An Augmented Reality Application based on Planar Tracking and Visual SLAM

Puyang Wang
puyang.wang@rutgers.edu

Lidan Wang
lidan.wang@rutgers.edu

Abstract

This report introduces the implementation of a novel proposed Head Mounted Augmented Reality (AR) Device. The whole project is divided into two parts. As for computer vision part, our objective is to accurately track the pose of cameras in a scene. The AKAZE planar tracking and ORB-SLAM algorithms are employed to compute the camera poses in our approach, then we utilize OpenGL to render a virtual object in the scene. Stereo cameras are used to achieve the stereoscopic AR effect. Firstly we discussed the methods that have been adopted and implemented in our approaches. Then a series of experiments were conducted to evaluate the system performance. The results are analyzed subsequently. At the end, we discuss the challenges in this project, future works, cost analysis and the current trends in the robotics field.

1. Introduction

Augmented Reality(AR), a technology where a 3-dimensional object are integrated into a real environment in real time, is a variation of Virtual Reality (VR) [9]. VR technologies completely immerse the user into a synthetic environment, so that the user cannot see the real world around him. In contrast, AR allows the user to see the real world, with virtual objects superimposed upon or composited with the real world. That is to say, AR supplements reality, rather than completely replacing it. Ideally, it would appear to the user that the virtual and real objects coexist in the same space. As defined in [3], AR system has three main characteristics:

Combines real and virtual,

Interactive in real time ,

Registered in 3-D. [3]

This technology has numerous potential applications in medical visualization, maintenance and repair, annotation, robot path planning, entertainment, and military aircraft navigation and targeting and so on.

Since nowadays most applications in AR field are 2-dimensional display, to provide users a better interactive experience with virtual objects that are merged into the real environment, we came up with an approach that allows users to feel 3-dimensional effect by using Google cardboard head mounted display.

The basic idea of this project is to replace the virtual environment in VR system to the real environment. Different from normal HDMs, our device contains a stereo camera system, an Android phone, and a Google cardboard. By attaching the stereo camera system to the HDM, the real environment can be captured and transmitted to the laptop. AR process is done on the laptop to merge a virtual object with the real environment. The final step is transmitting the AR-combined real environment real-time video stream to the Android phone and displaying it through Google cardboard to achieve 3-D AR display.

As the Computer vision part of this project, the task is to implement the real-time AR in the real environment. The key point of AR is finding where to locate the object and displaying accurately how the perspective of the object will change as camera pose changes. This leads our focus to the feature points finding and tracking problems. In our implementation, we choose Simultaneous Localization and Mapping (SLAM) algorithm as our approach.

SLAM was originally developed by Hugh Durrant-Whyte and John J. Leonard in [7]. It is the problem of concurrently constructing a map of an unknown environment by moving exteroceptive sensors while at the same time getting localized in the environment using the map. SLAM consists of multiple parts: Landmark extraction, data association, state estimation, state update and landmark update. There are many ways to solve each of the smaller parts. In this project, the SLAM algorithm is used to track and compute the camera pose.

The proposed approach of real-time AR has four steps. Step 1 is stereo camera system calibration and simple real-time AR based on a checkerboard, step 2 is real-time AR based on AKAZE planar tracking, step 3 is the implementation of ORB-SLAM, and step 4 is the real-time AR based on ORB-SLAM and planar tracking. After that, we will combine our part with another groups part to achieve the final goal of this project.

While testing the programs we use a simple cuboid as our 3D AR object, afterward, we utilize OpenGL to render

a more complex 3D object. The implementation will be discussed later in this report.

Our system is able to run under two modes which are planar tracking mode and ORB-SLAM mode. A Sponge Bob 3D model is rendered in the scene. The system can also reset when it loses tracking. The performance turns out very fast and robust. Finally, we connect our part with Google application development part.

The following sections will discuss the methods for each step and experimental results in details.

2. Methods and Implementation

2.1. Stereo Camera Calibration

Camera calibration is a crucial step in 3D computer vision in order to extract metric information from 2D images. As we known cameras have distortions, and camera calibration is to find the quantities internal the camera that affects the imaging process, for example, skew factor, lens distortion, different scaling factors for row pixels and column pixels and so on. This section discusses the method of camera calibration described in [17] and how the stereo camera calibration was implemented in this project.

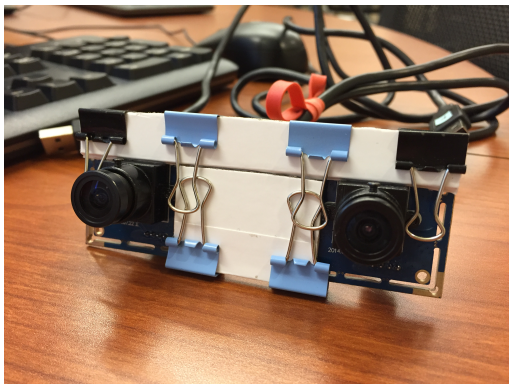

Figure 1: The model of stereo camera system. 2 USB cameras are employed as stereo camera system in this project.

2.1.1 Camera Calibration Method

According to the dimension of the calibration objects, the camera calibration technique is roughly classified into 4 categories, which are 3D reference object based calibration, 2D plane based calibration, 1D line based calibration and self-calibration, also can be considered as 0D approach. This project adopted 2D plane based calibration due to its flexibility. This technique only requires the camera to observe a planar pattern, the checkerboard patterns for instance, shown in a few different orientations.

First, we start with the notation used in this section. The functions in this section use a pinhole camera model. In this model, a scene view is formed by projecting 3D points into the image plane using a perspective transformation. A 2D projection point in pixels is denoted by $\mathbf{m} = [u, v]^T$, and a 3D point in the world coordinate space is denoted by $\mathbf{M} = [X, Y, Z]^T$. The relationship between a 3D point $\mathbf{M}$ and its image projection $\mathbf{m}$ is given by

$$s\tilde{\mathbf{m}} = \mathbf{A}[\mathbf{R}\ \mathbf{t}]\tilde{\mathbf{M}}, \tag{1}$$

where $\mathbf{A}$ is a camera matrix, or can be considered as a matrix of intrinsic parameters, is given by

$$\mathbf{A} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

with (u_0, v_0) the coordinates of the principle point, α and β the scale factors in image u and v axes, and γ the parameter describing the skewness of the two image axes. Moreover, s is an arbitrary scale factor, $\tilde{\mathbf{m}} = [u, v, 1]^T$ and $\tilde{\mathbf{M}} = [X, Y, Z, 1]^T$ are augmented vectors by adding 1 as the last element, and $(\mathbf{R}, \mathbf{t})$, called the extrinsic parameters, is the rotation and translation which relates the world coordination system to the camera coordinate system. Thus, given an image from the camera and it is scaled by a factor, all of the parameters above should be scaled bu the same factor. The matrix of extrinsic parameters $[\mathbf{R}, \mathbf{t}]$ is used to describe the camera motion around a static scene, or on the other hand, motion of an object in front of the still camera. That is to say, $[\mathbf{R}, \mathbf{t}]$ translates coordinates of a 3D point to a coordinate system, fixed with respect to the camera. Above transformation is equivalent to the following(with $z \neq 0$):

