# Mini Project Vision System

## Seed Lab Team 3 – Fall 2020

Trey S . - Localization
Jackson W. - Simulation
Geoff M - System Integration
Eric S. - Vision

The goal with our Mini Project vision system is to write code that we can later use in the other demos. While it would have been simplest to look at corner location relative to the screen, we instead wanted to focus on extracting a 3d position for the marker and use it to find our quadrant. Later on, this will make it much easier to know the relative position of the marker to the robot. This approach include a few caveats, mainly being requiring a camera calibration to accurately find the position of the marker.

### Camera Calibration Script

No camera, especially the ones we are using for SEED, has perfect optics that create no distortion in the image. Every camera will have some amount of fisheye distortion. This distortion is very visible on cameras such as GoPros. To accurately calculate the (x,y,z) coordinate of the ArUco markers, we have to account for this distortion. Open CV has built in functions to calculate and correct this distortion. Our camera calibration script captures the data needed for distortion correction, calculates the distortion, and then displays the undistorted image.

The code starts by importing the required libraries and gives the user some info about the script. Some constants are also set for the calibration grid that will be used for calibration.

```
#   Eric Sobczak
#   9/6/2022
#   SEED Lab Python Assignment 2+

# Peforming camera calibration is vital to accurately find the location of
# ArUco markers in 3D space. This script generates a .yaml calibration file
# that contains calibration data for any PiCamera. This scripts begins by
# running a photo application to collect calibration data. Simply aim the
# camera at a checkerboard and press "c" multiple times from different angles.
# After at least 6 images, press the d key. The program will look for the
# checkboard in each image and note the locations of all the corners.
# Afterwards opencv camera calibration script will run and generate a new
# calibration. This is save to a yaml file, and then reloaded. A preview of
# the calibration results is then shown to the user.
```

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import numpy as np
import time
import cv2

# Set the parameters for finding sub-pixel corners, max 30 cycles, max error
tolerance 0.001
subPixelCriteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
0.001)
#Set Size of Chessboard
gridW = 9   # 10 - 1
gridH = 6   # 7  - 1
checkerSpacing = 18.1 #Size of checkerboard points in mm
```

Arrays need to be setup that will hold the data needed to perform the calibration analysis.

```
#Define the checkerboard in world coordanites
checkerPoints = np.zeros((gridW*gridH, 3 ), np.float32)  #Create numpy array full
of zeros for all checkerboard points
checkerPoints[:,: 2 ] = np .mgrid[ 0 :gridW, 0 :gridH].T.reshape( -1 , 2 )  #Fill
the grid with coordainted of points
checkerPoints = checkerPoints * checkerSpacing #Multiply out grid by spacing

#Create holders 3D and 2D Coordanites
worldCordChess = [] # 3D points in the world coordinate system
imgCordChess = []   # 2D points in the image plane
```

This script simply sets up the camera and allows for an easy way to capture photos. It is a simple camera app.

```
#Capture images for calibration
camera = PiCamera()
camera.resolution = (1296, 976)
camera.framerate = 30
rawCapture = PiRGBArray(camera, size=(1296, 976))

# allow the camera to warmup
print("Starting PiCamera.....")
time.sleep(0.1)

#Idex for taking images
photoIndex = 0

print("Press 'c' to take a photo")
print("Press 'd' to finish and calibrate")

#Capture photos for calibration
for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
    imageVideo = frame.array

    #Display the video
    cv2.imshow("Frame", imageVideo)
    key = cv2.waitKey(1) & 0xFF

    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)

    # if the `d' key was pressed, break from the loop
    if key == ord("d"):
        cv2.destroyAllWindows()
        break

    # if the 'c' key was pressed, capture an image
    if key == ord("c"):
        fileName = "calibrationImage" + str(photoIndex) + ".jpg"
        cv2.imwrite(fileName, imageVideo)
        print ("Saving " + fileName)
        photoIndex += 1
```

This script loops through the photos that were just taken and find the points of the checkboard calibration grid. Comparing these points is how the calibration is calculated. Subpixel is used to get a more accurate point.

```
#Loop through all the captured photos and perform calibration steps
calibrateIndex = 0
while calibrateIndex < photoIndex:
    #Load from captured photo
    fileName = "calibrationImage" + str(calibrateIndex) + ".jpg"
    img = cv2.imread(fileName)

    #Get image parameters
    imgHeight, imgWidth = img.shape[0], img.shape[1]

    #Get grayscale image
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    #Perform scan for checkerbaord
    ret, corners = cv2.findChessboardCorners(gray, (gridW,gridH),None)
    if ret == True:
        print("Checkerboard found in " + fileName + ", running subpixel
correction")
        #Use subpixel constraints to get more accurate corners
        cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),subPixelCriteria)

        #Add corners to lists from earlier
        worldCordChess.append(checkerPoints)
        imgCordChess.append(corners)

        #Draw and display the corners on the chessbaord
        cv2.drawChessboardCorners(img, (gridW,gridH), corners, ret)
        cv2.namedWindow(('CheckerView - ' + fileName), cv2.WINDOW_NORMAL)
        cv2.resizeWindow(('CheckerView - ' + fileName), 640, 480)
        cv2.imshow(('CheckerView - ' + fileName),img)
        cv2.waitKey(2000)

    else:
        print("No checkerboard found in " + fileName)

    calibrateIndex += 1
cv2.destroyAllWindows()
```

Perform the analysis of all the found points and solve for the distortion. Then save it to a .yaml file.

