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
Department: Computer Engineering
Grade: 4th grade
Class: CSE307

Supervisor:
Mahmut SİNECEN

ABSTRACT

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

AUGMENTED REALITY WITH ARUCO MARKERS

1. METHOD

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

1.1 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,.....}
```

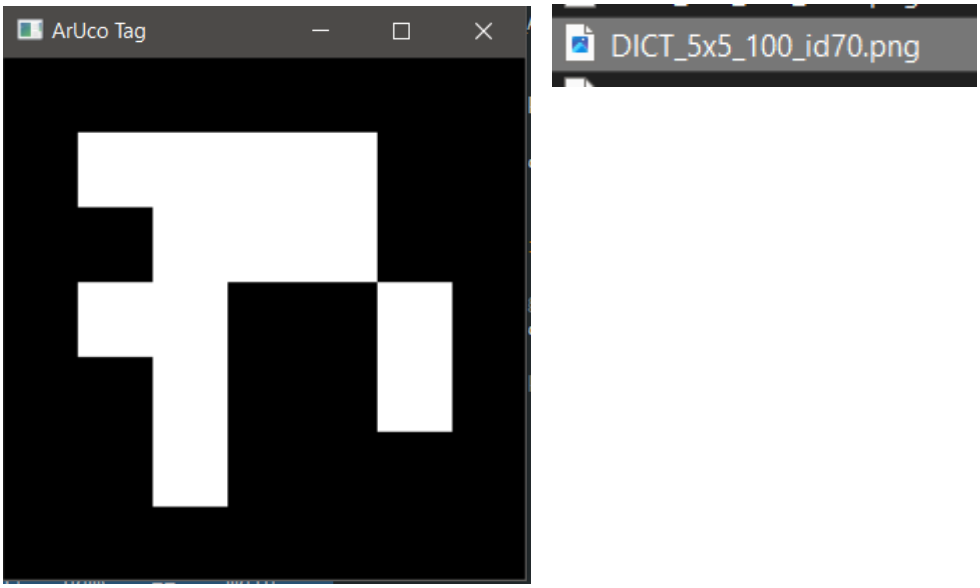
```
def generate_aruco_marker(output_path, marker_id, marker_type):
    # define the ArUco dict.
    aruco_dict = cv2.aruco.getPredefinedDictionary(ARUCO_DICT[marker_type])

    # 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.

1.1.1 Screenshots of Execution



1.2 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)
```

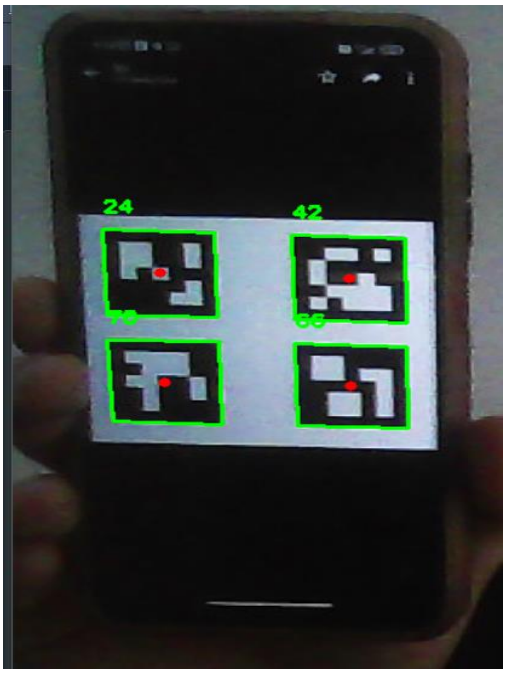
```
# Load the ArUCo dictionary, grab the ArUCo parameters, and create arucoDetector
print("[INFO] detecting '{}' tags...".format(aruco_type))
marker_ids, marker_corners, rejected_candidates = [], [], []
arucoDict = cv2.aruco.getPredefinedDictionary(ARUCO_DICT[aruco_type])
arucoParams = cv2.aruco.DetectorParameters()
arucoDetector = cv2.aruco.ArucoDetector(arucoDict,arucoParams)
```

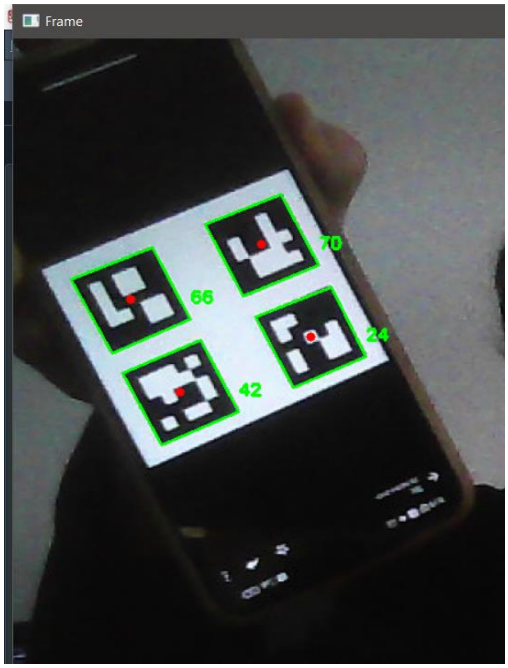
```
# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

