grants GR-2000028, GR-2000029, and Florida Polytechnic University startup grant GR-1900022.

Author information

Authors and Affiliations

Florida Polytechnic University, Lakeland, FL, 33805, USA

Onur Toker, Bayazit Karaman & Doga Demirel

Corresponding author

Correspondence to Onur Toker .

Editor information

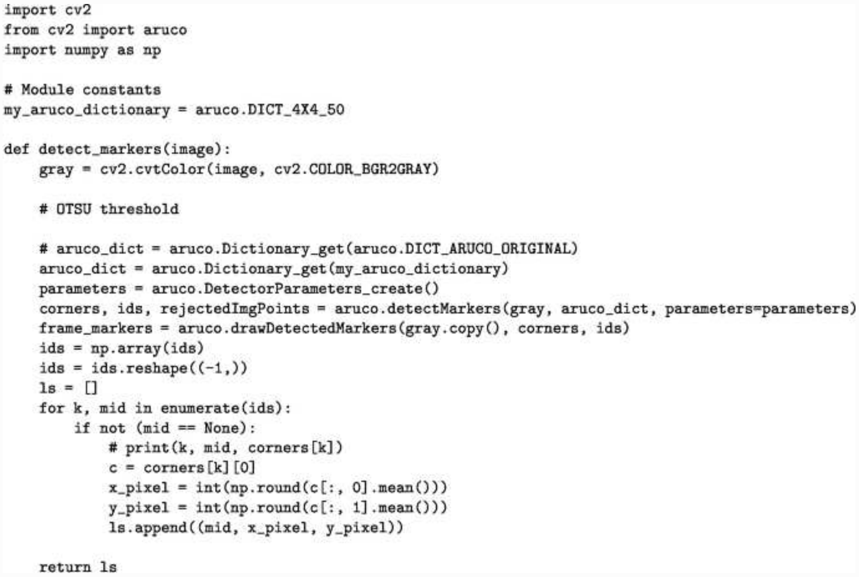
Editors and Affiliations

The Open University of Japan, Chiba, Japan

Masaaki Kurosu

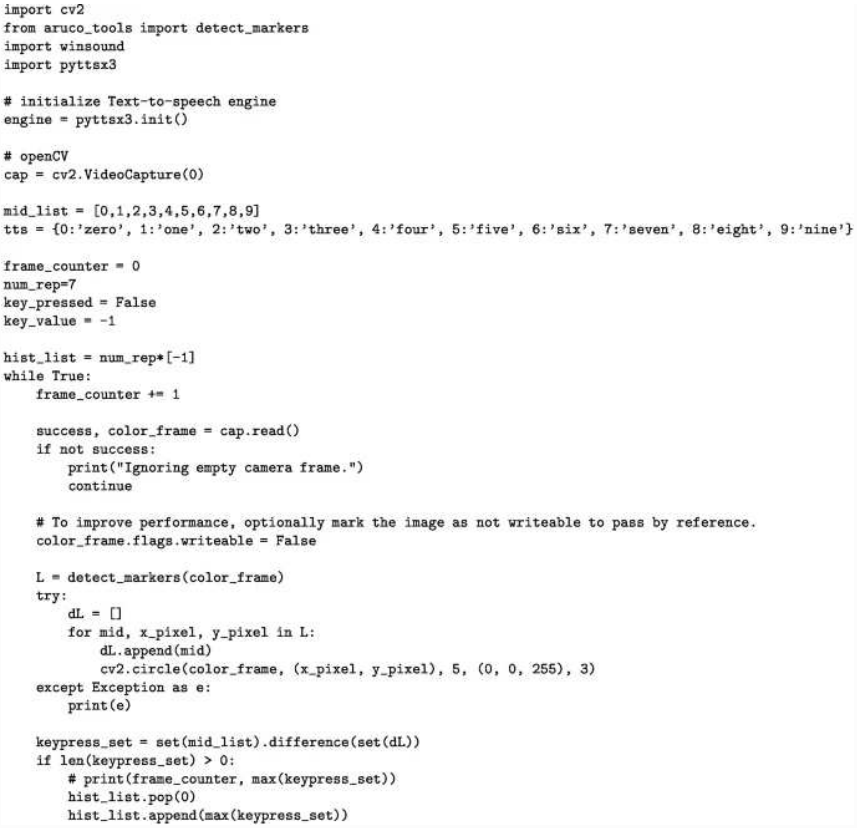
Appendices

Appendix I: ArUco Code Detection Module `aruco_tools.py`

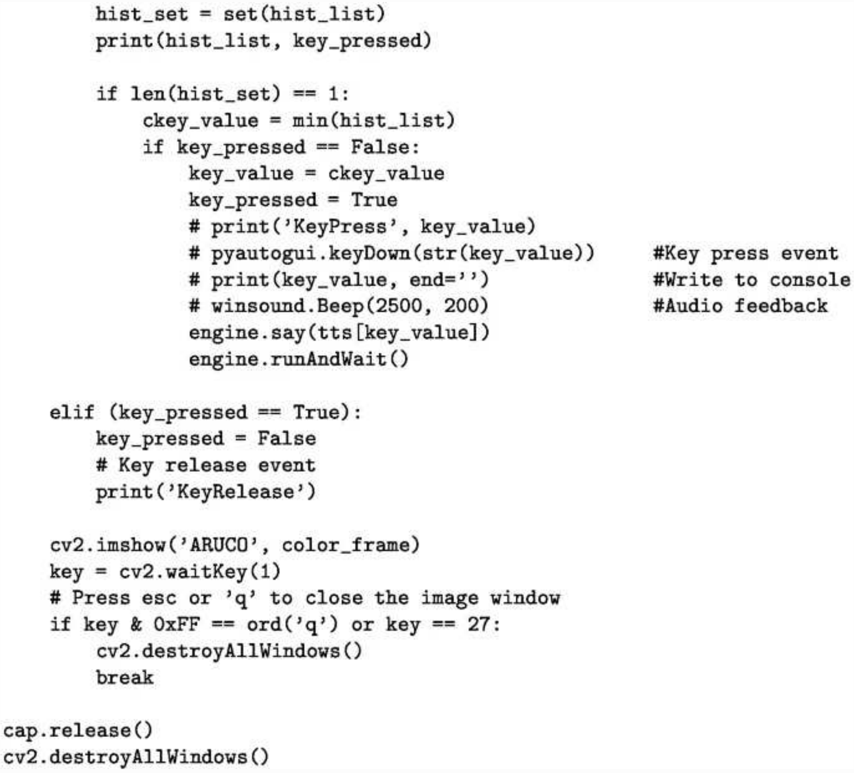


```
import cv2
from cv2 import aruco
import numpy as np

# Module constants
my_aruco_dictionary = aruco.DICT_4X4_50

def detect_markers(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # OTSU threshold

    # aruco_dict = aruco.Dictionary_get(aruco.DICT_ARUCO_ORIGINAL)
    aruco_dict = aruco.Dictionary_get(my_aruco_dictionary)
    parameters = aruco.DetectorParameters_create()
    corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)
    frame_markers = aruco.drawDetectedMarkers(gray.copy(), corners, ids)
    ids = np.array(ids)
    ids = ids.reshape((-1,))
    ls = []
    for k, mid in enumerate(ids):
        if not (mid == None):
            # print(k, mid, corners[k])
            c = corners[k][0]
            x_pixel = int(np.round(c[:, 0].mean()))