```
#Generate the calibration
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(worldCordChess, imgCordChess,
gray.shape[::-1], None, None)
#Generate a new camera matrix to account for a different frame size with the
distortion
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (imgWidth,
imgHeight), 1, (imgWidth, imgHeight))

#Save calibration to file
calibrationFile = "calibration_test2.yaml"
cv_file=cv2.FileStorage(calibrationFile, cv2.FILE_STORAGE_WRITE)
cv_file.write("camera_matrix", mtx)
cv_file.write("dist_coeff", dist)
cv_file.write("new_camera_matrix", newcameramtx)
cv_file.release()

#Load calibration file
calibrationFileLoad = "calibration_test.yaml"
cv_file_load = cv2.FileStorage(calibrationFileLoad, cv2.FILE_STORAGE_READ)
mtx_load = cv_file_load.getNode("camera_matrix").mat()
```

```
dist_load = cv_file_load.getNode("dist_coeff").mat()
newcameramtx_load = cv_file_load.getNode("new_camera_matrix").mat()
cv_file_load.release()
```

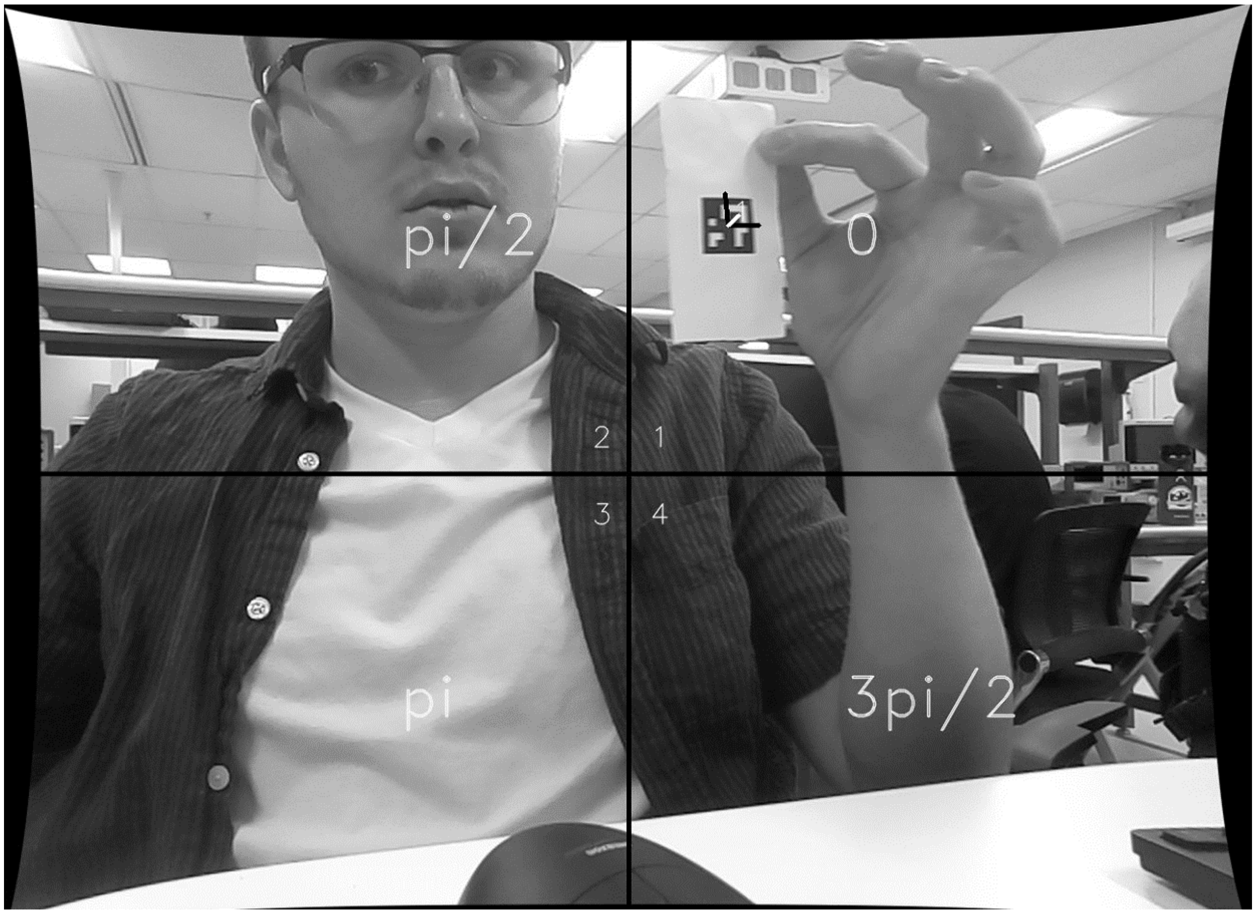
Give a preview of the calibration results.

