EGR 481 – 482 Senior Project Report

Flight Test Results of the Collision Avoidance System for a Fixed-Wing UAS using Stereoscopic Vision

Prepared By:

Anthony Tan

Jake Banuelos

Kishan Patel

Minwoong Chae

Jonathan Cratsley

Muhammad Moiz Khanqadri

Daniel Boebinger



**Table of Contents**

List of Figures iv

Introduction 1

2.0 Theory 2

2.1 Image Rectification 2-3

2.2 Camera Calibration 3

2.3 Disparity Map Generation 4

3.0 Hardware 5

3.1 Airframe 6

3.2 Flight Computer 6-7

3.3 Stereo Cameras 7-8

3.4 Autopilot 8-10

3.5 Ground Control Station 10-12

3.6 Xbee Communication 12-13

4.0 Software 14

4.1 Camera Calibration and Image Rectification 14

4.2 Disparity Map Generation 14-16

4.3 Intel NUC and Camera API 16-17

4.4 Collision Avoidance 18-20

5.0 Results 20

5.1 Ground and Static Tests 20-22

5.2 Flight Test- Post Processing 23-24

5.3 Flight Test- Real Time Processing 25-26

6.0 Conclusion and Future Work 28

7.0 Acknowledgements 29

8.0 References 30

9.0 Appendix A1

Appendix A A2-A6

Appendix B A7-A10

Appendix C A11-A17

Appendix D A18-A28

Appendix E A29

**List of Figures**

Figure 2.2 Epipolar geometry 3

Figure 2.3 Disparity map generation 4

Figure 3.1 System Architecture block diagram 5

Figure 3.2 Sig Kadet Senior Sport 6

Figure 3.3 Intel NUC D54250WYK kit 7

Figure 3.4 Point Grey Chameleon cameras for stereo vision 8

Figure 3.5 APM 2.6 9

Figure 3.6 Mission Planner Snapshot 11

Figure 3.7 Xbee Explorer & PX4 FMU 4-pin UART port 13

Figure 3.8 AMP 2.7 telemetry pin out 13

Figure 4.1 Left image 15

Figure 4.2 Right image 15

Figure 4.3 Blended image of figure 4.1 and figure 4.2 16

Figure 4.4 Ground truth disparity map of figure 4.1 and figure 4.2 16

Figure 4.5 Intel NUC operating system and camera API 17

Figure 4.6 Grid Generation with selected quadrant 19

Figure 4.7 Collision Avoidance Process 20

Figure 5.1 BM and SGBM disparity maps of picture 1 22

Figure 5.2 BM and SGBM disparity maps of picture 2 22

Figure 5.3 Disparity map generated from two images using BM 23-24

Figure 5.4 On Board Processing Disparity Maps 26

1. **Introduction**

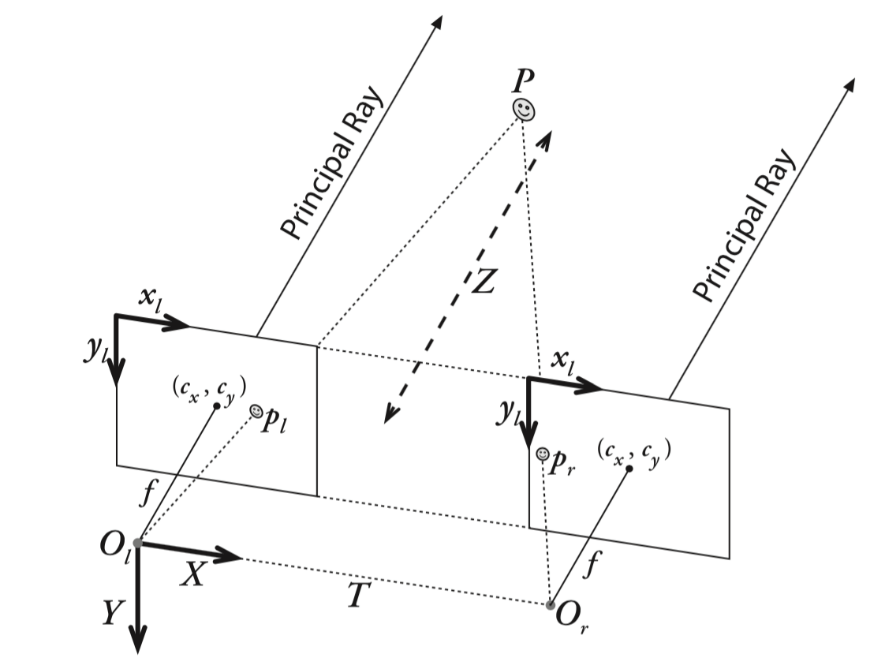
With the popularity of unmanned aerial systems (UAS) for both commercial and civilian use, the need for a collision avoidance has become apparent to be able to implement them into the Nation Airspace System (NAS). The Federal Aviation Association (FAA) has been pushing for safety among these unmanned aerial vehicles (UAV) which includes collision avoidance systems. There have been many proposed and researched ideas that have tried to solve the problem of collision avoidance for safety. One of these is use of automatic dependent surveillance- broadcast (ADS-B) sensors that would be implemented into each unmanned aerial system (UAS) to be able detect other ADS-B sensors from other UAS. Another technology that is being proposed is Lidar sensors which will map out the area and be able to detect things on the third dimensional plane. The one this paper proposes is stereoscopic vision detecting. This will implement the C++ library, OpenCV, which implements camera vision techniques similar to artificial intelligence. The thing that is special about stereoscopic vision is that it uses two cameras. The two cameras allows the system to be able to take two dimensional images and turn them into third dimensional images which are called depth, or disparity, maps. This is done by using an algorithm that makes use of parameters such as focal points and distances between the cameras. The reason why stereoscopic imaging would be a good collision avoidance system is that it can act as a redundant system to things like ADS-B and Lidar while also being implemented as a standalone system that is more affordable than the other sensors making it more available to the UAV enthusiast.

This paper will discuss the application the team has made to improve the previous code for the stereoscopic vision. It will also include the discussion of the results of the flight test and static ground tests. Future work and recommendations will then be discussed as well.

**2.0 Theory**

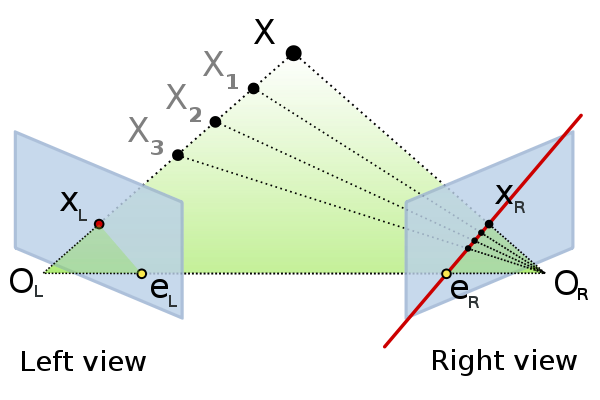
**2.1 Image Rectification**

The ideal stereovision system is shown in figure 2.1. The two image planes are coplanar to each other, and the projection point, and , are on the same row. However, the real stereovision system rig is barely reaching the requirement. It causes the corresponding projection points on the two image planes may not locate on the same row. Therefore, the purpose of image rectification is row-aligned, which means making the corresponding projection points be on the same pixel level. To do the image rectification, epipolar geometry is introduced.



**Figure 2.1 Stereovision system**

Epipolar geometry is shown in figure 2.2. The two rectangles represent the image plane of left camera and right camera respectively, and X represents the points captured by both cameras. and are optical centers. X projects on left image plane on point and projects on right image plane on point . Drawing a line from to , the line is called baseline. The intersection points of baseline and two image planes are called epipole, and . The lines formed by and is called epipolar line, and and are the same. The goal of rectification is making the two epipolar lines parallel to make and on the same pixel row. The next step would be



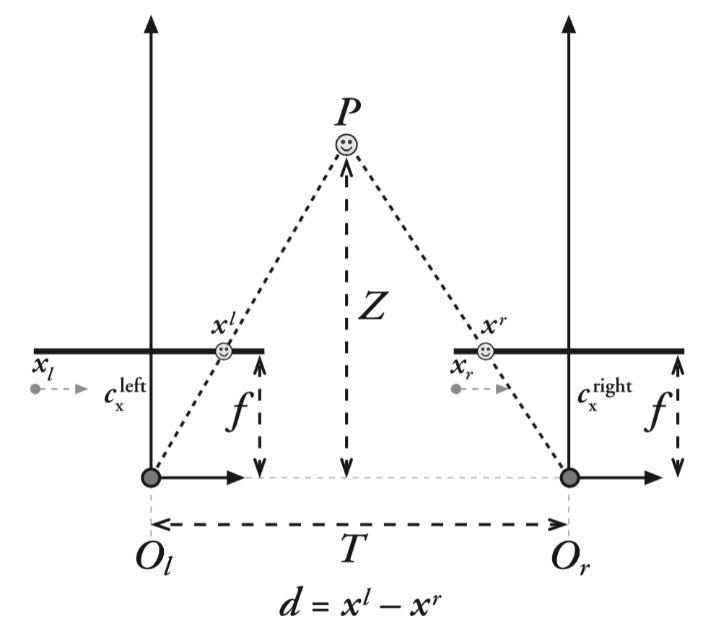
**Figure 2.2 Epipolar geometry**

**2.2 Camera Calibration**

Camera calibration produces parameters that will be used for image rectification. These parameters consist of intrinsic matrix and extrinsic matrix. Intrinsic matrix contains the information of the property of the camera and the relationship between the camera coordinate and image coordinate. Extrinsic matrix describes the relationship between the world coordinate and camera coordinate. Generating the distortion parameters and doing stereo calibration are also required in this step. Camera lens generates two types of distortion, which are radial distortion and tangent distortion. The distortion parameters would minimum the distortion. Stereo calibration is the process of calculating the geometrical relationship between the two cameras in physical world. After all the parameters are acquired, the next step is ready to process.

**2.3 Disparity Map Generation**

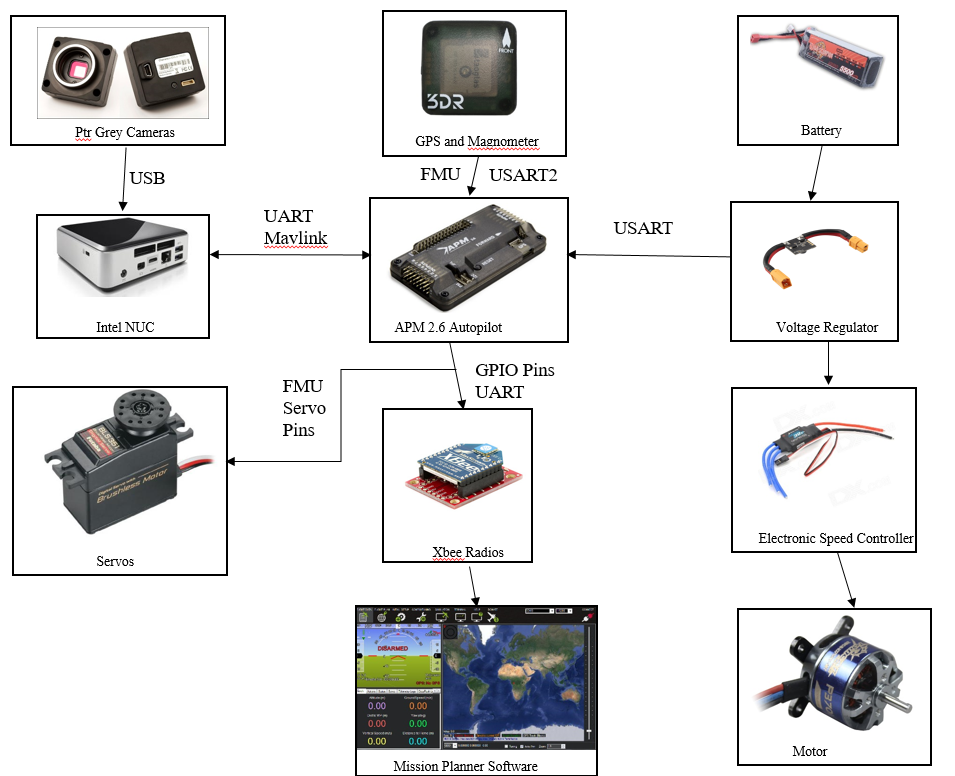
Given a point P in physical world, two projection points appears on both left image plane and right image plane. These two projection points are and , and the disparity between these two projection points is called disparity, as shown in figure 2.3. If the observed point is closer to the image planes, the disparity will be smaller. The larger disparity represent that the disparity between observed point and image planes is longer. Hence, disparity is utilized for determining the relative distance between object and cameras.



**Figure 2.3 Disparity map generation**

**3.0 Hardware**

Figure 3.1 shows the different hardware of the whole system used and will be integrated with each other. It shows how one connects to the other whether it be Universal Asynchronous Receiver/Transmitter (UART) or a hard connection such as Universal Serial Bus (USB). The following section will explain the different hardware used.



**Figure 3.1 System Architecture block diagram**

**3.1 Airframe**

The aircraft used for this project is the Sig Kadet Senior Sport which is shown in Figure 3.2. The Sig Kadet Senior Sport was chosen due to its light weight and great stability and maneuverability. The reason this aircraft is so light is because it consists of balsa wood and plywood. It also has long wings giving the aircraft a high-wing configuration which increases its stability in air. The wings are connected by a spar and then hooked into the fuselage. Nylon screws are then placed in the back of the wing to ensure the wing is fully attached and will break off to separate the wings in case of a crash. The tail surfaces and small dihedral angles give the aircraft great maneuverability so it is easy to fly. The servos control the ailerons, elevators, and rudders. The propeller is attached in the front of the aircraft as seen in Figure 3.2 to generate thrust. The Sig Kadet Senior Sport was the ideal option for this project due to its great characteristics.



**Figure 3.2 Sig Kadet Senior Sport**

**3.2 Flight Computer**

The flight computer is the Intel NUC D54250WYK kit, which is a powerful mini-computer designed by Intel. This particular NUC has the 4th generation Intel Core i5-4250U processor, we installed 8 GB of DDR3L RAM, and we also installed a 64GB mSATA Solid State Drive as the primary source of memory. This particular kit comes with four USB 3.0 ports and an Ethernet port. For external peripherals, we have a simple wireless mouse and keyboard. We have two ways to access this NUC, the first way is to connect it to an external monitor that supports either Mini-HDMI or HDMI; the second way is to use an external 10/100 MBPS desktop switch which we would then connect to the NUC and a personal computer or laptop using Ethernet cords and then open a Remote Desktop Connection. The cameras are connected into the back two USB 3.0 slots. We stripped the NUC of its casing in order to lose mass and so it would fit within the planes interior.