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{3}$$

$$x' = \frac{x}{z}, \quad y' = \frac{y}{z} \tag{4}$$

$$u = \alpha x' + u_0, \quad v = \beta y' + v_0 \tag{5}$$

Considering the lens distortion, the model above can be extended as:

$$x'' = x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2) \tag{6}$$

$$y'' = y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1(r^2 + 2y'^2 + 2p_2 x' y') \tag{7}$$

$$u = \alpha x'' + u_0, \;\; v = \beta y'' + v_0 \tag{8}$$

where $r^2 = x'^2 + y'^2$, k_1, k_2, k_3, k_4, k_5 and k_6 are radial distortion coefficients, p_1 and p_2 ate tangential distortion coefficients. Higher-order coefficients are not considered here. The following section introduces the implementation of stereo camera calibration that are used in this project.

2.1.2 Fisheye Camera Model

Before interim report, we were using two narrow-angle cameras whose field of view were 70 degrees, which is too narrow and easy to lose tracking while applying ORB-SLAM. Later we changed our stereo cameras to two wide-angle cameras with the field of view nearly 110 degrees. Wide angle lens has a drawback that it has so-called fisheye distortion. If we still use perspective camera model, we can observe that points of the image located far from the center of the image are distorted. Thus, we need to use fisheye camera model to calibrate the camera.

$$\theta = x' tan(r) \tag{9}$$

The fisheye distortion is:

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{10}$$

All possible coefficients (k_1, k_2, k_3, k_4) of fisheye distortion were optimized under calibration.

2.1.3 Stereo Camera Calibration Implementation

OpenCV was employed as a tool to implement this project. In order to minimize the reprojection error, each camera was calibrated independently firstly. After that, stereo cameras were calibrated simultaneously.

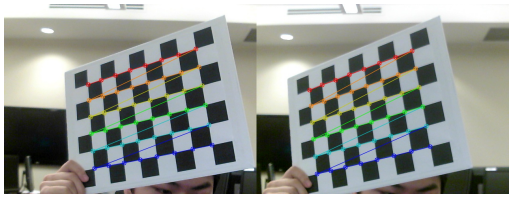

Figure 2: Stereo camera calibration using checkerboard pattern, the internal corners of the chessboard were detected and rendered.

Since Checkerboard patterns were used as the 2D pattern in the calibration process, we use OpenCV function $findChessboardCorners()$ to find the positions of internal corners of the chessboard. Then the function $calibrateCamera()$ was used to find the camera intrinsic and extrinsic parameters from several views of a calibration. The output parameters are camera matrices, distortion coefficients, transformation vectors and rotation vectors.

2.2. Planar Tracking

The goal of planar tracking is to track a plane in 3D space based on a set of corresponding points between two frames. With a set of corresponding points and the fact that those points are all on a plane, we can define the corresponding object points by simply taking z value as zero. Then we can find the camera pose from those 3D-2D point correspondences [4].

2.2.1 ORB Feature

The first step of planar tracking is to detect and match feature points between video frames.Feature matching is at the base of many computer vision problems, such as object recognition or structure from motion. In [13], Rublee *et al*. proposed a very fast binary descriptor based on BRIEF, called ORB, which is rotation invariant and resistant to noise. The efficiency was tested on many real-world applications including object detection and planar-tracking. Besides, its time efficiency is proven to be good enough for real-time applications on embedded systems.

ORB feature build on the well-known FAST key-point detector and BRIEF descriptor; for this reason it is called ORB (Oriented FAST and Rotated BRIEF).The main addition of ORB to FAST is the addition of a fast and accurate orientation component.

ORB uses a simple but effective measure of corner orientation, the intensity centroid. The intensity centroid assumes that a corner's intensity is offset from its center, and this vector may be used to impute an orientation. The moments of a patch are defined as:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y), \tag{11}$$

and with these moments we may find the centroid:

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{12}$$

We can construct a vector from the corner's center, O, to the centroid, $\vec{OC}$. The orientation of the patch then simply is:

$$\theta = atan2(m_{01}, m_{10}), \tag{13}$$

where $atan2$ is the quadrant-aware version of $arctan$. Compared with two gradient-based measures, BIN and MAX, the intensity centroid gives a uniformly good orientation even under large image noise.

2.2.2 AKAZE Feature

KAZE features which aim to detect and describe features in nonlinear scale spaces were introduced in [1]. This method increases repeatability and distinctiveness with respect to SIFT and SURF thanks to the use of nonlinear diffusion filtering. The main drawback of KAZE is that it is computationally intense.

To overcome this problem, Pablo F. Alcantarilla *et al.* introduced Accelerated KAZE Features (AKAZE) in [2]. They have proved that AKAZE has a excellent compromise between speed and performance compared to state-of-the-art methods such as BRISK, ORB, SURF, SIFT and KAZE.A-KAZE features are faster to compute than SURF, SIFT and KAZE due to the dramatic speed-up introduced by Fast Explicit Diffusion (FED) and also exhibit much better performance in detection and description than previous methods, including ORB and BRISK.

2.2.3 Brute-Force Feature Matching

After features are extracted, we have a feature descriptor for each key-point. For each video frame, we have a set of feature descriptors and we want to find point correspondences across frames. Therefore, a matcher should be called to find feature matches in two frames. The most simple one is Brute-Force matcher. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. Usually, we calculates l2 norms between two features in Brute-Force matcher. Given two features $u, v \in R^n$, the distance d between u and v is defined as:

$$d = \sqrt[2]{\sum_{i=1}^{n}(u_i - v_i)^2}. \tag{14}$$

By setting a threshold, we can define feature match as two features with a distance below the threshold.

