**AYDIN ADNAN MENDERES UNIVERSITY**

**ENGINEERING FACULTY  
COMPUTER ENGINEERING DEPARTMENT**



**AUGMENTED REALITY WITH ARUCO MARKERS**

*Name: Burak*

*Surname: TÜZEL*

*Student No: 191805057*

*E-mail:* [*buraktsoftware@gmail.com*](mailto:buraktsoftware@gmail.com)

*Department: Computer Engineering*

*Grade: 4th grade*

*Class: CSE307*

**Supervisor:**

*Mahmut SİNECEN*

**ABSTRAC**T

***AUGMENTED REALITY WITH ARUCO MARKERS***

**Burak TÜZEL**

**Supervisor: Mahmut SİNECEN**

ArUco Markers are 2D binary-encoded fiducial patterns designed to be quickly located by computer vision systems. ArUco Marker helps the camera to understand the angle, height, depth, and other parameters and finds its use case in cool computer vision and augmented reality tasks. Augmented Reality with ArUco Markers Project’s first part consists of 3 stages. The first stage is Creating ArUco Markers using OpenCV library. The second stage is Detecting ArUco Markers. And the last stage is a Building Augmented Reality Environment by detecting ArUco Markers.

**Keywords: ArUco Markers, Augmented Reality, OpenCV, Pattern, Computer Vision**

**TABLE OF CONTENTS**

[ABSTRACT ii](#_Toc155735645)

[AUGMENTED REALITY WITH ARUCO MARKERS 1](#_Toc155735646)

[1. METHOD 1](#_Toc155735647)

[1.1 CREATING ARUCO MARKERS 1](#_Toc155735648)

[1.1.1 Screenshots of Execution 2](#_Toc155735649)

[1.2 DETECTING ARUCO MARKERS 3](#_Toc155735650)

[1.2.1 Screenshots of Execution 5](#_Toc155735651)

[1.3 AUGMENTED REALITY WITH ARUCO MARKERS 6](#_Toc155735652)

[1.3.1 Screenshots of Execution 11](#_Toc155735653)

[1.4 AUGMENTED REALITY WITH ARUCO MARKERS APPLICATION 12](#_Toc155735654)

[1.4.1 Screenshots of Running Application (exe) 19](#_Toc155735655)

[2. LITERATURE 20](#_Toc155735656)

[3. REFERENCES 21](#_Toc155735657)

AUGMENTED REALITY WITH ARUCO MARKERS

# METHOD

The method of this project is creating, detecting and using ArUco Markers to build Augmented Reality environments.

## CREATING ARUCO MARKERS

Required Libraries: OpenCV,Numpy

***# import the necessary packages***

*import numpy as np*

*import argparse*

*import cv2*

*import sys*

***# define names of ArUco Dictionaries Supported by OpenCV***

*ARUCO\_DICT = {*

*"DICT\_4X4\_50": cv2.aruco.DICT\_4X4\_50,*

*"DICT\_4X4\_100": cv2.aruco.DICT\_4X4\_100,*

*"DICT\_4X4\_250": cv2.aruco.DICT\_4X4\_250,…..}*

*def generate\_aruco\_marker(output\_path, marker\_id, marker\_type):*

***# define the ArUco dict.***

*aruco\_dict = cv2.aruco.getPredefinedDictionary(ARUCO\_DICT[marker\_type])*

***# allocate memory for the output ArUco dict.***

*tag = np.zeros((300, 300, 1), dtype="uint8")*

***# draw the ArUco tag on the output image***

*cv2.aruco.generateImageMarker(aruco\_dict, marker\_id, 300, tag, 1)*

***# write the generated ArUco tag to disk***

*cv2.imwrite(output\_path, tag)*

***# display the ArUco tag***

*cv2.imshow("ArUco Tag", tag)*

*cv2.waitKey(0)*

*if \_\_name\_\_ == "\_\_main\_\_":*

***# Construct the argument parser and parse the arguments***

*parser = argparse.ArgumentParser(description="Generate ArUco marker.")*

*parser.add\_argument("--output", default="tags/DICT\_5x5\_100\_id70.png", help="Output file path for the ArUco marker")*

*parser.add\_argument("--id", type=int, default=70, help="ID of the ArUco marker")*

*parser.add\_argument("—type, default="DICT\_5X5\_100", choices=ARUCO\_DICT.keys(), help="Type of the ArUco marker")*

*args = parser.parse\_args()*

***# Verify that the supplied ArUco tag exists and is supported by OpenCV***

*if ARUCO\_DICT.get(args.type, None) is None:*

*print("[INFO] ArUco tag of '{}' is not supported".format(args.type))*

*sys.exit(0)*

***# Generate the ArUco marker***

*generate\_aruco\_marker(args.output, args.id, args.type)*

The result of the execution of this code is giving us a ArUco Marker which

is in the 5X5 dictionary with id 70.

