Name: Zhihao Cao

Class: CSCI 55700

Professor: Dr. Mihran Tuceryan

May 1, 2016

Augmented Reality

3D object insertion in real-time camera using OpenCV

Final report

Problem Statement

To achieve the goal to display a 3d object in the specific positon of the camera view, a marker is needed to help the program to locate the position.

Many techniques are designed to recognize the marker, but the accuracy of recognition is never enough.

The main reasons are:

- Changes of marker color due to the environment (illuminant)
- Changes of marker shape due to the angle and position of camera
- Part of Marker is blocked by some objects
- Marker moving too fast, thus camera unable to capture a clear input.

Literature Survey and previous work

Various marker detection techniques are proposed. They are basically in these two categories.

1. Measure the degree of correlation of a pattern (found inside an image) with known patterns (Feature detection with descriptor)

2. Use digital methods: read a binary code from the marker. The code stores a non-redundant ID for identification.

Correlation technique usually performs less robust than the digital code method.

Technical Description of Method

In this project, a synthetic square marker method is applied. (As figure 1 shown) The marker is composed by a wide black border and an inner binary matrix which determines its identifier (id) and original orientation (without asymmetry).

This project applies many techniques from the lectures. To identify the marker, a series of operations are needed on the camera frame image.

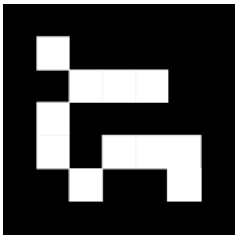


Figure 1 shows a sample of the synthetic square marker

First, the program captures a frame from camera. Then, in order to find the desired marker from the frame, the frame is converted to gray image. Then, an openCV function, adaptiveThreshold() is applied to obtain a binary image. In this way, since the marker has a wide black border which has high contrast to the background, in the binary image, the marker becomes sufficient clear.

Next, to reduce the noise, open operation is applied to the image. So that the whole image becomes low noise and the marker is clearer.

Next, an openCV function, findContours() is applied to extract the contours of all possible markers from the binary image as the figure 2 shown. Then, some criteria is needed to filter out non-markers from the contours. Here are two attributes of the contour of marker: There are only four corners for a marker, and the contour is convex. By applying these two criteria, we can filter out most of the non-markers. By far, markers in the camera frame are detected.

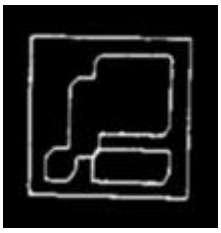


Figure 2 shows the contour of a marker

Next, markers are needed to be recognized. In order to read the cell code inside the marker correctly, the image is needed to be converted from contours image to binary image. The program applies Otsu's thresholding method to the image to obtain the binarization image.

Since the camera might be in certain position and rotation, the detected markers are not always in canonical form as figure 3 shown. The program needs to read the cell code inside the marker correctly, therefore, markers are needed to be converted to canonical form. Two openCV functions, getPerspectiveTransform() and warpPerspective are applied to obtain the canonical form of markers as figure 4 shown.

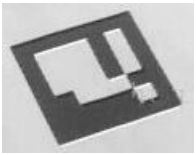
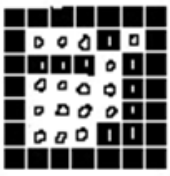


Figure 3 shows the marker in non-canonical form

Now, the marker is clear enough to read the cell code. An openCV function, countNonZero() is applied in each area of the cell code to convert the code to a bit matrix as the figure 5 shown.

Figure 4 shows the marker in canonical form



0	0	0	1	0
1	1	1	0	1
0	0	0	0	1
0	0	0	0	1
0	0	0	1	1

Figure 5 shows the marker in canonical form

Since the marker may appears in certain orientation, to read the correct information of the bit matrix, the right orientation is needed to be found. The first, third and fifth row of the matrix is designed to adjust the orientation, so the program will match the bit matrix with its given pattern matrix at four orientation. After the right

orientation is found, the program decodes the second and fourth row of the bit matrix to find the identification of marker.

