UNIVERSITA' DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII

Corso di Laurea Magistrale in Ingegneria Aerospaziale

DENSE MONOCULAR VISUAL ODOMETRY AIDED BY RANGE SENSORS FOR MOBILE ROBOTS

Relatore: Stefano Debei

Correlatore: Marco Pertile

Luca Brugiolo mat. 1063985

Anno Accademico 2017/2018

Table of Contents

Riassunto esteso

Gli strumenti tradizionalmente utilizzati per odometria visuale e ricostruzione densa possono essere poco adatti per veicoli di piccole dimensioni, a causa delle loro limitazioni di potenza, massa e volume.

Un configurazione monoculare è in grado di superare queste limitazioni. Un sistema monoculare è in grado di ricavare le pose della camera, e quindi il moto del veicolo; queste pose, tuttavia, non risulteranno scalate. Per superare questo problema si propone l'utilizzo del algoritmo di odometria monoculare ibrida sviluppato da Sebastiano Chiodini, Riccardo Giubilato, Marco Pertile and Stefano Debei [1]. Una volta che le pose sono accuratamente scalate, è possibile completare la ricostruzione densa, trattando una coppia di immagini successive come una coppia stereo.

Questa tesi si concentra primariamente nel definire un algoritmo di ricostruzione densa monoculare, scritto ed ideato per lavorare in concerto con l'algoritmo di odometria visuale monoculare ibrida sopra menzionato. La tesi è suddivisa in quattro capitoli principali.

In apertura si riprendono alcune delle basi teoriche fondamentali della visione artificiale, in particolare il modello pin-hole, la geometria epipolare e la triangolazione.

Nel secondo capitolo viene brevemente presentato l'algoritmo di odometria visuale monoculare ibrida scritto e progettato da Chiodini et al [1]. A questo punto, per superare l'incertezza di scala delle pose, viene proposto di accoppiare le misure ricavate da una singola camera RGB con le misure ottenute da una camera a tempo di volo di ridotto campo di vista. Questa configurazione ha l'obbiettivo di mantenere i pregi principali di una configurazione monoculare; ovverosia, bassa potenza e basso volume e massa occupati.

Il terzo capitolo, che rappresenta il cuore della tesi, descrive dettagliatamente l'algoritmo di ricostruzione densa monoculare. Quest'algoritmo usa le pose ricavate dall'odometria monoculare per analizzare coppie di immagini come farebbe un tradizionale algoritmo di ricostruzione stereo. Per poter far questo, l'algoritmo di

ricostruzione densa monoculare deve correggere la geometria epipolare delle immagini in modo che sia uguale a quella stereo. Quindi, esso ricava la geometria epipolare di ogni coppia di immagini, e la descrive tramite la matrice fondamentale. Dopodiché, ricerca e accoppia features tra le immagini, che saranno utilizzate durante la fase di rettifica e per costruire una nuvola di punti sparsa. A questo punto, l'algoritmo sceglie che metodo utilizzare per correggere la geometria epipolare delle immagini. Se questa è simile a quella stereo, allora viene utilizzata la rettifica lineare; se invece i punti epipolari sono vicini o interni alle immagini, l'algoritmo rettifica le immagini con un metodo polare. Completata la rettifica, ovvero l'operazione che trasforma la geometria epipolare delle immagini, l'algoritmo cerca corrispondenze dense tra le due immagini. Da questa informazione, l'algoritmo triangola la posizione dei punti nell'immagine, ottenendo così la loro corrispondente posizione tridimensionale. Questa viene usata per costruire una nuvola di punti densa, che poi viene filtrata per rimuovere rumore, e infine trasformata nel sistema di riferimento globale. Quest'operazione viene ripetuta per ogni coppia di immagini nella sequenza analizzata.

