Frédéric Uhrweiler Stéphane Vujasinovic

Author: Uhrweiller Frédéric and Vujasinovic Stéphane Matrikelnr.58566 EIT Matrikelnr .: 59092 Mechatronics

E-mail: uhfr1011@hs-karlsruhe.de E-mail: vust1011@hs-karlsruhe.de

Assistant Prof. Dr.-Ing. Ferdinand Olawsky

Project Coordinator MMT: Prof. Dr. Peter Weber

Project code: 17SS_OL_Remember

Deadline: 30.09.2017

Table of Contents

Abbildungsverzeichnis

Code

1. Introductionunderstand

Tothe stereo vision, you first need to find the In a simple way, explain a basic camera as it is basically constructed and what parameters need to be regulated.

A. What is stereo vision?

Stereoscopy is a procedure that takes up two images of the same scene and then creates a disparity map of the scene. From this disparity card it is possible to measure the distance to an object and create a 3D map from the scene.

For example, Stereo Vision inRobot The

the Clean Roomgoal of the great project is to develop a clean room robot for particle measurement throughout the clean room. In order to perform the measurement smoothly, the robot must be oriented without collisions. For this orientation, we decided to use only cameras and measure the distance to an object at least two cameras (Stereo Vision). The two cameras have a distance of 110 mm between each other.

Figure 1: Scheme to Stereo Camera

The larger the distance the better it is possible to evaluate the distance to the scene for objects that are far away.

A Python program that uses the OpenCV library was implemented in this project to calibrate the cameras and measure the distance to the objects of the scene. (See Appendix)

Figure 2: OpenCV and Python Logo

The stereo camera consists of two Logitech Webcam C170.

Figure 3: Photo from the Logitech Webcam C170 (Source: www.logitech.fr)

Video calls are optimal with images of 640x480 pixels. Focal length: 2.3mm. The self-built stereo camera:-built stereo camera

Figure 4: Photo of our self

2. Camera model

cameras capture the light rays of our environment. In principle, a camera like our eye works, the reflected light beams of our environment come in our eyes and you are collected on our retina. The "Pinhole Camera" is the simplest model. It's a good simplified model to understand how a camera works. In the model, all light beams are stopped by one surface. Only the beams that penetrate through the hole are configured reversely on a surface in the camera. The bottom picture illustrates this principle.

Figure 5: Functioning of a Camera

Source: https://funsizephysics.com/use-light-turn-world-upside/

This principle is very simple, but it is not a good way to get enough light at a fast Exposure. Therefore, lenses are used to collect more light beams at one point. The problem is that this lens brings distortion. One can distinguish two different types of distortion:

• radial distortion

• tangential distortion The radial distortion comes from the lens itself and the tangential distortion comes from the camera's geometry. The images can then be corrected using mathematical procedures. The calibration process allows a model of the camera's geometry and a model of the lens distortion. These models form the intrinsic parameters of a camera.

A. Focal length

The relative size of the image projected on the surface of the camera depends on the focal length. The pinhole model is the focal length, the distance between the hole and the area where the image is projected.

The Thales Theorem then gives: -x = f * (X / Z) With:

• x: image of the object (minus sign is reversed by the image)

• X: object size

• Z: distance from hole to object

• f : Focal length, distance from hole to image

Figure 6: Configured object

Source: Learning OpenCV 3 - O'Reilly

Since the lens is not perfectly centered, two parameters are introduced, Cx and Cy for each horizontal and vertical displacement the lens. The focal length on the X and Y axis is also different because the image area is rectangular. This gives you the following formula for positioning the object on the surface.

The projected points of the real world on the image area can thus be modeled as follows. Here M is the Intrinsic Matrix.

B. Distortion of the lens

Theoretically, it is possible to build a lens that does not cause distortion with a parabolic lens. In practice, however, it is much easier to produce a spherical lens than a parabolic lens. As previously discussed, there are two types of distortion. The radial distortion caused by the lens shape and the tangential distortion caused by the camera's assembly process.