By far, the marker is detected and recognized. The next step is to put 3d object into the image which using the world coordinate system. In order to do so, camera's intrinsic and extrinsic parameters are needed to calculate the projection from computer coordinate system to world coordinate system. Camara intrinsic and distortion parameters can be obtained by applying calibrateCamera(). With the camera intrinsic parameters and distortion parameters, the program applies solvePnP() to obtain the extrinsic parameters (rotation and translation).

Now, with camera intrinsic and extrinsic parameters, the program calls projectPoints() to project the 3d object points from computer coordinate system to world coordinate system. At the last, draws the projection points to the original frame image. By these procedure, the program can detect and recognize markers and display 3d object on the frame. It is exactly the idea of augmented reality.

Description of Experimental Setup and Data Collection

Experiment:

Use the program to detect and recognized markers with different degree of angles. Record the range the recognizable camera-marker angle.

Set up:

- Fix the camera 90-degree to the horizontal ground.
- Put a marker in front of camera starting from 90 degree respecting to the ground.

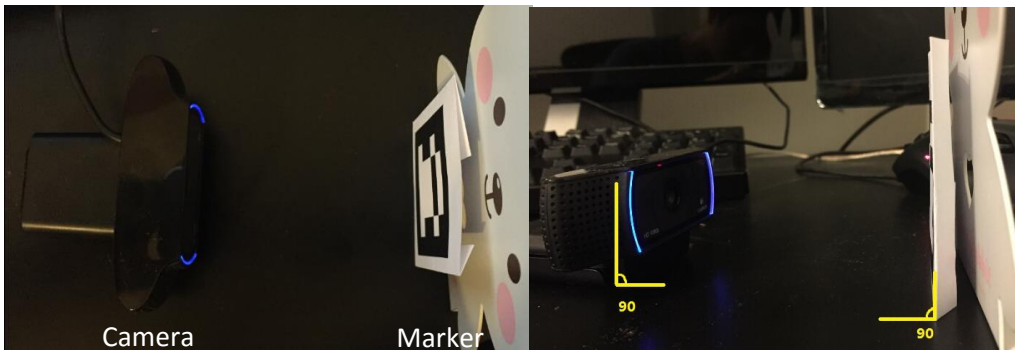


Figure 6 shows the camera and marker set up.

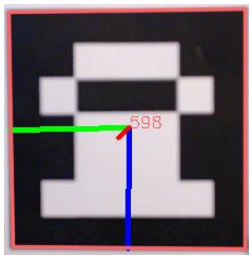


Figure 7 shows the result of the marker with 90 degree respecting to the ground.

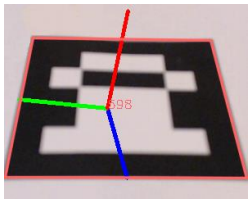


Figure 8 shows the result of the marker with 60 degree respecting to the ground.

Figure 8 shows the axis is a bit off the center of the marker, but the end points of the axis is still connecting to the center of the edge of marker. The problem should be the calculation of the center point. And will be fixed in future.

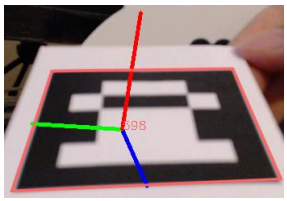


Figure 9 shows the result of the marker with 45 degree respecting to the ground

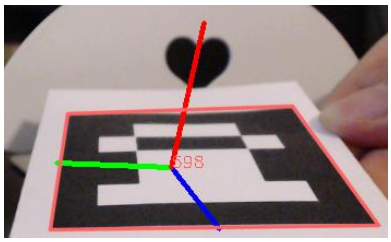


Figure 10 shows the result of the marker with 30 degree respecting to the ground.

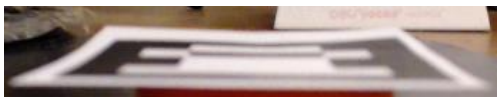


Figure 11 shows the result of the marker with 10 degree respecting to the ground.

In the experiment, when the marker with lower than 30 degree respecting to the ground, the program becomes weaker and weaker to recognize the marker. The reason is the detail of the marker is missing too much. The algorithm is incapable to reconstruct the canonical form of the marker.

In conclusion of this experiment, the algorithm of the program is only valid in the range of 90-30 degree respecting to the ground.