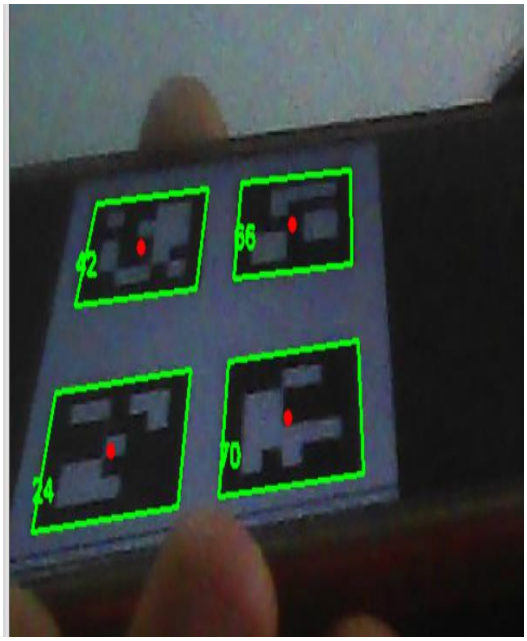
```
# loop over the frames from the video stream
while True:
   # grab the frame from the threaded video stream and resize it
   # to have a maximum width of 1000 pixels
   frame = vs.read()
   frame = imutils.resize(frame, width=1000)
   # detect ArUco markers in the input frame
   (corners, ids, rejected) = arucoDetector.detectMarkers(frame, marker_corners,
np.array(marker_ids),rejected_candidates)
```

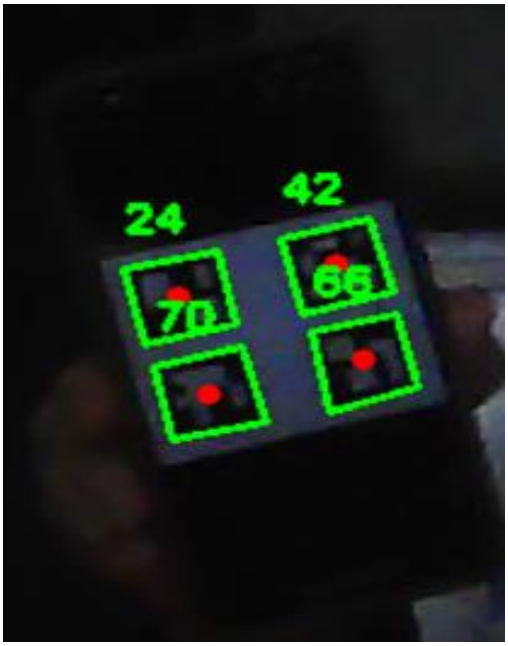
```
    # verify at least one ArUco marker was detected
    if len(corners) > 0:
        # flatten the ArUco IDs list
        ids = ids.flatten()
        # loop over the detected ArUCo corners
        for (markerCorner, markerID) in zip(corners, ids):
            # extract the marker corners (which are 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()
```

1.2.1 Screenshots of Execution









It can also work in dark environments.