### Screenshots of Execution

A screenshot of a computer

Description automatically generated

## DETECTING ARUCO MARKERS

Required Libraries: OpenCV, Numpy, imutils

***# import the necessary packages***

*import imutils*

*from imutils.video import VideoStream*

*import cv2*

*import sys*

*import numpy as np*

*import time*

***# Specify the type of ArUCo tag***

*aruco\_type = "DICT\_5X5\_100"*

*ARUCO\_DICT = {*

*"DICT\_4X4\_50": cv2.aruco.DICT\_4X4\_50,*

*"DICT\_4X4\_100": cv2.aruco.DICT\_4X4\_100,….}*

*# Verify that the supplied ArUCo tag exists and is supported by OpenCV*

*if ARUCO\_DICT.get(aruco\_type, None) is None:*

*print("[INFO] ArUCo tag of '{}' is not supported".format(aruco\_type))*

*sys.exit(0)*

***# Load the ArUCo*** *dictionary****, grab the ArUCo parameters, and create arucoDetector***

*print("[INFO] detecting '{}' tags...".format(aruco\_type))*

*marker\_ids, marker\_corners, rejected\_candidates = [], [], []*

*arucoDict = cv2.aruco.getPredefinedDictionary(ARUCO\_DICT[aruco\_type])*

*arucoParams = cv2.aruco.DetectorParameters()*

*arucoDetector = cv2.aruco.ArucoDetector(arucoDict,arucoParams)*

***# initialize the video stream and allow the camera sensor to warm up***

*print("[INFO] starting video stream...")*

*vs = VideoStream(src=0).start()*

*time.sleep(2.0)*

***# loop over the frames from the video stream***

*while True:*

***# grab the frame from the threaded video stream and resize it***

***# to have a maximum width of 1000 pixels***

*frame = vs.read()*

*frame = imutils.resize(frame, width=1000)*

***# detect ArUco markers in the input frame***

*(corners, ids, rejected) = arucoDetector.detectMarkers(frame, marker\_corners, np.array(marker\_ids),rejected\_candidates)*

***# verify at least one ArUco marker was detected***

*if len(corners) > 0:*

***# flatten the ArUco IDs list***

*ids = ids.flatten()*

***# loop over the detected ArUCo corners***

*for (markerCorner, markerID) in zip(corners, ids):*

***# extract the marker corners (which are always returned***

***# in top-left, top-right, bottom-right, and bottom-left***

***# order)***

*corners = markerCorner.reshape((4, 2))*

*(topLeft, topRight, bottomRight, bottomLeft) = corners*

***# convert each of the (x, y)-coordinate pairs to integers***

*topRight = (int(topRight[0]), int(topRight[1]))*

*bottomRight = (int(bottomRight[0]), int(bottomRight[1]))*

*bottomLeft = (int(bottomLeft[0]), int(bottomLeft[1]))*

*topLeft = (int(topLeft[0]), int(topLeft[1]))*

***# draw the bounding box of the ArUco detection***

*cv2.line(frame, topLeft, topRight, (0, 255, 0), 2)*

*cv2.line(frame, topRight, bottomRight, (0, 255, 0), 2)*

*cv2.line(frame, bottomRight, bottomLeft, (0, 255, 0), 2)*

*cv2.line(frame, bottomLeft, topLeft, (0, 255, 0), 2)*

***# compute and draw the center (x, y)-coordinates of the***

***# ArUco marker***

*cX = int((topLeft[0] + bottomRight[0]) / 2.0)*

*cY = int((topLeft[1] + bottomRight[1]) / 2.0)*

*cv2.circle(frame, (cX, cY), 4, (0, 0, 255), -1)*

***# draw the ArUco marker ID on the frame***

*cv2.putText(frame, str(markerID),*

*(topLeft[0], topLeft[1] - 15),*

*cv2.FONT\_HERSHEY\_SIMPLEX,*

*0.5, (0, 255, 0), 2)*

***# show the output frame***

*cv2.imshow("Frame", frame)*

*key = cv2.waitKey(1) & 0xFF*

***# if the `q` key was pressed, break from the loop***

*if key == ord("q"):*

*break*

***# cleanup***

*cv2.destroyAllWindows()*

*vs.stop()*

### Screenshots of Execution

A hand holding a cell phone

Description automatically generatedA hand holding a phone with a qr code

Description automatically generatedA hand holding a phone

Description automatically generatedA hand holding a cell phone

Description automatically generated

It can also work in dark environments.

