BAŞKENT ÜNİVERSİTESİ
BAŞKENT UNIVERSITY

SINAV SORU VE CEVAP KAĞIDI

QUESTION and ANSWER SHEET

Dersin Kodu: Course Code: **MAK540/SEY524/STS611**	Dersin Adı: Course Title: **MAKİNE MÜHENDİSLİĞİNDE ÖZEL KONULAR SAVUNMA SANAYİNDE OTONOM SİSTEMLER İNSANSIZ SAVUNMA SİSTEMLERİ**	Dersin Şubesi: Course Section: **01**
Ders Sorumlusu Öğretim Elemanının Adı Soyadı: Name of Lecturer: **ANDAÇ TÖRE ŞAMİLOĞLU**	Sınav Tarihi: Date of Exam: **08/ 06/ 2023**	Sınav Saati: Hour of Exam: **--:--**
Adı Soyadı: Name-Surname:	Öğrenci Numarası: Student Number:	
Fakülte/MYO/YO/Enstitü: Faculty/Vocational School/Institute:	Bölüm/Program: Department/Programme:	

2022-2023 Spring Semester

TAKEHOME FINAL EXAM (Due to 15.06.2023)

Duration: 1 Week

PROBLEM)

Use the same stereo camera set up in Midterm Examination. You are supposed to do the following:

1. Run the disparity_params.py code to adjust the disparity parameter using the GUI.
2. The disparity to depth linear relation (obtained by LSE method) is supplied in disparity_to_depth.py code. You should modify this code to record a lookup table. Use at least 5 different distances.
3. The obstacle avoidance example code is given. (obstacle_avoidance.py). This code finds the objects closer than the threshold and write warning messages. Modify this code to use look-up table you recorded in 2. The new code is supposed to draw the boundaries of the first 5 closest objects on the original image captured from left camera. And also write the distance on the middle of the objects.
4. Record a video of the code in 3 running.

- Submit your video and code project to moodle system (oys.baskent.edu.tr)
- If the file size is over 10MB, then please provide a download link.