            y_pixel = int(np.round(c[:, 1].mean()))
            ls.append((mid, x_pixel, y_pixel))

    return ls
```

Appendix II: Base System `minikdb_mono.py`

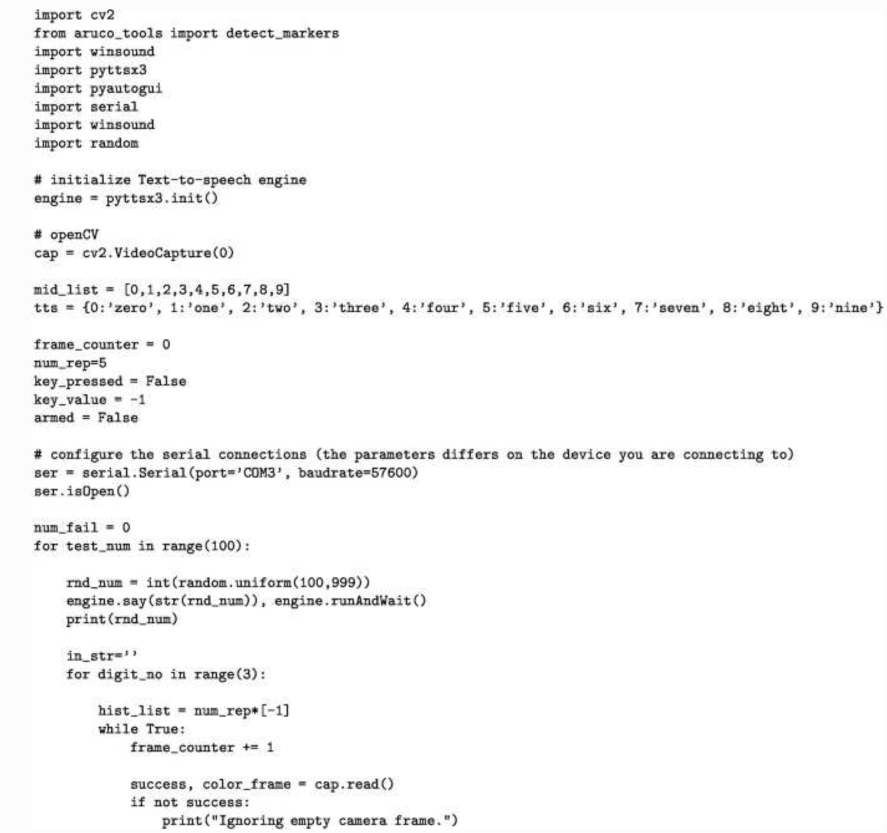


```
import cv2
from aruco_tools import detect_markers
import winsound
import pyttsx3

# initialize Text-to-speech engine
engine = pyttsx3.init()

# openCV
cap = cv2.VideoCapture(0)

mid_list = [0,1,2,3,4,5,6,7,8,9]
tts = {0:'zero', 1:'one', 2:'two', 3:'three', 4:'four', 5:'five', 6:'six', 7:'seven', 8:'eight', 9:'nine'}

frame_counter = 0
num_rep=7
key_pressed = False
key_value = -1

hist_list = num_rep*[-1]
while True:
    frame_counter += 1

    success, color_frame = cap.read()
    if not success:
        print("Ignoring empty camera frame.")
        continue

    # To improve performance, optionally mark the image as not writeable to pass by reference.
    color_frame.flags.writeable = False

    L = detect_markers(color_frame)
    try:
        dL = []
        for mid, x_pixel, y_pixel in L:
            dL.append(mid)
            cv2.circle(color_frame, (x_pixel, y_pixel), 5, (0, 0, 255), 3)
    except Exception as e:
        print(e)

    keypress_set = set(mid_list).difference(set(dL))
    if len(keypress_set) > 0:
        # print(frame_counter, max(keypress_set))
        hist_list.pop(0)
        hist_list.append(max(keypress_set))
```