## AUGMENTED REALITY WITH ARUCO MARKERS

Required Libraries: OpenCV, Numpy, imutils

***# import the necessary packages***

*import numpy as np*

*import cv2*

*from imutils.video import VideoStream*

*from collections import deque*

*import imutils*

*import time*

***# Function to calculate frames per second (FPS)***

*def calculate\_fps(start\_time, num\_frames):*

*elapsed\_time = time.time() - start\_time*

*fps = num\_frames / elapsed\_time*

*return fps*

***# initialize our cached reference points***

*CACHED\_REF\_PTS = None*

*def find\_and\_warp(frame, source, cornerIDs, arucoDetector, marker\_corners, marker\_ids, rejected\_candidates, useCache=False):*

***# grab a reference to our cached reference points***

*global CACHED\_REF\_PTS*

***# grab the width and height of the frame and source image, respectively***

*(imgH, imgW) = frame.shape[:2]*

*(srcH, srcW) = source.shape[:2]*

***# detect AruCo markers in the input frame***

*(corners, ids, rejected) = arucoDetector.detectMarkers(frame, marker\_corners, np.array(marker\_ids), rejected\_candidates)*

***# if we \*did not\* find our four ArUco markers, initialize an***

***# empty IDs list, otherwise flatten the ID list***

*ids = np.array([]) if len(corners) != 4 else ids.flatten()*

***# initialize our list of reference points***

*refPts = []*

***# loop over the IDs of the ArUco markers in top-left,***

***# top-right, bottom-right, and bottom-left order***

*for i in cornerIDs:*

***# grab the index of the corner with the current ID***

*j = np.squeeze(np.where(ids == i))*

***# if we receive an empty list instead of an integer index,***

***# then we could not find the marker with the current ID***

*if j.size == 0:*

*continue*

***# otherwise, append the corner (x, y)-coordinates to our list***

***# of reference points***

*corner = np.squeeze(corners[j])*

*refPts.append(corner)*

***# check to see if we failed to find the four ArUco markers***

*if len(refPts) != 4:*

***# if we are allowed to use cached reference points, fall***

***# back on them***

*if useCache and CACHED\_REF\_PTS is not None:*

*refPts = CACHED\_REF\_PTS*

***# otherwise, we cannot use the cache and/or there are no***

***# previous cached reference points, so return early***

*else:*

*return None*

***# if we are allowed to use cached reference points, then update***

***# the cache with the current set***

*if useCache:*

*CACHED\_REF\_PTS = refPts*

***# unpack our ArUco reference points and use the reference points***

***# to define the \*destination\* transform matrix, making sure the***

***# points are specified in top-left, top-right, bottom-right, and***

***# bottom-left order***

*(refPtTL, refPtTR, refPtBR, refPtBL) = refPts*

*dstMat = [refPtTL[0], refPtTR[1], refPtBR[2], refPtBL[3]]*

*dstMat = np.array(dstMat)*

***# define the transform matrix for the \*source\* image in top-left,***

***# top-right, bottom-right, and bottom-left order***

*srcMat = np.array([[0, 0], [srcW, 0], [srcW, srcH], [0, srcH]])*

***# compute the homography matrix and then warp the source image to***

***# the destination based on the homography***

*(H, \_) = cv2.findHomography(srcMat, dstMat)*

*warped = cv2.warpPerspective(source, H, (imgW, imgH))*

***# construct a mask for the source image now that the perspective***

***# warp has taken place (we'll need this mask to copy the source***

***# image into the destination)***

*mask = np.zeros((imgH, imgW), dtype="uint8")*

*cv2.fillConvexPoly(mask, dstMat.astype("int32"), (255, 255, 255), cv2.LINE\_AA)*