disparity_params.py

```
# Autonomous Mobile Robots II
# Perception
# Stereo Camera
# OpenCV example for Disparity Parameters
import numpy as np
import cv2

# Check for left and right camera IDs
# These values can change depending on the system
CamL_id = 0  # Camera ID for left camera
CamR_id = 1  # Camera ID for right camera

CamL = cv2.VideoCapture(CamL_id)
CamR = cv2.VideoCapture(CamR_id)

# Reading the mapping values for stereo image rectification
cv_file = cv2.FileStorage("improved_params2.xml", cv2.FILE_STORAGE_READ)
Left_Stereo_Map_x = cv_file.getNode("Left_Stereo_Map_x").mat()
Left_Stereo_Map_y = cv_file.getNode("Left_Stereo_Map_y").mat()
Right_Stereo_Map_x = cv_file.getNode("Right_Stereo_Map_x").mat()
Right_Stereo_Map_y = cv_file.getNode("Right_Stereo_Map_y").mat()
cv_file.release()


def nothing(x):
    pass


cv2.namedWindow('disp', cv2.WINDOW_NORMAL)
cv2.resizeWindow('disp', 600, 600)

cv2.createTrackbar('numDisparities', 'disp', 1, 17, nothing)
cv2.createTrackbar('blockSize', 'disp', 5, 50, nothing)
cv2.createTrackbar('preFilterType', 'disp', 1, 1, nothing)
cv2.createTrackbar('preFilterSize', 'disp', 2, 25, nothing)
cv2.createTrackbar('preFilterCap', 'disp', 5, 62, nothing)
cv2.createTrackbar('textureThreshold', 'disp', 10, 100, nothing)
cv2.createTrackbar('uniquenessRatio', 'disp', 15, 100, nothing)
cv2.createTrackbar('speckleRange', 'disp', 0, 100, nothing)
cv2.createTrackbar('speckleWindowSize', 'disp', 3, 25, nothing)
cv2.createTrackbar('disp12MaxDiff', 'disp', 5, 25, nothing)
cv2.createTrackbar('minDisparity', 'disp', 5, 25, nothing)

# Creating an object of StereoBM algorithm
stereo = cv2.StereoBM_create()

while True:

    # Capturing and storing left and right camera images
    retL, imgL = CamL.read()
    retR, imgR = CamR.read()

    # Proceed only if the frames have been captured
    if retL and retR:
        imgR_gray = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)
        imgL_gray = cv2.cvtColor(imgL, cv2.COLOR_BGR2GRAY)

        # Applying stereo image rectification on the left image
        Left_nice = cv2.remap(imgL_gray,
                              Left_Stereo_Map_x,
                              Left_Stereo_Map_y,
                              cv2.INTER_LANCZOS4,
                              cv2.BORDER_CONSTANT,
```

```
                                0)

        # Applying stereo image rectification on the right image
        Right_nice = cv2.remap(imgR_gray,
                               Right_Stereo_Map_x,
                               Right_Stereo_Map_y,
                               cv2.INTER_LANCZOS4,
                               cv2.BORDER_CONSTANT,
                               0)

        # Updating the parameters based on the trackbar positions
        numDisparities = cv2.getTrackbarPos('numDisparities', 'disp') * 16
        blockSize = cv2.getTrackbarPos('blockSize', 'disp') * 2 + 5
        preFilterType = cv2.getTrackbarPos('preFilterType', 'disp')
        preFilterSize = cv2.getTrackbarPos('preFilterSize', 'disp') * 2 + 5
        preFilterCap = cv2.getTrackbarPos('preFilterCap', 'disp')
        textureThreshold = cv2.getTrackbarPos('textureThreshold', 'disp')
        uniquenessRatio = cv2.getTrackbarPos('uniquenessRatio', 'disp')
        speckleRange = cv2.getTrackbarPos('speckleRange', 'disp')
        speckleWindowSize = cv2.getTrackbarPos('speckleWindowSize', 'disp')
* 2
        disp12MaxDiff = cv2.getTrackbarPos('disp12MaxDiff', 'disp')
        minDisparity = cv2.getTrackbarPos('minDisparity', 'disp')

        # Setting the updated parameters before computing disparity map
        stereo.setNumDisparities(numDisparities)
        stereo.setBlockSize(blockSize)
        stereo.setPreFilterType(preFilterType)
        stereo.setPreFilterSize(preFilterSize)
        stereo.setPreFilterCap(preFilterCap)
        stereo.setTextureThreshold(textureThreshold)
        stereo.setUniquenessRatio(uniquenessRatio)
        stereo.setSpeckleRange(speckleRange)
        stereo.setSpeckleWindowSize(speckleWindowSize)
        stereo.setDisp12MaxDiff(disp12MaxDiff)
        stereo.setMinDisparity(minDisparity)

        # Calculating disparity using the StereoBM algorithm
        disparity = stereo.compute(Left_nice, Right_nice)
        # NOTE: Code returns a 16bit signed single channel image,
        # CV_16S containing a disparity map scaled by 16. Hence it
        # is essential to convert it to CV_32F and scale it down 16 times.

        # Converting to float32
        disparity = disparity.astype(np.float32)

        # Scaling down the disparity values and normalizing them
        disparity = (disparity / 16.0 - minDisparity) / numDisparities

        # Displaying the disparity map
        cv2.imshow("disp", disparity)

        # Close window using esc key
        if cv2.waitKey(1) == 27:
            break

    else:
        CamL = cv2.VideoCapture(CamL_id)
        CamR = cv2.VideoCapture(CamR_id)

print("Saving depth estimation parameters ......")

cv_file = cv2.FileStorage("./data/depth_estmation_params_py2.xml",
cv2.FILE_STORAGE_WRITE)
cv_file.write("numDisparities",numDisparities)
cv_file.write("blockSize",blockSize)
```

```
cv_file.write("preFilterType",preFilterType)
cv_file.write("preFilterSize",preFilterSize)
cv_file.write("preFilterCap",preFilterCap)
cv_file.write("textureThreshold",textureThreshold)
cv_file.write("uniquenessRatio",uniquenessRatio)
cv_file.write("speckleRange",speckleRange)
cv_file.write("speckleWindowSize",speckleWindowSize)
cv_file.write("disp12MaxDiff",disp12MaxDiff)
cv_file.write("minDisparity",minDisparity)
cv_file.write("M",39.075)
cv_file.release()
```