2.2.4 Perspective-n-Point Problem

In the final step of planar tracking algorithm, we need to set the frame viewing the plane perpendicularly as the key-frame and each feature points as 3D object points with zero z values. Feature matches between every following frame and the key-frame is defined as corresponding 2D image points to the 3D object points. Therefore, we have a set 3D-2D point correspondences for every frame after the key-frame.

Perspective-n-Point is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. The camera pose consists of 6 degrees-of-freedom (DOF) which are made up of the rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world.

Given a set of n 3D points in a world reference frame and their corresponding 2D image projections as well as the calibrated intrinsic camera parameters, determine the 6 DOF pose of the camera in the form of its rotation and translation with respect to the world. This follows the perspective project model for cameras:

$$p_c = K[R|T]p_w, \tag{15}$$

where $p_w = [xyz1]^T$ is the homogeneous world point, $p_c = [uv1]^T$ is the corresponding homogeneous image point, K is the matrix of intrinsic camera parameters and R and T are the desired 3D rotation and translation of the camera (extrinsic parameters) that are being calculated. This leads to the following equation for the model:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 1 & r_1 2 & r_1 3 & t_1 \\ r_2 1 & r_2 2 & r_2 3 & t_2 \\ r_3 1 & r_3 2 & r_3 3 & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{16}$$

A commonly used solution to the problem exists for n = 3 called P3P, and many solutions are available for the general case of n 3. In this paper, we used OpenCV function $solvePnP()$ with RANSAC to solve the PnP problem.

2.3. ORB-SLAM

In this section we first introduce the SLAM problem and its basic methods. SLAM is more a concept than a specific algorithm. There are many algorithms involved in each part of the SLAM problems. Secondly, the ORB-SLAM algorithm will be discussed. At last, we provide the implementation of ORB-SLAM. ORB-SLAM is a new SLAM solution proposed and published by the robotics research group of University of Zaragoza [11]. It is an accurate SLAM solution for Monocular, Stereo and RGB-D cameras and is able to compute in real-time. Meanwhile, it includes an automatic and robust initialization from planar and non-planar scenes.

2.3.1 Introduction of SLAM Problem

SLAM problem involves a moving agent, for example a robot, which has at least one sensor able to gather its surrounding information. Those sensors include cameras, laser scanners, sonars and so on, and are called exteroceptive sensors. The moving agent can also incorporate other sensors such as wheel encoders, accelerometers and gyrometers, which are known as proprioceptive sensors, to measure its own movement. The SLAM problem was originally

defined for mobile robots navigation. But broadly speaking, one or more moving exteroceptive sensors connected to a computer can form a SLAM system.

SLAM consists of three basic operations, which are reiterated at each time step:

(1) Sensors move to a new point of view. Due to inevitable noise and errors, this motion gains the uncertainty of the localization. Thus a mathematical model called motion model is needed.

(2) Sensors extract interesting features in the environment and incorporate to the map. These features are known as landmarks. The location of those landmarks will also be uncertain because of the errors of the exteroceptive sensors. A model called observation model is needed here to determine the position of the landmarks in the scene from the data collected ed by the sensors.

(3) Sensors observe the landmarks that had been previously mapped and uses them to update its own localization and the localization of landmarks. This step decreases the uncertainties of both localization and landmarks.

2.3.2 Introduction of ORB-SLAM

ORB-SLAM is a keyframe-based SLAM system, which is presented by Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós.[11] It rises the reusability of the maps by directly mapping features that can be used for recognition. The system performs relocalisation and loop closing with a high invariance to viewpoint in real time.

Their paper gave us a solution for monocular SLAM problem. Monocular SLAM problem, as they defined, is to estimate the pose of camera and simultaneously reconstruct the world observed by it. Using monocular camera is more complex than using stereo cameras or RGB-D camera because the depth of the detected features are unknown and to solve the problem we need to use multi view geometry. Overall, the system is composed of three tasks: tracking, local mapping and loop closer, that run parallel in a multicore machine. The main feature of this system is its capability of relocalisation when track is lost. This relocalisation capability is the result of the integrated place recognition system. That place recognition was based on bags of binary words with ORB features[6], yielding a real time relocalisation system.

Another main feature of ORB SLAM is also due to the use of place recognition system and it is loop closing capacity. The importance of loop closing is that if a loop is detected we can do global bundle adjustment to finally optimize the whole map and generate a fully reusable map that can used to localize a different camera. We will briefly discuss them in the following contents.

The system stores several information from the real world, which are map points, ketframes and covisibility graph. Here we have a brief introduction of the implication of these information.

To form the structure of a 3D reconstruction, we need to obtain map points. Each map point corresponds to a textured planar patch in the world whose position has been triangulated from different views and refined by bundle adjustment.[11] Here the map points correspond to ORB features detected in the images and are the triangulation of FAST corners. Each map point ha associated a patch normal n, which is the mean vector of the viewing directions. A single map point may be associated with ORB features in different keyframes, thus the there will be different descriptors for this map point. In this system each map point stores a representative descriptor D, whose distance is least with respect to all associated descriptors. Also each map point stores the maximum d_{max} and minimum distance d_{min} at which it can be observed according to the scale and distance at which it has been observed.

Keyframes are some snapshots that store all the ORB features in the image and the camera pose. It can be discarded if there are enough other keyframes representing one area.

The covisibility information is stored as an undirected weighted graph where each node is a keyframe and an edge between two keyframes exists if they see at least θ common map points. Lower θ means the graph is more interconnected, thus more expensive to traverse. Note that the weight of the edges is the number of common map points.

The tracking is to localize the camera with every frame and decide when to insert a new keyframe. Before tracking, the map initialization is conducted to find the relation matrix from two near frames, for planar scenes it is homography. If the initial map exists, the tracking estimates the camera pose with every incoming frame. The tracking has several steps. The first step is extracting ORB features in the frame. As we mentioned before, the ORB features are extracted by using FAST algorithm. At first a scale image pyramid is computed and FAST key points are detected at each level, then only a subset of these keypoints will be described. If the tracking is successful in the last frame, it tries to get a first estimation of the camera pose from the last frame. Each ORB feature from last frame searches a match in the current frame in a small area near its previous position. If the camera happens a fast movement then no enough correspondences can be found, thus the matching process will repeat again with a bigger searching area. Now we have 3D to 2D correspondences and we are able to compute camera pose by solving PnP using RANSAC scheme. If the tracking is lost, the place recognition module is used to perform a global relocalization. Once there is an initial estimation of the camera pose and feature matches, a local visible map is retrieved using the covisibility graph of keyframes that is maintained by the system. Then matches with the local map

points are searched by reprojection, and camera pose is optimized again with all matches. Finally the tracking thread decide if a new keyframe is inserted.