***# this step is optional, but to give the source image a black***

***# border surrounding it when applied to the source image, you***

***# can apply a dilation operation***

*rect = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))*

*mask = cv2.dilate(mask, rect, iterations=2)*

***# create a three channel version of the mask by stacking it***

***# depth-wise, such that we can copy the warped source image***

***# into the input image***

*maskScaled = mask.copy() / 255.0*

*maskScaled = np.dstack([maskScaled] \* 3)*

***# copy the warped source image into the input image by***

***# (1) multiplying the warped image and masked together,***

***# (2) then multiplying the original input image with the***

***# mask (giving more weight to the input where there***

***# \*ARE NOT\* masked pixels), and (3) adding the resulting***

***# multiplications together***

*warpedMultiplied = cv2.multiply(warped.astype("float"), maskScaled)*

*imageMultiplied = cv2.multiply(frame.astype(float), 1.0 - maskScaled)*

*output = cv2.add(warpedMultiplied, imageMultiplied)*

*output = output.astype("uint8")*

***# return the output frame to the calling function***

*return output*

*input\_path = "AAA.mp4"*

*use\_cache = 1*

***# load the ArUCo dictionary and grab the ArUCo parameters***

*print("[INFO] initializing marker detector...")*

*marker\_ids, marker\_corners, rejected\_candidates = [], [], []*

*arucoDict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT\_5X5\_100)*

*arucoParams = cv2.aruco.DetectorParameters()*

*arucoDetector = cv2.aruco.ArucoDetector(arucoDict, arucoParams)*

***# initialize the video file stream***

*print("[INFO] accessing video stream...")*

*vf = cv2.VideoCapture(input\_path)*

***# initialize a queue to maintain the next frame from the video stream***

*Q = deque(maxlen=256)*

***# we need to have a frame in our queue to start our augmented reality***

***# pipeline, so read the next frame from our video file source and add***

***# it to our queue***

*(grabbed, source) = vf.read()*

*Q.appendleft(source)*

***# initialize the video stream and allow the camera sensor to warm up***

*print("[INFO] starting video stream...")*

*vs = VideoStream(src=0).start()*

*vs.stream.set(cv2.CAP\_PROP\_EXPOSURE,-6)*

*vs.stream.set(cv2.CAP\_PROP\_FPS,60)*

*time.sleep(2.0)*

***# loop over the frames from the video stream***

*num\_frames = 0*

*start\_time = time.time()*

*while len(Q) > 0:*

***# grab the frame from our video stream and resize it***

*frame = vs.read()*

*frame = imutils.resize(frame, width=600)*

***# attempt to find the ArUCo markers in the frame, and provided***

***# they are found, take the current source image and warp it onto***

***# input frame using our augmented reality technique***

*warped = find\_and\_warp(*

*frame, source,*

*cornerIDs=(24, 42, 66, 70),*

*arucoDetector=arucoDetector,*

*marker\_corners=marker\_corners,*

*marker\_ids=marker\_ids,*

*rejected\_candidates=rejected\_candidates,*

*useCache=use\_cache*

*)*

***# if the warped frame is not None, then we know (1) we found the***

***# four ArUCo markers and (2) the perspective warp was successfully***

***# applied***

*if warped is not None:*

***# set the frame to the output augment reality frame and then***

***# grab the next video file frame from our queue***

*frame = warped*

*source = Q.popleft()*

***# for speed/efficiency, we can use a queue to keep the next video***

***# frame queue ready for us -- the trick is to ensure the queue is***

***# always (or nearly full)***

*if len(Q) != Q.maxlen:*

***# read the next frame from the video file stream***

*(grabbed, nextFrame) = vf.read()*

***# if the frame was read (meaning we are not at the end of the***

***# video file stream), add the frame to our queue***

*if grabbed:*

*Q.append(nextFrame)*

***# show the output frame***

*cv2.imshow("Frame", frame)*

***# Calculate and display FPS***

*num\_frames += 1*

*if num\_frames % 30 == 0: # Update FPS every 30 frames*

*fps = calculate\_fps(start\_time, num\_frames)*

*print(f"FPS: {fps:.2f}")*

*key = cv2.waitKey(1) & 0xFF*

***# if the `q` key was pressed, break from the loop***

*if key == ord("q"):*

*break*

***# Cleanup***

*cv2.destroyAllWindows()*

*vs.stop()*

*vf.release()*

### A hand holding a phone with a screen on Description automatically generatedA hand holding a cell phone Description automatically generatedScreenshots of Execution

A cell phone with a screen

Description automatically generatedA person holding a cell phone

Description automatically generated

It can also work in dark environments.