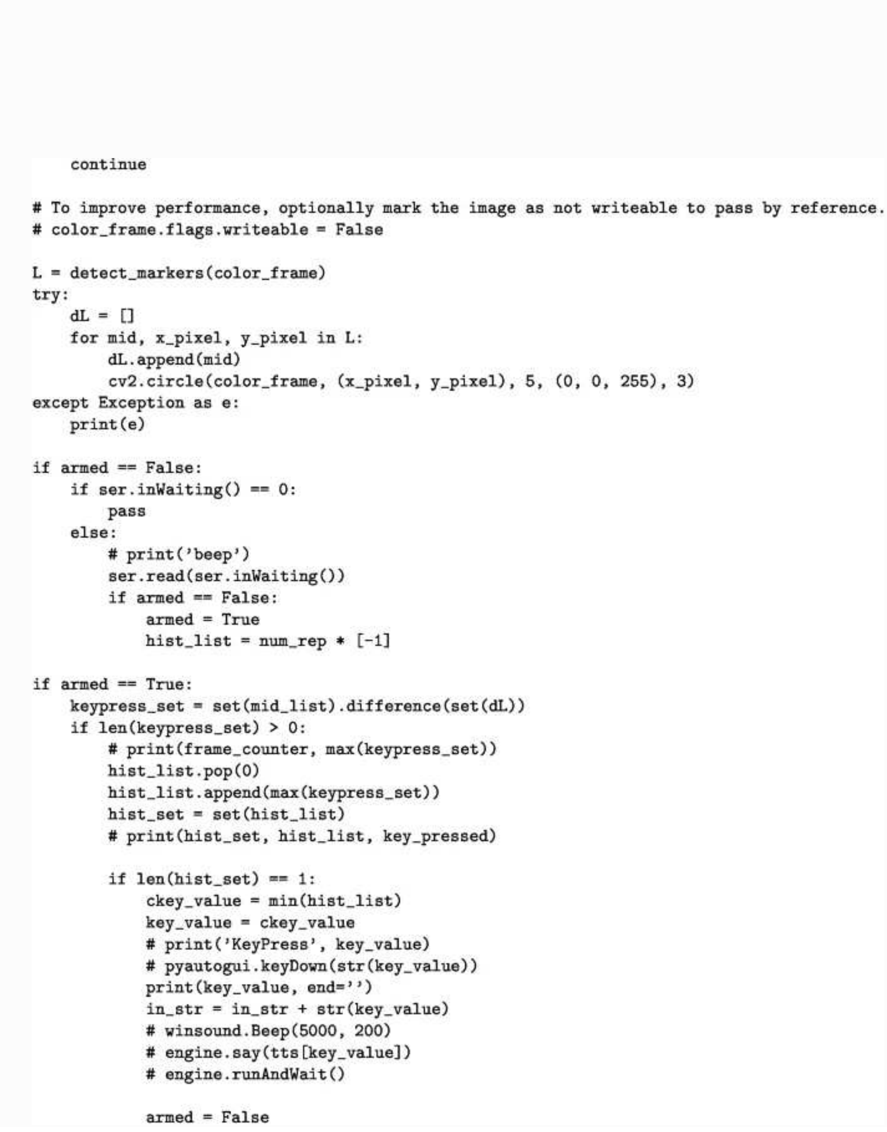


```
        hist_set = set(hist_list)
        print(hist_list, key_pressed)

        if len(hist_set) == 1:
            ckey_value = min(hist_list)
            if key_pressed == False:
                key_value = ckey_value
                key_pressed = True
                # print('KeyPress', key_value)
                # pyautogui.keyDown(str(key_value))     #Key press event
                # print(key_value, end='')              #Write to console
                # winsound.Beep(2500, 200)              #Audio feedback
                engine.say(tts[key_value])
                engine.runAndWait()

    elif (key_pressed == True):
        key_pressed = False
        # Key release event
        print('KeyRelease')

    cv2.imshow('ARUCO', color_frame)
    key = cv2.waitKey(1)
    # Press esc or 'q' to close the image window
    if key & 0xFF == ord('q') or key == 27:
        cv2.destroyAllWindows()
        break

cap.release()
cv2.destroyAllWindows()
```