****

**Figure 3.3 Intel NUC D54250WYK kit**

**3.3 Stereo Cameras**

For the first half of the project, the cameras of choice were the Logitech HD Webcam C615. Ultimately, these cameras were scrapped because they were low quality and produced far too much noise. Consumer grade webcams only come as USB 2.0 thus they would all be too low quality for this project. The project needed light-weight yet powerful USB 3.0 cameras and after some research the Point Grey Chameleon3 cameras seemed to be a good choice. These cameras can capture color images at 1.3 MP using USB 3.0 with resolutions up to 1280x1024 and up to 149 FPS. The dimensions of these cameras are 44mm x 35mm x 19.5mm with a mass of 54.9 grams. Special software and API are also required for these cameras to function (See **Software** section). These cameras also require a lens to be attached before use; Fujinon YV2.8×2.8SA-2, found on the Point Grey website, were chosen for their low profile design and for being light, 50 grams. These lens, however, must be manually focused for the desired range thus are slightly more challenging to properly set up. The USB 3.0 cables used were the ACC-01-2300 USB 3.0 Type-A to Micro-B cable found on the Point Grey website, these cables are 3 meters long and were slightly longer than was needed however, they had a locking mechanism which tightly secured them to the cameras so they would not unplug during the flight.



**Figure 3.4 Point Grey Chameleon cameras for stereo vision**

**3.4 Autopilot**

In order to detect potential collision threats that our aircraft may encounter while flying cameras and other sensors working with the autopilot are implemented into the aircraft. Collision threats include, but are not limited to, ground based structures and other aircraft. The autopilot that is being utilized for this aircraft is the Ardupilot APM 2.6 which comes with pre-loaded open source firmware that allows any fixed wing aircraft full autonomous capability. The firmware provides advanced functions such as support for hundreds of three-dimensional waypoints, automatic take-off and landing, as well as sophisticated mission planning and camera controls. It also allows for a compatibility with a variety of Ground Control Station software for programming and mission operations. For full autonomy, the APM 2.6 requires an external compass away from magnetic interference as well as a GPS unit. The APM 2.6 also has a built-in hardware failsafe processor that is will enable the aircraft to return to the launch site if it experiences radio loss. The APM 2.6 can be seen below in Figure 3.5.



**Figure 3.5 APM 2.6**

In order for communication to take place between the autopilot and the ground control station there needs to be some type of hardware to transmit and receive the data. In this case two XBee radios were the hardware of choice which wirelessly transmit and receive the packets of data which can be viewed and analyzed in the ground station. In addition, a 3DR GPS with magnetometer is connected to the Ardupilot using a 6-pin cable and the autopilot receives positioning data from the GPS. This GPS with magnetometer ensures that the autopilot will be fully functional and will give the aircraft the capability of being fully autonomous. The benefit of having a direct connection compatibility of the GPS with the autopilot is that it provides precise navigational data which are shown as waypoints on the ground control station. The orientation of the aircraft is also much more precise with this direct compatibility and ensures that there is minimal error when determining the current aircraft orientation. Because the purpose of this project is to have obstacle avoidance capabilities, accuracy of the autopilot is crucial and having a precise orientation and navigation data is required. The autopilot is being powered through the same power source as the NUC on board which is a separate power source from the propeller’s electric motor. The NUC and the autopilot are directly connected together so the collision avoidance algorithm stored can be implemented on board the aircraft. Having the NUC on board and directly connected to the autopilot enables the aircraft to detect an obstacle and immediately complete the necessary actions to change the current flight path. The autopilot will follow the waypoints generated through the user interface on the ground control station and this will allow the plane to autonomously fly the path necessary for the current mission. If the aircraft senses an obstacle in its path, it will use the collision avoidance algorithm to generate a new route to maneuver around the object and continue on its planned path which was initiated. If no object is sensed in its path, it will continue its route at the same desired elevation and orientation as it was programmed to.

**3.5 Ground Control Station**

The ground control station is where the main operations of the autonomous aircraft are implemented and this includes the input of the initial waypoints for the desired mission as well as the updated waypoints required for collision avoidance that are changed in real time. In this case Mission Planner was chosen as the ground station due to its full featured application already compatible for the Ardupilot. Mission Planner can be used as a configuration utility or as a dynamic control supplement for the autonomous aircraft. Mission Planner is free to download and quick to understand so it was installed and calibrated fairly quickly based off on the current location at Cal Poly Pomona. Once the firmware is configured on the Ardupilot it is possible to setup, configure, and tune the aircraft for optimum and desired performance. It is possible to plan, save, and load autonomous missions into the autopilot straight from Mission Planner. This can be done simply with just point and click-way point entry on interfaces such as Google Maps or any other sort of mapping tools. Between the capabilities of the autopilot and the ground station hundreds of three-dimensional waypoints can be supported for any given mission. The use of this ground control station can be seen in the snapshot of the Mission Planner in Figure 3.6 below.



**Figure 3.6 Mission Planner Snapshot**

With the telemetry hardware on the Ardupilot it is possible to monitor the aircrafts status while it is in operation. One major component is the ability to record the telemetry logs which contain much more information regarding the flight details and this will allow to further view and analyze telemetry logs. It is also possible to operate the aircraft in first person view if this is desired. Some other useful instruments available with Mission Planner are that it is possible to view the planes battery, altitude, speed, orientation, servo functions, real time waypoint generation, and the strength of the signal from telemetry. Someone who is not outside in the field to watch the actual flight, can essentially still watch the flight and all of its conditions straight from the computer using Mission Planner. Mission Planner will serve as the control center for the UAV just as there is a control center for commercial and military flights.

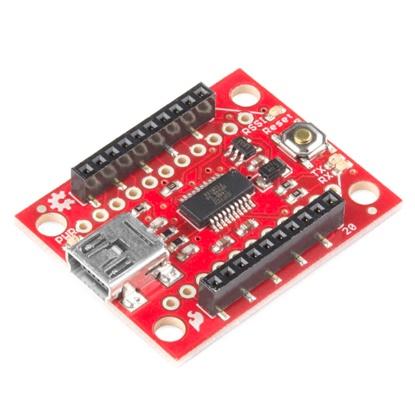
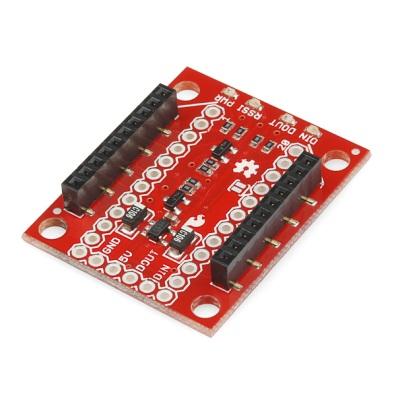
**3.6 Xbee Communication**

To get the XBee telemetry to work with the Mission Planner multiple parameters have to be configured on to the two XBees. The configuration was done by using XCTU, a software application designed by Digi that is used to write parameters onto the XBee. The XBees need to be connected to the computer to be able to implement the parameters on XCTU. The parameters that were changed from the default settings are:

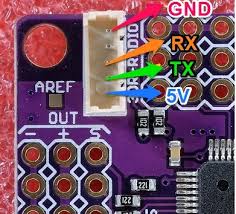
* Baud Rate: 57600
* Pan ID: 155
* Node Identifier: “Ground Station” & “OnBoard”
* Destination Address High:

With these parameters the XBees are ready for the radio telemetry to connect to the Mission Planner. The XBee with the Node Identifier “OnBoard” will be with the aircraft while the XBee with the Node Identifier “Ground Station” will be connected to the computer that is running Mission Planner. To be able to get the connection an XBee explorer and a FMU 4-pin UART port are needed can be found on Figure 3.7. The “Ground Station” XBee will be connected to the XBee explorer. The explorer is connected with a mini USB cable to the ground station computer. The 4-pin is connected to the “OnBoard” XBee. The 4-pin is connected directly to the APM 2.6 to provide the flight data to the “Ground Station” XBee.

When connecting everything on the on board module, the APM pin out is connected to the 4-pin with the pin layout for the APM can be found in Figure 3.8. This will allow the APM to power the “OnBoard” XBee for the telemetry to be viable. With the setup complete the 4-pin will be able to transmit data through the telemetry link.

- 

**Figure 3.7 XBee Explorer (left) & PX4 FMU 4-pin UART port (Right)**



**Figure 3.8 AMP 2.6 telemtry pin out**

The way the 4-pin is able to communicate with the APM 2.6 flight computer is through MAVLink. MAVLink is able to send data packets through the serial channels between the XBees. MAVLink is the main tool used by Mission Planner that allows for the communication between the ground station and the aircraft.

**4.0 Software**

The software of the project is the most important as it provides the core of the projects function: disparity map imaging, collision detection and collision avoidance. The language used to provide most of the coding was C++ for its object oriented programming and the main library to highlight is the Open Source Computer Vision (Open CV) that is used to develop the codes.

**4.1 Camera Calibration and Image Rectification**

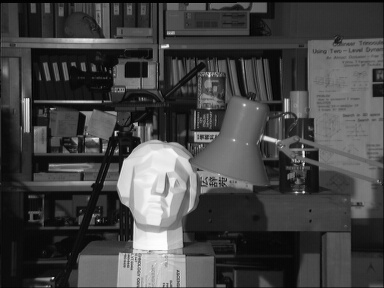
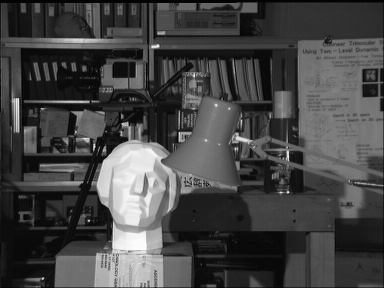
In order to calibrate each camera with OpenCV, each camera needs to capture a series of chessboard. The series of images include 15 different chessboard directions. The corners of the chessboard will be detected and utilized to generate parameters. To generate the distortion parameters, the images are used in this step. To do stereo calibration, the two cameras need to mount still, and the two cameras capture a series of chessboard images simultaneously.

After OpenCV calculates all the parameters, the prerequisite of image rectification is reached. The set of images will be rectified by using the parameters obtained by camera and stereo calibration.

**4.2 Disparity Map Generation**

Choosing a point, a pixel, from left image as a reference point, find a correspond point, a pixel, on the right image and calculate the distance from between the two pixels. Repeating the process over the entire image produces the disparity map. Figure 4.1 and figure 4.2 are two images captured by left camera and right camera, respectively. Figure 4.3 shows the blended image of figure 12 and figure 4.2. Comparing the box corners, in figure 4.3, showing in the left bottom red-circle and white-board angle in top right red-circle, the shifted distance of box corner is larger than the shifted distance of white-board corner. It represents that the box is closer to the camera than the white-board. Using this technique, OpenCV calculates the disparity map.

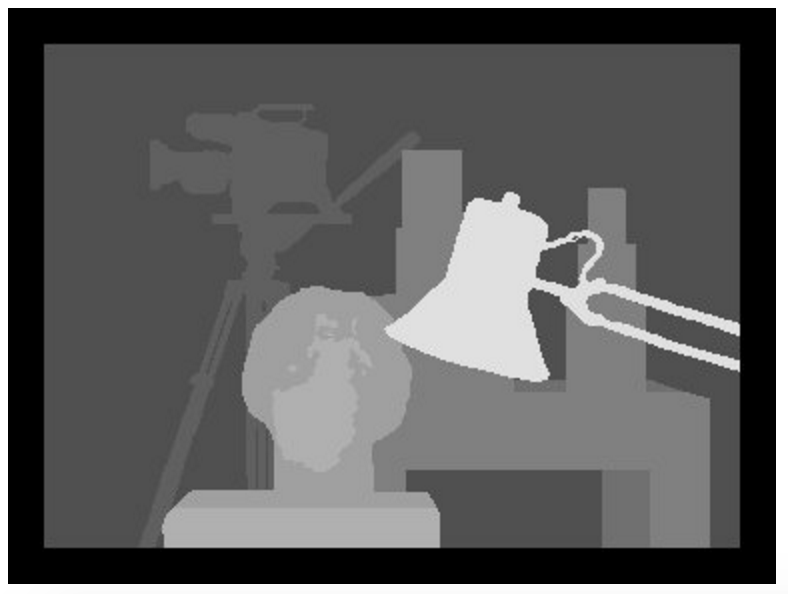
In order to simplify the computational difficulty, the images are converted to gray scale images. The gray image is scaled from 0 to 255. 0 represent black and 255 represents white. The longer of the pixel shifts, the whiter the object shows on the disparity map. Therefore, if the object is closer to the camera shot, it will look whiter. As shown in figure 4.1 and 4.2, because the table light is closer to the camera shot and video recorder is farer, the table light is whiter than the video recorder in figure 4.4. What OpenCV doing here is that it chooses a reference point on left image, finds a correspond point on right image, and calculates the distance to produce the disparity map. The process of finding correspond point is called matching problem. Because the pair of images is rectified, the searching direction is only on the horizontal pixel row. The approach OpenCV using is Sum of Absolute Difference (SAD). It compares two image boxes of left and right images by calculating the similarity of them; the center of the image boxes are reference pixel of left image and target pixel of right image. Choosing the most similarity boxes, the distance of the centers, reference pixel and target pixel, would be the value of the disparity.



**Figure 4.1 Left image Figure 4.2 Right image**



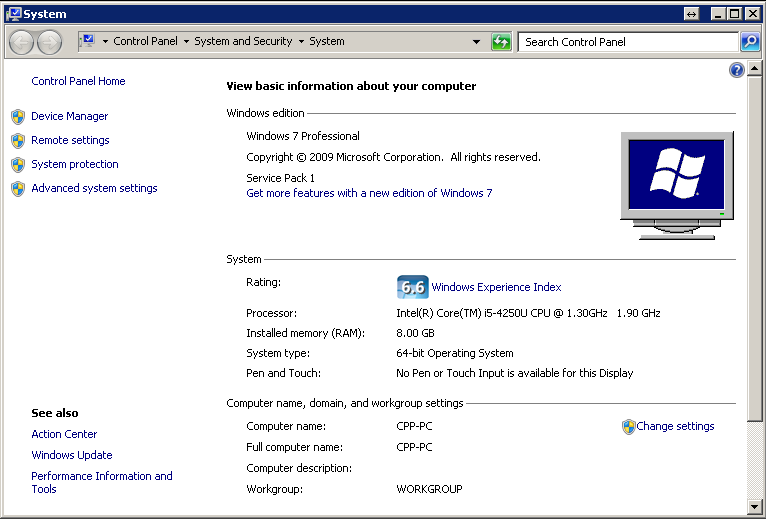
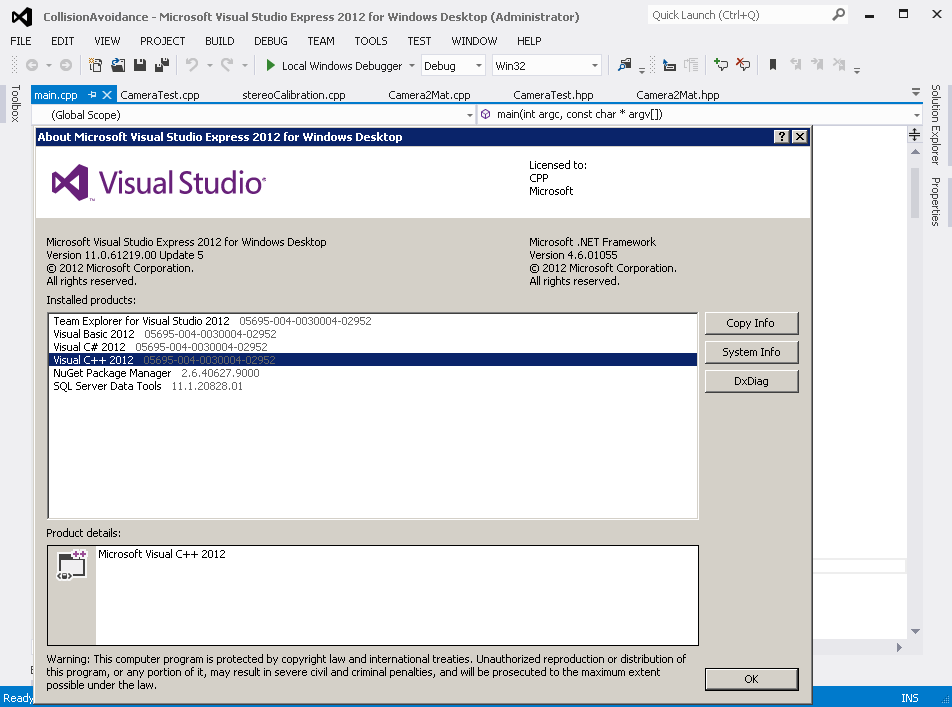
**Figure 4.3 Blended image of figure 12 and figure 13**