## AUGMENTED REALITY WITH ARUCO MARKERS APPLICATION

Required Libraries: OpenCV, Numpy, imutils,PyQt5

***# Import necessary libraries***

*import sys*

*from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout, QPushButton, QSlider*

*from PyQt5.QtGui import QImage, QPixmap*

*from PyQt5.QtCore import QTimer, Qt*

*import cv2*

*from collections import deque*

*import imutils*

*import time*

*import numpy as np*

*import os*

***# Global variable to cache reference points for marker detection***

*CACHED\_REF\_PTS = None*

***# Function to calculate frames per second (FPS) and reset interval***

*def calculate\_fps(start\_time, num\_frames, reset\_interval=2.0):*

*elapsed\_time = time.time() - start\_time*

*fps = num\_frames / elapsed\_time*

*if elapsed\_time > reset\_interval:*

*start\_time = time.time()*

*num\_frames = 0*

*return fps, start\_time, num\_frames*

***# Function to find ArUco markers in a frame, perform perspective transformation, and create an AR effect***

*def find\_and\_warp(frame, source, cornerIDs, arucoDetector, marker\_corners, marker\_ids, rejected\_candidates, useCache=False):*

*global CACHED\_REF\_PTS*

*gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)*

*(imgH, imgW) = frame.shape[:2]*

*(srcH, srcW) = source.shape[:2]*

***# Detect ArUco markers in the frame***

*(corners, ids, rejected) = arucoDetector.detectMarkers(gray, marker\_corners, np.array(marker\_ids), rejected\_candidates)*

*ids = np.array([]) if len(corners) != 4 else ids.flatten()*

*refPts = []*

***# Extract reference points based on specified corner IDs***

*for i in cornerIDs:*

*j = np.squeeze(np.where(ids == i))*

*if j.size == 0:*

*continue*

*corner = np.squeeze(corners[j])*

*refPts.append(corner)*

***# If the correct number of reference points is not found, use cached points (if available)***

*if len(refPts) != 4:*

*if useCache and CACHED\_REF\_PTS is not None:*

*refPts = CACHED\_REF\_PTS*

*else:*

*return None*

***# Update the cache if applicable***

*if useCache:*

*CACHED\_REF\_PTS = refPts*

***# Order reference points and perform perspective transformation***

*(refPtTL, refPtTR, refPtBR, refPtBL) = refPts*

*dstMat = [refPtTL[0], refPtTR[1], refPtBR[2], refPtBL[3]]*

*dstMat = np.array(dstMat)*

*srcMat = np.array([[0, 0], [srcW, 0], [srcW, srcH], [0, srcH]])*

*(H, \_) = cv2.findHomography(srcMat, dstMat)*

*warped = cv2.warpPerspective(source, H, (imgW, imgH), flags=cv2.INTER\_LINEAR, borderMode=cv2.BORDER\_CONSTANT, borderValue=(0, 0, 0))*

***# Create a mask for blending the AR effect into the original frame***

*mask = np.zeros((imgH, imgW), dtype="uint8")*

*cv2.fillConvexPoly(mask, dstMat.astype("int32"), 255, cv2.LINE\_AA)*

*mask = cv2.dilate(mask, None, iterations=2)*

*mask = mask / 255.0*

*mask = np.stack([mask] \* 3, axis=-1)*

***# Blend the AR effect into the original frame***

*result = frame \* (1 - mask) + warped \* mask*

*result = result.astype("uint8")*

*return result*

***# Class for the AR application GUI***

*class ARApp(QWidget):*

*def \_\_init\_\_(self):*

*super().\_\_init\_\_()*

***# Initialize parameters and components***

*self.use\_cache = 1*

*self.marker\_ids, self.marker\_corners, self.rejected\_candidates = [], [], []*

*self.arucoDict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT\_5X5\_100)*