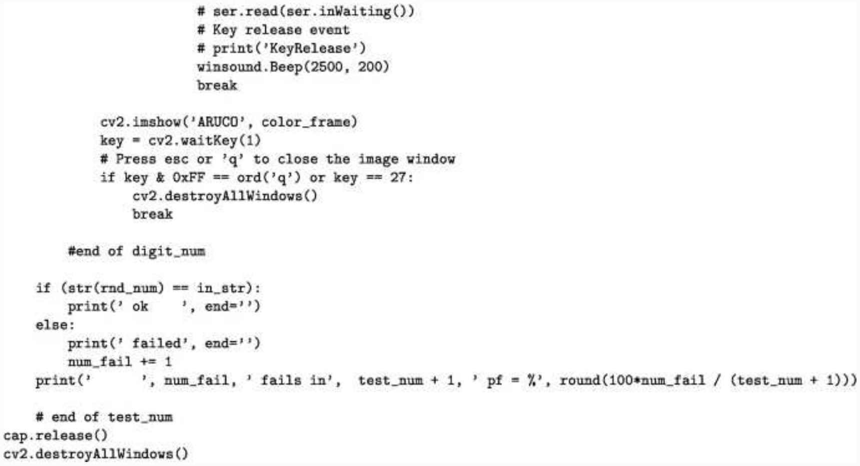
Appendix III: IMU Based System `minikbd_imu.py`



```
import cv2
from aruco_tools import detect_markers
import winsound
import pyttsx3
import pyautogui
import serial
import winsound
import random

# initialize Text-to-speech engine
engine = pyttsx3.init()

# openCV
cap = cv2.VideoCapture(0)

mid_list = [0,1,2,3,4,5,6,7,8,9]
tts = {0:'zero', 1:'one', 2:'two', 3:'three', 4:'four', 5:'five', 6:'six', 7:'seven', 8:'eight', 9:'nine'}

frame_counter = 0
num_rep=5
key_pressed = False
key_value = -1
armed = False

# configure the serial connections (the parameters differs on the device you are connecting to)
ser = serial.Serial(port='COM3', baudrate=57600)
ser.isOpen()

num_fail = 0
for test_num in range(100):

    rnd_num = int(random.uniform(100,999))
    engine.say(str(rnd_num)), engine.runAndWait()
    print(rnd_num)

    in_str=''
    for digit_no in range(3):

        hist_list = num_rep*[-1]
        while True:
            frame_counter += 1

            success, color_frame = cap.read()
            if not success:
                print("Ignoring empty camera frame.")
```