Disparity_to_depth.py

```
# Autonomous Mobile Robots II
# Perception
# Stereo Camera
# OpenCV example for Disparity to depth

import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt

# Check for left and right camera IDs
# These values can change depending on the system
CamL_id = 0  # Camera ID for left camera
CamR_id = 1  # Camera ID for right camera

CamL = cv2.VideoCapture(CamL_id)
CamR = cv2.VideoCapture(CamR_id)

# Reading the mapping values for stereo image rectification
cv_file = cv2.FileStorage("improved_params2.xml", cv2.FILE_STORAGE_READ)
Left_Stereo_Map_x = cv_file.getNode("Left_Stereo_Map_x").mat()
Left_Stereo_Map_y = cv_file.getNode("Left_Stereo_Map_y").mat()
Right_Stereo_Map_x = cv_file.getNode("Right_Stereo_Map_x").mat()
Right_Stereo_Map_y = cv_file.getNode("Right_Stereo_Map_y").mat()
cv_file.release()

# These parameters can vary according to the setup
# Keeping the target object at max_dist we store disparity values
# after every sample_delta distance.
max_dist = 100  # max distance to keep the target object (in cm)
min_dist = 30  # Minimum distance the stereo setup can measure (in cm)
sample_delta = 15  # Distance between two sampling points (in cm)

Z = max_dist
Value_pairs = []

disp_map = np.zeros((600, 600, 3))

# Reading the stored the StereoBM parameters
cv_file = cv2.FileStorage("./data/depth_estmation_params_py2.xml",
cv2.FILE_STORAGE_READ)
numDisparities = int(cv_file.getNode("numDisparities").real())
blockSize = int(cv_file.getNode("blockSize").real())
preFilterType = int(cv_file.getNode("preFilterType").real())
preFilterSize = int(cv_file.getNode("preFilterSize").real())
preFilterCap = int(cv_file.getNode("preFilterCap").real())
textureThreshold = int(cv_file.getNode("textureThreshold").real())
uniquenessRatio = int(cv_file.getNode("uniquenessRatio").real())
speckleRange = int(cv_file.getNode("speckleRange").real())
speckleWindowSize = int(cv_file.getNode("speckleWindowSize").real())
disp12MaxDiff = int(cv_file.getNode("disp12MaxDiff").real())
minDisparity = int(cv_file.getNode("minDisparity").real())
M = cv_file.getNode("M").real()
cv_file.release()


# Defining callback functions for mouse events
def mouse_click(event, x, y, flags, param):
    global Z
    if event == cv2.EVENT_LBUTTONDBLCLK:
        if disparity[y, x] > 0:
            Value_pairs.append([Z, disparity[y, x]])
```

```
            print("Distance: %r cm  | Disparity: %r" % (Z, disparity[y,
x]))
            Z -= sample_delta


cv2.namedWindow('disp', cv2.WINDOW_NORMAL)
cv2.resizeWindow('disp', 600, 600)
cv2.namedWindow('left image', cv2.WINDOW_NORMAL)
cv2.resizeWindow('left image', 600, 600)
cv2.setMouseCallback('disp', mouse_click)

# Creating an object of StereoBM algorithm
stereo = cv2.StereoBM_create()

while True:

    # Capturing and storing left and right camera images
    retR, imgR = CamR.read()
    retL, imgL = CamL.read()

    # Proceed only if the frames have been captured
    if retL and retR:
        imgR_gray = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)
        imgL_gray = cv2.cvtColor(imgL, cv2.COLOR_BGR2GRAY)

        # Applying stereo image rectification on the left image
        Left_nice = cv2.remap(imgL_gray,
                              Left_Stereo_Map_x,
                              Left_Stereo_Map_y,
                              cv2.INTER_LANCZOS4,
                              cv2.BORDER_CONSTANT,
                              0)

        # Applying stereo image rectification on the right image