1.3 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)
```

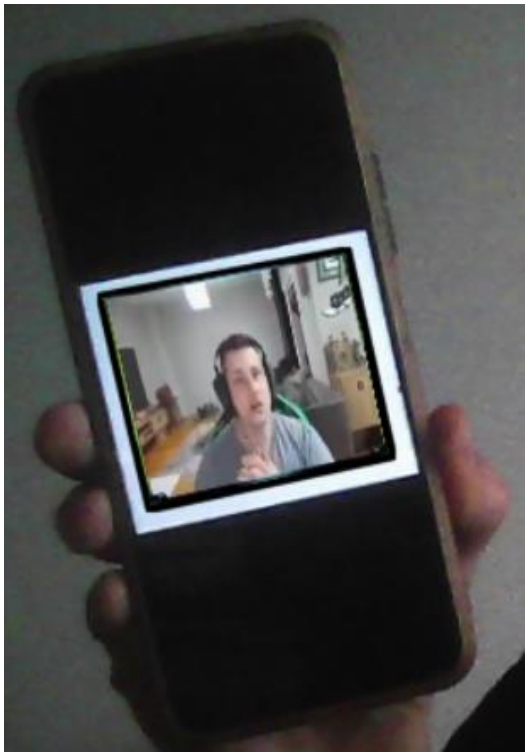
```
# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
vs.stream.set(cv2.CAP_PROP_EXPOSURE,-6)
vs.stream.set(cv2.CAP_PROP_FPS,60)
time.sleep(2.0)
```

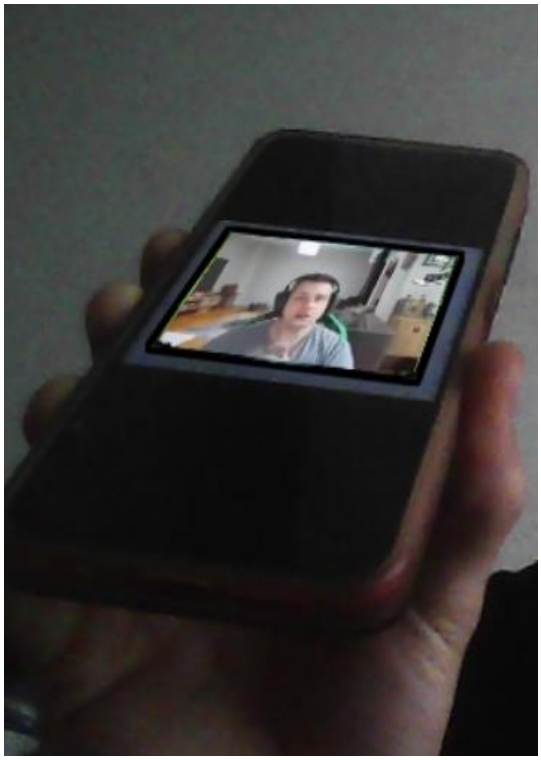
```
# loop over the frames from the video stream
num_frames = 0
start_time = time.time()