There is no radial distortion on the optical center and it is getting bigger as you approach the edges. In practice, this distortion is small, it is sufficient to make Taylor development up to the third term. It follows the following formula.

Figure 7: Radial Distortion (Source: Wikipedia)

x and y are the coordinates of the original point on the image area and thus the position of the corrected point is calculated.

There is also a tangential distortion because the lens is not perfectly parallel to the image surface. To correct this, two additional parameters are introduced, p1 and p2.

C. OpenCV Calibration The OpenCV

library allows us to calculate the intrinsic parameters using specific functions, which is called calibration. This is possible with various checks of a chessboard.

Figure 9: Photo taken during calibration

The program for recording images for later calibration is called "Take_images_for_calibration.py". When the corners of the chessboard are detected on both cameras, two windows with the recognized image will open for each Camera. The images are either saved or deleted by the user. Good pictures can be seen in which the corners are clearly recognizable. These images are used later for the calibration in the main program "Main_Stereo_Vision_Prog.py". OpenCV recommends at least 10 pictures for each camera to get a good calibration. We've gotten good results with 50 pictures for each camera. To calibrate the cameras, the Python Code searches the corners of the chess board on each image for each camera using the OpenCV function: cv2.findChesssboardCorners

The position of the corners for each image is then stored in an image vector and the 3d scene object points are another vector. These imgpoints and objpoints in the cv2.calibeCamera () function then use the camera matrix, the distortion coefficients, the rotation and translation vectors at the output.

Figure 8. Tangential Distortion (Source: Learning OpenCV 3 - O'Reilly)

The cv2.getOptimalNewCameraMatrix () function enables us to get precise camera matrices that we later use in the cv2.stereoRectify () function.

After the calibration with OpenCV we get the following matrix M for our camera:

Matrix M without rectification (right camera):

885,439 0 301,366 0 885,849 233,812 0 0

matrix M

directly

Rectified (right camera):

303,497 0 869,592 233,909 0 0

871,463

01

matrix M without rectification (left camera):

748,533 0 345,068 0 749,062 228,481 0 0

M matrix

directly

Rectified (left camera):

730,520 0 349,507 0 725,714 227,805 0 0

Stereoabbildung

A. Declaration

the stereo vision allows the depth in to recognize an image, to make measurements in the image and to make 3D localizations. For this purpose, points that match the two cameras must be found. From this, you can then divert the distance between the camera and the point. The system's geometry helps simplify the calculation.

The stereo imaging takes place these four steps:

1. Removal of radial and tangential distortion by mathematical

calculations. You get unmased images. 2. Rectify the angle and distance of the images. This step allows both images to be coplanar on the Y-axis, thus facilitating the search of correspondence, and you only need to search on a single axis (namely, on the X-axis). 3. Find the same feature in the right and left image. This gives you a disparity map representing the differences between the images on the X-axis. 4. The last step is triangulation. The disparitytransformed into

card isdistances by triangulation.

Step 1: Removing the Distortion Step 2: Rectifying Step 3: Finding the same feature in both pictures Step 4: Triangulation

B. Triangulation

In the last step, the triangulation assumes that both projection images are coplanar and that the horizontal pixel row of the left image with the corresponding horizontal pixel row of the left image. With the previous hypotheses, you can now build the following image.

The point P lies in the environment and is mapped to the left and right images on pl and pr, with the corresponding coordinates xl and xr. This allows us to introduce a new size, the disparity: d = xl - xr. It is evident that the further point P is removed, the smaller the size d. The disparity is, therefore, inversely proportional to the distance.

The following formula can calculate the distance: Z = f * T / (xl-xr) It can be seen

that there is a nonlinear relationship between disparity and distance. If the disparity is near 0, small disparity differences lead to a large range of differences. This is the opposite when the disparity is large. Small disparity differences do not lead to a large range of differences. From this you can conclude that the stereo vision has a high depth resolution, just for objects close to the camera.