Nel capitolo finale vengono presentati e descritti i risultati della ricostruzione densa effettuata su diverse sequenze. Il capitolo è diviso in due parti: nella prima, si descrivono i risultati per sequenze rettificate con il metodo lineare; nella seconda, i risultati di sequenze rettificate con il metodo polare. Le sequenze rettificate con il metodo lineare hanno una geometria epipolare facile da risolvere; la loro ricostruzione densa risulta accurata, e non ha presentato problemi notevoli. Di contro, le sequenze rettificate con metodo polare che progressivamente si allontanano da una geometria epipolare ideale peggiorano nei risultati. Nelle prime due sequenze, dove la camera è orientata rispettivamente a 60° e a 45° dalla direzione del moto, la ricostruzione è meno accurata del caso lineare; tuttavia, l'ambiente ricostruito nelle nuvole di punti dense si avvicina ancora a quello visto dalla camera. Nell'ultimo caso, la camera era montata con asse ottico parallelo alla direzione del moto. In queste condizioni, la geometria epipolare è troppo complessa da risolvere ed il metodo polare deforma sensibilmente le immagini. Le risultati nuvole di punti dense sono quindi poco accurate e poco rappresentative dell'ambiente visto dalla camera.

Introduction

Instruments traditionally used to compute Visual Odometry (VO) and Dense Reconstruction (DR) can be inadequate for small vehicles, due to their limitation to volume, mass, and power. This is especially true for rovers designed for planetary exploration.

Visual Odometry is a process that estimates the position and orientation of a vehicle during its motion using only information provided by the vision system mounted on the vehicle. To do this, the VO keeps track of same landmarks, registered in a sequence during a given period of time. Then, this information is used to compute the rotation and translation of the vision system at different time steps. From the motion estimate of the camera and the calibration between the vehicle and the vision system, the VO finds the motion of the vehicle. The most common VO algorithms utilize a stereo set-up as a vision system. However, this type of configuration is not always possible in exploration vehicles, given the strict constraint in terms of mass and volume.

The aim of Dense Reconstruction is to accurately represent in a 3D map the environment exactly as seen by the vehicle while moving. A DR algorithm can provide useful information about the objects and obstacles surrounding the vehicle, information that a sparse based method can not provide. Instruments that are usually employed to complete DR are range sensor with an ample Field of View (FoV). Two of such instruments are 3D LiDARs and RGB-D cameras. They are able to directly build a 3D map of the environment as seen inside their FoV. However, they have some specific limitations. While adequate for indoor application, an RGB-D camera has degraded performances if used outside. On the other hand, 3D LiDARs do not suffer from this limitation : however, they are bulky and heavy and are highly power consuming. An alternative is to use a stereo set-up, but the limits in volume on small vehicles render this set-up hardly feasible.

A solution, to surpass these limitations in small rovers and robots, is to use a monocular set-up to complete both VO and DR. However, a pure monocular set-up is only able to compute the camera poses up to scale. The solution we adopted in this thesis was designed by Sebastiano Chiodini, Riccardo Giubilato, Marco Pertile and Stefano Debei [1]. Following their method, we adopted a hybrid system composed of a small Time of Flight (ToF) camera and an RGB-camera to eliminate the scale uncertainty.

Once the poses are correctly scaled, we use that information in an algorithm that compute a dense reconstruction. This is done by considering every subsequent pair of pictures as a pair in a stereo set-up, and using the newfound scaled poses to adjust their epipolar geometry. Therefore, an algorithm written for a Dense Monocular Visual Odometry is composed of two parts:

1. The first part uses the information gathered through images taken by a monocular setup and by a range sensor to define the position and orientation of the rover through time. This is the Monocular Visual Odometry (MVO) step.
2. The second part uses the MVO information and the camera images obtained during the previous step to build a dense reconstruction of the environment as seen by the rover during its motion. This is the Dense Reconstruction part (DR).

The algorithm presented in this thesis focus on the second part. It was written in order to work in conjunction with the MVO written by Chiodini et al.

A part from this introduction the thesis consist of four main chapters. Initially, we introduce some preliminaries concepts regarding computer vision. Then, as a methodological basis, we report a brief overview on the MVO algorithm. Only at this point we are able to properly describe the DR algorithm. Finally, in the last parts, we report the results gathered with five different datasets, and we discuss them.

Related Works

For what concerns the Monocular VO approach, the algorithm was provided by Chiodini et al. For this reason, we invite the readers to consult their works for a thoughtful explanation of how the ego-motion is estimated [1].