Implementation/code

Main.cpp

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <iostream>
#include "MarkerDetector.h"

using namespace cv;

int main() {
	cv::VideoCapture capWebcam(0);				// initialize camera and select camera 0

	if (capWebcam.isOpened() == false) {				// check if valid
		std::cout << "can not access the webcam\n\n";
		return(0);
	// exit program
	}

	cv::Mat imgOriginal;				// input image
	cv::Mat imgGray;		// convert to Gray

	MarkerDetector m_detector;	//construct marker detector

	Point3f marker_corners[] ={	//marker corner 3d coordinate
		Point3f(-0.5f, -0.5f, 0),
		Point3f(-0.5f,  0.5f, 0),
		Point3f(0.5f,  0.5f, 0),
		Point3f(0.5f, -0.5f, 0)
	};

	std::vector<Point3f> axis ={	//axis 3d vector
		Point3f(0.5, 0, 0),	//x
		Point3f(0, 0.5, 0),	//y
		Point3f(0, 0, -0.5)	//z
	};

	float camera_matrix[] = {	//my camera matrix obtain from calibrateCamera()
		848.069f,     0.0f, 268.697f,
		   0.0f, 847.687f, 264.266f,
		   0.0f,     0.0f,     1.0f
	};

	float distortion[] = { -0.445957f, 0.278828f, -0.002213f, -0.000656f };	//distortion coefficients

	std::vector<Point3f> m_marker_corners;
	m_marker_corners = std::vector<Point3f>(marker_corners, marker_corners + 4); //marker corners vector
	cv::Mat m_camera_matrix;
	m_camera_matrix = Mat(3, 3, CV_32FC1, camera_matrix).clone();		//Camera Matrix
	cv::Mat m_distortion;
	m_distortion = Mat(1, 4, CV_32FC1, distortion).clone();		//Distortion coefficient

	std::vector<Point2f> imgpts;	//declear my axis projection points
```

```
        //float m_model_view_matrix[16];

        char charCheckForEscKey = 0;
        while (charCheckForEscKey != 27 && capWebcam.isOpened()) {                  //Esc key or webcam
lose connection to exit
                bool blnFrameReadSuccessfully = capWebcam.read(imgOriginal);            //load frame

                if (!blnFrameReadSuccessfully || imgOriginal.empty()) {             // check if frame load
successfully
                        std::cout << "fail to load frame from webcam\n";
                        break;

                }

                cv::cvtColor(imgOriginal, imgGray, CV_RGBA2GRAY);           //convet to gray img
                m_detector.startDetect(imgGray, 100);                       //load img into marker detector
component (input img, marker min-length)

                Mat m_rotation, m_translation;                                  //declear rotation and tranlation
                std::vector<Marker>& markers = m_detector.get_marker();         //obtain markers from the
marker detector component
                for (int i = 0; i < markers.size(); ++i){                   //for each detected marker
                        //use camera parameters to calculate the rotation and tranlation matrix
                        markers[i].marker_solvePnP(m_marker_corners, m_camera_matrix, m_distortion,
m_rotation, m_translation);
                        //call projectPoints() to project 3d object(axis) into the camera img
                        projectPoints(axis, m_rotation, m_translation, m_camera_matrix, m_distortion, imgpts);
//return imgpts (projection vector)
                        //draw to camera image
                        markers[i].drawImg(imgOriginal, 2, imgpts); //(input img, thickness, draw img)
                }

          // declare windows
                cv::namedWindow("imgOriginal", CV_WINDOW_AUTOSIZE);
                cv::imshow("imgOriginal", imgOriginal);                         // show results
                charCheckForEscKey = cv::waitKey(1);                            // waitKey
        }       // end while

        return(0);
}
```

MarkerDetector.h

```
#ifndef __MARKER_DETECTOR_H__
#define __MARKER_DETECTOR_H__

#include <vector>
#include <opencv2\core\core.hpp>

class Marker {
public:
	std::vector<cv::Point2f> marker_corners;
	int marker_id;