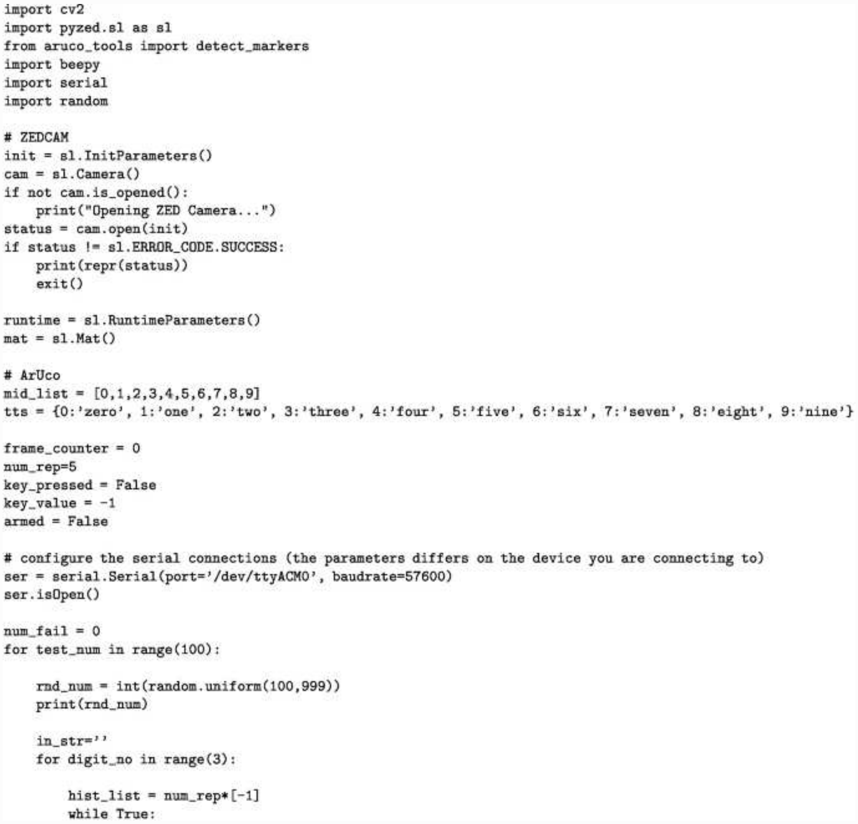
```
    continue

# To improve performance, optionally mark the image as not writeable to pass by reference.
# color_frame.flags.writeable = False

L = detect_markers(color_frame)
try:
    dL = []
    for mid, x_pixel, y_pixel in L:
        dL.append(mid)
        cv2.circle(color_frame, (x_pixel, y_pixel), 5, (0, 0, 255), 3)
except Exception as e:
    print(e)

if armed == False:
    if ser.inWaiting() == 0:
        pass
    else:
        # print('beep')
        ser.read(ser.inWaiting())
        if armed == False:
            armed = True
            hist_list = num_rep * [-1]

if armed == True:
    keypress_set = set(mid_list).difference(set(dL))
    if len(keypress_set) > 0:
        # print(frame_counter, max(keypress_set))
        hist_list.pop(0)
        hist_list.append(max(keypress_set))
        hist_set = set(hist_list)
        # print(hist_set, hist_list, key_pressed)

        if len(hist_set) == 1:
            ckey_value = min(hist_list)
            key_value = ckey_value
            # print('KeyPress', key_value)
            # pyautogui.keyDown(str(key_value))
            print(key_value, end='')
            in_str = in_str + str(key_value)
            # winsound.Beep(5000, 200)
            # engine.say(tts[key_value])
            # engine.runAndWait()

            armed = False
```



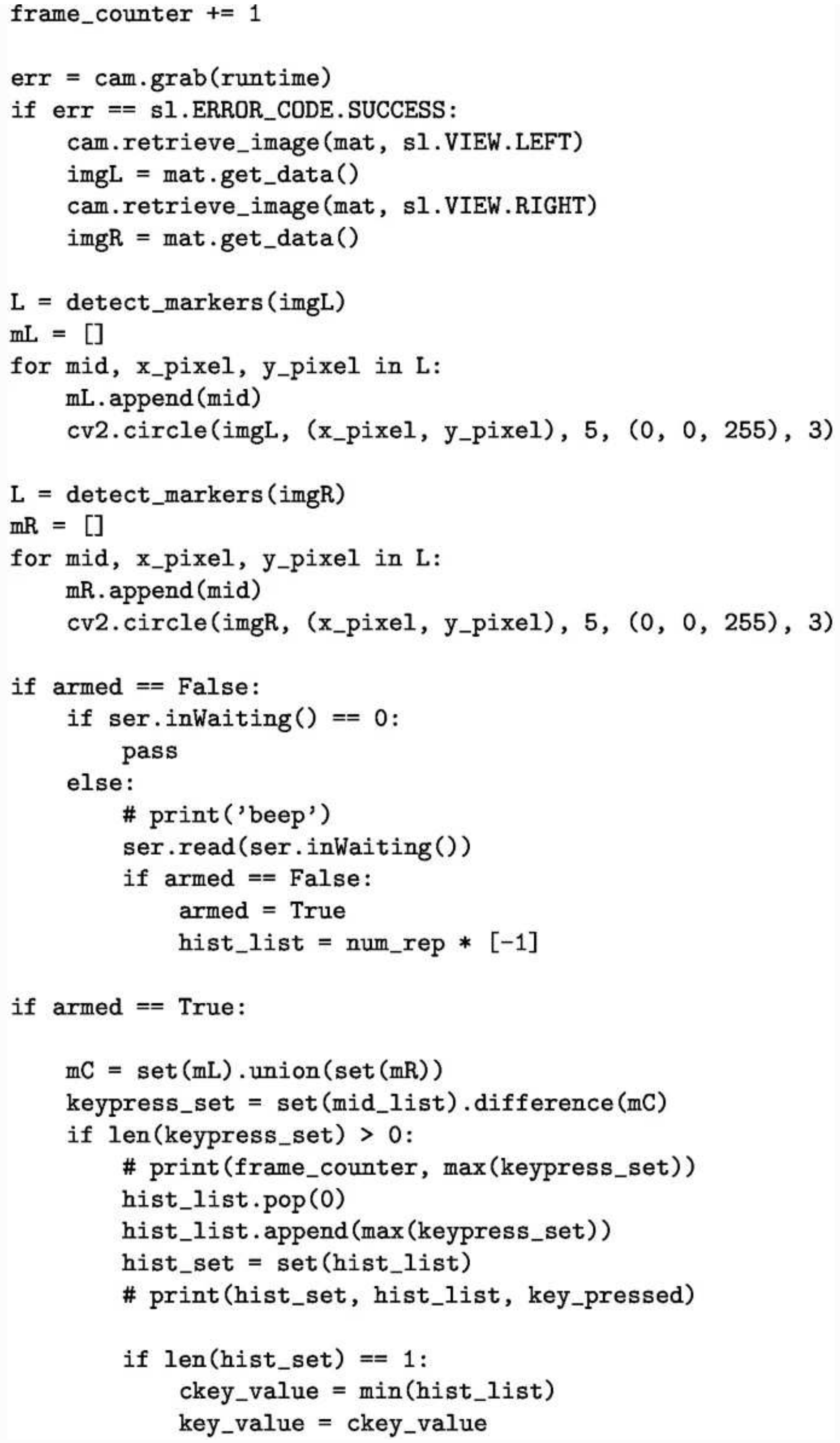
```
                        # ser.read(ser.inWaiting())
                        # Key release event
                        # print(’KeyRelease’)
                        winsound.Beep(2500, 200)
                        break

            cv2.imshow(’ARUCO’, color_frame)
            key = cv2.waitKey(1)
            # Press esc or ’q’ to close the image window
            if key & 0xFF == ord(’q’) or key == 27:
                cv2.destroyAllWindows()
                break

        #end of digit_num

    if (str(rnd_num) == in_str):
        print(’ ok    ’, end=’’)
    else:
        print(’ failed’, end=’’)
        num_fail += 1
    print(’      ’, num_fail, ’ fails in’,  test_num + 1, ’ pf = %’, round(100*num_fail / (test_num + 1)))

    # end of test_num
cap.release()
cv2.destroyAllWindows()
```