This method only works if the stereo camera configuration is ideal. In reality, however, this is not the case. That is why the left and right images are mathematically parallel. Of course, the cameras need to be positioned physically in parallel at least as closely as possible.

Figure 10: Triangulation (Source: Learning OpenCV 3 - O'Reilly)

Before we explain the method for the mathematical orientation of the images, we first need to understand the epipolar geometry.

C. Epipolar Geometry

The above image shows us the model of a non-perfect stereo camera consisting of two Pinhole camera models. By crossing the line of the projecting centers (Ol, Or) with the leveling projections, the epipolar points are created el and he. The lines (pl, el) and (pr, er) are called epipolarlines. The image of all possible points of a point at a projecting level is the epipolar line lying on the other image plane and passing through the epipolar point and the desired point. This allows you to limit the search of the point to a single dimension rather than on a whole level. You can therefore summarize the following points:

• Each 3D point in the view of a camera is contained in the epipolar plan

• A feature in one plane must be on the corresponding epipolar lines of the other plane (epipol condition)

• A two-dimensional search of a corresponding feature becomes a One-dimensional search is converted when you know the epipoloadometry.

• The sequence of the points is kept, ie, two points A and B are found in the same order on the epipolar lines of a plane as in the other plane.

D. Essential and Fundamental Matrices

To understand how the epipolar lines are calculated, one must first explain the essential and fundamental matrices (according to Matrize E and F). The Essential Matrix E contains the information on how the two cameras are physically arranged. It describes the location of the second camera relative to the first camera by translation and rotation parameters. These parameters are not directly readable in the matrix, as they are used for configuration. In the section Stereo Calibration we explain how to calculate R and T (rotation matrix and translation vector). Matrix F contains the information of the essential matrix E, the physical arrangement of the cameras and the information about the intrinsic parameters of the cameras.

The relation between the configured point on the left picture pl and the right image pr is defined as:

TEp p

r

l = 0

You might think that this formula is the link between the left and the right point completely describes. However, one must note that the 3x3 matrix E is of rank 2. That is, this formula is the equation of a straight line.

In order to fully define the relationship between points, you must therefore take into account the intrinsic parameters.

We remember the q = Mp, with the intrinsic matrix M by replacing in the previous equation yields: q

r

T (M l -1) TEM -1Q l l = 0

Man replaced: F = (M l

-1

And Thus, we obtain: q

r

-1) TEM l

TFq l = 0

E. Rotationand translation

matrixvector Now that we have explained the essential matrix E and the fundamental matrix F, we have to see how the rotation matrix and the translation vector are calculated.

We define the following notations:

• P l

defines the positions of the point in the coordinate system of the left or right cameras

• R l

and P

r

) define the rotation and translation of the camera to the point in the environment for the left (or right) camera.

• R and T are the rotation and the translation that brings the right camera's coordinate system into the left camera camera system.

It arise: Pl

and Tl

(R or

r

and t

r

= R + T l P l

and P

r

= R

r

+ P t

r

It has also: P l = RT (P

r

-T)

With these three DC introduces finally arises:

R = R

r

TT = t

r

r l -RT l

F. stereo rectification

Until now we have ourselves with the theme "stereo calibration" busy. There hand drops around the Descripti the geometrical arrangement of both cameras. The task of rectification is to project the two images so that they lie in exactly the same level

and accurately align the pixel rows, thereby making the epipolar lines horizontal to find the correspondence of a point in the two images more likely As a result of the alignment of both images, there are 8 terms, 4 for each camera:

• a distortion vector

• a rotation matrix R

rect

that needs to be applied to the image

• a rectified camera matrix M

rect

• a non- Rectified camera matrix M

OpenC V allows us to compute these terms with two algorithms: the Hartley algorithm and the bouguet algorithm.

1. Hartley Algorithm The Hartley algorithm will find the same points in the two images. He tries to minimize the disparities and find homographs that put epipoles in infinite. This method does not require the intrinsic parameters for each camera to be calculated.