The local mapping processes new keyframes and updates and optimizes their local neighborhood. First the keyframe is transformed into its bag of words representation. This process returns a bag of words vector that will be used in loop closing step. By triangulating ORB features in different keyframes we create the new map points. Different from PTAM, the points are triangulated with N neighbor keyframes in the covisibility graph that share most points. The next step is updating local area. After a map point is created, it is checked not to be wrong during the next three keyframes. After this since there is new measurement of points we need to recompute the covisibility graph and representative descriptor D for points that have new measurements. Then the local bundle adjustment is presented to optimize the currently processed keyframe, all the key frames that connected to it in the covisibility graph and all the map points seen by these keyframes. The keyframes that see the points but are not connected to in in the covisibility graph will stay fixed. After the optimization the normal n, scale invariance distances d_{min} and d_{max} for each map point will be recomputed.

The loop closing searches for loops with every new keyframe. If a loop is detected the system performs global optimization that maintain the global consistency of the map. The loop closing has following steps: Loop detection, detect the loop that is covisibility consistent; Compute the similarity transformation, too close the loop we need to compute the similarity transformation from the current keyframe to the loop keyframe which gives us the information that how much error are accumulated in the loop; Synchronization with local mapping thread; Covisibility pose graph optimization and Global Bundle Adjustment.

At this point we have introduced the basic methods that involved with ORB-SLAM. The following section introduces the application of ORB-SLAM in our system.

2.3.3 Implementation of ORB-SLAM

In this project, we built a modified version of ORB SLAM 2 whose source code was released under GPLv3 license to meet the purpose of this project. Compare to monocular version, the stereo version does not need much time for map initialization. We have fixed some error caused by library version update. The main contribution we made on the source code is that we convert the vocabulary file from $.txt$ to binary word, which reduces the vocabulary loading time. Our version of ORB SLAM is able to read from USB cameras in real time, run tracking without initialization and speed up the reading visual vocabulary process.

2.4. OpenGL 3D Rendering

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for the purpose of rendering real-time 3d graphics. In this project, we use OpenGL to render the 3D model.

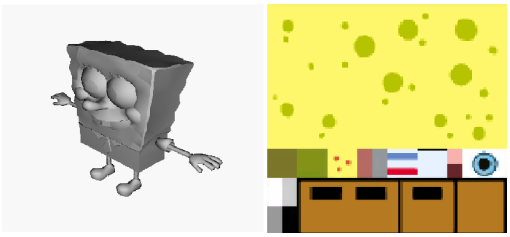

Figure 3: A $SpongeBob$ 3D model is shown at left and its texture is shown at right. The 3d model is stored in $.obj$ file. In $.obj$ files, vertices, texture coordinates and vertex normals are stored in order.

2.4.1 Rendering Pipeline

A rendering job starts from a set of one or more vertex buffers, which are filled with arrays of vertex attributes. These attributes are used as inputs to the vertex shader. Common vertex attributes include the location of the vertex in 3D space, and one or more sets of texture coordinates that map the vertex to a sample point on one or more textures. The set of vertex buffers supplying data to a rendering job are collectively called the vertex array. When a render job is submitted, we supply an additional element array, an array of indexes into the vertex array that select which vertices get fed into the pipeline.

The GPU begins by reading each selected vertex and pass it through the vertex shader, a program that takes a set of vertices attributes as input and outputs a new set of attributes. A typical vertex shader calculates the projected position of the vertex in screen space. Then GPU connects the projected vertices to form triangles by taking taking every three vertices. In this project, we also use vertex shader to generate texture corrdinates, for the rasterizer to blend across the surface of the triangles connecting the vertex.

The rasterizer takes each triangle, clips it and discards part that are outside of the screen, and breaks the remaining visible parts into pixel-sized fragments.

Finally, the generated fragments pass through anouther program called the fragment shader. The fragment shader receives the out by the vertex shader and interpolated by the rasterizer as inputs. It outputs color and depth values that get passed it the frame buffer. Common fragment shader operations include texture mapping and lighting. While we only use the most simple texture mapping fragment shader

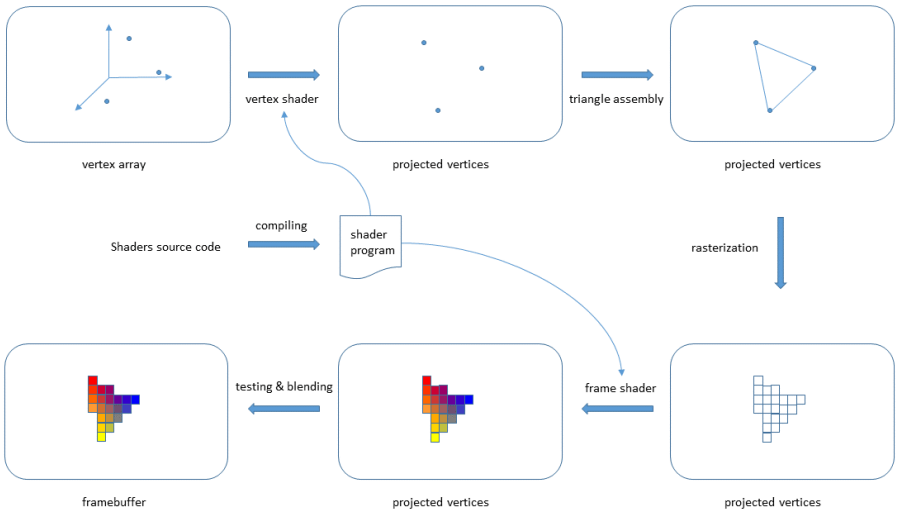

Figure 4: Shaders are programmed in a language called GLSL: GL Shader Language, which is part of OpenGL.Unlike C or Java, GLSL has to be compiled at tun time, which means that every time launching your application, all shaders are recompiled.

in this project.

Note that, after OpenGL 3, a modern rendering pipeline in which the vertex shader and the fragment shader are independent from the main program replace the legacy one that combines all shaders together. A complete diagram of the rendering pipeline is shown in Fig.

2.4.2 MVP Matrices

In OpenGL, the screen plane always stay on xy-plane and center at the origin $(0, 0, 0)$. As opposed in OpenCV where the Z axis is pointing forward, OpenGL has an unintuitive Z which is towards the back.