The Monocular Dense Reconstruction algorithm depends heavily from the polar rectification design by Marc Pollefeys, Reinhard Koch and Luc Van Gool [2]. Marc Pollefeys and Sudipta Sinha discuss the iso-disparity surfaces for a general camera motion in [3], and the uncertainty of reconstruction in the direction of motion due to the distortion of these surfaces. S. Cavegn, N. Haala, S. Nebiker, M. Rothermel and T. Zwölfer [4] in their six cameras set-up utilize the polar rectification to correct the epipolar geometry of forwarding facing pairs of cameras. Then, they use this data to compute a dense reconstruction in a way similar to the one proposed in this thesis.

One of the most advanced algorithms for Monocular Dense Reconstruction is REMODE [5] designed by Matia Pizzoli, Christian Forster and Davide Scaramuzza. REMODE is able to accurately reconstruct the scene in real time, to compute the depth maps it uses a probabilistic approach based on a Bayesian estimation. This is combined with a smoothing method in order to provide spatial regularity and to mitigate the effect of noisy camera localization.

W. Nicholas Greene, Kyel Ok, Peter Lommel, and Nicholas Roy [6] designed an accurate method to compute Monocular Dense Reconstruction that builds the depth maps, dividing the frames into images regions based on the available texture. This allows them to represent the different portions of the image differently based on the information it contains.

Chapter 1

Theoretical framework

Although this thesis is not eminently concerned with the speculative aspects of Visual Odometry and Dense Reconstruction, the theoretical grounding supporting these sets of algorithms will be recalled frequently during the text. For this reason, it could be of some use to briefly recollect the core elements of this theory.

1.1 Homogeneous Coordinates

Homogeneous coordinates are a type of coordinates used in projective geometry. They are largely employed in computer vision, where they greatly simplify the vast majority of the equations involved, leading to simpler formulations than their Cartesian counterpart.

They were firstly introduced by A. F. Mobius in his 1827 work *Der barycentrische Calcül*, as a set of coordinates capable to represent every point with finite coordinates. Beyond that, they are able to define even points at infinity. In order to do this, coordinates of N dimension in the Euclidean space are mapped to coordinates of N+1 dimension in the homogeneous coordinate.

For a given point **x** (x,y) a set of homogeneous coordinates is $\mathbf{x_h}$ (xZ,yZ,Z) for any not null value of Z. Therefore, a point in the Euclidean space is mapped to a series of equivalent homogeneous coordinates, which differ themselves only by a scalar Z. As such, even (x,y,1) represent the same point (x,y) in the Euclidean space.

At this point, it could be of use to recall briefly some of the cardinal proprieties of homogeneous coordinates:

- If we multiply the homogeneous coordinates for a non null scalar, we obtain a set of homogeneous coordinates mapping the same point in Euclidean space.
- A line in homogeneous coordinates is defined by N+1 variables;

  For the 2D space a line **l** in homogeneous coordinates is represented by:

$$\boldsymbol{l}=(a,b,c)^T \tag{1.1}$$

- l consist of the homogeneous points **x** (x,y,z) that satisfy:

$$\boldsymbol{x}^T\boldsymbol{l}=ax+by+cz=0 \tag{1.2}$$

- The line passing through two points $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ is given by the cross product:

$$\boldsymbol{l}=\boldsymbol{x}_1\wedge\boldsymbol{x}_2 \tag{1.3}$$

1.2 Pinhole model

A camera model projects points from the 3D space to a 2D plane, the image plane. The simplest of these models is the pinhole model.

This model is based upon the idealization of the thin lens model when the aperture shrinks to zero. In this conditions, all light rays are parallel to the optical axis and are focused to the focal point of the optical system. This is just an idealization, because in those conditions no light would actually pass through the lens: nevertheless, it can be used for points that are on focus.

We will take into account a 3D orthogonal coordinate system centered on the camera center, with its Z axis as the optical axis of the camera. The image plane, then, is a plane perpendicular to Z and distant f, the focal length, from the camera center. A line from the camera center to a point in 3D space intersect the image plane on a point **m**: the position of this point can be defined on the image plane through a 2D coordinate system, centered at the intersection of the image plane with Z and with axis x and y parallel to the axis X and Y of the 3D system.