Appendix IV: Stereo Camera Based System

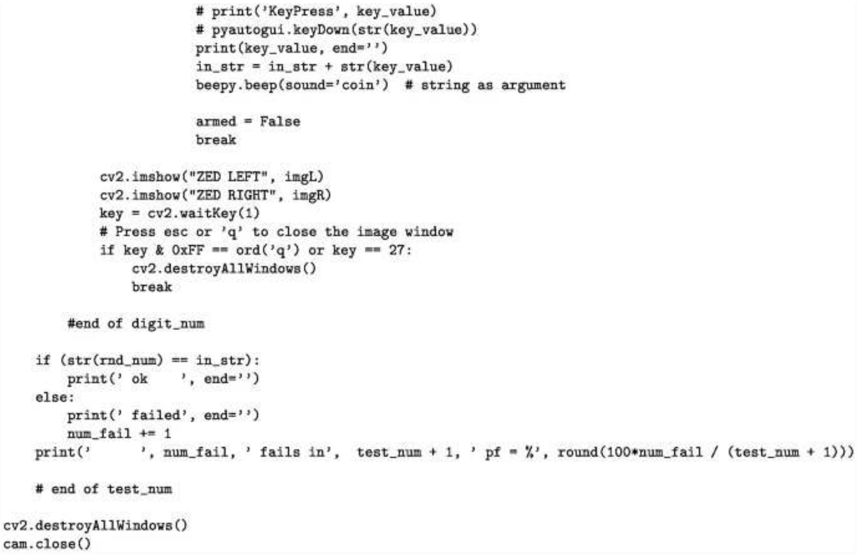
minikbd_zed.py



```
import cv2
import pyzed.sl as sl
from aruco_tools import detect_markers
import beepy
import serial
import random

# ZEDCAM
init = sl.InitParameters()
cam = sl.Camera()
if not cam.is_opened():
    print("Opening ZED Camera...")
status = cam.open(init)
if status != sl.ERROR_CODE.SUCCESS:
    print(repr(status))
    exit()

runtime = sl.RuntimeParameters()
mat = sl.Mat()

# ArUco
mid_list = [0,1,2,3,4,5,6,7,8,9]
tts = {0:’zero’, 1:’one’, 2:’two’, 3:’three’, 4:’four’, 5:’five’, 6:’six’, 7:’seven’, 8:’eight’, 9:’nine’}

frame_counter = 0
num_rep=5
key_pressed = False
key_value = -1
armed = False

# configure the serial connections (the parameters differs on the device you are connecting to)
ser = serial.Serial(port=’/dev/ttyACM0’, baudrate=57600)
ser.isOpen()

num_fail = 0
for test_num in range(100):

    rnd_num = int(random.uniform(100,999))
    print(rnd_num)

    in_str=’’
    for digit_no in range(3):

        hist_list = num_rep*[-1]
        while True:
```