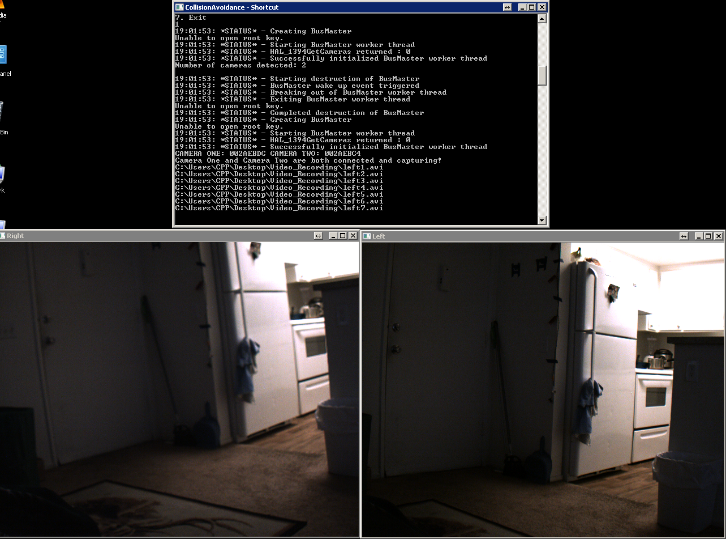


**Figure 4.4 Ground truth disparity map of figure 12 and figure 13**

**4.3 Intel NUC and Camera API**

The Intel NUC is running Microsoft Windows 7 as an operating system. The IDE, Integrated Development Environment, of choice was Microsoft Visual Studio 2012 Express. The two APIs, application program interface, used for this project are OpenCV and FlyCapture2. OpenCV, Open Source Computer Vision Library, is a library that is used for image manipulation, computer vision, and machine learning which was developed by a company called Itseez. The library has more than 2500 algorithms, is designed for Windows, Linux, Andriod, and Mac, and it has C++, C, Python, Java, and MATLAB interfaces. For this project, we are running Open CV 3.1.0, which has improved algorithms for Stereoscopic Vision and Calibration, for more information regarding OpenCV, see the *Stereoscopic Imaging and Camera Calibration* section. The Chameleon3 camera requires the special API called FlyCapture, which was developed by Point Grey team. This API is included, for free, within the FlyCapture2 SDK, software development kit, which includes drivers for the cameras, source code, examples, and the libraries required for the cameras.

****

****

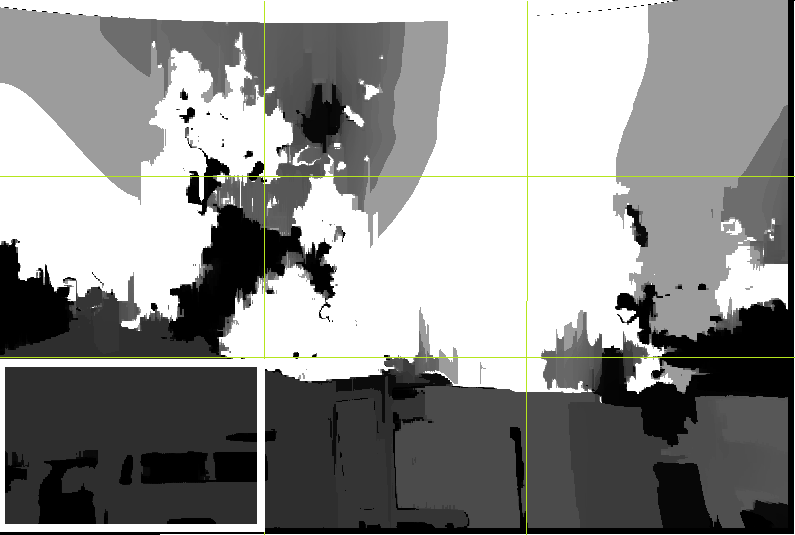
**Figure 4.5 Intel NUC operating system and camera API**

**4.4 Collision Avoidance**

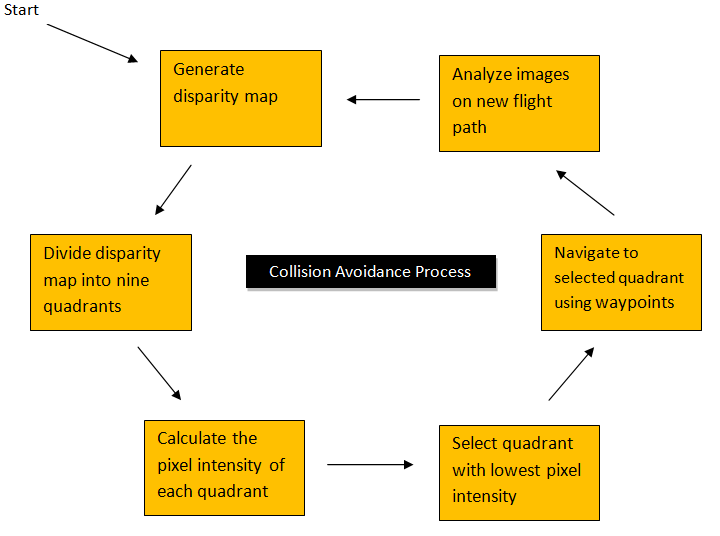
Collision avoidance for aircraft involves the detection of potential objects that are in the flight path of an aircraft, analyzing the trajectory and speed of the hazard, then executing a maneuver that will move the aircraft away from the threat. This concept was developed into a computer program that would execute these commands for an unmanned aerial vehicle (UAV). The algorithm requires the development of a disparity map, which utilizes two cameras to measure field depth, which provide the necessary information required, such as distance from the UAV, and velocity as that distance changes over time.

The collision avoidance process that is used in the algorithm and is broken into three sections, sense, detect, and avoid. The sensing phase is the first portion of the collision avoidance program. This involves the development of the disparity map which uses the camera systems to identify objects in front of the UAV and assigns a high pixel density of that object in order to denote the distance from the UAV. After the sensing phase comes the detection phase, where the possible threats to collision are categorized based on the path of the UAV and the speed, direction, and current location of the object. If one or more of the detected object are deemed as a collision threat, the avoidance phase is initiated. This calculates the safest path to travel based on the least pixel dense region of the disparity map. This will be continually updated as the path planning continues with new stereo data to constantly update the new path.

The collision avoidance algorithm that was used was developed by the previous team under the guidance of Dr. Subodh Bhandari. The collision avoidance algorithm works by developing a nine-quadrant layout of the disparity map. Each quadrant is analyzed by recording the pixel intensity and comparing them to one another. When a collision threat is determined, the algorithm will select the quadrant that has the least pixel density, which signifies the least possible chance of object collision6. When the quadrant is selected, the UAV will execute a maneuver by flying toward the selected quadrant. The disparity map and quadrant layout can be seen in figure 4.6 which is a disparity map that was generated during a ground test. The new heading and altitude were then determined using existing flight data and the angle of inclination to the center of the quadrant of the disparity map. Once the new heading was determined, the algorithm sets a GPS waypoint that the UAV will follow for a set distance to clear it of the obstacle. The collision avoidance process can be seen in figure 4.7.



**Figure 4.6 Grid generation with selected quadrant**

****

**Figure 4.7 Collision Avoidance Process**

**5.0 Results**

**5.1 Ground and Static Tests**

The project uses two algorithms to generate disparity map, which are block matching (BM) and semi-global block matching (SGBM). The block matching requires less time to produce a disparity map, but semi-global block matching yields more accuracy disparity map. Disparity map is generated by using a searching box on left image and finding a correspond box on right image. For the searching box, block matching uses 3 pixels by 3 pixels, and semi-global block matching uses 7 pixels by 7 pixels. Block matching generates one disparity map in 0.093 second, and semi-global block matching requires 0.18 to produce a disparity map. As showing in figure 5.1(c) and (d), the disparity that generated by semi-global block matching provides more details. OpenCV provides a function to refine the disparity map. It reduces noise and smooth the disparity map that yields a more accuracy disparity map. Comparing with the set of figure 5.1(a) and (b) with figure 5.2(a) and (b), figure 5.1(a) and (c) are captured by an aligned stereo rig, but figure 5.2(a) and (b) are captured by an unaligned stereo rig. The aligned stereo rig produces better disparity map. It shows more accuracy relative depth and without much noise.

To compute the disparity map, uses the left image as a reference image and search a correspond point on the right image. The process applies on both block matching and semi-global block matching. However, some conditions cause the error on the disparity map, such as texture-less areas, half-occlusions and regions near depth discontinuities. OpenCV provides a function, or called filter function, to minimum the error, and it requires two disparity maps from a set of left and right image. The first disparity map is same as usual, using left image as reference and searching on right image. And the second disparity map is using right image as reference image and searching left image. The filter function compares these two different disparity maps and minimize the errors.

According to the result, aligned stereo rig provide a good disparity map, so the future work would be reduce the time of producing disparity map. However, if the unaligned stereo rig is being used, an algorithm that has both features of fast and accuracy need to be developed and applied on the stereo system.

1. **Left image (b) Right image**

1. **BM disparity map (d) SGBM disparity map**

**Figure 5.1 BM and SGBM disparity maps generated of picture 1**

1. **Left image (b) Right image**

** **

1. **BM disparity map (d) SGBM disparity map**

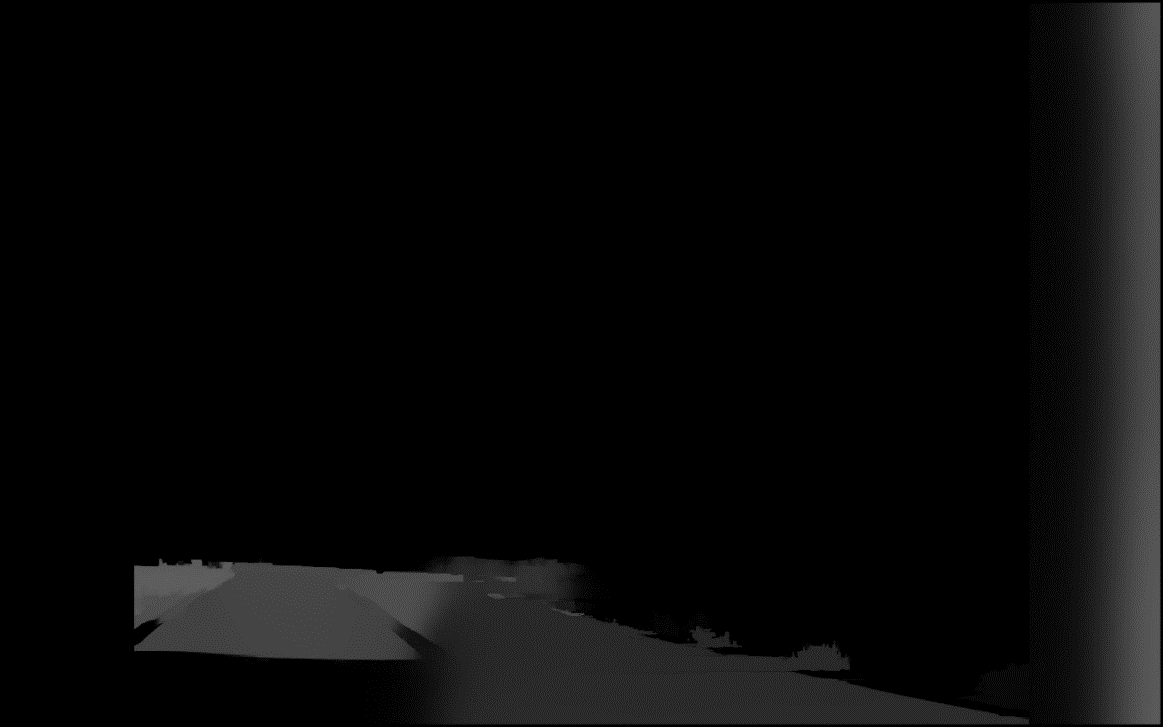
**Figure 5.2 BM and SGBM disparity maps generated of picture 2**

**5.2 Flight Test- Post Processing**

The purpose of the flight test was to test the new coding done with both filtering done as well as trying to increase the disparity maps generated per second. Due to API problems from the Point Grey Cameras, the flight was recorded then post processing images were generated. The cameras were put on top of the wing about 10 inches away from each other. The propeller was a predicted problem, but the team had hope that the image filtering would take care of the high speed blurry propeller. This would prove to actually be a problem since the propeller revolutions per minute were not as fast for the filtering. The image picked was the one with the most depth. There is no noise due to the noise filtering used. This day was particularly clear so the effectiveness of the code was of high quality. The disparity map is clear, although the depth of the image is not very far. The images could be processed around 20 disparity maps per second, but calibration of the cameras are preventing the disparity maps from getting better quality.