        Right_nice = cv2.remap(imgR_gray,
                               Right_Stereo_Map_x,
                               Right_Stereo_Map_y,
                               cv2.INTER_LANCZOS4,
                               cv2.BORDER_CONSTANT,
                               0)

        # Setting the updated parameters before computing disparity map
        stereo.setNumDisparities(numDisparities)
        stereo.setBlockSize(blockSize)
        stereo.setPreFilterType(preFilterType)
        stereo.setPreFilterSize(preFilterSize)
        stereo.setPreFilterCap(preFilterCap)
        stereo.setTextureThreshold(textureThreshold)
        stereo.setUniquenessRatio(uniquenessRatio)
        stereo.setSpeckleRange(speckleRange)
        stereo.setSpeckleWindowSize(speckleWindowSize)
        stereo.setDisp12MaxDiff(disp12MaxDiff)
        stereo.setMinDisparity(minDisparity)

        # Calculating disparity using the StereoBM algorithm
        disparity = stereo.compute(Left_nice, Right_nice)
        # NOTE: compute returns a 16bit signed single channel image,
        # CV_16S containing a disparity map scaled by 16. Hence it
        # is essential to convert it to CV_16S and scale it down 16 times.

        # Converting to float32
        disparity = disparity.astype(np.float32)

        # Scaling down the disparity values and normalizing them
        disparity = (disparity / 16.0 - minDisparity) / numDisparities
```

```
        # Displaying the disparity map
        cv2.imshow("disp", disparity)
        cv2.imshow("left image", imgL)

        if cv2.waitKey(1) == 27:
            break

        if Z < min_dist:
            break

    else:
        CamL = cv2.VideoCapture(CamL_id)
        CamR = cv2.VideoCapture(CamR_id)

# solving for M in the following equation
# ||    depth = M * (1/disparity)   ||
# for N data points coeff is Nx2 matrix with values
# 1/disparity, 1
# and depth is Nx1 matrix with depth values

value_pairs = np.array(Value_pairs)
z = value_pairs[:, 0]
disp = value_pairs[:, 1]
disp_inv = 1 / disp

# Plotting the relation depth and corresponding disparity
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
ax1.plot(disp, z, 'o-')
ax1.set(xlabel='Normalized disparity value', ylabel='Depth from camera
(cm)',
        title='Relation between depth \n and corresponding disparity')
ax1.grid()
ax2.plot(disp_inv, z, 'o-')
ax2.set(xlabel='Inverse disparity value (1/disp) ', ylabel='Depth from
camera (cm)',
        title='Relation between depth \n and corresponding inverse
disparity')
ax2.grid()
plt.show()

# Solving for M using least square fitting with QR decomposition method
coeff = np.vstack([disp_inv, np.ones(len(disp_inv))]).T
ret, sol = cv2.solve(coeff, z, flags=cv2.DECOMP_QR)
M = sol[0, 0]
C = sol[1, 0]
print("Value of M = ", M)

# Storing the updated value of M along with the stereo parameters
cv_file = cv2.FileStorage("./data/depth_estmation_params_py2.xml",
cv2.FILE_STORAGE_WRITE)
cv_file.write("numDisparities", numDisparities)
cv_file.write("blockSize", blockSize)
cv_file.write("preFilterType", preFilterType)
cv_file.write("preFilterSize", preFilterSize)
cv_file.write("preFilterCap", preFilterCap)
cv_file.write("textureThreshold", textureThreshold)
cv_file.write("uniquenessRatio", uniquenessRatio)
cv_file.write("speckleRange", speckleRange)
cv_file.write("speckleWindowSize", speckleWindowSize)
cv_file.write("disp12MaxDiff", disp12MaxDiff)
cv_file.write("minDisparity", minDisparity)
cv_file.write("M", M)
cv_file.release()
```