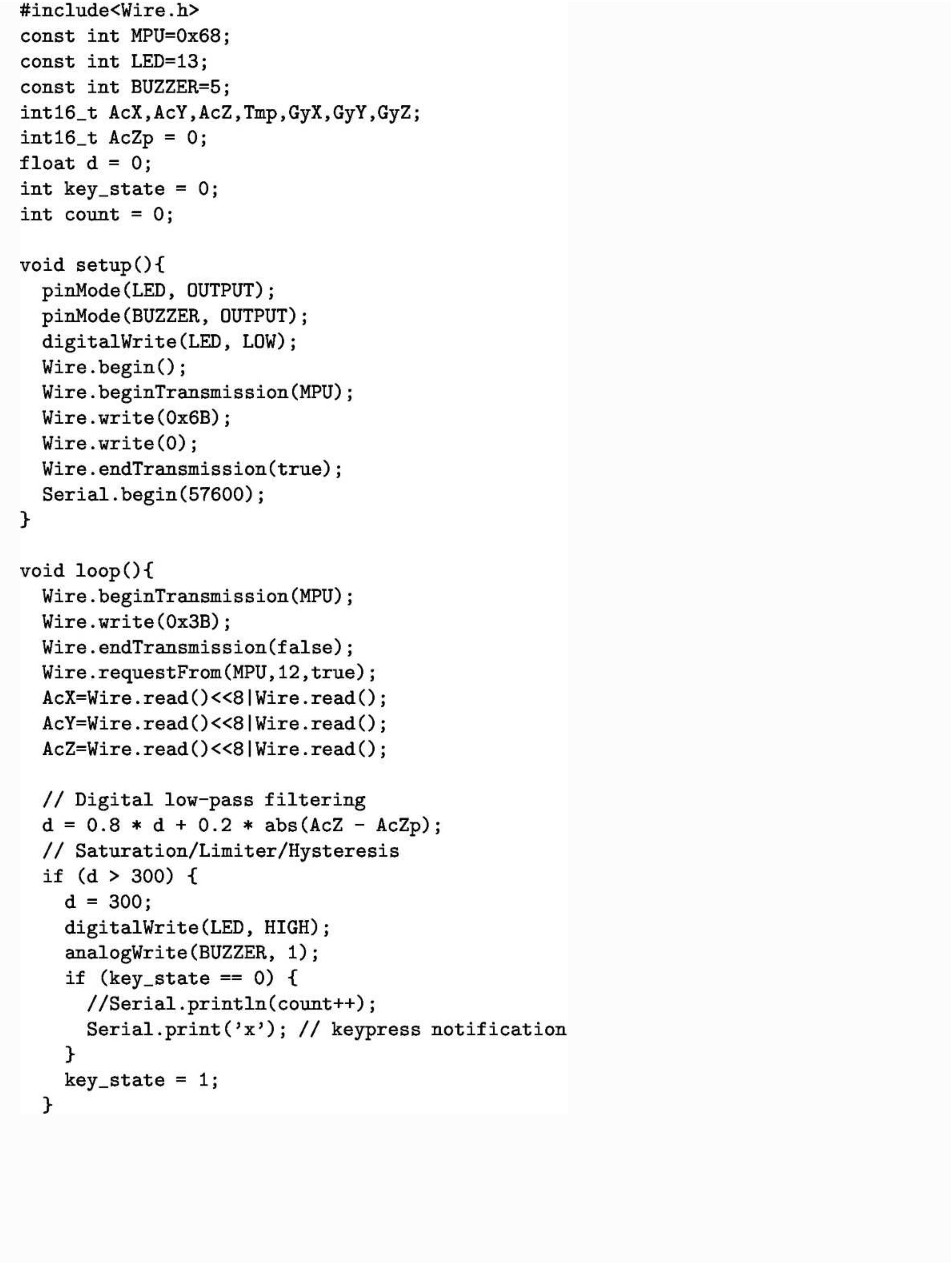


```
frame_counter += 1

err = cam.grab(runtime)
if err == sl.ERROR_CODE.SUCCESS:
    cam.retrieve_image(mat, sl.VIEW.LEFT)
    imgL = mat.get_data()
    cam.retrieve_image(mat, sl.VIEW.RIGHT)
    imgR = mat.get_data()

L = detect_markers(imgL)
mL = []
for mid, x_pixel, y_pixel in L:
    mL.append(mid)
    cv2.circle(imgL, (x_pixel, y_pixel), 5, (0, 0, 255), 3)

L = detect_markers(imgR)
mR = []
for mid, x_pixel, y_pixel in L:
    mR.append(mid)
    cv2.circle(imgR, (x_pixel, y_pixel), 5, (0, 0, 255), 3)

if armed == False:
    if ser.inWaiting() == 0:
        pass
    else:
        # print('beep')
        ser.read(ser.inWaiting())
        if armed == False:
            armed = True
            hist_list = num_rep * [-1]

if armed == True:

    mC = set(mL).union(set(mR))
    keypress_set = set(mid_list).difference(mC)
    if len(keypress_set) > 0:
        # print(frame_counter, max(keypress_set))
        hist_list.pop(0)
        hist_list.append(max(keypress_set))
        hist_set = set(hist_list)
        # print(hist_set, hist_list, key_pressed)

        if len(hist_set) == 1:
            ckey_value = min(hist_list)
            key_value = ckey_value
```