1. **Left image (b) Right image**

****

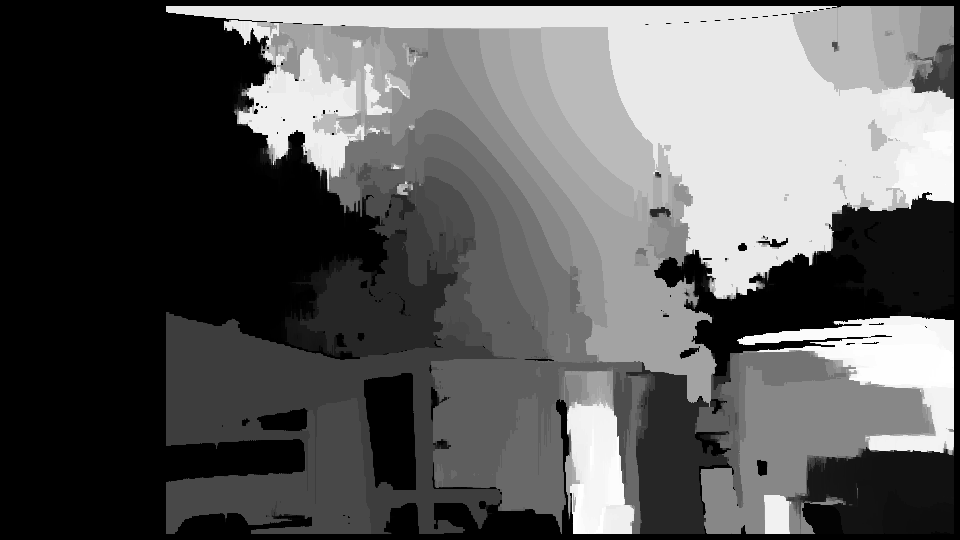
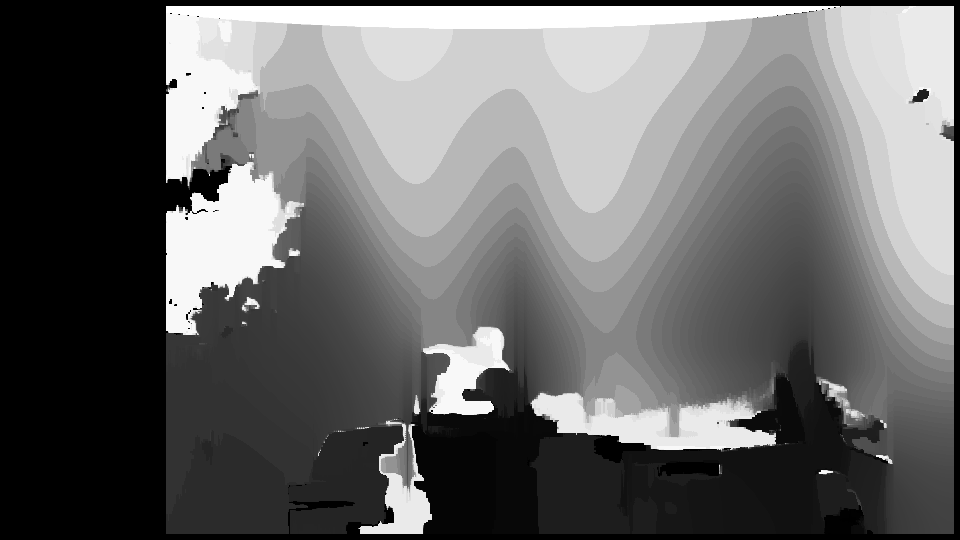
**(c) Disparity map generated**

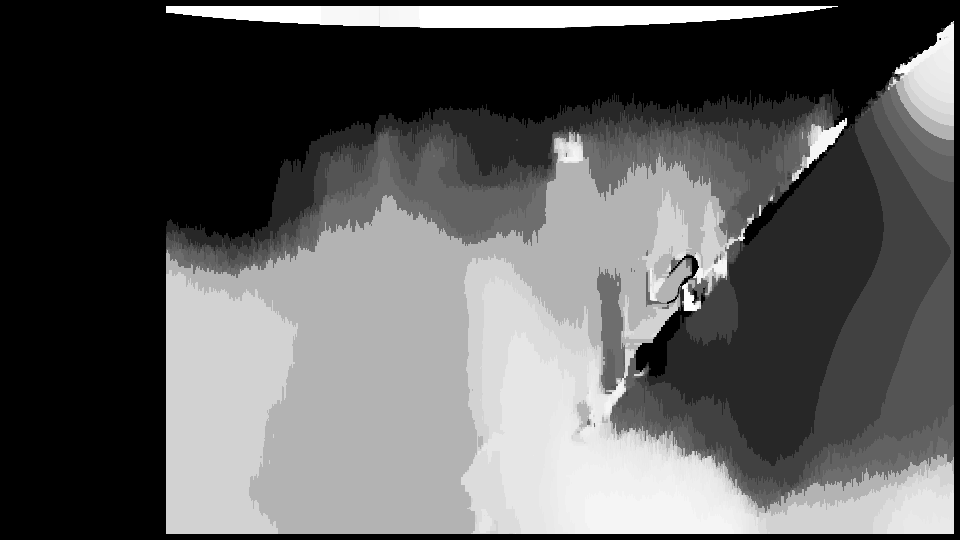
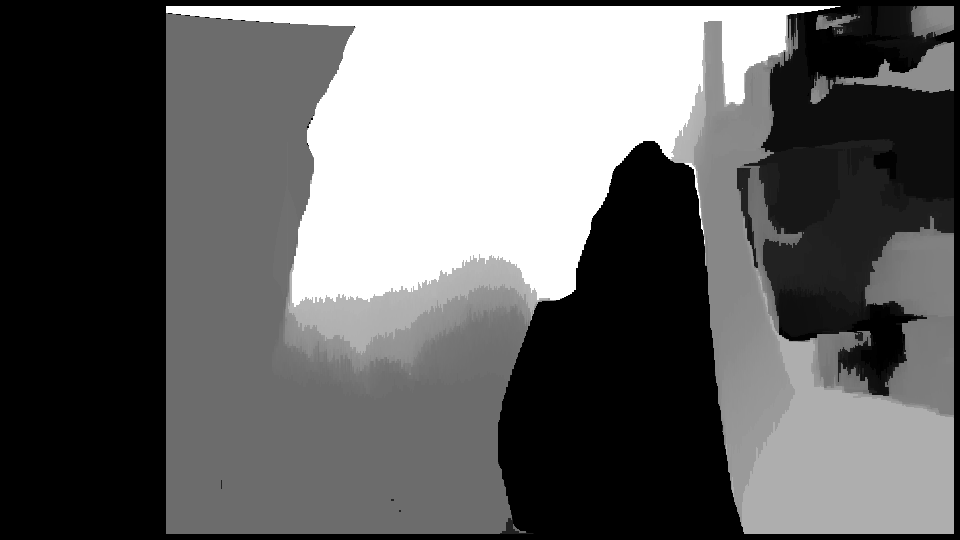
**Figure 5.3 Disparity map generated from two images using BM**

The image seems to only generate about 20 feet in front of it. Also to note, the image of the fence was no generated as an object as well which can be a problem for any meshed object that can cause crashes. The road has a clear image, but on the right side of the lane the map gets really dark and will give the avoidance code bad readings meaning bad quadrant picking. The good that came out of this is that noise did not seem to be a big factor in this seeing that there is not many random white spots on the disparity map meaning for a more accurate map to be generated, but the code does need to be optimized for speed, or processing, and to be able to read further depth since head to head collisions will move relatively more faster so must be detected earlier.

**5.3 Flight Test- Real Time Processing**

This flight test is using the on board Intel NUC to processing while flying. The restriction of this flight testwas that the NUC was not able to record the actual flight image since imaging processing itself took up most of the processing power. By itself, the image processing could get about 5 disparity maps per second and even less if the recording were processed. The images were not as good as desired, but shows where there must be improvement. The image on the top left shows the ground image on the approach with a fence nearby which shows some good depth, but a thing that applies to all these images is there is still noise. The noise might still be a big factor and must be taken care of. The top right and bottom left image show in flight images. The day of the flight test was pretty clear with not much trees nearby so the whitest parts of the image show possible noise that might have interfered. The images do have a good detail on the depth despite the noise. The bottom right image shows that the propeller shows up on the disparity map which is not a good thing. The hope the team had was that because the propeller is so fast the imaging program would filter out it or the cameras would not pick it up due to not having enough frames per second to catch it, but since it did this is going to be a problem. The easiest fix is to position the cameras somewhere where it cannot detect the propellers.

**** ****

**** ****

**Figure 5.4 On Board Processing Disparity Maps**

The problem that might be happening during the flight test is since the NUC does not processing the disparity maps as quickly as hoped the movement of the plane is giving major problems. The movement of the aircraft must be accounted for. The code is more geared towards static images meaning that the code takes processing the same image over and over, but now needs to be changed so that it can handle movement compensation of real aircrafts. This can be a problem which things like vibration and gusts.

**6.0 Conclusion and Future Work**

In the end, the code was vastly improved with the introduction of better imaging processing through filtering as well as different processing to go through for the disparity map. The work was mainly done on getting the disparity map clearer which proved to be successful. A problem that was run into was the stereo calibrations were not properly aligned so the code would give improper readings for the disparity maps. This will need to be improved so that the avoidance code can get correct data from the map on the physical world.

Future work for the project includes testing with real avoidance. Implementing test procedures to be able to run a real avoidance program might be difficult to keep safe. It will also be limited if the code cannot read further than 20 feet since the safest way to test is having an object further away. Also, the movement of the system needs to be accounted for since the images are not appearing as fast as hoped the movement might be causing lots of interference and noise. Not only those, but processing images has been seen to be a big problem so the code must be optimized so that it will use as little processing power for anything other than the image processing so that the disparity maps can be constantly generated for accurate readings. One suggestion can be the use of Lubuntu like the previous team before. This year’s team could not use it because the API for the cameras were not updated for the newest Lubuntu. The application of the Census transform may be used as well for to be able to speed up the processing of the stereo matching since Census masks are large and sparse and perform with the same processing as small dense masks.

As for the improvement of the code, there is three levels that were implemented last year on the collision avoidance that were hoped to be reached. The first level was achieved last year and this year the second level, although not the exact methods, was achieved.

* ***Level 1:*** *Pure quadrant-based navigation → Set waypoint to least-occupied quadrant (completed)*
* ***Level 2:*** *Navigation based on improved disparity map noise-filter & hole filling, such as through Mashhour Solh’s hierarchical hole-filling and depth-adaptive hole-filling methods.(completed)*
* ***Level 3:*** *Navigation using blob detection or the morphological filtering approach used by Carnie et al.01 for obstacle velocity estimation → Set waypoint based on velocity of incoming obstacle (identified using blob detection algorithm and range finding method),and selecting optimal path from multiple low-cost regions of the disparity map.*

A method proposed using a third camera to increase the accuracy of the disparity maps while using block-based and wireframe modeling techniques. In turn, using the block-based imaging technique can create a by producting for compensating 2-D and 3-D motion as well which will improve the on board processing to the autopilot.

The biggest need for future work is the improvement of hardware. The imaging processing is very stressful on the Intel NUC. The Intel NUC used this project is only an I5 processor, so the next upgrade would be a Sixth Generation Intel I7 processor NUC to be able to handle all the image generation. A good number to reach for disparity maps per second would be 15 at the very least to be able to detect objects and read the relative motion of the object. The suggestion that would improve the project immensely is making a mini ITX board, which is essentially a mini computer that would be stronger than the Intel NUC. The main benefit to the min ITX board would be the fact one can implement a graphics processing unit (GPU) instead of having an integrated graphics processing unit like the Intel NUC. The graphics card should give the hardware more variety of choosing what is needed as well as making the imaging processing easier for the processing since the GPU will be taking some of the processing power itself meaning more image and faster disparity maps produced.

A suggestion for testing would be trying to use a multirotor to test since Cal Poly Pomona might have the possibility of being to fly on campus soon. Since there are no runways on the campus it is not easy to fly a fixed wing aircraft so using a multirotor can have us easily test any time. One would test on the multirotor then implement it onto a fixed wing aircraft to do real flight testing.

**7.0 Acknowledgements**

The team that worked on this project would like to express our thanks to our advisor, Dr. Bhandari, as well as the Cal Poly Pomona Aerospace Engineering department staff for the support during the course of this project.

**8.0 References**

1. Schaeffer, R., “A Standards-Based Approach to Sense-and-Avoid Technology,” *AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*, September, 2004.
2. Saneyyoshi, K., “Stereo Vision System on Automobile for Collision Avdoiance,” *IAPR Conference on Machine Vision Applications*, June, 2011.
3. Blin, Richards, et al. “Obstacle Avoidance System for UAVs using Computer Vision.” Proceedings ofAIAA Invotech@Aerospace Conference, FL, 5-9 January, 2015.
4. Park, Jongho, and Youdan Kim. "Obstacle Detection and Collision Avoidance of Quadrotor UAV Using Depth Map of Stereo Vision." *AIAA Guidance, Navigation, and Control (GNC) Conference* (2013).
5. Lacher, A. R., Maroney, D. R., and Zeitlin, A. D., “Unmanned Aircraft Collision Avoidance - Technology Assessment and Evaluation Methods,” *The MITRE Corporation*, 2007.
6. Hrabar, S. "3D Path Planning and Stereo-based Obstacle Avoidance for Rotorcraft UAVs." *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2008).
7. Bhandari, S., Srinivasan, T., Gray, J., Torstenbo, M., Corral, J., Brown, N., Mcdorman, N., Wood, J., and Raheja, A., “Collision Avoidance System using Stereoscopic Vision for Unmanned Aerial Systems,” *AIAA Infotech @ Aerospace*, 2016.
8. Humenberger, M., Zinner, C., Weber, M., Kubinger, W., and Vincze, M., "A Fast Stereo Matching Algorithm Suitable for Embedded Real-time Systems," *Computer Vision and Image Understanding*, Vol. 114, No. 11, pp. 1180-1202, November 2010.
9. Solh, M., and Alregib, G., "Hierarchical Hole-Filling for Depth-Based View Synthesis in FTV and 3D Video." *IEEE Journal of Selected Topics in Signal Processing IEEE*, Vol. 6, No. 5, pp. 495-504, June 2012.
10. Carnie, R. J.,Walker, R. A., and Corke, P. I. “Computer Vision  Based Collision Avoidance for UAVs.” *Proceedings of 11th Australian International Aerospace Congress*, Melbourne, Australia, 2005.
11. Lai, X., Wang, H., and Xu, Y., "A Real-time Range Finding System with Binocular Stereo Vision," *International Journal of Advanced Robotic System*, 2012.
12. Tzovaras, Dimitrios, Nikos Grammalidis, and Michael G. Strintzis. "Disparity Field and Depth Map Coding for Multiview 3D Image Generation." *Signal Processing: Image Communication* 11.3 (1998): 205-30.
13. Bradski, Gray, Kaehler, Adrian. *Learning OpenCV*. California: O’Reilly Media, Inc., 2008. Print.
14. Alvarez, H., L.M. Paz, J. Sturm, and D. Cremers. "Collision Avoidance for Quadrotors with a Monocular Camera." Print.
15. XCTU. (2015). Retrieved May 31, 2015, from http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu
16. Oborne, M. (2015). Mission Planner. Retrieved May 31, 2015, from http://planner.ardupilot.com/

**9.0 Appendix**

**Appendix A: Camera API Implementation**

**Appendix B: Filtered Disparity Map**

**Appendix C: Real Time Disparity Map**

**Appendix D: Single Camera Calibration**

**Appendix E: Stereo Calibration**

**Appendix A: Camera API Implementation**

#include "Camera2Mat.hpp"

#include "FlyCapture2.h"

#include "opencv2/core.hpp"

#include <opencv2/core/utility.hpp>

#include "opencv2/imgproc.hpp"

#include "opencv2/calib3d.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/videoio.hpp"

#include "opencv2/highgui.hpp"

#include "stereoCalibration.hpp"

#include <iostream>

**using** **namespace** FlyCapture2**;**

Camera2Mat**::**Camera2Mat**()**

**{**

**if(**checkNumOfCameras**())**

**{**

cameraOne **=** **new** Camera**();**

cameraTwo **=** **new** Camera**();**

connectCameras**();**

**}**

**}**

Camera2Mat**::~**Camera2Mat**()**

**{**

safelyDelete**();**

**}**

bool Camera2Mat**::**checkNumOfCameras**()**

**{**

BusManager busMgr**;**

FlyCapture2**::**Error error **=** busMgr**.**GetNumOfCameras**(&**numCameras**);**