Furthermore, OpenGL uses homogeneous coordinates and that leads to $4by4$ transformation matrix defined as

$$\mathbf{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

where r_{ij} represent the rotation and xyz are the translation.

Transformation matrix is usually decomposed into the multiplication of three matrices, the model, view and projection matrices. The Model, View and Projection (MVP) matrices are designed to separate different kinds of transformation clearly. By applying the model matrix to all vertices of the model, we derive a scaled, rotated and translated model. The following step is to set the camera pose. View matrix $i.e.$ the transformation matrix of the camera, as defined above, decides the camera pose which is obtained from the SLAM system. Finally, a perspective projection should be applied to project those vertices onto the screen plane.

Note that, in OpenGL, the y and z axis are inverted compared to OpenCV. Due to this reason, A additional matrix that can invert y and z axis should be applied in front of model matrix and view matrix. In conclusion, the MVP matrix are computed as

$$\mathbf{MVP} = \mathbf{P} \cdot \mathbf{V} \cdot \mathbf{M} \tag{18}$$

where **M,V,P** are model, view and projection matrix respectively.

3. Experimental Results

The implementation results for each part discussed above are provided and discussed in this section. Firstly the performance of stereo camera calibration is evaluated. Secondly the performance of AKAZE and ORB planar tracking are compared. Then the ORB-SLAM system is presented by testing on the TUM RGB-D SLAM dataset[15] At the end we show the results of our Sponge Bob AR result.

3.1. Reprojection Error

This experiment investigates the performance with respect to the number of images of model plane (here is the

chessboard pattern). The number varies from 1 to 20 and the result is shown in Fig.5.

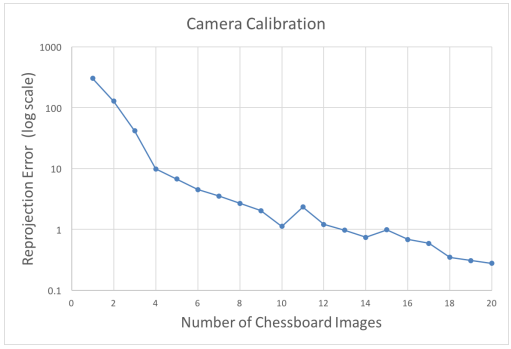

Figure 5: Reprojection Error v.s. Number of Chessboard (model plane) Images. The reprojection error decreases with increasing model plane images. That is to say, the calibration becomes more accurate with more model plane images.

The reprojection error decreases when more images are used and can reach 0.2771 with 20 images, which indicates the highly accuracy of calibration.

3.2. Planar Tracking

This experiment compares the AKAZE and ORB features based planar tracking. By testing on a sample video, the number of matches and runtime are presented for each frame. Fig.6 shows the testing on a sample video. During the test, the AKAZE planar tracking was observed to be more stable than ORB planar tracking when we move the marker fast.

Fig.7 shows the matches for both algorithms on each frame. From observation, the AKAZE has around 200 more matches than ORB, which means that it is far more stable than ORB. Fig.8 shows the runtime of both algorithms on each frame. The runtime of AKAZE is around 80 ms, and for ORB the runtime is around 15 ms. This indicates that ORB feature is more suitable for real-time application. Table 1 compares the mean matches and mean runtime of AKAZE and ORB.

In brief, ORB is faster than AKAZE while AKAZE has more matches than ORB. But in practice, the matching ability for both algorithms is enough for normal application. Considering that ORB has faster running speed than AKAZE, we choose ORB feature for real-time AR.

Fig.9 shows the real-time AR based on planar tracking. A cuboid is attached on the planar and the visual angle will change as cameras changing their poses. In addition, the camera pose can still be tracked even though part of the planar is blocked because enough number (more than the threshold) of feature points can still be detected.

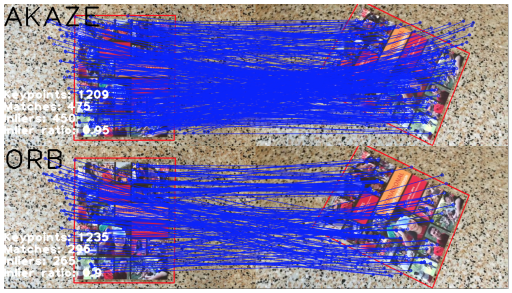

Figure 6: Testing of AKAZE planar tracking and ORB planar tracking. Note that ORB has less match points than AKAZE, thus the tracking performance of AKAZE is more stable than ORB.

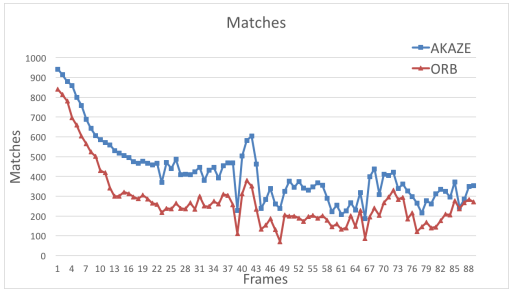

Figure 7: Matches comparison between AKAZE planar tracking and ORB planar tracking. Note that ORB has less match points than AKAZE, thus the tracking performance of AKAZE is more stable than ORB.

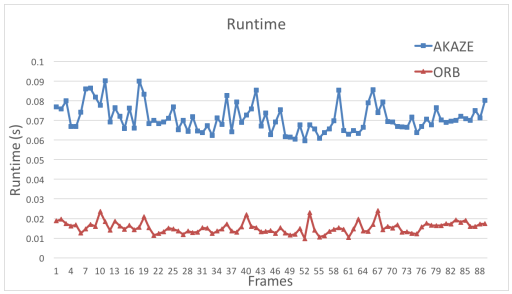

Figure 8: Runtime comparison between AKAZE planar tracking and ORB planar tracking of each frame. Note that ORB is faster than AKAZE, thus the tracking performance of ORB is more efficient than AKAZE.

3.3. Evaluation of ORB-SLAM

To evaluate the ORB-SLAM performance, we adopted a dataset provided by the computer vision group of Technical University of Munich. The dataset contains RGB-D data and ground-truth data with the goal to establish a novel

	AKAZE	ORB
Mean Matches	419	277
Mean Runtime (s)	0.077	0.015

Table 1: Mean matches and mean runtime of AKAZE and ORB. AkAZE has more matches than ORB while ORB has less runtime than AKAZE.