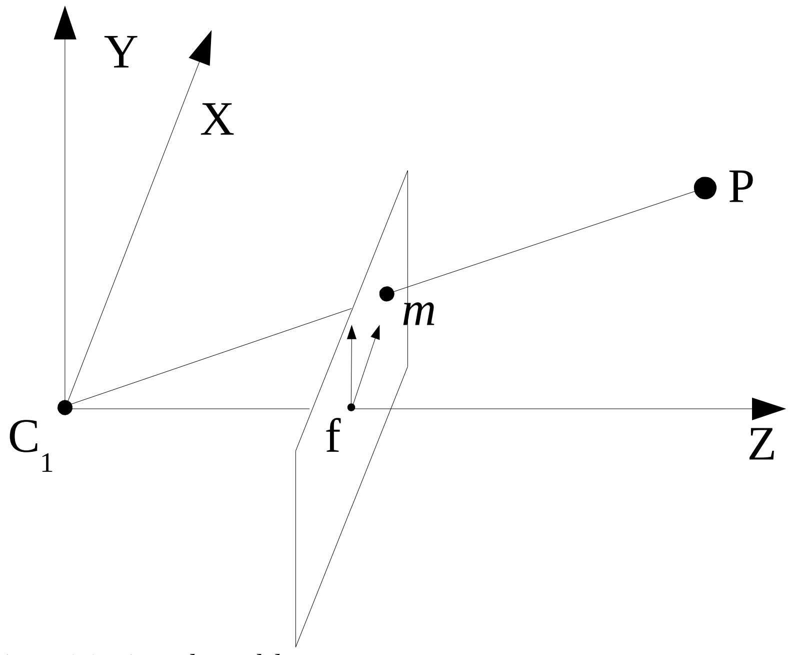


Figure 1.1: Pin-Hole model

Let **P** be the point in the 3D space with coordinates:

$$\boldsymbol{P}=[X_p, Y_p, Z_p]^T \tag{1.4}$$

and **m** the projection of **P** on the image plane with coordinates in the 2D plane:

$$\boldsymbol{m}=[x_m, y_m]^T \tag{1.5}$$

Then, through similar triangles, a relationship between the 3D and 2D coordinate systems can be defined:

$$x_m=f\,X_P/Z_P \qquad y_m=f\,Y_P/Z_P \tag{1.6}$$

This relation in homogeneous coordinates can be written as follow:

$$\lambda \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ 1 \end{bmatrix} \tag{1.7}$$

Where λ is the depth factor and is equal to Z_p.

However, points in the image plane are not normally defined through a system centered at the intersection with the Z axis. Also, they are measured in pixels and not in meters: this is due to the fact that the image plane is the sensor of the camera. Thus, the position on the sensor is discretized in pixel, and the coordinates are centered on the top left corner of the sensor. The conversion to the pixel coordinates is easily done; if the camera is calibrated accordingly, then:

$$u = k_x\, x_m + c_x \qquad\qquad v = k_y\, y_m + c_y \tag{1.8}$$

Where *u* is the coordinate in the direction parallel to *x*, *v* parallel to *y*. c_x and c_y are the positions of the center of the sensor in pixel coordinates. k_x and k_y are scale factors from meters to pixels in the two directions.

Equation 1.7 then becomes:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & k_s & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ 1 \end{bmatrix} \tag{1.9}$$

Where k_s is the skew factor. If the sensor axes are not orthogonal, then this factor must be taken into consideration. However, in the majority of the cases, k_s is null.

The product of the first two matrices on the right side of the equation is called calibration matrix:

$$K=\begin{bmatrix} k_x & k_s & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} k_x f & k_s f & c_x \\ 0 & k_y f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

K is also called the intrinsic parameters matrix, because it contains the information on the intrinsic parameters of the camera.

If the camera system does not coincide with the world system, we need to account for it. Therefore, we transform the coordinate from the camera system to the world system. Let *R* and **t** be the rotation and the translation that bring the coordinates from the world system to the camera system, then:

$$\boldsymbol{X_c}=[R|\boldsymbol{t}]\boldsymbol{X_w} \tag{1.11}$$

Where $\boldsymbol{X_c}$ are the coordinates in the camera system, the one used until this point, and $\boldsymbol{X_w}$ are the world coordinates.