	Marker();
	Marker(int input_id, cv::Point2f input_0, cv::Point2f input_1, cv::Point2f input_2, cv::Point2f input_3);
	void drawImg(cv::Mat& img, float t, std::vector<cv::Point2f> imgpts);
	void marker_solvePnP(std::vector<cv::Point3f> input_corners, cv::Mat& camera_matrix, cv::Mat&
distortion, cv::Mat& rotation, cv::Mat& translation);
};

class MarkerDetector {
public:
	MarkerDetector();
	int startDetect(cv::Mat& input_img, int img_size, int img_length = 10);
	std::vector<Marker>& get_marker();

private:
	std::vector<Marker> marker_result;
	std::vector<cv::Point2f> marker_position;

	void validate_marker(cv::Mat& input_img, std::vector<Marker>& marker_found, std::vector<Marker>&
true_marker);
	void find_marker(cv::Mat& img_gray, std::vector<Marker>& marker_found, int min_size, int img_length);
	void refine_mraker(cv::Mat& input_img, std::vector<Marker>& true_marker);
	int decodeID(cv::Mat& input_matrix);
	int orientation(cv::Mat& input_matrix);
	cv::Mat rotate_matrix(cv::Mat& input_matrix);
};
#endif
```

MarkerDetector.cpp

```
#include <iostream>
#include <sstream>
#include <opencv2\calib3d\calib3d.hpp>
#include <math.h>
#include <opencv2\imgproc\imgproc.hpp>
#include <string>
#include "MarkerDetector.h"

#define CELL_LENGTH 10
#define MARKER_LENGTH (7*CELL_LENGTH)

using namespace std;
using namespace cv;

//MarkerDetector class
MarkerDetector::MarkerDetector() {
	//assign marker 2d position, counterclockwise
	marker_position.push_back(Point2f(0, 0));
	marker_position.push_back(Point2f(0, MARKER_LENGTH - 1));
	marker_position.push_back(Point2f(MARKER_LENGTH - 1, MARKER_LENGTH - 1));
	marker_position.push_back(Point2f(MARKER_LENGTH - 1, 0));
}

int MarkerDetector::startDetect(Mat& input_img, int img_size, int img_length) {
	CV_Assert(!input_img.empty());		//check if imput image is valid
	CV_Assert(input_img.type() == CV_8UC1);

	Mat my_img = input_img.clone();		//real copy
	//increase the image contrast
	equalizeHist(my_img, my_img);
	//declare the marker vector to hold found markers
	vector<Marker> marker_found;
	//find the markers
	find_marker(my_img, marker_found, img_size, img_length);
	//filter out the wrong markers and decode markers
	validate_marker(my_img, marker_found, marker_result);
	//refine marker corners
	refine_mraker(my_img, marker_result);

	return marker_result.size();
}

void MarkerDetector::find_marker(Mat& img_gray, vector<Marker>& marker_found, int img_size, int img_length) {
	Mat binary_image;			//declare binary image
	int threshold = (img_size / 4) * 2 + 1;	//calculate the size threshold of finding markers
	//apply openCV adaptiveThreshold() to obtain the binary image
	adaptiveThreshold(img_gray, binary_image, 255, ADAPTIVE_THRESH_GAUSSIAN_C,
THRESH_BINARY_INV, threshold, threshold / 3);
	//use open operation to remove noise
	morphologyEx(binary_image, binary_image, MORPH_OPEN, Mat());

	vector<vector<Point>> contour_result;	//contours found
	vector<vector<Point>> valid_contour;	//valid marker contours
	//apply openCV findContours() to obtain image contours
	findContours(binary_image, contour_result, CV_RETR_LIST, CV_CHAIN_APPROX_NONE);

	//filter out non-marker contours by size
	for (int i = 0; i < contour_result.size(); ++i) {
		if (contour_result[i].size() > img_size) {
			valid_contour.push_back(contour_result[i]);
```

```
                }
        }

        //approximate curves of polygonal in the contour
        vector<Point> poly_curves;  //declare the pologon curve variable
        for (int i = 0; i < valid_contour.size(); ++i) {
                double epsilon = valid_contour[i].size()*0.08;
                //apply openCV approxPolyDP() to approximate the number of curves in a polygon
                approxPolyDP(valid_contour[i], poly_curves, epsilon, true);

                //filter out not 4-curve polygon
                if (poly_curves.size() != 4) continue;

                //filter out non-convex contours
                if (!isContourConvex(poly_curves)) continue;