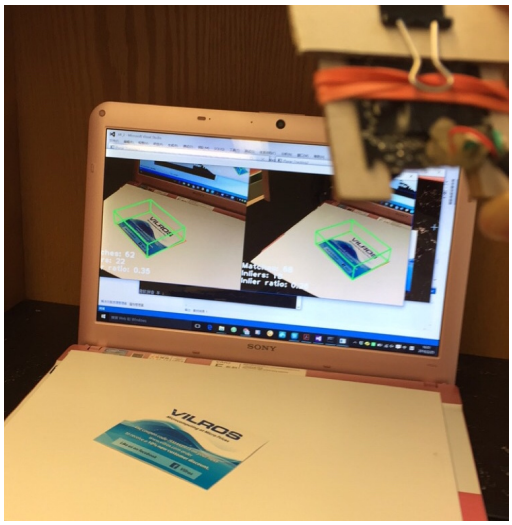

Figure 9: Planar tracking based real-time AR. The planar need to be put on a relatively clean (does not contain interest features), meanwhile the planar need to be complex enough (contains enough number of feature points). Two sides of the window show the different visual angle of left and right camera respectively.

	RMSE	mean	std	min	max
Translation (cm)	1.48	1.29	0.72	0.12	4.20
Rotation (deg)	1.44	1.29	0.62	0.14	2.92

Table 2: The table shows translation error and rotation error of ORB-SLAM. Four measurements are presented which are root mean square error(RMSE), mean error, standard error, minimum error and maximum error.

benchmark. The data was recorded at full frame rate (30 Hz) and sensor resolution (640480).

Fig.10 shows the translation error between estimated camera trajectory and the given ground truth. We can easily observed that the overall error distance is less than 0.07m and mostly it is around 0.02m. A tentative inference on those high peaks and noise is that they are associated to the fast camera movement caused lost of tracking. Fig.11 shows the ground truth and estimated camera movement trail and the difference between them. Table 2 presents translation and rotation error under different measurements.

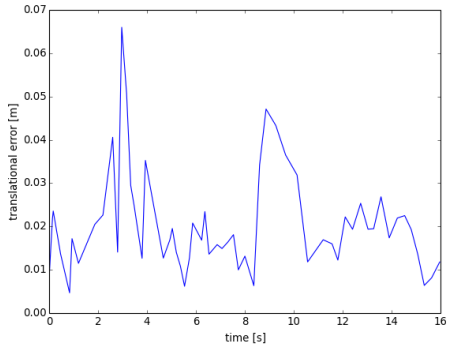

Figure 10: Translation error between estimated camera trajectory and ground truth. Errors are less than 0.07m and mostly are around 0.02m, indicates the ORB-SLAM is robust. The high peak and noise come from the lost of tracking caused by large motion.

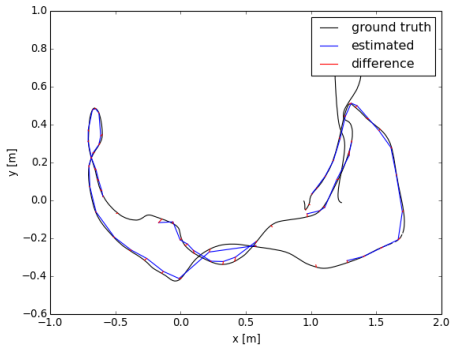

Figure 11: This figure shows the real and estimated camera trails and the difference between estimated results and ground truth. Note that black line is ground truth, blue line is estimated camera trail and red line is difference between two sets of data. From the plots we can observe that there is little difference between estimated result and ground truth, which indicates ORB-SLAM has high accuracy.

3.4. ORB-SLAM AR

In this section, we will show the final result of our part in the whole project. The system has two running modes, which are Planar tracking mode and ORB-SLAM mode. It starts with Planar tracking mode, once the planar is found, a Sponge Bob 3D model will be rendered above the planar. As long as there are enough detected feature points on the planar marker, the model will attach on the marker and will move as the marker moves. We can also switch to ORB-SLAM mode, at this point, the system will start using ORB-SLAM to estimate and tracking the camera pose as well as

build a map of the current environment. Once the system loses tracking, we can reset it to the Planar tracking mode. We are also able to control the model to spin and scale the size of it. The performance turns out to be very fast and robust. Fig.12 shows the planar tracking mode. On the left, it shows that stereo cameras capture the environment and the marker in the environment, on the right is the processed frame that a Sponge Bob model is rendered on the marker.

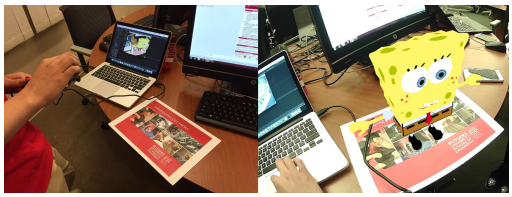

Figure 12: AR effect under Planar tracking mode. A planar marker is put on the desk, stereo cameras capture the scene and detect marker in the scene (left image). Once the feature points on the planar marker are detected, a 3D model will be rendered on it (right image).

Fig.13 shows the ORB-SLAM mode. The left image shows that stereo cameras are capturing the environment, note there is no marker in the environment, the right image shows a Sponge Bob model rendered in the environment.

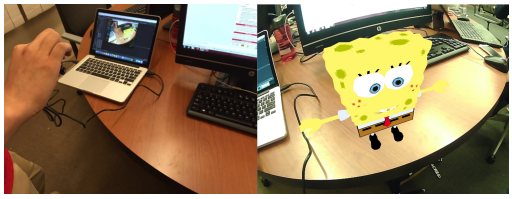

Figure 13: AR effect under ORB-SLAM mode. Note there is no marker in the environment. Stereo cameras capture the scene, detect feature points in the environment and build a map of the current environment (left image). Once the feature points are detected, the camera pose will be computed and we will render a 3D model in the environment (right image).

Both approaches to estimate and track camera poses have several drawbacks. For planar tracking, firstly a specific marker is necessary, this indicates that to ensure system running the environment has to contain markers. Secondly, the marker should not be too small, too far away from cameras or too simple. In brief, the marker captured by the camera and showed in the frame should ensure that enough feature points can be detected, otherwise, the camera will lose tracking. Compared with the planar tracking approach, ORB-SLAM does not need markers in the scene, thus, it is very flexible that it can almost run in any arbitrary scene. However, there is a big drawback of ORB-SLAM which is that the whole background/environment or most part of it has to be still. This is because ORB-SLAM will optimize feature points and update the map periodically. If the environment has a lot of moving objects, the map will be updated after it recomputes the map points, therefore, we will lose tracking, meanwhile, the object will not be able to stay at the same position as in previous frames. In brief, the planar tracking mode has a limitation that a marker has to be in the scene while ORB-SLAM mode has a limitation that the environment needs to be still. Therefore, planar tracking is more suitable for dynamic backgrounds whereas ORB-SLAM is better for static environments.