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()
```

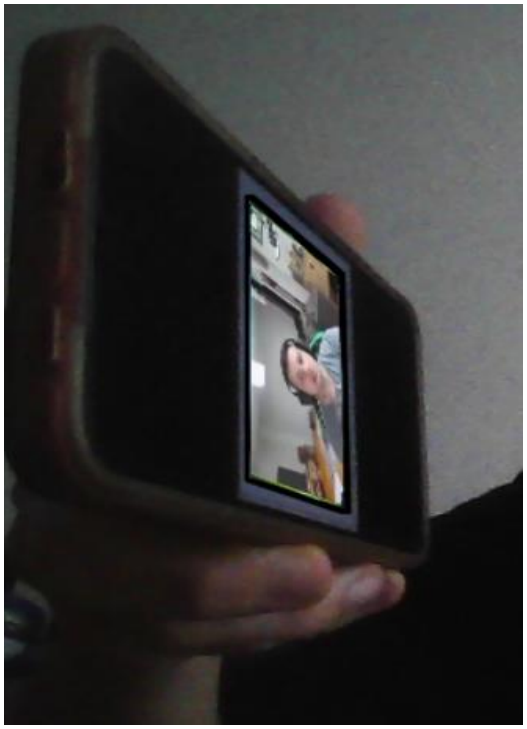
```
# for speed/efficiency, we can use a queue to keep the next video
# frame queue ready for us -- the trick is to ensure the queue is
# always (or nearly full)
if len(Q) != Q.maxlen:
    # read the next frame from the video file stream
    (grabbed, nextFrame) = vf.read()
    # if the frame was read (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
```

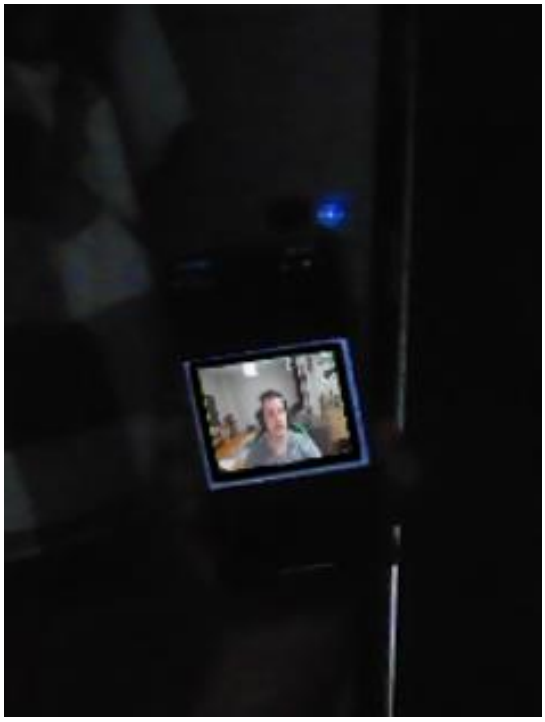
```
# Cleanup
cv2.destroyAllWindows()
vs.stop()
vf.release()
```

1.3.1 Screenshots of Execution









It can also work in dark environments.

1.4 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
```

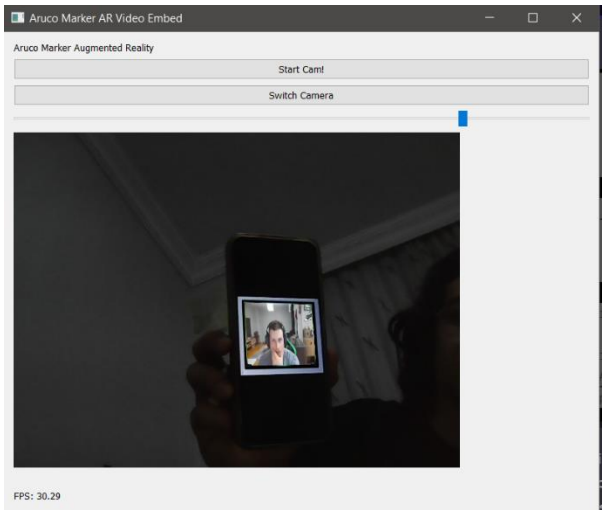
```
    # Handle the close event of the application
    def closeEvent(self, event):
        self.stop_video_streams()
        event.accept()
```

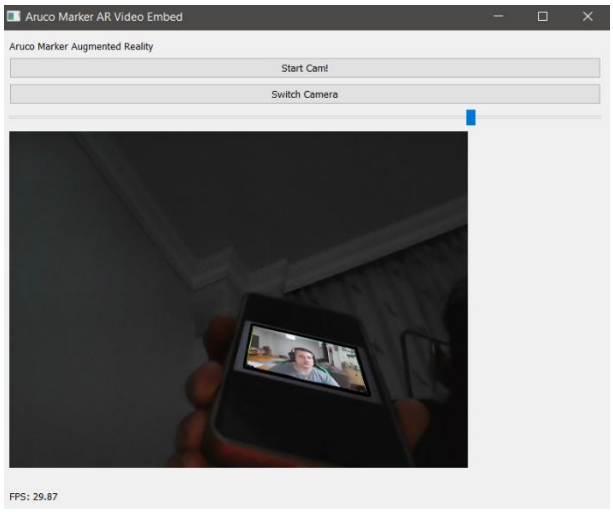
```
    # Update the GUI and perform AR processing
    def update(self):
        if hasattr(self, 'vs') and self.vs is not None:
            if len(self.Q) > 0:
                _, frame = self.vs.read()

                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)
```