An advantage of this method is that calibration is possible only thanks to the observation of points in the scene. But a great disadvantage is that you have no scaling of the image, you only have the relative distance information. You can not measure exactly how far an object is removed from the cameras.

2. Bouguet Algorithm The Bouguet algorithm uses the calculated rotation matrix and the Translation vector to rotate two projected levels around half a turn so they are in the same plane. Thus, the main beams are parallel and the planets Koplanar are not yet aligned. This will be done later. In the project we have used the Bouguet algorithm.

4. Operation of stereo images

As already mentioned, the program is encoded in Python and the OpenCV library is being used. How have we decided for the Python language and the OpenCV library because we have already had experience and because there is a lot of documentation about it. Another argument for this decision is that we just wanted to work with "Open Source" libraries.

Two Python programs were developed for this project.

The first "Take_images_for_calibration.py" is used to capture good pictures later in calibration of the two cameras (distortion calibration and stereo calibration).

The second program and thus the main program "Main_Stereo_Vision_Prog.py" is used for the stereo image. We calibrate in this program the cameras with the

images that were recorded, generate a disparity card and thanks to a straightforward experiment, we can find the measuring distance for each pixel. An WLS filter is used to detect the edges of objects better.

The Python programs are listed in the appendix.

A. Used Packages The

following packages were imported into the program:

• The version of OpenCV.3.2.0 with opencv_contrib (includes the stereo functions) called "cv2" in Python contains:

o the image processing library o the stereo vision features

• Numpy.1.12.

o Used for matrices (images consist of matrices)

• Workbook of openpyxl

o Package to write data in an Excel file

• "normalize" the library sklearn 0.18.1

o sklearn enables machine learning but in this project you

only use the WLS Filter

Code 1: Package Importation

B. Video Loop

In order to work with the cameras you must first activate it. The cv2.VideoCapture () function activates both cameras by entering the number of ports from each camera (two objects are generated in the program that can use the methods of the class cv2.VideoCapture ().

Code 2: Activating cameras with OpenCV

To get an image from the cameras, the cv2.VideoCapture (). Read () method is used as an output to get an image of the scene that the camera looks at when calling this function . In order to get a video you have to call this method repeatedly in an infinite loop. To be more efficient, it is recommended to convert the BGR images into Gray images, which is performed using the cv2.cvtColor () function.

Code 3: Image Processing

To view the video on the PC, the cv2.imshow () function is used to open a window where the video can be displayed.

Code 4: Display Images

To get out of the infinite loop, a break is used. This will be activated whenever the user presses the space bar. The recognition that you pressed a key is performed using the cv2.waitKey () function. In the end, the two used cameras have to be deactivated using the cv2.VideoCapture () release () method and the open windows are closed with the function cv2.destroyAllWindows ().

Code 5: Typische Exit für ein Programm

C. Funktionsweise vom Programm “Take_images_for_calibration.py”

Wenn dieses Programm gestartet wird, werden beide Kameras aktiv und zwei Fenster werden geöffnet damit der Benutzer sehen kann wo das Schachbrett in den Bildern positioniert ist.

Diagramm 1: Funktionsweise von "Take_images_for_calibration.py"

1. Vektoren für die Kalibrierung Die Funktion cv2.findChessboardCorners() wird eine definierte Anzahl von Schachbrett-Ecken suchen und es werden folgende Vektoren erzeugt:

• imgpointsR: enthält die Koordinate der Ecken im Rechtem Bild (im Bildraum)

• imgpointsL: enthält die Koordinate der Ecken im Linkem Bild (im Bildraum)

• objpoints: enthält die Koordinate der Ecken im Objektraum

Die Präzision der Koordinate der gefundenen Ecken wird erhöht in dem die Funktion cv2.cornerSubPix() eingesetzt wird.