**if** **(**error **!=** PGRERROR\_OK**)**

**{**

std**::**cout **<<** error**.**GetDescription**()** **<<** std**::**endl**;**

**return** **false;**

**}**

std**::**cout **<<** "Number of cameras detected: " **<<** numCameras **<<** std**::**endl **<<** std**::**endl**;**

**if** **(** numCameras **!=** 2 **)**

**{**

std**::**cout **<<** "Insufficient number of cameras... press Enter to exit." **<<** std**::**endl **;**

std**::**cin**.**ignore**();**

**return** **false;**

**}**

**return** **true;**

**}**

void Camera2Mat**::**connectCameras**()**

**{**

BusManager busMgr**;**

//Each camera has a specific ID so lets retrieve them

PGRGuid guid0**;**

PGRGuid guid1**;**

//GET ID FOR CAM ONE

FlyCapture2**::**Error error **=** busMgr**.**GetCameraFromIndex**(**0**,** **&**guid0**);**

//Error check

**if(** error **!=** PGRERROR\_OK **)**

**{**

std**::**cout **<<** error**.**GetDescription**()** **<<** std**::**endl**;**

**return;**

**}**

//GET ID FOR CAM TWO

error **=** busMgr**.**GetCameraFromIndex**(**1**,** **&**guid1**);**

//Error check

**if(** error **!=** PGRERROR\_OK **)**

**{**

std**::**cout **<<** error**.**GetDescription**()** **<<** std**::**endl**;**

**return;**

**}**

//Display IDs

std**::**cout **<<** "CAMERA ONE: " **<<** guid0**.**value **<<** " CAMERA TWO: " **<<** guid1**.**value **<<** std**::**endl**;**

// CONNECT CAM ONE WITH GUID0

error **=** cameraOne**->**Connect**(&**guid0**);**

//Error check

**if(** error **!=** PGRERROR\_OK **)**

**{**

std**::**cout **<<** error**.**GetDescription**()** **<<** std**::**endl**;**

**return;**

**}**

// CONNECT CAM TWO WITH GUID1

error **=** cameraTwo**->**Connect**(&**guid1**);**

//Error check

**if(** error **!=** PGRERROR\_OK **)**

**{**

std**::**cout **<<** error**.**GetDescription**()** **<<** std**::**endl**;**

**return;**

**}**

//Now that both cameras are connected, lets start the capture!

//START CAPTURE OF CAM 1

error **=** cameraOne**->**StartCapture**();**

//Error check

**if** **(** error **==** PGRERROR\_ISOCH\_BANDWIDTH\_EXCEEDED **)**

**{**

std**::**cout **<<** "Bandwidth exceeded" **<<** std**::**endl**;**

**return;**

**}**

**if(** error **!=** PGRERROR\_OK **)**

**{**

std**::**cout **<<** error**.**GetDescription**();**

**return;**

**}**

//START CAPTURE OF CAM 2

error **=** cameraTwo**->**StartCapture**();**

//Error check

**if** **(** error **==** PGRERROR\_ISOCH\_BANDWIDTH\_EXCEEDED **)**

**{**

std**::**cout **<<** "Bandwidth exceeded" **<<** std**::**endl**;**

**return;**

**}**

**if(** error **!=** PGRERROR\_OK **)**

**{**

std**::**cout **<<** error**.**GetDescription**();**

**return;**

**}**

//Both cameras are now connected and capturing!

std**::**cout **<<** "Camera One and Camera Two are both connected and capturing!" **<<** std**::**endl**;**

**}**

double Camera2Mat**::**getFPS**()**

**{**

double fps **=** 0.0**;**

**return** fps**;**

**}**

void Camera2Mat**::**displayImage**()**

**{** // capture loop

// capture loop

char key **=** 0**;**

**while(**key **!=** 'q'**)**

**{**

// Get the image from cam 1

Image rawLeftImage**;**

FlyCapture2**::**Error error **=** cameraOne**->**RetrieveBuffer**(** **&**rawLeftImage **);**

**if** **(** error **!=** PGRERROR\_OK **)**

**{**

std**::**cout **<<** error**.**GetDescription**()** **<<** std**::**endl**;**

**return;**

**}**

// convert to rgb

Image rgbLeftImage**;**

rawLeftImage**.**Convert**(** FlyCapture2**::**PIXEL\_FORMAT\_BGR**,** **&**rgbLeftImage **);**

// convert to OpenCV Mat

unsigned int rowLeftBytes **=** **(**double**)**rgbLeftImage**.**GetReceivedDataSize**()/(**double**)**rgbLeftImage**.**GetRows**();**

cv**::**Mat left **=** cv**::**Mat**(**rgbLeftImage**.**GetRows**(),** rgbLeftImage**.**GetCols**(),** CV\_8UC3**,** rgbLeftImage**.**GetData**(),**rowLeftBytes**);**

cv**::**resize**(**left**,** left**,** cv**::**Size**(**640**,**520**));**// 1280x1040

cv**::**imshow**(**"Left"**,** left**);**

// Get the image from cam 2

Image rawRightImage**;**

error **=** cameraTwo**->**RetrieveBuffer**(** **&**rawRightImage **);**

**if** **(** error **!=** PGRERROR\_OK **)**

**{**

std**::**cout **<<** "capture error" **<<** std**::**endl**;**

**return;**

**}**

// convert to rgb

Image rgbRightImage**;**

rawRightImage**.**Convert**(** FlyCapture2**::**PIXEL\_FORMAT\_BGR**,** **&**rgbRightImage **);**

// convert to OpenCV Mat

unsigned int rowRightBytes **=** **(**double**)**rgbRightImage**.**GetReceivedDataSize**()/(**double**)**rgbRightImage**.**GetRows**();**

cv**::**Mat right **=** cv**::**Mat**(**rgbRightImage**.**GetRows**(),** rgbRightImage**.**GetCols**(),** CV\_8UC3**,** rgbRightImage**.**GetData**(),**rowRightBytes**);**

cv**::**resize**(**right**,** right**,** cv**::**Size**(**640**,**520**));**// 1280x1040

cv**::**imshow**(**"Right"**,** right**);**

key **=** cv**::**waitKey**(**30**);**

**}**

**}**

cv**::**Mat Camera2Mat**::**imageFromCameraOne**()**

**{**

Image rawLeftImage**;**

FlyCapture2**::**Error error **=** cameraOne**->**RetrieveBuffer**(** **&**rawLeftImage **);**

**if** **(** error **!=** PGRERROR\_OK **)**

**{**

cv**::**Mat null**;**

std**::**cout **<<** error**.**GetDescription**()** **<<** std**::**endl**;**

**return** null**;**

**}**

// convert to rgb

Image rgbLeftImage**;**

rawLeftImage**.**Convert**(** FlyCapture2**::**PIXEL\_FORMAT\_BGR**,** **&**rgbLeftImage **);**

//rgbLeftImage.DeepCopy(&rgbLeftImage);

// convert to OpenCV Mat

unsigned int rowLeftBytes **=** **(**double**)**rgbLeftImage**.**GetReceivedDataSize**()/(**double**)**rgbLeftImage**.**GetRows**();**

cv**::**Mat converted **=** cv**::**Mat**(**rgbLeftImage**.**GetRows**(),** rgbLeftImage**.**GetCols**(),** CV\_8UC3**,** rgbLeftImage**.**GetData**(),**rowLeftBytes**);**

//cv::resize(left, left, cv::Size(640,520));// 1280x1040

cv**::**Mat image**(**converted**.**size**(),** converted**.**type**());**

converted**.**copyTo**(**image**);**

//cv::imwrite("one.jpg", image);

**return** image**;**

**}**

cv**::**Mat Camera2Mat**::**imageFromCameraTwo**()**

**{**

// Get the image from cam 2

Image rawRightImage**;**

FlyCapture2**::**Error error **=** cameraTwo**->**RetrieveBuffer**(** **&**rawRightImage **);**

**if** **(** error **!=** PGRERROR\_OK **)**

**{**

cv**::**Mat null**;**

std**::**cout **<<** "capture error" **<<** std**::**endl**;**

**return** null**;**

**}**

// convert to rgb

Image rgbRightImage**;**

rawRightImage**.**Convert**(** FlyCapture2**::**PIXEL\_FORMAT\_BGR**,** **&**rgbRightImage **);**

//rgbRightImage.DeepCopy(&rgbRightImage);

// convert to OpenCV Mat

unsigned int rowRightBytes **=** **(**double**)**rgbRightImage**.**GetReceivedDataSize**()/(**double**)**rgbRightImage**.**GetRows**();**

cv**::**Mat converted **=** cv**::**Mat**(**rgbRightImage**.**GetRows**(),** rgbRightImage**.**GetCols**(),** CV\_8UC3**,** rgbRightImage**.**GetData**(),**rowRightBytes**);**

//cv::resize(right, right, cv::Size(640,520));// 1280x1040

cv**::**Mat image**(**converted**.**size**(),** converted**.**type**());**

converted**.**copyTo**(**image**);**

//cv::imwrite("two.jpg", image);

**return** image**;**

**}**

void Camera2Mat**::**safelyDelete**()**

**{**

cameraOne**->**StopCapture**();**

cameraOne**->**Disconnect**();**

cameraTwo**->**StopCapture**();**

cameraTwo**->**Disconnect**();**

**delete** cameraOne**;**

**delete** cameraTwo**;**

**}**

**Appendix B: Filtered Disparity Map**

#include "filteredDisparityMap.hpp"

#include "opencv2/calib3d.hpp"

#include "opencv2/imgproc.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/highgui.hpp"

#include "opencv2/core/utility.hpp"

#include "opencv2/ximgproc/disparity\_filter.hpp"

#include <iostream>

#include <string>

**using** **namespace** cv**;**

**using** **namespace** cv**::**ximgproc**;**

**using** **namespace** std**;**

//Mat filteredDisparityMap(Mat left, Mat right, int bm\_sgbm=0)

Mat filteredDisparityMap**::**disparityMap**(**Mat left**,** Mat right**,** int bm\_sgbm**)**

**{**

String algo**;**

bool no\_display **=** **false;**

bool no\_downscale **=** **false;**

int max\_disp **=** 160**;**

double lambda **=** 8000.0**;**

double sigma **=** 1.5**;**

double vis\_mult **=** 1.0**;**

**if(**bm\_sgbm**==**0**)**

algo **=** "bm"**;**

**else**

algo **=** "sgbm"**;**

int wsize**;**

**if(**algo**==**"sgbm"**)**

wsize **=** 3**;** //default window size for SGBM

**else** **if(!**no\_downscale **&&** algo**==**"bm"**)**// && filter=="wls\_conf")

wsize **=** 7**;** //default window size for BM on downscaled views (downscaling is performed only for wls\_conf)

**else**

wsize **=** 15**;** //default window size for BM on full-sized views

Mat left\_for\_matcher**,** right\_for\_matcher**;**

Mat left\_disp**,**right\_disp**;**

Mat filtered\_disp**;**

Mat conf\_map **=** Mat**(**left**.**rows**,**left**.**cols**,**CV\_8U**);**

conf\_map **=** Scalar**(**255**);**

Rect ROI**;**

Ptr**<**DisparityWLSFilter**>** wls\_filter**;**

double matching\_time**,** filtering\_time**;**

**if(**max\_disp**<=**0 **||** max\_disp**%**16**!=**0**)**

**{**

cout**<<**"Incorrect max\_disparity value: it should be positive and divisible by 16"**;**

**}**

**if(**wsize**<=**0 **||** wsize**%**2**!=**1**)**

**{**

cout**<<**"Incorrect window\_size value: it should be positive and odd"**;**

**}**

// if(filter=="wls\_conf") // filtering with confidence (significantly better quality than wls\_no\_conf)

// {

**if(!**no\_downscale**)**

**{**

// downscale the views to speed-up the matching stage, as we will need to compute both left

// and right disparity maps for confidence map computation

//! [downscale]

max\_disp**/=**2**;**

**if(**max\_disp**%**16**!=**0**)**

max\_disp **+=** 16**-(**max\_disp**%**16**);**

resize**(**left **,**left\_for\_matcher **,**Size**(),**0.5**,**0.5**);**

resize**(**right**,**right\_for\_matcher**,**Size**(),**0.5**,**0.5**);**

//! [downscale]

**}**

**else**

**{**

left\_for\_matcher **=** left**.**clone**();**

right\_for\_matcher **=** right**.**clone**();**

**}**

**if(**algo**==**"bm"**)**

**{**

//! [matching]

// cout<< "max\_disp: "<< max\_disp<<endl;cout<< "wsize: "<< wsize<< endl;//max\_disp=16;

Ptr**<**StereoBM**>** left\_matcher **=** StereoBM**::**create**(**max\_disp**,**wsize**);**

wls\_filter **=** createDisparityWLSFilter**(**left\_matcher**);**

Ptr**<**StereoMatcher**>** right\_matcher **=** createRightMatcher**(**left\_matcher**);**

cvtColor**(**left\_for\_matcher**,** left\_for\_matcher**,** COLOR\_BGR2GRAY**);**

cvtColor**(**right\_for\_matcher**,** right\_for\_matcher**,** COLOR\_BGR2GRAY**);**

matching\_time **=** **(**double**)**getTickCount**();**

left\_matcher**->** compute**(**left\_for\_matcher**,** right\_for\_matcher**,**left\_disp**);**

right\_matcher**->**compute**(**right\_for\_matcher**,**left\_for\_matcher**,** right\_disp**);**

matching\_time **=** **((**double**)**getTickCount**()** **-** matching\_time**)/**getTickFrequency**();**

//! [matching]

**}**

**else** **if(**algo**==**"sgbm"**)**

**{**

Ptr**<**StereoSGBM**>** left\_matcher **=** StereoSGBM**::**create**(**0**,**max\_disp**,**wsize**);**

left\_matcher**->**setP1**(**24**\***wsize**\***wsize**);**

left\_matcher**->**setP2**(**96**\***wsize**\***wsize**);**

left\_matcher**->**setPreFilterCap**(**63**);**

left\_matcher**->**setMode**(**StereoSGBM**::**MODE\_SGBM\_3WAY**);**

wls\_filter **=** createDisparityWLSFilter**(**left\_matcher**);**

Ptr**<**StereoMatcher**>** right\_matcher **=** createRightMatcher**(**left\_matcher**);**

matching\_time **=** **(**double**)**getTickCount**();**

left\_matcher**->** compute**(**left\_for\_matcher**,** right\_for\_matcher**,**left\_disp**);**

right\_matcher**->**compute**(**right\_for\_matcher**,**left\_for\_matcher**,** right\_disp**);**

matching\_time **=** **((**double**)**getTickCount**()** **-** matching\_time**)/**getTickFrequency**();**

**}**

//! [filtering]

wls\_filter**->**setLambda**(**lambda**);**

wls\_filter**->**setSigmaColor**(**sigma**);**

filtering\_time **=** **(**double**)**getTickCount**();**

wls\_filter**->**filter**(**left\_disp**,**left**,**filtered\_disp**,**right\_disp**);**

filtering\_time **=** **((**double**)**getTickCount**()** **-** filtering\_time**)/**getTickFrequency**();**

//! [filtering]

conf\_map **=** wls\_filter**->**getConfidenceMap**();**

// Get the ROI that was used in the last filter call:

