A Paper-based Keyboard using ArUco Codes: ArUco Keyboard

Onur Toker, Bayazit Karaman, and Doga Demirel

Florida Polytechnic University
Lakeland, FL 33805
{otoker,bkaraman,ddemirel}@floridapoly.edu

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

1 Introduction

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.

Our setup is straightforward and consists of a webcam and a piece of paper with a keyboard-like pattern printed on it, see [4]. Basically, there is a numeric keypad with rectangular regions labeled from 0 to 9, and each region has the ArUco code for the corresponding key value. When the system is running in "live" mode, users can use this printed paper as a keypad. All "touched" key values will be translated to keypress events and the printed paper will act as a regular keyboard. This system needs both computer vision and smoothing/filtering techniques which can be fine-tuned for an average user or a specific user.

In this paper, we propose using a real-time OpenCV-based computer vision approach and a specific state-machine based fast smoothing/filtering algorithm. The filter has a parameter, N, which represents the filter strength. We have first created a dataset using six-digit numbers typed by the same user using this paper-based keyboard. Then we varied the filter strength parameter N from 1 to 10 and measured the accuracy of the proposed paper-based keyboard. For a specific trained user, and for a specific dataset of size ten, the system accuracy is measured as 0.0 for N less than 4, 0.6 for $N = 4$, 1.0 for $N = 5, 6, 7$, 0.3 for $N = 8$, 0.10 for $N = 9$, and finally 0.0 for $N = 10$. Optimal values seem to be $N = 5, 6, 7$, but if we eliminate $N = 5$ and 7 as potential boundary cases, we get $N = 6$ as the optimal choice for this specific trained user.

The ArUco keyboard used in this study is shown in Fig. 1, and the base system demo is presented in Fig. 4. 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. The specific stereo camera used for this research was a USB3 ZED camera, see Fig. 2, tested with a GeForce GTX 1050 Ti Max-Q 4GB laptop running Ubuntu 18 LTS. It has been observed that the stereo camera reduces occlusion related issues, and results more robust detection performance. The IMU sensor used in this research is a GY-521 accelerometer and gyro sensor, see Fig. 3, interfaced to an Arduino Uno board over the SPI interface. The IMU sensor detects keypress/touch related vibrations and sends this information to the host computer. Most of the mobile devices used today do have a camera(s) and IMU sensors, therefore the proposed extensions to our base system is quite realistic. Basically, the IMU sensor detects vibrations which in turn simplifies the KeyPress detection problem.