2. Erfassung der Bilder für die Kalibrierung Hat das Programm die Position der Schachbrett-Ecken auf beide Bilder erkannt, werden zwei neue Fenster geöffnet wo man die aufgenommenen Bilder bewerten kann. Wenn die Bilder nicht verschwommen sind und gut aussehen muss man auf die “s” (Save) Taste drücken um die Bildern zu speichern. Wenn das Gegenteil der Fall ist kann man auf die “c” (Cancel) Taste

drücken damit die Bilder nicht gespeichert werden. Mit der Funktion cv2.drawChessboardCorners() überlagert das Programm ein Schachbrett-Muster auf die Bilder.

Abbildung 11: Erkennung der Ecken von einem Schachbrett

D. Funktionsweise vom Programm “Main_Stereo_Vision_Prog.py”

In der Initialisierung werden die Kameras zuerst einzeln Kalibriert, zur Entfernung der Distorsion. Dann wird die Stereo Kalibration durchgeführt (Rotation beseitigen, Alignierung der Epipolarlinien). In eine unendliche Schleife werden die Bilder bearbeitet und eine Disparitätskarte wird erzeugt.

Diagramm 2: Funktionsweise von " Main_Stereo_Vision_Prog.py"

1. Kalibrierung der Distorsion Für die Korrektur der Distorsion der Kameras werden die Bilder verwendet die mit dem Programm “Take_images_for_calibration.py” aufgenommen wurden. Diese Kalibrierung basiert sich auf “Take_images_for_calibration.py” indem in Imgpoints und Objpoints die Position der Schachbrett-Ecken gespeichert sind. Die Funktion cv2.calibrateCamera() wird eingesetzt um neue Kamera Matrizen (Die Kamera Matrix beschreibt die Projektion von einem Punkt der 3D Welt in einem 2D Bild), Distorsion Koeffizienten, Rotation und Translation Vektoren für jede einzelne Kamera zu bekommen die später gebraucht werden um die Distorsion von jeder Kamera zu beseitigen. Um die Optimale Kamera Matrizen für jede Kamera zu bekommen wird die Funktion cv2.getOptimalNewCameraMatrix() benutzt (Erhöhung der Präzision).

Code 6: Kalibrierung der Distorsion in Python

2. Kalibrierung der Stereokamera Für die Stereo-Kalibrierung wird die Funktion cv2.StereoCalibrate() verwendet, sie berechnet die Transformation zwischen beiden Kameras (eine Kamera dient als Referenz für die andere).

Die Funktion cv2.stereoRectify() ermöglicht, die Epipolarlinien der beiden Kameras auf derselben Ebene zu bringen. Diese Transformation erleichtert die Arbeit für die Funktion die die Disparitätskarte erstellt, weil dann die Blockenübereinstimmung nur noch in einer Dimension gesucht werden muss. Mit dieser Funktion bekommt man auch die Essentielle Matrix und die Fundamentale Matrix die in der nächsten Funktion gebraucht werden.

Abbildung 12: Prinzip der Epipolarlinien

Die Funktion cv2.initUndistortRectifyMap() ergibt ein Bild das keine Distorsion hat. Diese Bilder werden dann in der Berechnung der Disparitätskarte verwendet.

Abbildung 13: Ohne kalibrierung

Abbildung 14: Mit Kalibrierung

3. Berechnung der Disparitätskarte Um die Disparitätskarte zu berechnen wird ein StereoSGBM Objekt erzeugt mit der Funktion cv2.StereoSGBM_create(). Diese Klasse benutzt ein semi-global Matching Algorithmus (Hirschmüller, 2008) um eine Stereo Übereinstimmung zwischen den Bildern der Rechten Kamera und der Linken zu erhalten.