ROI **=** wls\_filter**->**getROI**();**

**if(!**no\_downscale**)**

**{**

// upscale raw disparity and ROI back for a proper comparison:

resize**(**left\_disp**,**left\_disp**,**Size**(),**2.0**,**2.0**);**

left\_disp **=** left\_disp**\***2.0**;**

ROI **=** Rect**(**ROI**.**x**\***2**,**ROI**.**y**\***2**,**ROI**.**width**\***2**,**ROI**.**height**\***2**);**

**}**

// }

//collect and print all the stats:

// cout.precision(2);

cout**<<**"Matching time: "**<<**matching\_time**<<**"s"**<<**endl**;**

cout**<<**"Filtering time: "**<<**filtering\_time**<<**"s"**<<**endl**;**

// cout<<endl;

Mat filtered\_disp\_vis**;**

**if(!**no\_display**)**

**{**

// namedWindow("left", WINDOW\_AUTOSIZE);

// imshow("left", left);

// namedWindow("right", WINDOW\_AUTOSIZE);

// imshow("right", right);

//! [visualization]

// Mat raw\_disp\_vis;

// getDisparityVis(left\_disp,raw\_disp\_vis,vis\_mult);

// namedWindow("raw disparity", WINDOW\_AUTOSIZE);

// imshow("raw disparity", raw\_disp\_vis);

getDisparityVis**(**filtered\_disp**,**filtered\_disp\_vis**,**vis\_mult**);**

namedWindow**(**"filtered disparity"**,** WINDOW\_AUTOSIZE**);**

equalizeHist**(**filtered\_disp\_vis**,** filtered\_disp\_vis**);**

// imshow("filtered disparity", filtered\_disp\_vis);

// imwrite("disparity\_map.jpg", filtered\_disp\_vis);////////////////////////////////////////

//! [visualization]

**}**

**return** filtered\_disp\_vis**;**

**}**

**Appendix C: Real Time Disparity Map**

#include "realtimeDisparityMap.hpp"

#include <stdio.h>

#include <iostream>

#include "opencv2/core.hpp"

#include <opencv2/core/utility.hpp>

#include "opencv2/imgproc.hpp"

#include "opencv2/calib3d.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/videoio.hpp"

#include "opencv2/highgui.hpp"

**using** **namespace** cv**;**

**using** **namespace** std**;**

void realtimeDisparityMap**::**readingData**()**

**{**

FileStorage leftFile**(**"left.yml"**,** FileStorage**::**READ**);**

FileStorage rightFile**(**"right.yml"**,** FileStorage**::**READ**);**

FileStorage stereoFile**(**"extrinsics.yml"**,** FileStorage**::**READ**);**

leftFile**[**"camera\_matrix"**]>>** leftCameraMatrix**;**

leftFile**[**"distortion\_coefficients"**]>>** leftDistCoeffs**;**

rightFile**[**"camera\_matrix"**]>>** rightCameraMatrix**;**

rightFile**[**"distortion\_coefficients"**]>>** rightDistCoeffs**;**

stereoFile**[**"R1"**]>>** R\_left**;**

stereoFile**[**"R2"**]>>** R\_right**;**

stereoFile**[**"P1"**]>>** P\_left**;**

stereoFile**[**"P2"**]>>** P\_right**;**

**}**

void realtimeDisparityMap**::**bm**()**

**{**

//1: the usb on the left hand side of computer

//0: the usb on the right hand side of computer

//the value may be different on different computer

VideoCapture left**(**1**),** right**(**0**);**

**if(** **!(**left**.**isOpened**()&&**right**.**isOpened**())** **)**

**{**

cout**<<** "Camera is not opened properly."**<<** endl**;**

**return;**

**}**

readingData**();**

Mat left\_map1**,** left\_map2**;**

Mat right\_map1**,** right\_map2**;**

initUndistortRectifyMap**(**leftCameraMatrix**,** leftDistCoeffs**,** R\_left**,** P\_left**,** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_16SC2**,** left\_map1**,** left\_map2**);**

initUndistortRectifyMap**(**rightCameraMatrix**,** rightDistCoeffs**,** R\_right**,** P\_right**,** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_16SC2**,** right\_map1**,** right\_map2**);**

int key **=** 0**;**

Mat leftRawImg**,**rightRawImg**;**

Mat leftImage**,** rightImage**;**

// create the image in which we will save our disparities

Mat imgDisparity16S **=** Mat**(** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_16S **);**

imgDisparity8U **=** Mat**(** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_8UC1 **);**

int number\_disparities **=** DEFAULT\_NUMBER\_DISPARITIES**;**

int block\_size **=** DEFAULT\_BLOCK\_SIZE**;**

Ptr**<**StereoBM**>** bm\_realtime **=** StereoBM**::**create**(**number\_disparities**,** block\_size**);**

//key = 32 means pressing the spacebar

//stopping capturing by pressing spacebar

**while(**key **!=** 32**){**

left **>>** leftRawImg**;**

right **>>** rightRawImg**;**

cvtColor**(**leftRawImg**,** leftRawImg**,** CV\_BGR2GRAY**);**

cvtColor**(**rightRawImg**,** rightRawImg**,** CV\_BGR2GRAY**);**

resize**(**leftRawImg**,** leftRawImg**,** Size**(**imgSize\_width**,** imgSize\_height**));**

resize**(**rightRawImg**,** rightRawImg**,** Size**(**imgSize\_width**,** imgSize\_height**));**

remap**(**leftRawImg**,** leftImage**,** left\_map1**,** left\_map2**,** INTER\_LINEAR**);**

remap**(**rightRawImg**,** rightImage**,** right\_map1**,** right\_map2**,** INTER\_LINEAR**);**

bm\_realtime**->**compute**(** leftImage**,** rightImage**,** imgDisparity16S **);**

double minVal**;** double maxVal**;**

minMaxLoc**(** imgDisparity16S**,** **&**minVal**,** **&**maxVal **);**

imgDisparity16S**.**convertTo**(** imgDisparity8U**,** CV\_8UC1**,** 255**/(**maxVal **-** minVal**));**

imshow**(**"BM: real-time disparity map"**,**imgDisparity8U**);**

key **=** waitKey**(**10**);**

**}**

**}**

Mat realtimeDisparityMap**::**bm**(**Mat left**,** Mat right**)**

**{**

readingData**();**

Mat left\_map1**,** left\_map2**;**

Mat right\_map1**,** right\_map2**;**

//initUndistortRectifyMap(leftCameraMatrix, leftDistCoeffs, R\_left, P\_left, Size(imgSize\_width, imgSize\_height), CV\_16SC2, left\_map1, left\_map2);

//initUndistortRectifyMap(rightCameraMatrix, rightDistCoeffs, R\_right, P\_right, Size(imgSize\_width, imgSize\_height), CV\_16SC2, right\_map1, right\_map2);

int key **=** 0**;**

Mat leftRawImg**,**rightRawImg**;**

Mat leftImage**,** rightImage**;**

// create the image in which we will save our disparities

Mat imgDisparity16S **=** Mat**(** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_16S **);**

Mat imgDisparity8U **=** Mat**(** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_8UC1 **);**

int number\_disparities **=** DEFAULT\_NUMBER\_DISPARITIES**;**

int block\_size **=** DEFAULT\_BLOCK\_SIZE**;**

Ptr**<**StereoBM**>** bm\_realtime **=** StereoBM**::**create**(**number\_disparities**,** block\_size**);**

//key = 32 means pressing the spacebar

//stopping capturing by pressing spacebar

// while(key != 32){

cvtColor**(**left**,** left**,** CV\_BGR2GRAY**);**

cvtColor**(**right**,** right**,** CV\_BGR2GRAY**);**

resize**(**left**,** left**,** Size**(**imgSize\_width**,** imgSize\_height**));**

resize**(**right**,** right**,** Size**(**imgSize\_width**,** imgSize\_height**));**

left**.**copyTo**(**leftImage**);**

right**.**copyTo**(**rightImage**);**

//remap(left, leftImage, left\_map1, left\_map2, INTER\_LINEAR);

//remap(right, rightImage, right\_map1, right\_map2, INTER\_LINEAR);

bm\_realtime**->**compute**(** leftImage**,** rightImage**,** imgDisparity16S **);**

double minVal**;** double maxVal**;**

minMaxLoc**(** imgDisparity16S**,** **&**minVal**,** **&**maxVal **);**

imgDisparity16S**.**convertTo**(** imgDisparity8U**,** CV\_8UC1**,** 255**/(**maxVal **-** minVal**));**

**return** imgDisparity8U**;**

//imshow("BM: real-time disparity map",imgDisparity8U);

waitKey**(**10**);**

//}

**}**

void realtimeDisparityMap**::**sgbm**()**

**{**

//1: the usb on the left hand side of computer

//0: the usb on the right hand side of computer

//the value may be different on different computer

VideoCapture left**(**1**),** right**(**0**);**

**if(** **!(**left**.**isOpened**()&&**right**.**isOpened**())** **)**

**{**

cout**<<** "Camera is not opened properly."**<<** endl**;**

**return;**

**}**

readingData**();**

Mat left\_map1**,** left\_map2**;**

Mat right\_map1**,** right\_map2**;**

initUndistortRectifyMap**(**leftCameraMatrix**,** leftDistCoeffs**,** R\_left**,** P\_left**,** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_16SC2**,** left\_map1**,** left\_map2**);**

initUndistortRectifyMap**(**rightCameraMatrix**,** rightDistCoeffs**,** R\_right**,** P\_right**,** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_16SC2**,** right\_map1**,** right\_map2**);**

int key **=** 0**;**

Mat leftRawImg**,**rightRawImg**;**

Mat leftImage**,** rightImage**;**

Mat leftTemp**,** rightTemp**;**

// create the image in which we will save our disparities

Mat imgDisparity16S **=** Mat**(** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_16S **);**

imgDisparity8U **=** Mat**(** Size**(**imgSize\_width**,** imgSize\_height**),** CV\_8UC1 **);**

int minimum\_disparity **=** DEFAULT\_MINIMUM\_DISPARITY**;**

int number\_disparities **=** DEFAULT\_NUMBER\_DISPARITIES**;**

int block\_size **=** DEFAULT\_BLOCK\_SIZE**;**

Ptr**<**StereoSGBM**>** sgbm\_realtime **=** StereoSGBM**::**create**(**minimum\_disparity**,** number\_disparities**,** block\_size**,** P1**,** P2**);**

//key = 32 means pressing the spacebar

//stopping capturing by pressing spacebar

**while(**key **!=** 32**){**

left **>>** leftRawImg**;**

right **>>** rightRawImg**;**

resize**(**leftRawImg**,** leftTemp**,** Size**(**displaySize\_width**,** displaySize\_height**));**

resize**(**rightRawImg**,** rightTemp**,** Size**(**displaySize\_width**,** displaySize\_height**));**

imshow**(**"left"**,**leftTemp**);**

imshow**(**"right"**,** rightTemp**);**

cvtColor**(**leftRawImg**,** leftRawImg**,** CV\_BGR2GRAY**);**

cvtColor**(**rightRawImg**,** rightRawImg**,** CV\_BGR2GRAY**);**

resize**(**leftRawImg**,** leftRawImg**,** Size**(**imgSize\_width**,** imgSize\_height**));**

resize**(**rightRawImg**,** rightRawImg**,** Size**(**imgSize\_width**,** imgSize\_height**));**

remap**(**leftRawImg**,** leftImage**,** left\_map1**,** left\_map2**,** INTER\_LINEAR**);**

remap**(**rightRawImg**,** rightImage**,** right\_map1**,** right\_map2**,** INTER\_LINEAR**);**

sgbm\_realtime**->**compute**(** leftImage**,** rightImage**,** imgDisparity16S **);**

double minVal**;** double maxVal**;**

minMaxLoc**(** imgDisparity16S**,** **&**minVal**,** **&**maxVal **);**

imgDisparity16S**.**convertTo**(** imgDisparity8U**,** CV\_8UC1**,** 255**/(**maxVal **-** minVal**));**

imshow**(**"SGBM: real-time disparity map"**,**imgDisparity8U**);**

key **=** waitKey**(**10**);**

**}**

**}**

////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////

/////

/////

/////

////////////////////////////////////////////////////////////////////////////////////////////////////////////

void realtimeDisparityMap**::**bm\_safest**()**

**{**

//1: the usb on the left hand side of computer

//0: the usb on the right hand side of computer

//the value may be different on different computer

VideoCapture left**(**1**),** right**(**0**);**

**if(** **!(**left**.**isOpened**()&&**right**.**isOpened**())** **)**

**{**

cout**<<** "Camera is not opened properly."**<<** endl**;**

**return;**

**}**

readingData**();**

Mat left\_map1**,** left\_map2**;**

Mat right\_map1**,** right\_map2**;**

initUndistortRectifyMap**(**leftCameraMatrix**,** leftDistCoeffs**,** R\_left**,** P\_left**,** Size**(**960**,** 540**),** CV\_16SC2**,** left\_map1**,** left\_map2**);**

initUndistortRectifyMap**(**rightCameraMatrix**,** rightDistCoeffs**,** R\_right**,** P\_right**,** Size**(**960**,** 540**),** CV\_16SC2**,** right\_map1**,** right\_map2**);**

int key **=** 0**;**

Mat leftRawImg**,**rightRawImg**;**

Mat leftImage**,** rightImage**;**

//-- And create the image in which we will save our disparities

Mat imgDisparity16S **=** Mat**(** Size**(**960**,** 540**),** CV\_16S **);**

imgDisparity8U **=** Mat**(** Size**(**960**,** 540**),** CV\_8UC1 **);**

int number\_disparities **=** 10**\***16**;**

int block\_size **=** 21**;**

Ptr**<**StereoBM**>** bm\_realtime **=** StereoBM**::**create**(**number\_disparities**,** block\_size**);**