                //check if two point is far enough
                float edge_min = FLT_MAX;
                for (int j = 0; j < 4; ++j){
                        Point edge = poly_curves[j] - poly_curves[(j + 1) % 4];
                        edge_min = min(img_size, edge.dot(edge));
                }
                //filter out too small contours
                if (edge_min < img_length*img_length) continue;

                //assign the corners in counterclockwise order
                Marker marker_final = Marker(0, poly_curves[0], poly_curves[1], poly_curves[2],
poly_curves[3]); //(id,corner1-4)
                Point2f corner_1 = marker_final.marker_corners[1] - marker_final.marker_corners[0];
                Point2f corner_2 = marker_final.marker_corners[2] - marker_final.marker_corners[0];
                if (corner_1.cross(corner_2) > 0) {  //swap y
                        swap(marker_final.marker_corners[1], marker_final.marker_corners[3]);
                }
                marker_found.push_back(marker_final);
        }
}

void MarkerDetector::validate_marker(cv::Mat& input_img, vector<Marker>& marker_found, vector<Marker>&
true_markers) {
        //clear the marker vector
        true_markers.clear();

        Mat img_marker;                 //declare the marker image
        Mat marker_binary_matrix(5, 5, CV_8UC1);        //declare the marker code matrix
        //for each marker
        for (int i = 0; i < marker_found.size(); ++i) {
                //obtain the marker canonical form by openCV getPerspectiveTransform() and warpPerspective()
                Mat myMat = getPerspectiveTransform(marker_found[i].marker_corners, marker_position);
                warpPerspective(input_img, img_marker, myMat, Size(MARKER_LENGTH,
MARKER_LENGTH));
                //obtain the binarization marker img by openCV threshold()
                threshold(img_marker, img_marker, 125, 255, THRESH_BINARY | THRESH_OTSU);

                //A marker must has a whole black border.
                for (int y = 0; y < 7; ++y) {           //for each y-cell of the marker
                        int increment = (y == 0 || y == 6) ? 1 : 6;
                        int cellY = y*CELL_LENGTH;

                        for (int x = 0; x < 7; x += increment) {  //for each yx-cell of the marker
                                int cellX = x*CELL_LENGTH;
                                //apply openCV countNonZero to obtain the number of non zero elements
```

```
                                        int count_result = countNonZero(img_marker(Rect(cellX, cellY,
CELL_LENGTH, CELL_LENGTH)));
                                        if (count_result > CELL_LENGTH*CELL_LENGTH / 4) goto marker_fail;
                                }
                        }

                        //read the marker code
                        for (int y = 0; y < 5; ++y) {     //for each row
                                int cellY = (y + 1)*CELL_LENGTH;

                                for (int x = 0; x < 5; ++x) { //for each col
                                        int cellX = (x + 1)*CELL_LENGTH;
                                        //apply openCV countNonZero to obtain the number of non zero elements
                                        int count_result = countNonZero(img_marker(Rect(cellX, cellY,
CELL_LENGTH, CELL_LENGTH)));
                                        //conver the marker code into binary matrix
                                        if (count_result > CELL_LENGTH*CELL_LENGTH / 2)
                                                marker_binary_matrix.at<uchar>(y, x) = 1;
                                        else
                                                marker_binary_matrix.at<uchar>(y, x) = 0;
                                }
                        }

                        //obtain the marker orientation
                        bool valid_marker = false;
                        int rot_counter;
                        for (rot_counter = 0; rot_counter < 4; ++rot_counter) {// loop 4 orientation
                                if (orientation(marker_binary_matrix) == 0){
                                        valid_marker = true;
                                        break;
                                }
                                marker_binary_matrix = rotate_matrix(marker_binary_matrix);
                        }
                        //fail to find right orientation
                        if (!valid_marker) goto marker_fail;

                        //decode the marker identification
                        Marker& true_marker = marker_found[i];
                        true_marker.marker_id = decodeID(marker_binary_matrix);
                        //rotate the marker to right orientation
                        std::rotate(true_marker.marker_corners.begin(), true_marker.marker_corners.begin() + rot_counter,
true_marker.marker_corners.end());
                        //store to marker vector
                        true_markers.push_back(true_marker);