```
#Start video capture again
print("Running preview of distortion correction, press 'd' to exit")
for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
    imageVideo = frame.array

    dst1 = cv2.undistort(imageVideo, mtx_load, dist_load, None,
newcameramtx_load)

    #Perform the ArUco detection and display the video
    cv2.imshow("Corrected", dst1)
    cv2.imshow("Uncorrected", imageVideo)
    key = cv2.waitKey(1) & 0xFF

    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)

    # if the `d' key was pressed, break from the loop
    if key == ord("d"):
        cv2.destroyAllWindows()
        break

# The following tutorial provide info on how to perform these calibration steps.
# I also utilized the resources from the other documents
#https://stackoverflow.com/questions/39432322/what-does-the-
getoptimalnewcameramatrix-do-in-opencv
#https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
```

## ArcUo Detection Script

The ArUco detection script uses the calibration file generated above to find markers and get their world position. This world position is relative to the camera, where the x and y is centered at center of the frame. Our detection scheme uses the s and y to find what quadrant the marker is in, and then output this quadrant over i2c.

The following image shows what the user sees on the Pis display when this code is running.



We start by importing our necessary libraires and setting up the display, i2c, the camera, and opencv. Most of this code is setting constants that will be used later.

```
#   Team 3 - SEED LAB - Fall 2022
#   10/01/2022
#   SEED Lab mini Project

# The following code calculate in what grid quadrant an ArUco
```

```python
# marker is locaetd in, and then sends the quadrant over i2c.
# This code also display the quadrant on an adafruit display.

from picamera.array import PiRGBArray
from picamera import PiCamera
import numpy as np
import time
import cv2
import math
from smbus2 import SMBus
import sys
import board
import busio
import adafruit_character_lcd.character_lcd_rgb_i2c as character_lcd


#Load calibration file
calibrationFileLoad = "camera_calibration.yaml"
cv_file_load = cv2.FileStorage(calibrationFileLoad, cv2.FILE_STORAGE_READ)
mtx_load = cv_file_load.getNode("camera_matrix").mat()
dist_load = cv_file_load.getNode("dist_coeff").mat()
newcameramtx_load = cv_file_load.getNode("new_camera_matrix").mat()
cv_file_load.release()

#Constants for ArUco detection
arucoDict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
arucoParams = cv2.aruco.DetectorParameters_create()

font = cv2.FONT_HERSHEY_SIMPLEX #font for displaying text (below)
fontBold = cv2.FONT_HERSHEY_DUPLEX #font for displaying text (below)

#Set camera parameters
camera = PiCamera()
camera.resolution = (1296, 976)
#camera.framerate = 30
rawCapture = PiRGBArray(camera, size=(1296, 976))

# allow the camera to warmup
print("Starting PiCamera.....")
time.sleep(0.1)

#For Miniproject
quadrant = 0
quadrantPrev = 0
quadrantNew = False
```

```
centerX = 1296/2
centerY = 976/2

#LCD Setup
lcd_columns = 16
lcd_rows = 2
i2c = busio.I2C(board.SCL, board.SDA)
lcd = character_lcd.Character_LCD_RGB_I2C(i2c, lcd_columns, lcd_rows)
lcd.color = [0, 127, 255]
#GIve a nice welcome message
smile = chr(0b10111100)
lcd.message = " " + smile + "    ArUco   " + smile + " \n     Testing     "
time.sleep(2)
lcd.clear()
redColorIter = 0
blueColorIter = 127
greenColorIter = 255

#I2C Setup
bus = SMBus(1)
addr = 0x8
return_data = 0
```

This the primary code loop. First the ArUco detection occurs using the calibration file generated earlier. The first found marker is taken, and it position is analyzed. Depending on the quadrant it is in, the quadrant value will be set. Then, we run many draw functions to get a pretty display output. This includes drawing the marker location and the quadrant lines/values.