Obstacle_avoidance.py

```
# Autonomous Mobile Robots II
# Perception
# Stereo Camera
# OpenCV example for Obstacle Avoidance

import numpy as np
import cv2

# Check for left and right camera IDs
# These values can change depending on the system
CamL_id = 0  # Camera ID for left camera
CamR_id = 1  # Camera ID for right camera

CamL = cv2.VideoCapture(CamL_id)
CamR = cv2.VideoCapture(CamR_id)

# Reading the mapping values for stereo image rectification
cv_file = cv2.FileStorage("improved_params2.xml", cv2.FILE_STORAGE_READ)
Left_Stereo_Map_x = cv_file.getNode("Left_Stereo_Map_x").mat()
Left_Stereo_Map_y = cv_file.getNode("Left_Stereo_Map_y").mat()
Right_Stereo_Map_x = cv_file.getNode("Right_Stereo_Map_x").mat()
Right_Stereo_Map_y = cv_file.getNode("Right_Stereo_Map_y").mat()
cv_file.release()

disparity = None
depth_map = None

# These parameters can vary according to the setup
max_depth = 200  # maximum distance the setup can measure (in cm)
min_depth = 30  # minimum distance the setup can measure (in cm)
depth_thresh = 50.0  # Threshold for SAFE distance (in cm)

# Reading the stored the StereoBM parameters
cv_file = cv2.FileStorage("./data/depth_estmation_params_py2.xml",
cv2.FILE_STORAGE_READ)
numDisparities = int(cv_file.getNode("numDisparities").real())
blockSize = int(cv_file.getNode("blockSize").real())
preFilterType = int(cv_file.getNode("preFilterType").real())
preFilterSize = int(cv_file.getNode("preFilterSize").real())
preFilterCap = int(cv_file.getNode("preFilterCap").real())
textureThreshold = int(cv_file.getNode("textureThreshold").real())
uniquenessRatio = int(cv_file.getNode("uniquenessRatio").real())
speckleRange = int(cv_file.getNode("speckleRange").real())
speckleWindowSize = int(cv_file.getNode("speckleWindowSize").real())
disp12MaxDiff = int(cv_file.getNode("disp12MaxDiff").real())
minDisparity = int(cv_file.getNode("minDisparity").real())
M = cv_file.getNode("M").real()
cv_file.release()


# mouse callback function
def mouse_click(event, x, y, flags, param):
    global Z
    if event == cv2.EVENT_LBUTTONDBLCLK:
        print("Disparity= %.4f" % disparity[y, x])
        print("Distance = %.4f cm" % depth_map[y, x])


cv2.namedWindow('disp', cv2.WINDOW_NORMAL)
cv2.resizeWindow('disp', 600, 600)
cv2.setMouseCallback('disp', mouse_click)
```

```
output_canvas = None

# Creating an object of StereoBM algorithm
stereo = cv2.StereoBM_create()


def obstacle_avoid():
    # Mask to segment regions with depth less than threshold
    mask = cv2.inRange(depth_map, 10, depth_thresh)

    # Check if a significantly large obstacle is present and filter out
smaller noisy regions
    if np.sum(mask) / 255.0 > 0.01 * mask.shape[0] * mask.shape[1]:

        # Contour detection
        contours, _ = cv2.findContours(mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
        cnts = sorted(contours, key=cv2.contourArea, reverse=True)

        # Check if detected contour is significantly large (to avoid
multiple tiny regions)
        if cv2.contourArea(cnts[0]) > 0.01 * mask.shape[0] * mask.shape[1]:
            x, y, w, h = cv2.boundingRect(cnts[0])

            # finding average depth of region represented by the largest
contour
            mask2 = np.zeros_like(mask)
            cv2.drawContours(mask2, cnts, 0, (255), -1)

            # Calculating the average depth of the object closer than the
safe distance
            depth_mean, _ = cv2.meanStdDev(depth_map, mask=mask2)