                        marker_fail: continue;
                }
}

vector<Marker>& MarkerDetector::get_marker() {
                return marker_result;
}

//decode the marker identification
int MarkerDetector::decodeID(Mat& input_matrix) {
                int matrix_id = 0;   //initialize the id
                                                                //for each row
                for (int r = 0; r < 5; ++r) {
                        matrix_id <<= 1;        //left bit shift
                        matrix_id |= input_matrix.at<uchar>(r, 1); //insert second row of bit value

                        matrix_id <<= 1;        //left bit shift
```

```
                matrix_id |= input_matrix.at<uchar>(r, 3); //insert fourth row of bit value
        }
        return matrix_id;
}

//rotate matrix by counterclockwise
Mat MarkerDetector::rotate_matrix(cv::Mat& input_matrix) {
        Mat result = input_matrix.clone();    //get real copy
        int r = input_matrix.rows;
        int c = input_matrix.cols;
        //rotate by counterclockwise
        for (int i = 0; i<r; ++i) {       //for each row
                for (int j = 0; j<c; j++) {   //for each col
                        result.at<uchar>(i, j) = input_matrix.at<uchar>(c - j - 1, i);
                }
        }
        return result;
}

//find the orientation
int MarkerDetector::orientation(Mat& input_matrix) {
        const int codex[4][5] = {
                { 1,0,0,0,0 },          // 00
                { 1,0,1,1,1 },          // 01
                { 0,1,0,0,1 },          // 10
                { 0,1,1,1,0 }           // 11
        };

        int result = 0;
        //for each row
        for (int y = 0; y < 5; ++y) {
                int minimum_sum = INT_MAX;
                for (int p = 0; p < 4; ++p) {
                        int m_sum = 0;
                        //match the input_matrix with the codex, sum up the result
                        for (int x = 0; x < 5; ++x) {
                                m_sum += !(input_matrix.at<uchar>(y, x) == codex[p][x]); //return 1 if fail
                        }
                        minimum_sum = min(minimum_sum, m_sum);
                }
                //sum up each loop result
                result += minimum_sum;
        }
        return result;  //expect 0 result which means all match
}

//refine marker corners
void MarkerDetector::refine_mraker(cv::Mat& input_img, vector<Marker>& true_marker) {
        //for each true marker
        for (int i = 0; i < true_marker.size(); ++i) {
                vector<Point2f>& corner_vec = true_marker[i].marker_corners;
                //refines the corner locations by calling openCV cornerSubPix()
                cornerSubPix(input_img, corner_vec, Size(5, 5), Size(-1, -1),
TermCriteria(CV_TERMCRIT_ITER, 30, 0.1));
        }
}



//Marker class
Marker::Marker() {
        //initialize marker id
```

```
		marker_id = -1;
		marker_corners.resize(4, Point2f(0.f, 0.f));
}

//construct marker object
Marker::Marker(int input_id, cv::Point2f input_0, cv::Point2f input_1, cv::Point2f input_2, cv::Point2f input_3) {
		marker_id = input_id;
		marker_corners.reserve(4);
		marker_corners.push_back(input_0);
		marker_corners.push_back(input_1);
		marker_corners.push_back(input_2);
		marker_corners.push_back(input_3);
}

//draw the result into camera image
void Marker::drawImg(cv::Mat& img, float t, vector<Point2f> imgpts) {
		//calculate the center coordinate of marker
		Point center_point = marker_corners[0] + marker_corners[2];
		center_point.x /= 2;
		center_point.y /= 2;

		//draw the 3d-axis
		line(img, center_point, imgpts[0], cv::Scalar(255, 0, 0), 3);
		line(img, center_point, imgpts[1], cv::Scalar(0, 255, 0), 3);
		line(img, center_point, imgpts[2], cv::Scalar(0, 0, 255), 3);

		//draw marker borders
		line(img, marker_corners[0], marker_corners[1], cv::Scalar(100, 100, 255), t, CV_AA);
		line(img, marker_corners[1], marker_corners[2], cv::Scalar(100, 100, 255), t, CV_AA);
		line(img, marker_corners[2], marker_corners[3], cv::Scalar(100, 100, 255), t, CV_AA);
		line(img, marker_corners[3], marker_corners[0], cv::Scalar(100, 100, 255), t, CV_AA);