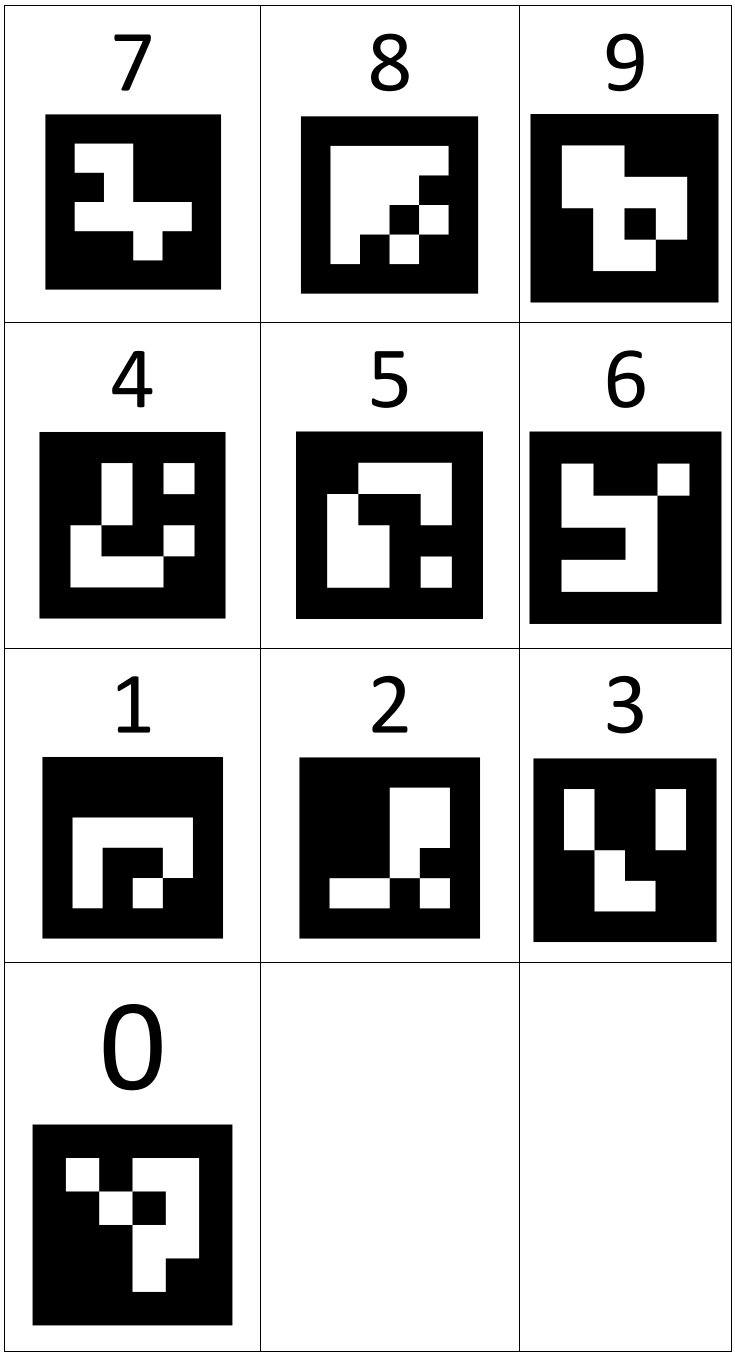

Fig. 1. ArUco keyboard.

In summary, it has been observed that use of any of these additional sensors, i.e. additional camera and/or IMU sensor, improves the overall system performance.

Fig. 2. Stereo camera (USB3 ZED camera) based ArUco keyboard system.

2 Base system

Our base system [1] shown in, Fig. 4, has a single webcam. The algorithm used in this base implementation is shown in Algorithm 1. In each OpenCV frame, we first detect all visible ArUco markers and then determine all blocked ArUco markers. For each frame, we also determine the highest blocked marker value. If the highest blocked marker is the same during the past N frames, then we generate a KeyPress event. A KeyRelease event is generated in the first frame having all ArUco markers visible.

The detection performance of the system depends on the value of N. For a specific trained user, $N = 6$ value is found to be optimal for a webcam running at 25 frames/sec. In general, the optimal N value depends on the frame rate and the user.

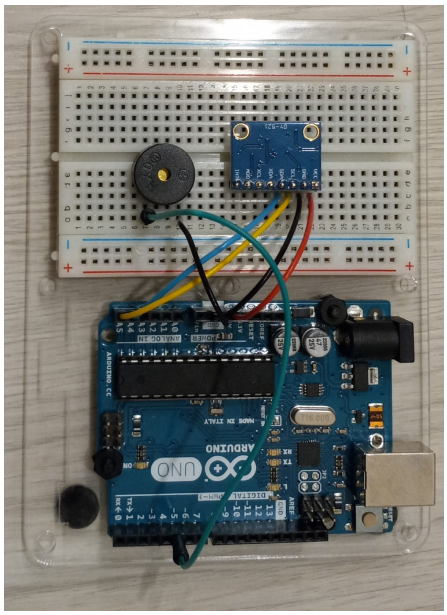

Fig. 3. GY521: InvenSense MPU-6050 based IMU sensor board interfaced to an Arduino Uno over SPI.

Algorithm 1: Algorithm used for the base system

Input: OpenCV frame `img` as a `numpy` array
Result: Keyboard event and key code

1 Find all visible ArUco markers in the frame `img`
2 Determine all blocked ArUco markers
3 Memorize the list of blocked markers for the past N frames
4 Find the highest blocked marker for all of the past N frames
5 If the highest blocked marker does not change, trigger a KeyPress event and use the highest blocked marker as the key value
6 Generate a KeyRelease event in the first frame having all ArUco markers visible
7 Goto Step 1