            # Display warning text
            cv2.putText(output_canvas, "WARNING !", (x + 5, y - 40), 1, 2,
(0, 0, 255), 2, 2)
            cv2.putText(output_canvas, "Object at", (x + 5, y), 1, 2, (100,
10, 25), 2, 2)
            cv2.putText(output_canvas, "%.2f cm" % depth_mean, (x + 5, y +
40), 1, 2, (100, 10, 25), 2, 2)

    else:
        cv2.putText(output_canvas, "SAFE!", (100, 100), 1, 3, (0, 255, 0),
2, 3)

    cv2.imshow('output_canvas', output_canvas)


while True:
    retR, imgR = CamR.read()
    retL, imgL = CamL.read()

    if retL and retR:

        output_canvas = imgL.copy()

        imgR_gray = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)
        imgL_gray = cv2.cvtColor(imgL, cv2.COLOR_BGR2GRAY)

        # Applying stereo image rectification on the left image
        Left_nice = cv2.remap(imgL_gray,
                              Left_Stereo_Map_x,
                              Left_Stereo_Map_y,
                              cv2.INTER_LANCZOS4,
                              cv2.BORDER_CONSTANT,
                              0)
```

```
        # Applying stereo image rectification on the right image
        Right_nice = cv2.remap(imgR_gray,
                               Right_Stereo_Map_x,
                               Right_Stereo_Map_y,
                               cv2.INTER_LANCZOS4,
                               cv2.BORDER_CONSTANT,
                               0)

        # Setting the updated parameters before computing disparity map
        stereo.setNumDisparities(numDisparities)
        stereo.setBlockSize(blockSize)
        stereo.setPreFilterType(preFilterType)
        stereo.setPreFilterSize(preFilterSize)
        stereo.setPreFilterCap(preFilterCap)
        stereo.setTextureThreshold(textureThreshold)
        stereo.setUniquenessRatio(uniquenessRatio)
        stereo.setSpeckleRange(speckleRange)
        stereo.setSpeckleWindowSize(speckleWindowSize)
        stereo.setDisp12MaxDiff(disp12MaxDiff)
        stereo.setMinDisparity(minDisparity)

        # Calculating disparity using the StereoBM algorithm
        disparity = stereo.compute(Left_nice, Right_nice)
        # NOTE: compute returns a 16bit signed single channel image,
        # CV_16S containing a disparity map scaled by 16. Hence it
        # is essential to convert it to CV_16S and scale it down 16 times.

        # Converting to float32
        disparity = disparity.astype(np.float32)

        # Normalizing the disparity map
        disparity = (disparity / 16.0 - minDisparity) / numDisparities

        depth_map = M / (disparity)  # for depth in (cm)

        # print(np.min(disparity))
        # print(np.max(disparity))
        # print('-------------------------------------------------------')
        # print(np.min(depth_map))
        # print(np.max(depth_map))
        # print('-------------------------------------------------------')

        mask_temp = cv2.inRange(depth_map, min_depth, max_depth)
        depth_map = cv2.bitwise_and(depth_map, depth_map, mask=mask_temp)

        # print(np.min(depth_map))
        # print(np.max(depth_map))
        # print('---------------------***************----------------------
')

        obstacle_avoid()

        cv2.resizeWindow("disp", 700, 700)
        cv2.imshow("disp", disparity)

        if cv2.waitKey(1) == 27:
            break

    else:
        CamL = cv2.VideoCapture(CamL_id)
        CamR = cv2.VideoCapture(CamR_id)
```