```
#The loop starts here
for frame in camera.capture_continuous(rawCapture, format="rgb",
use_video_port=True):

    #Get image and convert
    image = frame.array
    grayOff = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.undistort(grayOff, mtx_load, dist_load, None, newcameramtx_load)
    #flip the image
    #gray = cv2.flip(gray, 1)

    #Run ArUco detection
    corners, ids, rejected = cv2.aruco.detectMarkers(gray, arucoDict,
parameters=arucoParams)
    #Check if any marker was found
    if len(corners) > 0:
        rvec, tvec, _ = cv2.aruco.estimatePoseSingleMarkers(corners, 0.05,
mtx_load, dist_load)
        (rvec-tvec).any()

        #Look at only the first marker
        index = 0

        #Grab corner data
        cornerInfo = corners[index]

        #Calculate center of marker
        cX = int((cornerInfo[0][0][0] + cornerInfo[0][2][0]) / 2.0)
        cY = int((cornerInfo[0][0][1] + cornerInfo[0][2][1]) / 2.0)

        ###### DRAW QUADRANT DATA #####
        if (cX < centerX) & (cY < centerY):
            quadrant = 2
        elif (cX > centerX) & (cY < centerY):
            quadrant = 1
        elif (cX < centerX) & (cY > centerY):
            quadrant = 3
        else:
            quadrant = 4

        if (quadrant == quadrantPrev):
            quadrantNew = 0
        else:
            quadrantNew = 1
        quadrantPrev = quadrant


        cv2.putText(gray, str(quadrant), (cX+4,cY-4), font, 1,
(255,255,255),1,cv2.LINE_AA)

        for i in range(rvec.shape[0]):
            cv2.drawFrameAxes(gray, mtx_load, dist_load, rvec[i, :, :], tvec[i,
:, :], 0.03)
            #cv2.drawDetectedMarkers(gray, corners)

    #Draw functions to make output look pretty
    #Draw lines:
    cv2.line(gray, (648,0), (648,976), (0, 255, 0), 3)
    cv2.line(gray, (0,488), (1296,488), (0, 255, 0), 3)
    #Draw Numbers
    cxb = 640
    cyb = 500
    hob = 40
    wob = 30
    cv2.putText(gray, "1", (cxb+wob,cyb-hob), font, 1,
(255,255,255),1,cv2.LINE_AA)
    cv2.putText(gray, "2", (cxb-wob,cyb-hob), font, 1,
(255,255,255),1,cv2.LINE_AA)
    cv2.putText(gray, "3", (cxb-wob,cyb+hob), font, 1,
(255,255,255),1,cv2.LINE_AA)
    cv2.putText(gray, "4", (cxb+wob,cyb+hob), font, 1,
(255,255,255),1,cv2.LINE_AA)

    #Draw Values
    cx = 640
    cy = 500
    ho = 240
    wo = 230
    cv2.putText(gray, "0", (cx+wo,cy-ho), fontBold, 2,
(255,255,255),1,cv2.LINE_AA)
    cv2.putText(gray, "pi/2", (cx-wo,cy-ho), fontBold, 2,
(255,255,255),1,cv2.LINE_AA)
    cv2.putText(gray, "pi", (cx-wo,cy+ho), fontBold, 2,
(255,255,255),1,cv2.LINE_AA)
    cv2.putText(gray, "3pi/2", (cx+wo,cy+ho), fontBold, 2,
(255,255,255),1,cv2.LINE_AA)

    cv2.imshow("ArUco 3D Detection", gray)

    #Capture any key press
    key = cv2.waitKey(1) & 0xFF
    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)

    # if the `d' key was pressed, break from the loop
    if key == ord("d"):
        cv2.destroyAllWindows()
        break
```

With quadrant calculated, the following code sends it over i2c to the Arduino. We also write the value to the LCD.

```
#quadrant will give a value between 1 and 4 for the quadrant
    #quadrantNew will be true for one loop cycle if there is a new quadrant

    if (quadrantNew):
        #Code here if you want the single exectuation
        try:
            pi = chr(0b11110111)
            bus.write_byte_data(addr, 0, quadrant)
            if(quadrant == 1):
                pos = "0"
                lcd.color = [255, 0, 0]
            elif (quadrant == 2):
                pos = pi + "/2"
                lcd.color = [255, 255, 0]
            elif (quadrant == 3):
                pos = pi
                lcd.color = [0, 255, 0]
            elif (quadrant == 4):
                pos = "3" + pi + "/2"
                lcd.color = [0, 127, 255]
            else:
                lcd.message = "Sent: " + str(quadrant) + "\nPosition: " + pos +
"      "
        except:
            print("I2C no longer go brr")
        quadrantNew = 0
        #lcd.color = [redColorIter, greenColorIter, blueColorIter]
        lcd.message = "Sent: " + str(quadrant) + "\nPosition: " + pos + "      "
```

The Python code was written entirely using our previous assignments. No additional resources were used beyond out previous assignments.