Funktionsweise des Semi-Global Matching Algorithmus:

Folgende Szene ist der Stereokamera präsentiert:

Abbildung 15: Beispiel einer Stereokamera die eine Szene beobachtet

Man definiert in den Eingangsparametern die Größe der Blöcke. Diese Blöcke ersetzen die Pixels wenn die Größe (Block Size) größer als 1 ist. Das erzeugte SGBM Objekt vergleicht die Blöcke von einem Referenzbild mit den Blöcken vom Match Bild. Wenn die Stereokalibrierung gut gemacht wurde muss zum Beispiel ein Block der Reihe Vier vom Referenzbild mit alle Blocken des Match Bild verglichen werden die nur auf der Vierten Reihe sind. Auf diese Weise wird die Rechnung der Disparitätskarte effizienter.

Nehmen wir das vorherige Beispiel mit den Blöcken der Vierten Reihe um zu erklären wie die Disparität-Karte hergestellt wird. In der Unteren Abbildung muss man den Block (4,7) der vierten Reihe, siebten Spalte vom Basis Bild mit alle anderen Blocken(4,i) der Vierten Reihe vergleichen(Gleichen Epipolarlinie) vom Match Bild.

Abbildung 16: Übereinstimmung von Blocken suchen mit StereoSGBM

Je größer die Einstimmung zwischen dem Referenz Block mit dem Match Block desto wahrscheinlicher muss er der Selbe Punkt in der Umwelt sein.

Abbildung 17: Erkennung von Selben Blocken mit StereoSGBM

Man erkennt in diesem Beispiel dass der Referenz Block(4,7) den Höchsten Übereinstimmungsgrad mit dem Match Block(4,4) hat. In der Theorie soll es so ablaufen aber in der Praxis werden noch 4 andere Richtung Standard- mäßig in OpenCV noch bearbeitet um präziser zu sein mit derselben Methode wie für eine Richtung.

Abbildung 18: Beispiel für die Fünf Richtungen vom StereoSGBM Algorithmus bei OpenCV

Um die Disparität zu finden, werden die Koordinaten des Match Blocks mit dem Referenz Block subtrahiert, dann wird der absolute wert vom Ergebnis genommen und je größer dieser Wert desto näher an der Stereokamera ist das Objekt.

Das Programm benutzt kalibrierte schwarzweiß-Bilder für die Berechnung der Disparitätskarte, es ist auch möglich mit BGR Bilder zu arbeiten aber das würde mehr Zeitaufwand für den Computer bedeuten. Die Rechnung der Karte erfolgt mit einer Methode von unserem Stereo erzeugten Objekt, cv2.StereoSGBM_create().compute().

Code 7: Die Disparitätskarte zeigen

Mit unseren Parametern die in der Initialisierung festgelegt wurden bekommen wir folgendes Ergebnis für die disparitätskarte.

Code 8: Parametern für die Instanz von StereoSGBM

Abbildung 19: Ergebnis für die Disparitätskarte

In dieser Disparitätskarte gibt es noch sehr viel Rauschen, um den zu eliminieren wird ein morphologischer Filter eingesetzt. Ein “Closing” Filter wird verwendet mit der OpenCV Funktion cv2.morphologyEx(cv2.MORPH_CLOSE) um die kleinen schwarzen Punkte zu entfernen.

Abbildung 20: Beispiel von einem Closing Filter

Abbildung 21: Ergebnis von eine Disparität Karte nach einem Closing Filter

Ein anderes Beispiel mit derselben Szene um den Unterschied bessere zu erkennen.

Abbildung 22: Normale Szene gesehen von der Linken Kamera ohne Rektifikation und Kalibration

Abbildung 23: Disparität Karte von der Oberen Szene ohne Closing Filter und mit

4. Einsatz des WLS (Weighted Least Squares) Filters Das Ergebnis ist nicht schlecht aber es ist trotzdem sehr schwer die Rande von Objekten zu erkennen wegen des Rauschens, deswegen wird ein WLS Filter eingesetzt. Als erstes müssen die Parameter des Filters in der Initialisierung festgelegt werden.