*self.arucoParams = cv2.aruco.DetectorParameters()*

*self.arucoDetector = cv2.aruco.ArucoDetector(self.arucoDict, self.arucoParams)*

*script\_dir = os.path.dirname(os.path.realpath("\_\_file\_\_"))*

*self.vf = cv2.VideoCapture(os.path.join(script\_dir, "AAA.mp4"))*

*self.num\_frames = 0*

*self.start\_time = time.time()*

*self.Q = deque(maxlen=256)*

*self.current\_webcam\_index = 0*

*self.initUI()*

***# Initialize the GUI components***

*def initUI(self):*

*self.setWindowTitle('Aruco Marker AR Video Embed')*

*self.setGeometry(100, 100, 800, 600)*

*layout = QVBoxLayout()*

*self.tv\_placeholder = QLabel('Aruco Marker Augmented Reality')*

*layout.addWidget(self.tv\_placeholder)*

*btn\_start\_cam = QPushButton('Start Cam!')*

*btn\_start\_cam.clicked.connect(self.on\_button\_press)*

*layout.addWidget(btn\_start\_cam)*

*btn\_switch\_cam = QPushButton('Switch Camera')*

*btn\_switch\_cam.clicked.connect(self.on\_switch\_camera)*

*layout.addWidget(btn\_switch\_cam)*

***# Slider for adjusting camera exposure***

*self.exposure\_slider = QSlider(Qt.Horizontal)*

*self.exposure\_slider.setMinimum(-15)*

*self.exposure\_slider.setMaximum(-1)*

*self.exposure\_slider.setValue(-5)*

*self.exposure\_slider.valueChanged.connect(self.on\_exposure\_change)*

*layout.addWidget(self.exposure\_slider)*

*self.image\_label = QLabel()*

*layout.addWidget(self.image\_label)*

*self.video\_label = QLabel()*

*layout.addWidget(self.video\_label)*

***# Timer for updating the GUI at regular intervals***

*self.timer = QTimer(self)*

*self.timer.timeout.connect(self.update)*

*self.timer.start(11)*

*self.fps\_label = QLabel('FPS: N/A')*

*layout.addWidget(self.fps\_label)*

*self.setLayout(layout)*

*self.show()*

***# Handle exposure change based on slider value***

*def on\_exposure\_change(self, value):*

*if hasattr(self, 'vs') and self.vs is not None:*

*exposure\_value = value*

*self.vs.set(cv2.CAP\_PROP\_EXPOSURE, exposure\_value)*

*time.sleep(0.5)*

***# Switch between different cameras (webcams)***

*def on\_switch\_camera(self):*

*self.vs.release()*

*self.current\_webcam\_index = (self.current\_webcam\_index + 1) % 2*

*self.vs = cv2.VideoCapture(self.current\_webcam\_index)*

*self.vs.set(cv2.CAP\_PROP\_EXPOSURE, -4.5)*

*time.sleep(2.0)*

***# Start the webcam and AR application***

*def on\_button\_press(self):*

*self.use\_cache = 1*

*self.marker\_ids, self.marker\_corners, self.rejected\_candidates = [], [], []*

*self.arucoDict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT\_5X5\_100)*