4. Discussion

Up to now, we have successfully achieved our goals step by step, using ORB-SLAM to achieve a robust real-time AR. All the implementations so far were done on PC. Considering the portability of our head mounted AR device, we may try to implement real-time AR on Raspberry-Pi in the future, then attach the Raspberry-Pi on the Google cardboard. We can also add more functionality to the system such as combining hand gesture recognition to enable some control activities.

For any visual systems, the illumination condition of the environment is always the biggest challenge. During our tests, we have observed the system is easy to lose tracking in some lighting condition even though human eyes feel it is easy to recognize interest points. This problem is highly concerned by researchers and still has no perfect solutions yet. Another big problem we have faced is that cameras will change exposure level as lighting condition changes so that it influences our system performance a lot. We have not found a right way to control it.

5. Cost Analysis

Our head mounted AR device is supposed to be an economical entertainment solution. Comparing with the price to be released Microsoft HoloLens, which is predicted to cost around 3000 USD, the cost of our system is very low. We are using a cheap Google cardboard kit(19 USD or DIY) and a smartphone, which is available for almost everyone, as our head mounted display platform. Combining with two USB cameras (40 USD), we can achieve the stereoscopic AR effect. This solution is able to provide users a cheap (under 100 USD in total) and easy-to-get entertainment device to let them interact with enriched world achieved by the human imagination.

6. Current Trends in Robotics

Robotics is the field that combines mechanical engineering, electrical and computer engineering, computer science and other research fields that deals with the design, construction, operation, and application of robots. These technologies can take the place of humans in dangerous environments or manufacturing processes, or look and behave like humans. They may even have cognition like humans. Today robots are applied in almost every industry and have brought human huge benefit.

Different from industrial applications, much of the research in robotics focuses on investigations into new types of robots, alternative ways to think about or design robots, and new ways to manufacture them. Robotics is changing the way human live and interact with the environment. It contains a bunch of components such as power source, actuation, sensing, manipulation, locomotion, environmental interaction, navigation, human-robot interaction and so on.

Nowadays robots are supposed to have human intelligence. Artificial intelligence is an area strongly related to robotics. It creates computers and machines that are able to behave intelligently. The central problems of AI research include reasoning, knowledge, planning, learning, natural language processing, interaction, cognition and the ability to move and manipulate objects.

Localization, or knowing where am I is a central topic of research in mobile robotics today. It is the essential part of navigating a robot to where it supposed to go. One of the very hot topics is the research in autonomous cars. Obviously, future cars able to drive themselves must also know where they are so they can maneuver in time to take a given exit at the highway, respect lanes in one-way streets and so on. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage.[5][6] Autonomous cars have control systems that are capable of analyzing sensory data to distinguish between different cars on the road, which is very useful in planning a path to the desired destination.[7] For autonomous cars there exists two challenges: (1) the car must stay localized against some kind of map and (2) localization should be efficient, robust and in real-time. We all know that the exist GPS-based navigation systems only have the meter-order accuracy. Since urban navigation applications require localization with centimeter-order accuracy, GPS will not be a proper solution.

The most researches today is focusing on how to build an accurate map of the environment while the car is moving, then we can be able to navigate it according to the map in the environment. As we mentioned above, SLAM is a hot topic of how to solve localization and mapping problems for robotics. Since the it was first proposed fifteen years ago, there are numerous of algorithms were proposed, and it has not been completely solved. today there is still considerable research going on in the field. Published approaches are employed in self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers, newly emerging domestic robots and even inside the human body [10]. Related methods include EKF SLAM, FastSLAM, GraphSLAM, ORB-SLAM, MonoSLAM and so on.

Jess Levinson, Michael Montemerlo, Sebastian Thrun from Stanford Artificial Intelligence Laboratory have proposed a technique for high-accuracy localization of moving vehicles that utilizes map of urban environments. [8] This technique is applied to Google's self-driving car project. They employed GPS, IMU, wheel odometry, and LIDAR data acquired by an instrumented vehicle to generate high-resolution environment maps. LIDAR is a 3D laser scanner, it is currently the most valuable sensor for autonomous vehicles. It can provide a 3D scan of the environment, with range (depth) and reflectivity information for each direction in the space. They also employed offline relaxation techniques which are similar to recent SLAM methods.

Their work discussed a key problem that barely discussed in most SLAM literature, which is the dynamic environment. Because urban environments are dynamic, for successful localization the system has to detect the static part of the environment. Previous works on SLAM in dynamic environments mostly focused on tracking or removing objects that are moving at the time of mapping. Differently, their work reduces the map to features that with very high likelihood are static. In their approach only the flat road surface is retained. The resulting map is simply an overhead image of the road surface. After the map has been built, a particle filter is used to localize the vehicle in real-time.

Jakob Engel, Dr. Jrg Stckler, Prof. Dr. Daniel Cremers from Technical University of Munich also proposed a method named LSD-SLAM, which means Large-Scale Direct Monocular SLAM.[5] It is a novel, direct monocular SLAM technique. This work is in the field named Visual SLAM. Instead of using keypoints as some Visual SLAM methods, it directly operates on image intensities both for tracking and mapping. The camera is tracked using direct image alignment. Meanwhile the system builds semi-dense depth maps of the environment by filtering over many pixelwise stereo comparisons.

LSD-SLAM can run in real-time on a CPU, and even on a modern smartphone. As direct method, LSD-SLAM uses all information in the image, including edges, while keypoint-based approaches can only use small patches around corners. This leads to higher accuracy and more robustness in sparsely textured environments (e.g. indoors), and a much denser 3D reconstruction. Further, as the proposed pixelwise depth-filters incorporate many small-baseline stereo comparisons instead of only few large-baseline frames, there are much less outliers. LSD-SLAM

builds a pose-graph of keyframes, each containing an estimated semi-dense depth map. [5]

In [16] LSD-SLAM are used to reconstruct 3D model of the street scenes. It is different from the approach that we have mentioned above of street-scene 3D reconstruction from a driving car, which relies on 3D laser scanning or complex offline computation from visual images. Here, images captured by the camera are the only sensing information from the world. In this paper, they provided us a comparison of real-time capable 3D reconstruction method using a stereo extension of LSD-SLAM with laser-based maps and traditional stereo reconstructions based on processing individual stereo frames. In this reconstruction, small-baseline comparison over several subsequent frames are fused with fixed-baseline disparity from the stereo camera setup. They also demonstrate that SLD-SLAM provides an excellent compromise between speed and accuracy, generating visually pleasing and globally consistent semi-dense reconstructions of the environment in real-time on a single CPU.