Code 9: Parametern für ein WLS Filter

Lambda wird typisch auf 8000 gesetzt, je größer dieser Wert desto mehr werden die Formen der Disparitätskarte zu den Formen des Referenz Bild angehaftet, wir haben es auf 80000 gesetzt weil es bessere Resultate mit diesem Wert gab. Sigma beschreibt wie stark unser Filter an den Randen von Objekten präzise sein muss. Ein anderes Stereo Objekt wird erzeugt mit cv2.ximgproc.createRightMatcher() und basiert sich auf dem Ersten. Diese zwei Instanzen werden dann in den WLS Filter benutzt um eine Disparitätskarte zu erzeugen. Eine Instanz des Filters wird mit der Funktion cv2.ximgproc.createDisparityWLSFilter() hergestellt.

Code 10: Erzeugung von einem WLS Filter in Python

Um dann die Instanz des WLS Filter anzuwenden wird folgende Methode aufgerufen cv2.ximgproc.createRightMatcher().filter(), die Werte von unserem Filter werden dann normalisiert mit cv2.normalize().

Code 11: Einsetzung des WLS Filter in Python

Eine Ocean ColorMap wurde verwendet um eine bessere Visualisierung zu bekommen mit cv2.applyColorMap(). Je dunkler die Farbe desto weiter ist unser Objekt zu der Stereokamera.

Abbildung 24: Ocean ColorMap

Abbildung 25: WLS Filter auf eine Disparitätskarte mit einem Ocean Map Color und ohne

Auf diese Weise bekommen wir ein Bild, dass gut die Rande zeigen kann aber nicht mehr präzise genug ist und nicht mehr die gute Disparität Werte enthält (Die Disparitätskarte ist in float32 kodiert aber das WLS Resultat ist in uint8 kodiert). Um die gute Werte zu verwenden um später den Abstand zu Objekten zu messen, werden die Koordinaten x und y des WLS filtered Bild mit einem doppelklicken genommen. Diese (x, y) Koordinaten werden dann benutzt um den Disparität Wert von der Disparitätskarte zu bekommen um dann den Abstand zu messen. Der zurückgegebene Wert ist der Mittelwert der Disparität einer 9x9 Pixel-Matrix.

Abbildung 26: Rechnung des Abstand mit dem WLS Filter und die Disparitätskarte

5. Messen des Abstands Nachdem die Disparitätskarte erzeugt wurde, muss der Abstand bestimmt werden. Die Arbeit besteht darin, die Relation zwischen den Disparitätwert und den Abstand zu finden. Um dies zu erledigen haben wir die Disparitätwerte an mehreren Stellen experimentell gemessen um daraus eine Regression zu bestimmen.

210 200 190 180 170 160 150 140 130 120

y = -593,97x3 + 1506,8x2 - 1373,1x + 522,06

110 100 90 80 70 60

0,35 0,4 0,45 0,5 0,55 0,6 0,65 0,7 0,75 0,8 0,85 0,9 0,95 1

Geradengleichung in das Python Programm

Code 12: Die Regression Formel in „Main_Stereo_Vision_Prog.py“

Um diese Geradengleichung der Regression zu bekommen wurde der Packlage openpyxl eingesetzt um die Werte der Disparität in einer Excel Datei zu speichern. Die Linien in dem Programm die dafür gesorgt haben, dass die Werte gespeichert werden, wurden in dem Programm kommentiert, aber man kann sie unkommentierten wenn man eine neue Geradengleichung braucht. Die Abstandsmessung gilt nur von einem Abstand von 67cm bis zu 203cm um gute Ergebnisse zu bekommen. Die Präzision der Messung hängt auch mit der Qualität der Kalibrierung ab. Mit unsere Stereokameras haben den Abstand zu einem Objekt mit einer Präzision von +/ 5 cm messen können.

Abbildung 27: Experimentalen Messung der Disparität in Abhängigkeit vom Abstand zum Objekt

A

b

s

t

a

n

d

[

c

m

Abstand zum Objekten [cm] in abhängigkeit von den Disparitäts Werten

]

Disparität

6. Mögliche Verbesserungen Mögliche Verbesserungen für das Programm:

• Die Form vom WLS Filter nehmen und die, auf der Disparitätskarte zu projektieren. Diese Projektion würde dann genommen werden um alle Disparität Werte die sich in der Form befinden zu nehmen und der Wert der am häufigsten vorkommt wird dann als Wert für die gesamten Fläsche gesetzt.