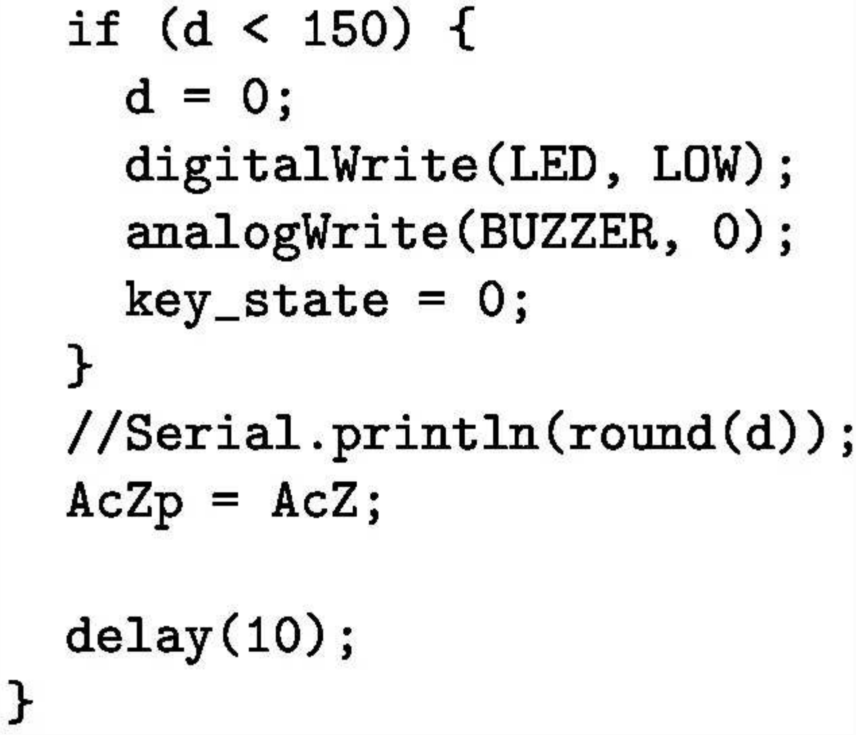


```
                        # print('KeyPress', key_value)
                        # pyautogui.keyDown(str(key_value))
                        print(key_value, end='')
                        in_str = in_str + str(key_value)
                        beepy.beep(sound='coin')  # string as argument

                        armed = False
                        break

            cv2.imshow("ZED LEFT", imgL)
            cv2.imshow("ZED RIGHT", imgR)
            key = cv2.waitKey(1)
            # Press esc or 'q' to close the image window
            if key & 0xFF == ord('q') or key == 27:
                cv2.destroyAllWindows()
                break

        #end of digit_num

    if (str(rnd_num) == in_str):
        print(' ok    ', end='')
    else:
        print(' failed', end='')
        num_fail += 1
    print('      ', num_fail, ' fails in',  test_num + 1, ' pf = %', round(100*num_fail / (test_num + 1)))

    # end of test_num

cv2.destroyAllWindows()
cam.close()
```

Appendix V: IMU Sensor Code for Arduino Uno



```
#include<Wire.h>
const int MPU=0x68;
const int LED=13;
const int BUZZER=5;
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
int16_t AcZp = 0;
float d = 0;
int key_state = 0;
int count = 0;

void setup(){
  pinMode(LED, OUTPUT);
  pinMode(BUZZER, OUTPUT);
  digitalWrite(LED, LOW);
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
  Serial.begin(57600);
}

void loop(){
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,12,true);
  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read();

  // Digital low-pass filtering
  d = 0.8 * d + 0.2 * abs(AcZ - AcZp);
  // Saturation/Limiter/Hysteresis
  if (d > 300) {
    d = 300;
    digitalWrite(LED, HIGH);
    analogWrite(BUZZER, 1);
    if (key_state == 0) {
      //Serial.println(count++);
      Serial.print(’x’); // keypress notification
    }
    key_state = 1;
  }
```



```
  if (d < 150) {
    d = 0;
    digitalWrite(LED, LOW);
    analogWrite(BUZZER, 0);
    key_state = 0;
  }
  //Serial.println(round(d));
  AcZp = AcZ;

  delay(10);
}
```

Rights and permissions

Reprints and Permissions

Copyright information

About this paper

Cite this paper

Toker, O., Karaman, B., Demirel, D. (2022). A Paper-Based Keyboard Using ArUco Codes: ArUco Keyboard. In: Kurosu, M. (eds) Human-Computer Interaction. Technological Innovation. HCII 2022. Lecture Notes in Computer Science,

vol 13303. Springer, Cham. https://doi.org/10.1007/978-3-031-05409-9_15

.RIS .ENW .BIB

DOI

https://doi.org/10.1007/978-3-031-05409-9_15

Published	Publisher Name	Print ISBN
16 June 2022	Springer, Cham	978-3-031-05408-2

Online ISBN	eBook Packages
978-3-031-05409-9	Computer Science Computer Science (R0)