		//draw marker id
		stringstream myText;
		myText << marker_id;
		putText(img, myText.str(), center_point, FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(100, 100, 255));
}

//calculate the rotation and translation vector
void Marker::marker_solvePnP(vector<Point3f> input_corners, cv::Mat& camera_matrix, cv::Mat& distortion,
cv::Mat& rotation, cv::Mat& translation) {
		Mat rotation_vector;
		//call openCV solvePnP() to obtain rotation and translation vector
		bool res = solvePnP(input_corners, marker_corners, camera_matrix, distortion, rotation_vector, translation);
		//apply Rodrigues to set the rotation vector
		Rodrigues(rotation_vector, rotation);
}
```

Results and Discussion

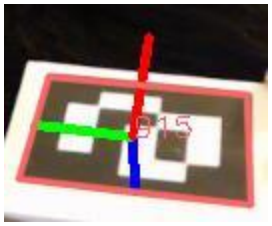


Figure 12 shows a marker with certain camera rotation and translation

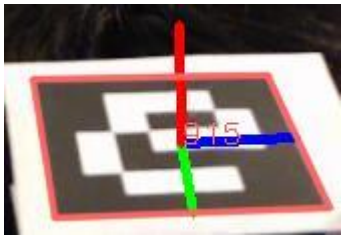


Figure 13 shows the xyz-axis specifies the orientation of the marker.

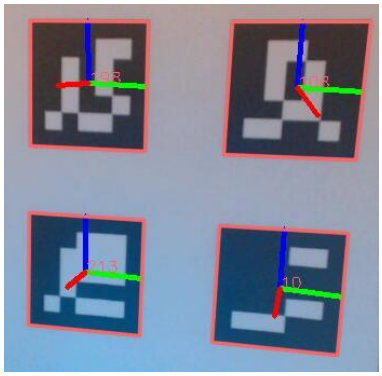


Figure 14 shows the program recognizes multiple markers at the same time.

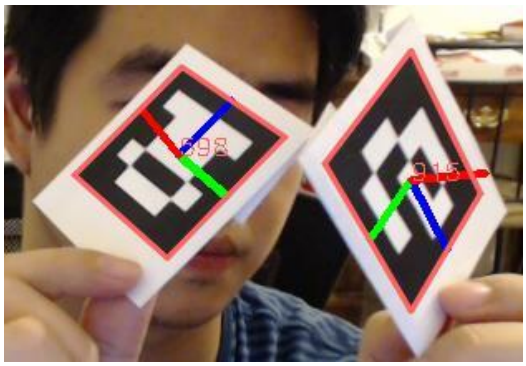


Figure 15 shows the program recognizes markers with different orientations and positions

As the figures shown, the program is capable to recognize markers with different orientations and positions.

Conclusions/Future Work

In this project, a program to detect and recognize markers is successfully implemented. The program is capable to detect and recognize markers with different orientations and positions, but the experiment shows that the valid angle range of recognizing the marker is 30-90 degree (marker respecting to the ground). The validity of the program is also significantly affected by the quality of the input image which means the performance of the camera device. When the input frame is clear without any fuzzy, the accuracy of recognizing marker must be highly improved.

For the future work

- The accuracy of recognizing markers should be improved. This is done by upgrading camera device and applying better algorithm to process the input frame.
- Apply more advanced 3D model to the camera window. This is done by applying the OpenGL library.

References

Camera Calibration and 3D Reconstruction

http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

Detection of ArUco Markers

http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html#gsc.tab=0

ArUco: a minimal library for Augmented Reality applications based on OpenCV

http://www.uco.es/investiga/grupos/ava/node/26

Chris Dahms -OpenCV_3_Windows_10_Installation_Tutorial

https://github.com/MicrocontrollersAndMore/OpenCV_3_Windows_10_Installation_Tutorial

SIMPLAR: AUGMENTED REALITY FOR OPENCV BEGINNERS

http://dsynflo.blogspot.com/2010/06/simplar-augmented-reality-for-opencv.html

Augmented Reality SDK with OpenCV –stackoverflow

http://stackoverflow.com/questions/12283675/augmented-reality-sdk-with-opencv