*self.arucoParams = cv2.aruco.DetectorParameters()*

*self.arucoDetector = cv2.aruco.ArucoDetector(self.arucoDict, self.arucoParams)*

***# Stop existing video streams***

*self.stop\_video\_streams()*

***# Start the default webcam (index 0)***

*self.vs = cv2.VideoCapture(0)*

*self.vs.set(cv2.CAP\_PROP\_EXPOSURE, -4.5)*

*time.sleep(2.0)*

*self.num\_frames = 0*

*self.start\_time = time.time()*

*self.Q = deque(maxlen=256)*

*self.source = None*

***# Load the AR video source (AAA.mp4)***

*self.vf = cv2.VideoCapture("AAA.mp4")*

*(grabbed, self.source) = self.vf.read()*

*self.Q.appendleft(self.source)*

***# Stop all video streams when closing the application***

*def stop\_video\_streams(self):*

*if hasattr(self, 'vs') and self.vs is not None:*

*self.vs.release()*

*del self.vs*

*if hasattr(self, 'ar\_video\_source') and self.ar\_video\_source is not None:*

*self.ar\_video\_source.release()*

*del self.ar\_video\_source*

*if hasattr(self, 'vf') and self.vf is not None:*

*self.vf.release()*

*del self.vf*

***# Handle the close event of the application***

*def closeEvent(self, event):*

*self.stop\_video\_streams()*

*event.accept()*

***# Update the GUI and perform AR processing***

*def update(self):*

*if hasattr(self, 'vs') and self.vs is not None:*

*if len(self.Q) > 0:*

*\_, frame = self.vs.read()*

*if frame is not None and frame.size > 0:*

*frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)*

*frame\_rgb = imutils.resize(frame\_rgb, width=600)*

*live\_warped = find\_and\_warp(*

*frame\_rgb, self.source,*

*cornerIDs=(24, 42, 66, 70),*

*arucoDetector=self.arucoDetector,*

*marker\_corners=self.marker\_corners,*

*marker\_ids=self.marker\_ids,*

*rejected\_candidates=self.rejected\_candidates,*

*useCache=self.use\_cache*

*)*

*if live\_warped is not None:*

*frame\_rgb = live\_warped*

*self.source = self.Q.popleft()*

*if len(self.Q) != self.Q.maxlen:*

*(grabbed, nextFrame) = self.vf.read()*

*if grabbed and nextFrame is not None and nextFrame.size > 0:*

*nextFrame = cv2.cvtColor(nextFrame, cv2.COLOR\_BGR2RGB)*

*self.Q.append(nextFrame)*

*else:*

*self.vf.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)*

*height, width, channel = frame\_rgb.shape*

*bytes\_per\_line = 3 \* width*

*q\_img = QImage(frame\_rgb.data, width, height, bytes\_per\_line, QImage.Format\_RGB888)*

*pixmap = QPixmap.fromImage(q\_img)*

*self.image\_label.setPixmap(pixmap)*

*self.num\_frames += 1*

*self.fps, self.start\_time, self.num\_frames = calculate\_fps(self.start\_time, self.num\_frames)*

*if self.num\_frames % 30 == 0:*

*self.fps\_label.setText(f'FPS: {self.fps:.2f}')*

***# Entry point of the application***

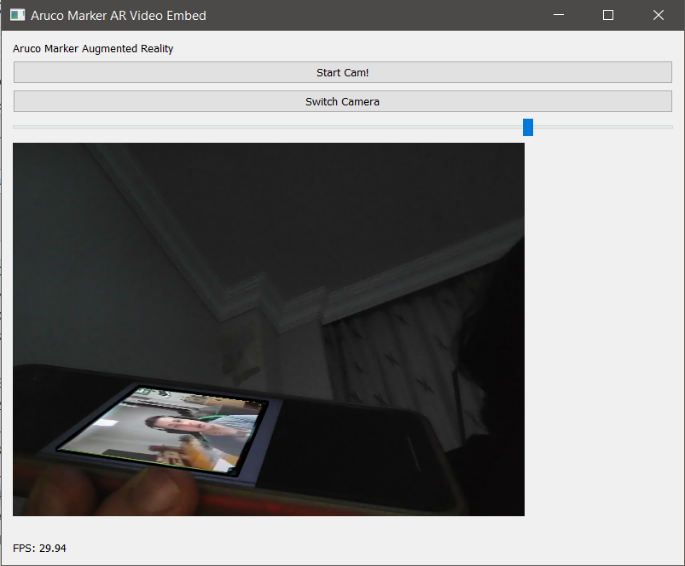
*if \_\_name\_\_ == '\_\_main\_\_':*

*app = QApplication(sys.argv)*

*ar\_app = ARApp()*

*sys.exit(app.exec\_())*

### Screenshots of Running Application (exe)



A screenshot of a computer

Description automatically generated

# LITERATURE

Youtube Videos:

- <https://www.youtube.com/watch?v=sg1bVJBjbng>

- <https://www.youtube.com/watch?v=UlM2bpqo_o0>

- <https://www.youtube.com/watch?v=GEWoGDdjlSc>

Websites:

- <https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html>

- <https://pyimagesearch.com/2020/12/14/generating-aruco-markers-with-opencv-and-python/>

- <https://pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/>

- <https://pyimagesearch.com/2021/01/11/opencv-video-augmented-reality/>

- <https://fab.cba.mit.edu/classes/865.21/people/zach/arucomarkers.html>

- <https://mecaruco2.readthedocs.io/en/latest/notebooks_rst/Aruco/aruco_basics.html>

# REFERENCES

[1] <https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html>

[2] <https://pyimagesearch.com/2020/12/14/generating-aruco-markers-with-opencv-and-python/>

[3] <https://pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/>

[4] <https://pyimagesearch.com/2021/01/11/opencv-video-augmented-reality/>