Equation 1.11 can then be written as:

$$\lambda\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}= K\ [R|\boldsymbol{t}]\boldsymbol{X_w} \tag{1.12}$$

Where the product between the calibration matrix and the roto-translation matrix is called the projection matrix:

$$Pj= K\ [R|\boldsymbol{t}] \tag{1.13}$$

And thus :

$$\dot{\boldsymbol{x}}_{\boldsymbol{m}}=[Pj]\boldsymbol{X}_w \qquad where \quad \dot{\boldsymbol{x}}_{\boldsymbol{m}}=\begin{bmatrix}\lambda u\\ \lambda v\\ \lambda\end{bmatrix} \tag{1.14}$$

1.2.1 Distortions

The lens on a camera tends to distort the image. This distortion is defined through seven coefficients, grouped in two different set:

- k_1, k_2, k_3, k_4, k_5 are radial distortion coefficients.
- p_1, p_2 are tangential distortion coefficients.

We obtain these coefficients through the calibration of the camera. Then, we correct the distortions as follows:

$$r^2=x_m^2+y_m^2$$

$$\begin{aligned} x_m'&=x_m\frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}+2p_1x_my_m+p_2(r^2+2x_m^2)\\ y_m'&=y_m\frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6}+2p_2x_my_m+p_1(r^2+2y_m^2)\end{aligned} \tag{1.15}$$

$$u=fk_xx_m'+c_x \qquad v=fk_yy_m'+c_y$$

1.3 Epipolar Geometry

The epipolar geometry is used to describe relationship between corresponding points in different camera views. For the sake of simplicity simplicity, we have decided to explain the epipolar geometry by referencing to stereo setup.

We define the epipolar geometry with a set of planes that have a common axis: the baseline. These planes are called epipolar planes: the intersection of those plane with the image planes define the epipolar lines, and the intersection of the baseline with the image planes define the epipolar points. The baseline is the line passing through the two camera center, so the epipolar point of one image is the projection on that image plane of the camera center of the other image.

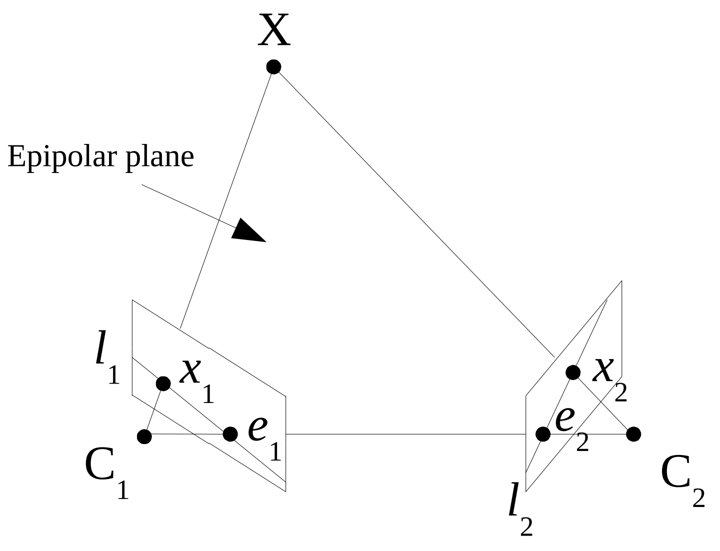


Figure 1.2: Epipolar geometry

Let us identify a point in the 3D space as **X**. This point and his projection on the left and right image planes, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, lie on the same epipolar plane. Considering another 3D point the epipolar plane and lines change, but the epipolar points remain the same. Therefore, all the epipolar lines on one image plane intersect the epipolar point for that image. If only the position of $\boldsymbol{x}_1$ and the epipolar geometry are known, then the epipolar plane passing through $\boldsymbol{x}_1$ is defined, and so are the epipolar lines $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$. Therefore, we can search for the point $\boldsymbol{x}_2$ only along the epipolar line $\boldsymbol{l}_2$ instead of the whole right image. This correlation between points on different images is called epipolar constraint and greatly simplifies the research and match of corresponding points in an image pairs.