There are still many researches going on to built safe and smart self-driving cars. Despite the huge progress in computer vision in recent several years, we all know all current methods are not comparable to human brain in interpreting and understanding images. That is to say, it is still not safe enough to rely self-driving cars on cameras and computer vision techniques. And due to the extremely high price of sensors required by state-of-the-art prototypes, we are still far from seen autonomous cars in our cities.

Since modern robots are supposed to have the ability to understand the world, firstly they need to recognize the world. Object recognition is therefore a vital component in a robots all kinds of skills. It is one of the most widely studied topics in artificial intelligence, currently, even the best object detectors still fail much of the time. Recently some researchers began to employ SLAM methods to achieve more reliable recognition.

Sudeep Pillai and John J. Leonard from MIT Computer Science and Artificial Intelligence Laboratory published a paper in June 2015 [12] that have developed a monocular SLAM supported object recognition which is able to achieve considerably stronger recognition performance as compared to classical object recognition system that function on a frame-by-frame basis.

They classify systems into "SLAM-aware" and "SLAM-oblivious" system. SLAM-aware system is one that has access to the map of its surroundings as it builds it gradually and the location of its camera simultaneously. This is in contrast to classical recognition system that are SLAM-oblivious, which that detects and recognizes objects on a frame-by-frame basis without being cognizant of the map of its environment. The location of its camera, or the objects may be tracked and computed within these maps.

In their research they developed the ability for a SLAM-aware system to robustly recognize objects in its environment using an RGB camera as its only sensory input. The combination of this system with a monocular visual-SLAM (VSLAM) also enables us to take advantage of both the reconstructed map and camera location to significantly robust recognition result.

Despite using only the output of an ordinary video camera, the systems performance is already comparable to that of special-purpose robotic object-recognition systems. And because the system can blend information obtained from different camera angles, it works much better than object-recognition systems trying to identify objects in single-angle images. Because a SLAM map is three-dimensional, it does a better job of distinguishing objects that are near each other than single-perspective analysis can.

More important, the SLAM data let the system correlate the segmentation of images captured from different perspectives. Analyzing image segments that likely depict the same objects from different angles improves the systems performance.

Very recently VR and AR have been very hot topics. They are technologies change the way human interact with the surrounding environment, and are generally believed to become the next big computing platform. They have the potential application in education, entertainment, health care, retail, engineering and military. SLAM methods also have applications in those fields.

In our project we employed ORB-SLAM to compute and track the camera pose, thus implement Augmented Reality. The LSD-SLAM can also be used for AR. As described in [14], the estimated semi-dense depth maps are in-painted and completed with an estimated ground-plane, which then allows to implement basic physical interaction with the environment.We present a direct monocular visual odometry system which runs in real-time on a smartphone. Being a direct method, it tracks and maps on the images themselves instead of extracted features such as keypoints. New images are tracked using direct image alignment, while geometry is represented in the form of a semi-dense depth map. Depth is estimated by filtering over many small-baseline, pixel-wise stereo comparisons. This leads to significantly less outliers and allows to map and use all image regions with sufficient gradient, including edges.

With the development of robotics, the robot nowadays can behave extremely close to human beings, even has better ability than humans in some scenarios. In face recognition field, some algorithms even have a higher accuracy compare with humans. Those examples give us the awareness that machines have the ability to beat humans, but on the other hand, employment of robots in industries has increased productivity and efficiency savings and is typically seen as a long term investment for benefactors.

References

[1] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. Kaze features. In *Computer Vision–ECCV 2012*, pages 214–227. Springer, 2012. 4

[2] P. F. Alcantarilla and T. Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell*, 34(7):1281–1298, 2011. 4

[3] R. T. Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997. 1

[4] J. Corso, D. Burschka, and G. Hager. Direct plane tracking in stereo images for mobile navigation. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 875–880. IEEE, 2003. 3

[5] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, September 2014. 11, 12

[6] D. Gálvez-López and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *Robotics, IEEE Transactions on*, 28(5):1188–1197, 2012. 5

[7] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *Robotics and Automation, IEEE Transactions on*, 7(3):376–382, 1991. 1

[8] J. Levinson, M. Montemerlo, and S. Thrun. Map-based precision vehicle localization in urban environments. In *Robotics: Science and Systems*, volume 4, page 1. Citeseer, 2007. 11

[9] T. Mazuryk and M. Gervautz. Virtual reality-history, applications, technology and future. 1996. 1

[10] P. Mountney, D. Stoyanov, A. Davison, and G.-Z. Yang. Simultaneous stereoscope localization and soft-tissue mapping for minimal invasive surgery. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2006*, pages 347–354. Springer, 2006. 11

[11] R. Mur-Artal, J. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *Robotics, IEEE Transactions on*, 31(5):1147–1163, 2015. 4, 5

[12] S. Pillai and J. Leonard. Monocular slam supported object recognition. *arXiv preprint arXiv:1506.01732*, 2015. 12

[13] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011. 3

[14] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for AR on a smartphone. In *International Symposium on Mixed and Augmented Reality*, September 2014. 12

[15] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012. 7

[16] V. Usenko, J. Engel, J. Stueckler, and D. Cremers. Reconstructing street-scenes in real-time from a driving car. In *Proc. of the Int. Conference on 3D Vision (3DV)*, Oct. 2015. 12

[17] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, Nov. 2000. 2

Appendix

Project Source Code:

We published our source code on github, so everyone can contribute to this project.

https://github.com/XwK-P/ORB_AR

AKAZE and ORB Planar Tracking:

We learned how to implement AKAZE and ORB planar tracking from this document provided by OpenCV. Then we modified a real-time planar tracking version for our system.

http://docs.opencv.org/3.0-beta/doc/tutorials/features2d/akaze_tracking/akaze_tracking.html

ORB-SLAM Source Code:

We use these source code to understand ORB-SLAM method and developed our modified version for this project.

https://github.com/raulmur/ORB_SLAM

https://github.com/raulmur/ORB_SLAM2

OpenGL Tutorial:

The code on this tutorial page gave us the idea of how to render 3D objects. We used their code to study the implementation and wrote our own program to render a 3D Sponge Bob model.

http://www.opengl-tutorial.org/