**This is quite a tedious process. If you ever get stuck, reach out to me:**

[**suhail.prasathong@gmail.com**](mailto:suhail.prasathong@gmail.com)

[**sxp9646@rit.edu**](mailto:sxp9646@rit.edu)

**main.cpp function by function breakdown:**

|  |
| --- |
| **#include "opencv2\core.hpp" #include "opencv2\imgcodecs.hpp" #include "opencv2\imgproc.hpp" #include "opencv2\highgui.hpp" #include "opencv2\aruco.hpp" #include "opencv2\calib3d.hpp"  #include <sstream> #include <iostream> #include <fstream>  using namespace std; using namespace cv;  const float calibrationSquareDimension = 0.024f; //meters const float arucoSquareDimension = .106f; //meters const Size chessboardDimensions = Size(9, 6);** |

This block has required include header files from the opencv library to implement aruco markers.

Constants are defined as

(0.024) - width between each block in an aruco marker in meters

**NOTE:** different printers print these differently so please measure to the tee.

(0.0106) - this is the length of one side of the whole aruco square

(9,6) - are the pre-defined chessboard dimensions for camera calibration

**NOTE: Check out those 1,11-21 youtube playlist videos to learn more**

[**https://www.youtube.com/watch?v=fIpTks0G2m0&t=355s**](https://www.youtube.com/watch?v=fIpTks0G2m0&t=355s)

|  |
| --- |
| **void createArucoMarkers() {  Mat outputMarker;   Ptr<aruco::Dictionary> markerDictionary = aruco::getPredefinedDictionary(aruco::PREDEFINED\_DICTIONARY\_NAME::DICT\_4X4\_50);   for (int i = 0; i < 50; i++) {  aruco::drawMarker(markerDictionary, i, 500, outputMarker, 1);  ostringstream convert;  string imageName = "4x4Marker\_";  convert << imageName << i << ".jpg";  imwrite(convert.str(), outputMarker);  } }** |

This block of code generates 50 aruco markers of 4x4 type. Each of these markers is stored with a unique ID in the backend in a giant look up table for use later. You can create different marker dimensions (5x5, 6x6) etc. Read up OpenCV aruco documentation (https://docs.opencv.org/3.1.0/d9/d6a/group\_\_aruco.html) to learn more about that.

|  |
| --- |
| void createKnownBoardPosition(Size boardSize, float squareEdgeLength, vector<Point3f>&corners) {  for (int i = 0; i < boardSize.height; i++) {  for (int j = 0; j < boardSize.width; j++) {  corners.push\_back(Point3f(j\*squareEdgeLength, i\*squareEdgeLength, 0.0f));  }  } }  void getChessboardCorners(vector<Mat> images, vector<vector<Point2f>>& allFoundCorners, bool showResults = false) {  for (vector<Mat>::iterator iter = images.begin(); iter != images.end(); iter++) {  vector<Point2f> pointBuf;  bool found = findChessboardCorners(\*iter, Size(9,6), pointBuf, CV\_CALIB\_CB\_ADAPTIVE\_THRESH | CV\_CALIB\_CB\_NORMALIZE\_IMAGE);   if (found) {  allFoundCorners.push\_back(pointBuf);  }   if (showResults) {  drawChessboardCorners(\*iter, Size(9,6), pointBuf, found);  imshow("Looking for corners", \*iter);  waitKey(0);  }  } } |

These functions are mainly for camera calibration and are called in the cameraCalibrationProcess function (internal use among other functions only, never gets called in main)

|  |
| --- |
| int startWebcamMonitoring(const Mat& cameraMatrix, const Mat& distanceCoefficients, float arucoSquareDimensions){  Mat frame;    vector<int> markerIds;  vector<vector<Point2f>> markerCorners, rejectedCandidates;  aruco::DetectorParameters parameters;   Ptr < aruco::Dictionary> markerDictionary = aruco::getPredefinedDictionary(aruco::PREDEFINED\_DICTIONARY\_NAME::DICT\_4X4\_50);  VideoCapture vid(0);   if (!vid.isOpened()) {  return -1;  }  namedWindow("Webcam", CV\_WINDOW\_AUTOSIZE);  vector<Vec3d> rotationVectors, translationVectors;   while (true) {  if (!vid.read(frame)) {  break;  }  aruco::detectMarkers(frame, markerDictionary, markerCorners, markerIds);  aruco::estimatePoseSingleMarkers(markerCorners, arucoSquareDimension, cameraMatrix, distanceCoefficients, rotationVectors, translationVectors);   for (int i = 0; i < markerIds.size(); i++) {  aruco::drawAxis(frame, cameraMatrix, distanceCoefficients, rotationVectors[i], translationVectors[i], 0.023f);  cout << "Rotation X Y Z";  cout << rotationVectors[i] << "\n\n";   cout << "Translation X Y Z";  cout << translationVectors[i] << "\n";  }  imshow("Webcam", frame);  if (waitKey(30) >= 0) break;  }  return 1; } |

This block starts the camera and recognizes markers then calculates and outputs its translation and rotation vectors.

|  |
| --- |
| cameraCalibration() cameraCalibrationProcess() loadCameraCalibration() |

These functions are designed to implement the camera calibration process (using the chessboard -> refer to playlist videos to learn more)

Then, the generated calibration is loaded into the algo using loadCameraCalibration.

|  |
| --- |
| int main(int argv, char\*\* argc) {   Mat cameraMatrix = Mat::eye(3, 3, CV\_64F);  Mat distanceCoefficients;    //cameraCalibrationProcess(cameraMatrix, distanceCoefficients);  loadCameraCalibration("Camera Calibration", cameraMatrix, distanceCoefficients);  startWebcamMonitoring(cameraMatrix, distanceCoefficients, 0.099f);     return 0; } |

This is the main method. Either run cameraCalibrationProcess on its own to calibrate your camera or run loadCameraCalibration and startWebcamMonitoring to load the calibration and try to start recognizing your aruco markers.