//key = 32 means pressing the spacebar

//stopping capturing by pressing spacebar

**while(**key **!=** 32**){**

left **>>** leftRawImg**;**

right **>>** rightRawImg**;**

cvtColor**(**leftRawImg**,** leftRawImg**,** CV\_BGR2GRAY**);**

cvtColor**(**rightRawImg**,** rightRawImg**,** CV\_BGR2GRAY**);**

resize**(**leftRawImg**,** leftRawImg**,** Size**(**960**,** 540**));**

resize**(**rightRawImg**,** rightRawImg**,** Size**(**960**,** 540**));**

remap**(**leftRawImg**,** leftImage**,** left\_map1**,** left\_map2**,** INTER\_LINEAR**);**

remap**(**rightRawImg**,** rightImage**,** right\_map1**,** right\_map2**,** INTER\_LINEAR**);**

bm\_realtime**->**compute**(** leftImage**,** rightImage**,** imgDisparity16S **);**

double minVal**;** double maxVal**;**

minMaxLoc**(** imgDisparity16S**,** **&**minVal**,** **&**maxVal **);**

imgDisparity16S**.**convertTo**(** imgDisparity8U**,** CV\_8UC1**,** 255**/(**maxVal **-** minVal**));**

imshow**(**"BM: real-time disparity map"**,**imgDisparity8U**);**

key **=** waitKey**(**10**);**

**}**

**}**

void realtimeDisparityMap**::**sgbm\_safest**()**

**{**

//1: the usb on the left hand side of computer

//0: the usb on the right hand side of computer

//the value may be different on different computer

VideoCapture left**(**1**),** right**(**0**);**

**if(** **!(**left**.**isOpened**()&&**right**.**isOpened**())** **)**

**{**

cout**<<** "Camera is not opened properly."**<<** endl**;**

**return;**

**}**

readingData**();**

Mat left\_map1**,** left\_map2**;**

Mat right\_map1**,** right\_map2**;**

initUndistortRectifyMap**(**leftCameraMatrix**,** leftDistCoeffs**,** R\_left**,** P\_left**,** Size**(**960**,** 540**),** CV\_16SC2**,** left\_map1**,** left\_map2**);**

initUndistortRectifyMap**(**rightCameraMatrix**,** rightDistCoeffs**,** R\_right**,** P\_right**,** Size**(**960**,** 540**),** CV\_16SC2**,** right\_map1**,** right\_map2**);**

int key **=** 0**;**

Mat leftRawImg**,**rightRawImg**;**

Mat leftImage**,** rightImage**;**

//-- And create the image in which we will save our disparities

Mat imgDisparity16S **=** Mat**(** Size**(**960**,** 540**),** CV\_16S **);**

imgDisparity8U **=** Mat**(** Size**(**960**,** 540**),** CV\_8UC1 **);**

int minimum\_disparity **=** 0**;**

int number\_disparities **=** 10**\***16**;**

int block\_size **=** 21**;**

Ptr**<**StereoSGBM**>** sgbm\_realtime **=** StereoSGBM**::**create**(**minimum\_disparity**,** number\_disparities**,** block\_size**);**

//key = 32 means pressing the spacebar

//stopping capturing by pressing spacebar

**while(**key **!=** 32**){**

left **>>** leftRawImg**;**

right **>>** rightRawImg**;**

cvtColor**(**leftRawImg**,** leftRawImg**,** CV\_BGR2GRAY**);**

cvtColor**(**rightRawImg**,** rightRawImg**,** CV\_BGR2GRAY**);**

resize**(**leftRawImg**,** leftRawImg**,** Size**(**960**,** 540**));**

resize**(**rightRawImg**,** rightRawImg**,** Size**(**960**,** 540**));**

remap**(**leftRawImg**,** leftImage**,** left\_map1**,** left\_map2**,** INTER\_LINEAR**);**

remap**(**rightRawImg**,** rightImage**,** right\_map1**,** right\_map2**,** INTER\_LINEAR**);**

sgbm\_realtime**->**compute**(** leftImage**,** rightImage**,** imgDisparity16S **);**

double minVal**;** double maxVal**;**

minMaxLoc**(** imgDisparity16S**,** **&**minVal**,** **&**maxVal **);**

imgDisparity16S**.**convertTo**(** imgDisparity8U**,** CV\_8UC1**,** 255**/(**maxVal **-** minVal**));**

imshow**(**"SGBM: real-time disparity map"**,**imgDisparity8U**);**

key **=** waitKey**(**10**);**

**}**

**}**

//realtimeDisparityMap::safest\_quadrant()

//{

// for(int i=0; i<)

//}

realtimeDisparityMap**::**realtimeDisparityMap**()**

**{**

DEFAULT\_MINIMUM\_DISPARITY **=** 0**;**

DEFAULT\_NUMBER\_DISPARITIES **=** 5**\***16**;**

DEFAULT\_BLOCK\_SIZE **=** 21**;**

P1 **=** 0**;**

P2 **=** 0**;**

imgSize\_width **=** 960**;**

imgSize\_height **=** 540**;**

displaySize\_width **=** 960**;**

displaySize\_height **=** 540**;**

readingData**();**

**}**

**Appendix D: Single Camera Calibration**

#include "singleCameraCalibration.hpp"

#include "Camera2Mat.hpp"

#include "opencv2/core.hpp"

#include <opencv2/core/utility.hpp>

#include "opencv2/imgproc.hpp"

#include "opencv2/calib3d.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/videoio.hpp"

#include "opencv2/highgui.hpp"

#include <iostream>

#include <cctype>

#include <stdio.h>

#include <string.h>

#include <time.h>

**using** **namespace** cv**;**

**using** **namespace** std**;**

SingleCameraCalibration**::**SingleCameraCalibration**()**

**{**

**}**

double SingleCameraCalibration**::**computeReprojectionErrors**(**

const vector**<**vector**<**Point3f**>** **>&** objectPoints**,**

const vector**<**vector**<**Point2f**>** **>&** imagePoints**,**

const vector**<**Mat**>&** rvecs**,** const vector**<**Mat**>&** tvecs**,**

const Mat**&** cameraMatrix**,** const Mat**&** distCoeffs**,**

vector**<**float**>&** perViewErrors **)**

**{**

vector**<**Point2f**>** imagePoints2**;**

int i**,** totalPoints **=** 0**;**

double totalErr **=** 0**,** err**;**

perViewErrors**.**resize**(**objectPoints**.**size**());**

**for(** i **=** 0**;** i **<** **(**int**)**objectPoints**.**size**();** i**++** **)**

**{**

projectPoints**(**Mat**(**objectPoints**[**i**]),** rvecs**[**i**],** tvecs**[**i**],**

cameraMatrix**,** distCoeffs**,** imagePoints2**);**

err **=** norm**(**Mat**(**imagePoints**[**i**]),** Mat**(**imagePoints2**),** NORM\_L2**);**

int n **=** **(**int**)**objectPoints**[**i**].**size**();**

perViewErrors**[**i**]** **=** **(**float**)**std**::**sqrt**(**err**\***err**/**n**);**

totalErr **+=** err**\***err**;**

totalPoints **+=** n**;**

**}**

**return** std**::**sqrt**(**totalErr**/**totalPoints**);**

**}**

void SingleCameraCalibration**::**calcChessboardCorners**(**Size boardSize**,** float squareSize**,** vector**<**Point3f**>&** corners**)**

**{**

corners**.**resize**(**0**);**

**for(** int i **=** 0**;** i **<** boardSize**.**height**;** i**++** **)**

**for(** int j **=** 0**;** j **<** boardSize**.**width**;** j**++** **)**

corners**.**push\_back**(**Point3f**(**float**(**j**\***squareSize**),** float**(**i**\***squareSize**),** 0**));**

**}**

bool SingleCameraCalibration**::**runCalibration**(** vector**<**vector**<**Point2f**>** **>** imagePoints**,**

Size imageSize**,** Size boardSize**,**

float squareSize**,** float aspectRatio**,**

int flags**,** Mat**&** cameraMatrix**,** Mat**&** distCoeffs**,**

vector**<**Mat**>&** rvecs**,** vector**<**Mat**>&** tvecs**,**

vector**<**float**>&** reprojErrs**,**

double**&** totalAvgErr**)**

**{**

cameraMatrix **=** Mat**::**eye**(**3**,** 3**,** CV\_64F**);**

**if(** flags **&** CALIB\_FIX\_ASPECT\_RATIO **)**

cameraMatrix**.**at**<**double**>(**0**,**0**)** **=** aspectRatio**;**

distCoeffs **=** Mat**::**zeros**(**8**,** 1**,** CV\_64F**);**

vector**<**vector**<**Point3f**>** **>** objectPoints**(**1**);**

calcChessboardCorners**(**boardSize**,** squareSize**,** objectPoints**[**0**]);**

objectPoints**.**resize**(**imagePoints**.**size**(),**objectPoints**[**0**]);**

double rms **=** calibrateCamera**(**objectPoints**,** imagePoints**,** imageSize**,** cameraMatrix**,**

distCoeffs**,** rvecs**,** tvecs**,** flags**|**CALIB\_FIX\_K4**|**CALIB\_FIX\_K5**);**

printf**(**"RMS error reported by calibrateCamera: %g\n"**,** rms**);**

bool ok **=** checkRange**(**cameraMatrix**)** **&&** checkRange**(**distCoeffs**);**

totalAvgErr **=** computeReprojectionErrors**(**objectPoints**,** imagePoints**,**

rvecs**,** tvecs**,** cameraMatrix**,** distCoeffs**,** reprojErrs**);**

**return** ok**;**

**}**

void SingleCameraCalibration**::**saveCameraParams**(** const string**&** filename**,**

Size imageSize**,** Size boardSize**,**

float squareSize**,** float aspectRatio**,** int flags**,**

const Mat**&** cameraMatrix**,** const Mat**&** distCoeffs**,**

const vector**<**Mat**>&** rvecs**,** const vector**<**Mat**>&** tvecs**,**

const vector**<**float**>&** reprojErrs**,**

const vector**<**vector**<**Point2f**>** **>&** imagePoints**,**

double totalAvgErr **)**

**{**

FileStorage fs**(** filename**,** FileStorage**::**WRITE **);**

time\_t tt**;**

time**(** **&**tt **);**

struct tm **\***t2 **=** localtime**(** **&**tt **);**

char buf**[**1024**];**

strftime**(** buf**,** **sizeof(**buf**)-**1**,** "%c"**,** t2 **);**

fs **<<** "calibration\_time" **<<** buf**;**

**if(** **!**rvecs**.**empty**()** **||** **!**reprojErrs**.**empty**()** **)**

fs **<<** "nframes" **<<** **(**int**)**std**::**max**(**rvecs**.**size**(),** reprojErrs**.**size**());**

fs **<<** "image\_width" **<<** imageSize**.**width**;**

fs **<<** "image\_height" **<<** imageSize**.**height**;**

fs **<<** "board\_width" **<<** boardSize**.**width**;**

fs **<<** "board\_height" **<<** boardSize**.**height**;**

fs **<<** "square\_size" **<<** squareSize**;**

**if(** flags **&** CALIB\_FIX\_ASPECT\_RATIO **)**

fs **<<** "aspectRatio" **<<** aspectRatio**;**

**if(** flags **!=** 0 **)**

**{**

sprintf**(** buf**,** "flags: %s%s%s%s"**,**

flags **&** CALIB\_USE\_INTRINSIC\_GUESS **?** "+use\_intrinsic\_guess" **:** ""**,**

flags **&** CALIB\_FIX\_ASPECT\_RATIO **?** "+fix\_aspectRatio" **:** ""**,**

flags **&** CALIB\_FIX\_PRINCIPAL\_POINT **?** "+fix\_principal\_point" **:** ""**,**

flags **&** CALIB\_ZERO\_TANGENT\_DIST **?** "+zero\_tangent\_dist" **:** "" **);**

**}**

fs **<<** "flags" **<<** flags**;**

fs **<<** "camera\_matrix" **<<** cameraMatrix**;**

fs **<<** "distortion\_coefficients" **<<** distCoeffs**;**

fs **<<** "avg\_reprojection\_error" **<<** totalAvgErr**;**

**if(** **!**reprojErrs**.**empty**()** **)**

fs **<<** "per\_view\_reprojection\_errors" **<<** Mat**(**reprojErrs**);**

**if(** **!**rvecs**.**empty**()** **&&** **!**tvecs**.**empty**()** **)**

**{**

CV\_Assert**(**rvecs**[**0**].**type**()** **==** tvecs**[**0**].**type**());**

Mat bigmat**((**int**)**rvecs**.**size**(),** 6**,** rvecs**[**0**].**type**());**

**for(** int i **=** 0**;** i **<** **(**int**)**rvecs**.**size**();** i**++** **)**

**{**

Mat r **=** bigmat**(**Range**(**i**,** i**+**1**),** Range**(**0**,**3**));**

Mat t **=** bigmat**(**Range**(**i**,** i**+**1**),** Range**(**3**,**6**));**

CV\_Assert**(**rvecs**[**i**].**rows **==** 3 **&&** rvecs**[**i**].**cols **==** 1**);**

CV\_Assert**(**tvecs**[**i**].**rows **==** 3 **&&** tvecs**[**i**].**cols **==** 1**);**

r **=** rvecs**[**i**].**t**();**

t **=** tvecs**[**i**].**t**();**

**}**

fs **<<** "extrinsic\_parameters" **<<** bigmat**;**

**}**

**if(** **!**imagePoints**.**empty**()** **)**

**{**

Mat imagePtMat**((**int**)**imagePoints**.**size**(),** **(**int**)**imagePoints**[**0**].**size**(),** CV\_32FC2**);**

**for(** int i **=** 0**;** i **<** **(**int**)**imagePoints**.**size**();** i**++** **)**

**{**

Mat r **=** imagePtMat**.**row**(**i**).**reshape**(**2**,** imagePtMat**.**cols**);**

Mat imgpti**(**imagePoints**[**i**]);**

imgpti**.**copyTo**(**r**);**

**}**

fs **<<** "image\_points" **<<** imagePtMat**;**

**}**

**}**

bool SingleCameraCalibration**::**readStringList**(** const string**&** filename**,** vector**<**string**>&** l **)**

**{**

l**.**resize**(**0**);**

FileStorage fs**(**filename**,** FileStorage**::**READ**);**

**if(** **!**fs**.**isOpened**()** **)**

**return** **false;**

FileNode n **=** fs**.**getFirstTopLevelNode**();**

**if(** n**.**type**()** **!=** FileNode**::**SEQ **)**

**return** **false;**

FileNodeIterator it **=** n**.**begin**(),** it\_end **=** n**.**end**();**

**for(** **;** it **!=** it\_end**;** **++**it **)**

l**.**push\_back**((**string**)\***it**);**

**return** **true;**

**}**

bool SingleCameraCalibration**::**runAndSave**(**const string**&** outputFilename**,**

const vector**<**vector**<**Point2f**>** **>&** imagePoints**,**

Size imageSize**,** Size boardSize**,** float squareSize**,**

float aspectRatio**,** int flags**,** Mat**&** cameraMatrix**,**

Mat**&** distCoeffs**,** bool writeExtrinsics**,** bool writePoints **)**

**{**

vector**<**Mat**>** rvecs**,** tvecs**;**

vector**<**float**>** reprojErrs**;**

double totalAvgErr **=** 0**;**

bool ok **=** runCalibration**(**imagePoints**,** imageSize**,** boardSize**,** squareSize**,**

aspectRatio**,** flags**,** cameraMatrix**,** distCoeffs**,**

rvecs**,** tvecs**,** reprojErrs**,** totalAvgErr**);**

printf**(**"%s. avg reprojection error = %.2f\n"**,**

ok **?** "Calibration succeeded" **:** "Calibration failed"**,**

totalAvgErr**);**

**if(** ok **)**

saveCameraParams**(** outputFilename**,** imageSize**,**

boardSize**,** squareSize**,** aspectRatio**,**

flags**,** cameraMatrix**,** distCoeffs**,**

writeExtrinsics **?** rvecs **:** vector**<**Mat**>(),**

writeExtrinsics **?** tvecs **:** vector**<**Mat**>(),**

writeExtrinsics **?** reprojErrs **:** vector**<**float**>(),**

writePoints **?** imagePoints **:** vector**<**vector**<**Point2f**>** **>(),**

totalAvgErr **);**

**return** ok**;**

**}**

int SingleCameraCalibration**::**singleMain**(** int argc**,** char**\*\*** argv **)**