Abbildung 28:Mögliche Verbesserung mit dem Ersten Vorschlag

• Verwendung von einem Bilateral Filter auf die kalibrierten Bilder bevor die Disparitätskarte erzeugt wird, auf dieser weise wäre es möglich keinen WLS Filter einzusetzen. Dass muss überprüft werden, aber der WLS Filter dient nur um die Rändern von Objekten gut zu erkennen, vielleicht gibt es da noch ein besseren weg.

• Um die Berechnungszeit für die Erzeugung der Disparitätskarte zu verringern sollte man die kalibrierten Bilder verkleinern mit der Funktion cv2.resize(cv2:INTER_AREA), man sollte dann achten dass die Werte der Essentiellen Matrizen, Fundamentalen Matrizen auch proportional verkleinert werden.

• Die Erzeugung von einer Tiefen-Karte könnte auch von Vorteil sein.

• Eine Kamera die stabiler ist als unsere wo die Rotation von den Köpfen wirklich verhindert werden kann, auf dieser weise bräuchte man nur die Werte der Matrizen von der Stereokalibrierung zu speichern um die wieder einsetzen zu können. Sehr viel Zeit könnte damit in der Initialisierung gespart werden.

• Das Programm auf dem GPU laufen zu lassen würde uns auch ermöglichen glättere Bilder zu bekommen, wenn die Stereokamera in Bewegung ist

5. Schlussfolgerung

A. Zusammenfassung

Diese Projektarbeit hat uns ermöglicht mit der Python Sprache zu arbeiten und mehr über die OpenCV Bibliothek zu lernen. Wir haben die Gelegenheit gehabt, uns mit dem Thema Stereo Vision in diesem Projekt zu vertrauen. Ein Neues Thema, dass uns beide sehr interessiert und dass noch sehr neu ist in Vergleich zu anderen Abstandsmessung Techniken die verfügbar sind. Es existieren auch andere Methoden um den Abstand mit Bildverarbeitung zu messen, zB Time of Light(TOF) Kameras, die allerdings teuer sind und mit der es sehr wenig Dokumentation gibt. Wir haben uns auch entschieden einfache Kameras zu benutzen wegen dem Preis.

B. Fazit

In der hier ausgeführten Projektarbeit ist es mit dem entwickelten Programm möglich ein Abstand zu Rechnen auf der Basis von einer Disparität Karte.

C. Ausblick

Damit die gerechneten Abstandswerte immer korrekt bleiben, sollte man ein neues System für die Kameras entwickeln das alle freie Bewegungen der Kameras vermeidet. Somit würden nur Matrizen Werte verwendet um die Kalibration schneller durchzuführen. Die benutzte Geradengleichung würde immer den exakten Abstand zum Objekt zurückgeben mit einer größeren Präzision, mit weniger Aufwand.

Literaturverzeichnis

OpenCV-Python Tutorials

OpenCV Dokumentation

Stack Overflow

RDMILLIGAN auf seine Internet Seite rdmilligan.wordpress.com

Stereo Vision: Algorithms and Applications von Stefano Mattoccia, Department of Computer Science(DISI), University of Bologna

Stereo Matching von Nassir Navab und Slides prepared von Christian Unger

Oreilly Learning OpenCV

6. Anhang

Video zu unserem Projekt: https://youtu.be/xjx4mbZXaNc Die Python Programme können im Ordner „Python_Prog_Stereo_Vision“ gefunden werden. Dieser Besteht aus den Programmen:

- **Take_images_for_calibration.py”**
- ”Main_Stereo_Vision_Prog.py” Die zwei Programme in einer .txt Version können auch im selben Ordner gefunden werden wenn man nicht Python auf seinem Computer installiert hat.