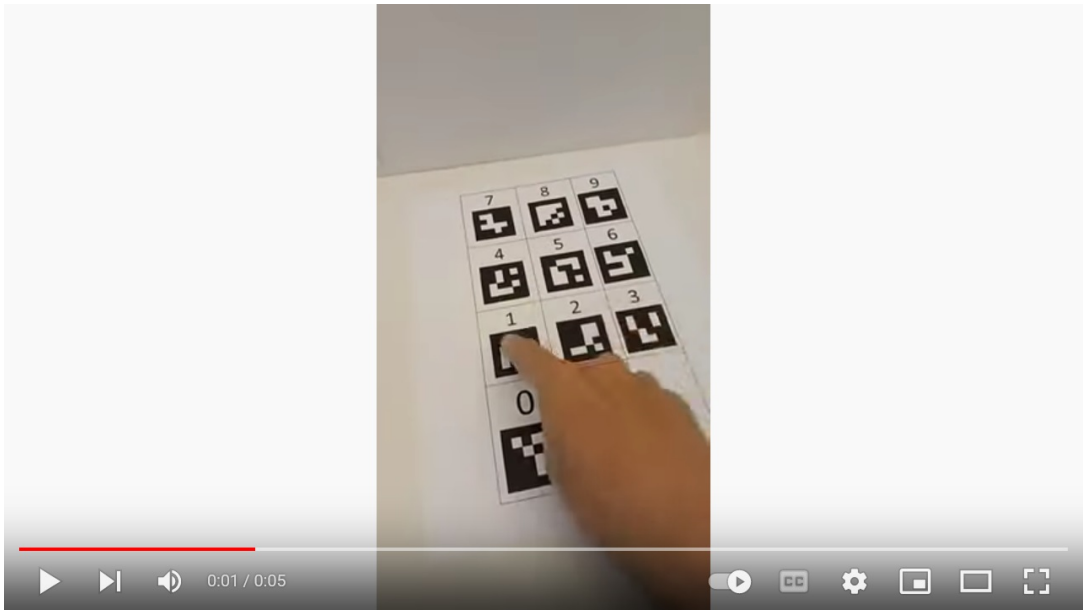

Fig. 4. ArUco keyboard: Base system, `https://youtu.be/tnKc6zvXliY`

3 IMU sensor based system

The base system presented in the previous section works by detecting blocked ArUco markers in each frame. However, this single camera based system cannot differentiate between blocked without touch and blocked because of touch cases. Because of this technical difficulty, a user should be trained not to keep his/her hand stationary for a "long" period of time (5/25 sec) while being visible by the camera. Although this is technically possible, and the training process is observed to be not that difficult, we have developed an alternative approach to overcome this problem.

This new approach [2] is based on using an IMU sensor, see Fig. 5, to differentiate between blocked without touch and blocked because of touch. IMU sensors have acceleration sensors in x, y and z directions, and can be used to detect even a slight tap on a surface. We have used an InvenSense MPU6050 chip as our IMU sensor. A first order digital low-pass filter is used for smoothing, and a thresholding with hysteresis is used for tap detection. In this case, the microcontroller sends the tap events to the host device, and only after this stage the host device starts executing Algorithm 1. See the full source code given in the appendix for digital low-pass filter, thresholding and hysteresis parameters.