1.3.1 Fundamental Matrix

The fundamental matrix is a 3x3 matrix of rank 2 and the algebraic representation of the epipolar geometry. Given **X** and the corresponding points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ written as homogeneous coordinates, then:

$$\boldsymbol{x}_2^T F \boldsymbol{x}_1 = 0 \tag{1.16}$$

where *F* is the fundamental matrix. Equation 1.16 algebraically represents the epipolar constrain.

Given a couple of images there is only one set of fundamental matrix able to satisfy Equation 1.16 for all corresponding points $\boldsymbol{x}_1$, $\boldsymbol{x}_2$. This set of matrices differentiate themselves only for a scale factor.

This matrix can be seen as a sort of map from one point on an image to the correspondent epipolar line on the other image:

$$\boldsymbol{l}_1 = F^T \boldsymbol{x}_2 \qquad \boldsymbol{l}_2 = F \boldsymbol{x}_1 \tag{1.17}$$

All the epipolar lines on one image intersect on the epipolar point, $\boldsymbol{e}_i$ of that image. Thus:

$$\boldsymbol{e}_1^T \boldsymbol{l}_1 = \boldsymbol{e}_1^T F^T \boldsymbol{x}_2 = 0 \qquad \boldsymbol{e}_2^T \boldsymbol{l}_2 = \boldsymbol{e}_2 F \boldsymbol{x}_1 = 0 \tag{1.18}$$

for every point $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, it follows that:

$$F \boldsymbol{e}_1 = 0 \qquad F^T \boldsymbol{e}_2 = 0 \tag{1.19}$$

The fundamental matrix can be derived algebraically using the projection matrices P_{j1} and P_{j2}. All the points that project from the 3D space to the point $\boldsymbol{x_1}$ in the image plane of the first camera can be defined by [7]:

$$\boldsymbol{X}(\lambda)=Pj_1^{+}\boldsymbol{x_1}+\lambda\boldsymbol{C_1} \tag{1.20}$$

Where: Pj_1^{+} is the pseudo-inverse of the projection matrix Pj_1, and $\boldsymbol{C_1}$ is the position in homogeneous coordinates of the camera center of the first camera.

Equation 1.20 represents a parametrization through λ of the ray back projecting from $\boldsymbol{x_1}$ to the 3D space. In particular, two points lays on this ray, the camera center $\boldsymbol{C_1}$ and $Pj_1^{+}\boldsymbol{x_1}$. These two points are represented on the image plane of the second camera on $Pj_2 C_1$ and $Pj_2 Pj_1^{+}\boldsymbol{x_1}$. A line on the image plane of the second camera $\boldsymbol{l_2}=Pj_2\boldsymbol{C_1}\wedge Pj_1^{+}\boldsymbol{x_1}$ passes through these two points; this has to be an epipolar line, since $Pj_2\boldsymbol{C_1}$ is the epipolar point $\boldsymbol{e_2}$. Thus:

$$\boldsymbol{l_2}=Pj_2\boldsymbol{C_1}\wedge Pj_1^{+}\boldsymbol{x_1}=[\boldsymbol{e_2}]_{\wedge}Pj_1^{+}\boldsymbol{x_1}=F\boldsymbol{x_1} \tag{1.21}$$

and so from:

$$F=Pj_2\boldsymbol{C_1}\wedge Pj_1^{+}=[\boldsymbol{e_2}]_{\wedge}Pj_1^{+} \tag{1.22}$$

the Fundamental matrix is obtained, once the projective matrices of the two cameras are known.