**{**

Size boardSize**,** imageSize**;**

boardSize**.**width **=** 9**;**//////////////////

boardSize**.**height **=** 6**;**//////////////////

float squareSize**,** aspectRatio**;**

Mat cameraMatrix**,** distCoeffs**;**

string outputFilename**;**

string inputFilename **=** ""**;**

int i**,** nframes**;**

bool writeExtrinsics**,** writePoints**;**

bool undistortImage **=** **false;**

int flags **=** 0**;**

VideoCapture capture**;**

bool flipVertical**;**

bool showUndistorted**;**

bool videofile**;**

int delay**;**

clock\_t prevTimestamp **=** 0**;**

int mode **=** DETECTION**;**

int cameraId **=** 0**;**

vector**<**vector**<**Point2f**>** **>** imagePoints**;**

vector**<**string**>** imageList**;**

Pattern pattern **=** CHESSBOARD**;**

cv**::**CommandLineParser parser**(**argc**,** argv**,**

"{help ||}{w||}{h||}{pt|chessboard|}{n|10|}{d|1000|}{s|1|}{o|out\_camera\_data.yml|}"

"{op||}{oe||}{zt||}{a|1|}{p||}{v||}{V||}{su||}"

"{@input\_data|0|}"**);**

pattern **=** CHESSBOARD**;**

squareSize **=** parser**.**get**<**float**>(**"s"**);**

// cout<< "squareSize: "<< squareSize<< endl;

nframes **=** parser**.**get**<**int**>(**"n"**);**

// cout<< "nframes: "<< nframes<< endl;

aspectRatio **=** parser**.**get**<**float**>(**"a"**);**

// cout<< "aspectRatio: "<< aspectRatio<< endl;

delay **=** parser**.**get**<**int**>(**"d"**);**

// cout<< "delay: "<< delay<< endl;

writePoints **=** parser**.**has**(**"op"**);**

// cout<< "writePoints: "<< writePoints<< endl;

writeExtrinsics **=** parser**.**has**(**"oe"**);**

// cout<< "writeExtrinsics: "<< writeExtrinsics<< endl;

**if** **(**parser**.**has**(**"a"**))**

flags **|=** CALIB\_FIX\_ASPECT\_RATIO**;**

**if** **(** parser**.**has**(**"zt"**)** **)**

flags **|=** CALIB\_ZERO\_TANGENT\_DIST**;**

**if** **(** parser**.**has**(**"p"**)** **)**

flags **|=** CALIB\_FIX\_PRINCIPAL\_POINT**;**

flipVertical **=** parser**.**has**(**"v"**);**

videofile **=** parser**.**has**(**"V"**);**

**if** **(** parser**.**has**(**"o"**)** **)**

outputFilename **=** parser**.**get**<**string**>(**"o"**);**

showUndistorted **=** parser**.**has**(**"su"**);**

**if** **(** isdigit**(**parser**.**get**<**string**>(**"@input\_data"**)[**0**])** **)**

cameraId **=** parser**.**get**<**int**>(**"@input\_data"**);**

**else**

inputFilename **=** parser**.**get**<**string**>(**"@input\_data"**);**

**if** **(** squareSize **<=** 0 **)**

**return** fprintf**(** stderr**,** "Invalid board square width\n" **),** **-**1**;**

**if** **(** nframes **<=** 3 **)**

**return** printf**(**"Invalid number of images\n" **),** **-**1**;**

**if** **(** aspectRatio **<=** 0 **)**

**return** printf**(** "Invalid aspect ratio\n" **),** **-**1**;**

**if** **(** delay **<=** 0 **)**

**return** printf**(** "Invalid delay\n" **),** **-**1**;**

**if** **(** boardSize**.**width **<=** 0 **)**

**return** fprintf**(** stderr**,** "Invalid board width\n" **),** **-**1**;**

**if** **(** boardSize**.**height **<=** 0 **)**

**return** fprintf**(** stderr**,** "Invalid board height\n" **),** **-**1**;**

mode **=** CAPTURING**;**

readStringList**(**inputFilename**,** imageList**);**

nframes **=** **(**int**)**imageList**.**size**();**//nframes = 15

namedWindow**(** "Image View"**,** 1 **);**

**for(**i **=** 0**;;**i**++)**

**{**

Mat view **=** Mat**::**zeros**(**0**,**0**,**0**),** viewGray **=** Mat**::**zeros**(**0**,**0**,**0**);**

bool blink **=** **false;**

**if(** i **<** **(**int**)**imageList**.**size**()** **)**

**{**

view **=** imread**(**imageList**[**i**],** 1**);**

**if(**view**.**data **==** **NULL)**

cout **<<** "View is empty\n"**;**

**}**

imageSize **=** view**.**size**();**

vector**<**Point2f**>** pointbuf**;**

cvtColor**(**view**,** viewGray**,** COLOR\_BGR2GRAY**);**

bool found**;**

found **=** findChessboardCorners**(** view**,** boardSize**,** pointbuf**,**

CALIB\_CB\_ADAPTIVE\_THRESH **|** CALIB\_CB\_FAST\_CHECK **|** CALIB\_CB\_NORMALIZE\_IMAGE**);**

// improve the found corners' coordinate accuracy

// cornerSubPix( viewGray, pointbuf, Size(11,11), Size(-1,-1), TermCriteria( TermCriteria::EPS+TermCriteria::COUNT, 30, 0.1 ));

imagePoints**.**push\_back**(**pointbuf**);**

prevTimestamp **=** clock**();**

blink **=** capture**.**isOpened**();**

**if(**found**)**

drawChessboardCorners**(** view**,** boardSize**,** Mat**(**pointbuf**),** found **);**

imshow**(**"Image View"**,** view**);**

int key **=** 0xff **&** waitKey**(**capture**.**isOpened**()** **?** 50 **:** 500**);**

**if(** **(**key **&** 255**)** **==** 27 **)**

**break;**

**if(** imagePoints**.**size**()** **>=** **(**unsigned**)**nframes **)**

**{**

runAndSave**(**outputFilename**,** imagePoints**,** imageSize**,**

boardSize**,** squareSize**,** aspectRatio**,**

flags**,** cameraMatrix**,** distCoeffs**,**

writeExtrinsics**,** writePoints**);**

**break;**

**}**

**}**

**return** 0**;**

**}**

void SingleCameraCalibration**::**openCamera\_left**(**Camera2Mat**\*** cam**)**

**{**

int displaySize\_width **=** 640**;**

int displaySize\_height **=** 512**;**

Mat frame **=** Mat**::**zeros**(**0**,**0**,**0**);**

//VideoCapture camera(1);

//if( !camera.isOpened() )

//{

// cout<< "camera is not opened properly."<< endl;

// return;

//}

int key **=** 0**;**

//Mat frame;

int numberOfImage **=** 1**;**

int maxNumberOfImages **=** 20**;**

//key = 32 means pressing the spacebar

//stopping capturing by pressing spacebar

**while(**key **!=** 32**)**

**{**

// camera >> frame;

frame **=** cam**->**imageFromCameraOne**();**

resize**(**frame**,** frame**,** Size**(**displaySize\_width**,** displaySize\_height**));**

//find the corner

**if(**findCorner\_left**(**frame**,** numberOfImage**))**

**{**

numberOfImage**++;**

**}**

//reach the max number of images, break the loop

**if(**numberOfImage**>**maxNumberOfImages**)**

**{**

**break;**

**}**

//for the purpose of displaying on the screen

//resize the frame size

imshow**(**"Camera"**,**frame**);**

key **=** waitKey**(**10**);**

**}**

**}**

bool SingleCameraCalibration**::**findCorner\_left**(**Mat image**,** int numberOfImage**)**

**{**

int board\_w **=** 9**;**

int board\_h **=** 6**;**

int numBoards **=** 20**;** //25

int displaySize\_width **=** 640**;**

int displaySize\_height **=** 512**;**

// int number = 1;

vector**<**Point2f**>** corners**;**

Size board\_sz **=** Size**(**board\_w**,** board\_h**);**

bool found**;**

found **=** findChessboardCorners**(**image**,** board\_sz**,** corners**,** CV\_CALIB\_CB\_ADAPTIVE\_THRESH **|** CV\_CALIB\_CB\_FILTER\_QUADS**);**

**if(**found**)**

**{**

// when chessboard corners are found on both images, store the iamges first

**if(**imwrite**(**"left\_single"**+**std**::**to\_string**(**numberOfImage**)+**".jpg"**,** image**))**

cout**<<** "writing left\_single image successfully"**<<** endl**;**

**else**

cout**<<** "writing left\_single image false"**<<** endl**;**

// display the images on the screen

drawChessboardCorners**(**image**,** board\_sz**,** corners**,** found**);**

resize**(**image**,** image**,** Size**(**displaySize\_width**,** displaySize\_height**));**

imshow**(**"Camera"**,** image**);**

cout**<<** std**::**to\_string**(**numberOfImage**)+**" is found"**<<** endl**;**

waitKey**(**0**);**

**}**

**else**

**{**

cout**<<** "not found"**<<** endl**;**

**}**

**return** found**;**

**}**

void SingleCameraCalibration**::**imageListCreator\_left**(**string outputName**,** int maxNumberOfimages**)**

**{**

Mat m **=** imread**(**outputName**);**

FileStorage fs**(**outputName**,** FileStorage**::**WRITE**);**

fs **<<** "images" **<<** "["**;**

**for(**int i **=** 1**;** i **<=** maxNumberOfimages**;** i**++){**

fs **<<** string**(**"left\_single"**+**std**::**to\_string**(**i**)+**".jpg"**);**

**}**

fs **<<** "]"**;**

**}**

void SingleCameraCalibration**::**wholePackage**()**

**{**

Camera2Mat**\*** cameras **=** **new** Camera2Mat**();**

openCamera\_left**(**cameras**);**

int maxNumberOfImages **=** 20**;**

imageListCreator\_left**(**"left.xml"**,** maxNumberOfImages**);**

int num **=** 7**;**

char **\***a **=** "./single"**;**

char **\***b **=** "-w=9"**;**

char **\***c **=** "-h=6"**;**

char **\***d **=** "-o=left.yml"**;**

char **\***e **=** "-op"**;**

char **\***f **=** "-oe"**;**

char **\***g **=** "left.xml"**;**

char **\***argument**[**7**]** **=** **{**a**,** b**,** c**,** d**,** e**,** f**,** g**};**

singleMain**(**num**,** argument**);**

destroyAllWindows**();**

openCamera\_right**(**cameras**);**

imageListCreator\_right**(**"right.xml"**,** maxNumberOfImages**);**

char **\***aa **=** "./single"**;**

char **\***bb **=** "-w=9"**;**

char **\***cc **=** "-h=6"**;**

char **\***dd **=** "-o=right.yml"**;**

char **\***ee **=** "-op"**;**

char **\***ff **=** "-oe"**;**

char **\***gg **=** "right.xml"**;**

char **\***argument2**[**7**]** **=** **{**aa**,** bb**,** cc**,** dd**,** ee**,** ff**,** gg**};**

singleMain**(**num**,** argument2**);**

**delete** cameras**;**

**}**

void SingleCameraCalibration**::**openCamera\_right**(**Camera2Mat**\*** cam**)**

**{**

int displaySize\_width **=** 640**;**

int displaySize\_height **=** 512**;**

// VideoCapture camera(0);

// if( !camera.isOpened() )

// {

// cout<< "camera is not opened properly."<< endl;

// return;

// }

Mat frame **=** Mat**::**zeros**(**0**,**0**,**0**);**

int key **=** 0**;**

// Mat frame;

int numberOfImage **=** 1**;**

int maxNumberOfImages **=** 20**;**

//key = 32 means pressing the spacebar

//stopping capturing by pressing spacebar

**while(**key **!=** 32**)**

**{**

frame **=** cam**->**imageFromCameraTwo**();**

//camera >> frame;

resize**(**frame**,** frame**,** Size**(**displaySize\_width**,** displaySize\_height**));**

//find the corner

**if(**findCorner\_right**(**frame**,** numberOfImage**))**

**{**

numberOfImage**++;**

**}**

//reach the max number of images, break the loop

**if(**numberOfImage**>**maxNumberOfImages**)**

**{**

**break;**

**}**

//for the purpose of displaying on the screen

//resize the frame size

imshow**(**"Camera"**,**frame**);**

key **=** waitKey**(**10**);**

**}**

**}**

bool SingleCameraCalibration**::**findCorner\_right**(**Mat image**,** int numberOfImage**)**

**{**

int board\_w **=** 9**;**

int board\_h **=** 6**;**

int numBoards **=** 20**;** //25

int displaySize\_width **=** 640**;**

int displaySize\_height **=** 512**;**

// int number = 1;

vector**<**Point2f**>** corners**;**

Size board\_sz **=** Size**(**board\_w**,** board\_h**);**

bool found**;**

found **=** findChessboardCorners**(**image**,** board\_sz**,** corners**,** CV\_CALIB\_CB\_ADAPTIVE\_THRESH **|** CV\_CALIB\_CB\_FILTER\_QUADS**);**

**if(**found**)**

**{**

// when chessboard corners are found on both images, store the iamges first

**if(**imwrite**(**"right\_single"**+**std**::**to\_string**(**numberOfImage**)+**".jpg"**,** image**))**

cout**<<** "writing right\_single image successfully"**<<** endl**;**

**else**

cout**<<** "writing right\_single image false"**<<** endl**;**

// display the images on the screen

drawChessboardCorners**(**image**,** board\_sz**,** corners**,** found**);**

resize**(**image**,** image**,** Size**(**displaySize\_width**,** displaySize\_height**));**

imshow**(**"Camera"**,** image**);**

cout**<<** std**::**to\_string**(**numberOfImage**)+**" is found"**<<** endl**;**

waitKey**(**0**);**

**}**

**else**

**{**

cout**<<** "not found"**<<** endl**;**

**}**

**return** found**;**

**}**

void SingleCameraCalibration**::**imageListCreator\_right**(**string outputName**,** int maxNumberOfimages**)**

**{**

Mat m **=** imread**(**outputName**);**

FileStorage fs**(**outputName**,** FileStorage**::**WRITE**);**

fs **<<** "images" **<<** "["**;**

**for(**int i **=** 1**;** i **<=** maxNumberOfimages**;** i**++){**

fs **<<** string**(**"right\_single"**+**std**::**to\_string**(**i**)+**".jpg"**);**

**}**

fs **<<** "]"**;**

**}**

**Appendix E: Stereo Calibration**

#ifndef stereoCalibration\_hpp

#define stereoCalibration\_hpp

#include <stdio.h>

#include <string>

#include <vector>

#include "opencv2/imgcodecs.hpp"

**using** **namespace** std**;**

**using** **namespace** cv**;**

#endif /\* stereoCalibration\_hpp \*/

// capture images

// produce extrinsics.yml

class stereoCalibration

**{**

public**:**

void captureImage**();**

void startCalibrate**(**string**);**

void StereoCalib**(**const vector**<**string**>&** imagelist**,** Size boardSize**,** bool useCalibrated**=true,** bool showRectified**=true);**

bool readStringList**(** const string**&** filename**,** vector**<**string**>&** l **);**

stereoCalibration**();**

private**:**

int board\_w**;**

int board\_h**;**//5

int numBoards**;** //25

int imgSize\_width**;** //960\*540 //480\*270 //1280\*720 //1920\*1080

int imgSize\_height**;**

int displaySize\_width**;**

int displaySize\_height**;**

**};**