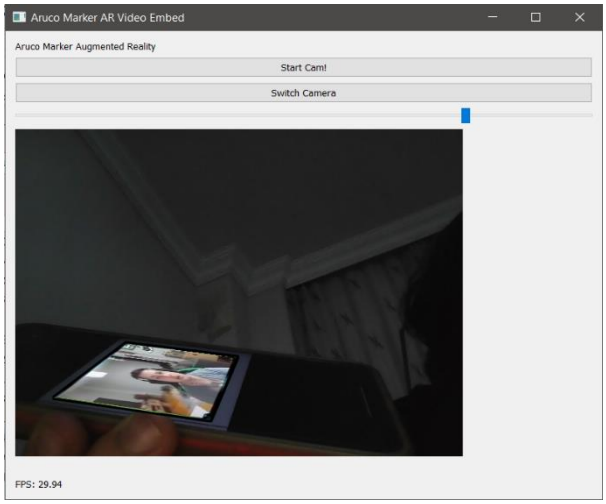
```
            pixmap = QPixmap.fromImage(q_img)
            self.image_label.setPixmap(pixmap)

            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}')
```

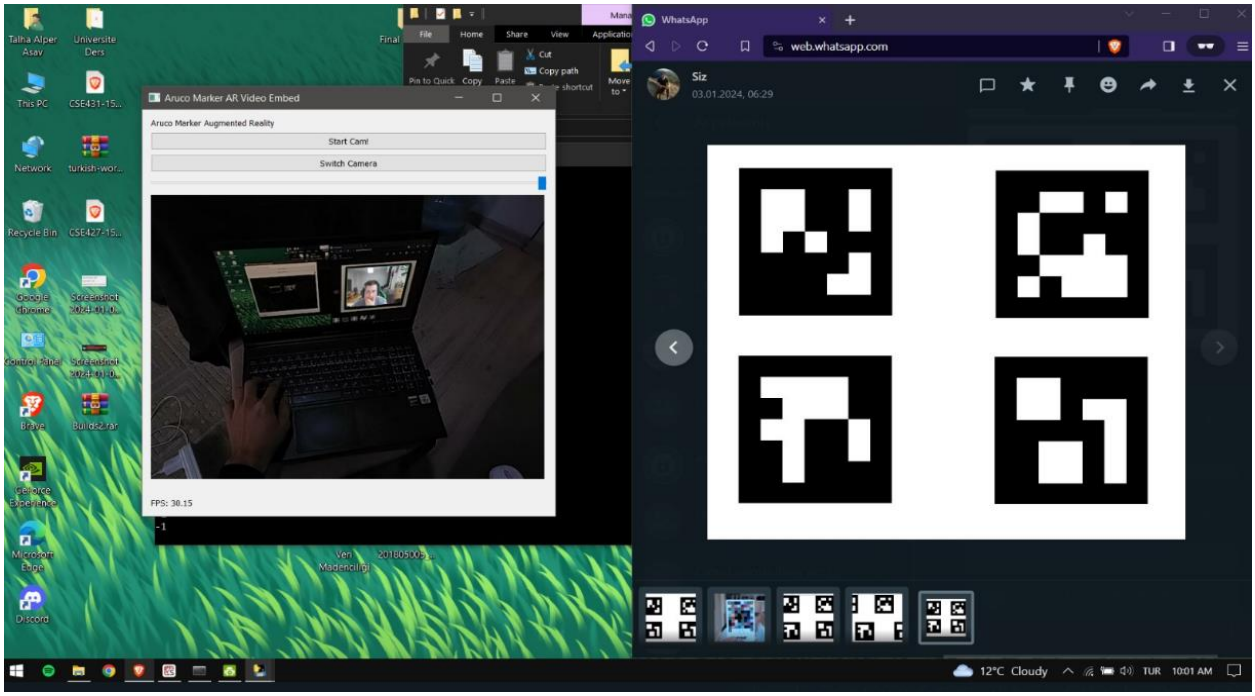
```
# Entry point of the application
if __name__ == '__main__':

    app = QApplication(sys.argv)

    ar_app = ARApp()

    sys.exit(app.exec_())
```

1.4.1 Screenshots of Running Application (exe)









2. 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

3. 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/