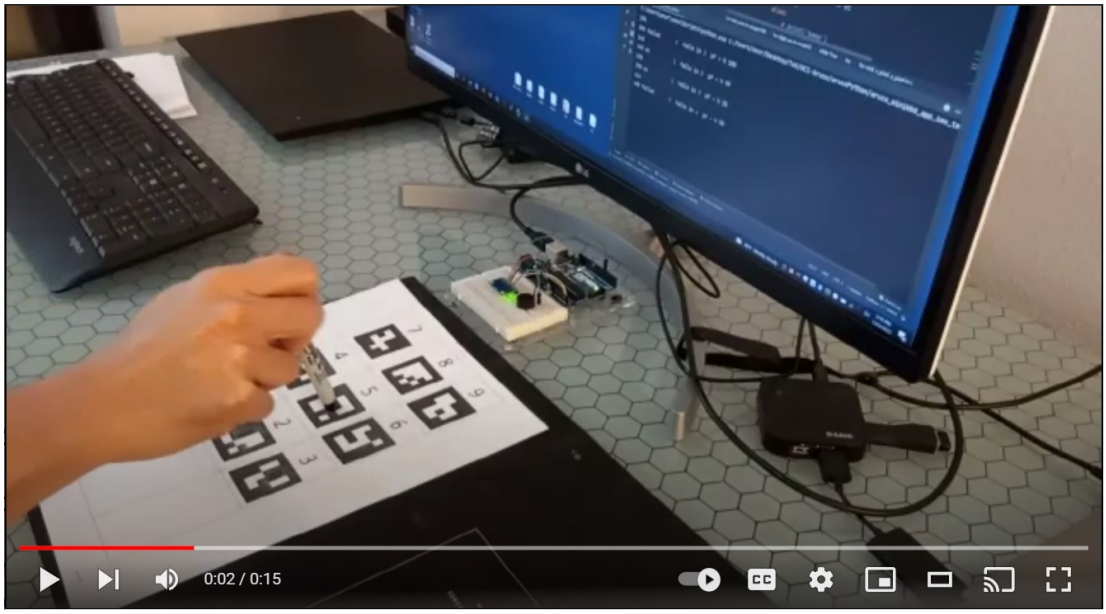

Fig. 5. ArUco keyboard: IMU sensor based version, `https://youtu.be/sIuhZQpu0AE`

4 Stereo camera based system

As a final improvement of the proposed ArUco keyboard system, we have implemented a stereo camera based solution [3] shown in Fig. 6. A stereo camera based system provides more data which can be used to improve the overall system performance, and this is true with or without using an IMU sensor. Sometimes, we may have certain ArUco markers being blocked because of occlusion, and not because of touch. Basically, after a touch or tap is detected we may still have multiple ArUco markers being blocked. The priority scheme used in Algorithm 1 seems to work for most cases, but the failure rate is non-zero and becomes more noticeable if the ArUco keyboard is rotated significantly. A stereo camera greatly improves detection performance for such cases.

If both cameras report a particular ArUco marker as not detected, then the probability of failure, i.e. being not-detected because of occlusion, will be smaller compared to a single camera system. Therefore, use of a stereo camera reduces false KeyPress events and also key value errors. But it requires more processing power and more complex hardware which may not be practical for all possible use cases.

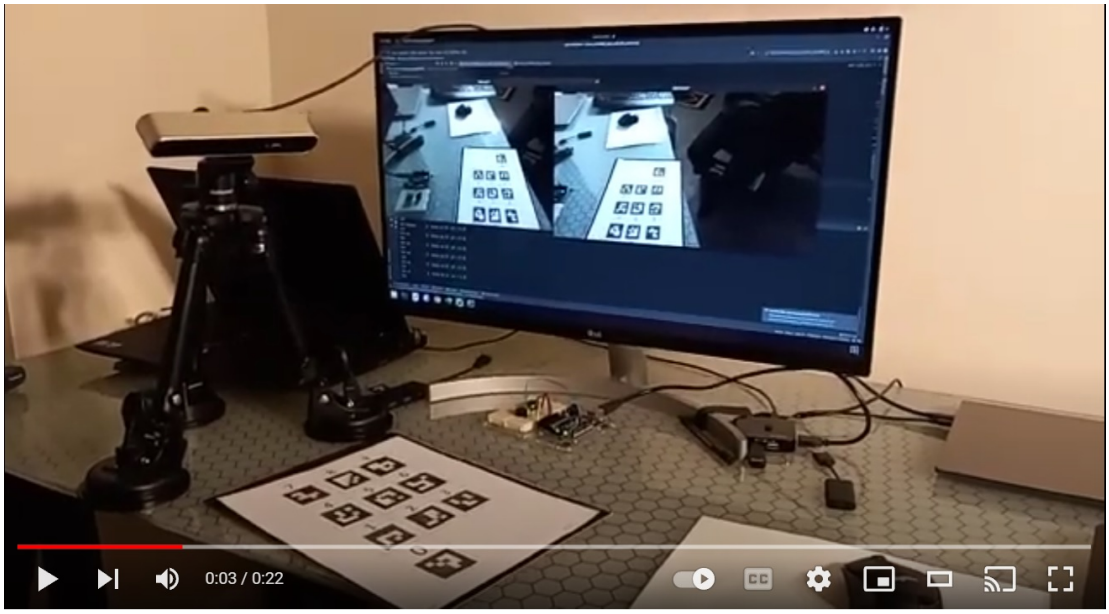

Fig. 6. ArUco keyboard: Stereo camera version (USB3 ZED camera), `https://youtu.be/ssbv2NqfAJg`

5 Conclusion

In this paper, we have presented a paper based numeric keypad using using ArUco markers. Full details of all source codes are given in the appendix. This system can be quite useful as a low-cost disposable keyboard for VR systems and mobile devices equipped with a camera. It has been observed that, the use of an IMU sensor greatly improves the overall system performance. Since almost all mobile devices, whether it is a phone or a tablet, do have IMU sensors, the improved IMU based keyboard can be used without any additional sensor or equipment. We have also implemented a stereo camera based system, but to the best of our knowledge mobile devices with stereo cameras are not widely available. The stereo camera based implementation is a feasible alternative for VR systems.

Acknowledgments.

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

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. Intelligent Service Robotics **10**, 185–194 (2017), https://doi.org/10.1007/s11370-017-0219-8
5. Billinghurst, M., Clark, A., Lee, G.: A survey of augmented reality. Foundations and Trends in Human–Computer Interaction **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 Recognition **47**(6), 2280–2292 (2014), https://doi.org/10.1016/j.patcog.2014.01.005

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);
}
```