As a simple example, we will consider a calibrated stereo set-up, with the left camera reference system coinciding with the world camera system. In this example, therefore:

$$Pj_1=K_1[I|\boldsymbol{0}] \qquad\qquad Pj_2=K_2[R|\boldsymbol{t}] \tag{1.23}$$

$$Pj_1^+ = \begin{bmatrix} K_1^{-1} \\ \mathbf{0} \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{C}_1 = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{1.24}$$

$$F = [Pj_2 \boldsymbol{C}_1]_\wedge Pj_1^+ = [K_2 \boldsymbol{t}]_\wedge K_2 R K_1^{-1} = K_2^{-T} [\boldsymbol{t}]_\wedge R K_1^{-1} \tag{1.25}$$

$$F = K_2^{-T} [\boldsymbol{t}]_\wedge R K_1^{-1} = K_2^{-T} R [R^T \boldsymbol{t}]_\wedge K_1 = K_2^{-T} R K_1^T [K_1 R^T \boldsymbol{t}]_\wedge \tag{1.26}$$

1.4 Triangulation

Triangulation is the process that let us determine the position of a point in the 3D space by matching points in the image planes. Essentially, if $\boldsymbol{x}_1$ is a point in first image and $\boldsymbol{x}_2$ is the corresponding point on the second image, then the line from the camera center of the first image, $\boldsymbol{C}_1$, that pass through $\boldsymbol{x}_1$ and the line from the second camera center $\boldsymbol{C}_2$ and $\boldsymbol{x}_2$ will intersect in a position in the 3D space. Furthermore, these two lines intersect one another only if the two points satisfy the epipolar constraint.

Solving the triangulation problem means finding the **X** that satisfy:

$$\dot{\boldsymbol{x}}_1 = Pj_1 \boldsymbol{X} \qquad \dot{\boldsymbol{x}}_2 = Pj_2 \boldsymbol{X} \tag{1.27}$$

For every given couple of points $\dot{\boldsymbol{x}}_1$ and $\dot{\boldsymbol{x}}_2$ on the image plane.

1.4.1 Triangulation: Ideal Case

In an ideal stereo set-up, the two cameras must have the optical axes parallel one another and perpendicular to the baseline. The rotation matrix between these cameras is the identity matrix, and the translation is only in the horizontal direction with a magnitude equal to the baseline. Also, the cameras must have the same intrinsic parameters.

We will consider an ideal stereo set-up, where the world coordinate system coincides with the left camera coordinate system. In this case, the position of a point **P** in the 3D space can be related to its correspondent position on the two images planes through Equation 1.12. Therefore:

$$\lambda_1 \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = K \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ 1 \end{bmatrix}$$

$$\lambda_2 \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = K \begin{bmatrix} 1 & 0 & 0 & -b \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ 1 \end{bmatrix} \tag{1.28}$$

these equations are easily solved for X_p , Y_p and Z_p :

$$X_p = b \frac{u_1 - c_x}{u_1 - u_2}$$

$$Y_p = b \frac{k_x}{k_y} \frac{v_1 - c_y}{u_1 - u_2} \tag{1.29}$$

$$Z_p = b k_x \frac{f}{u_1 - u_2}$$

Chapter 2

Monocular Visual Odometry (MVO)

In this chapter, we are going to describe the Monocular Visual Odometry (VO) algorithm, necessary for the correct operating of the Dense Monocular Reconstruction algorithm.
The MVO algorithm was created by Sebastiano Chiodini, Riccardo Giubilato, Marco Pertile and Stefano Debei [1]. They propose an Hybrid solution to the scaling problems concerning a typical Monocular Visual Odometry. Namely, they couple the information provided by an RGB-camera to the data gathered with a Time-of-Flight camera of small Field of View.

The aim of a Monocular Visual Odometry algorithm is to provide an accurate ego-motion estimate of a moving vehicle. The information obtained can then be passed to the Dense monocular reconstruction algorithm to build a 3D representation of the environment viewed by the camera. This chapter provides only a brief explanation of how the Monocular Visual Odometry algorithm of Chiodini et al. Works.

2.1 Hybrid MVO

In a monocular set-up, the information provided by an RGB camera is not sufficient to produce an accurate VO. An RGB camera, in fact, is only able to define the VO up to scale. Therefore, to accurately scale the VO we need a range sensor.

The hybrid Monocular VO uses a low-resolution Time-of-Flight (ToF) camera to compensate the scale ambiguity of a monocular VO. This hybrid set-up preserves the main perks of a monocular VO. Namely, these are three: low power usage, low mass, and low volume occupied. The authors of this algorithm provide also a method to calibrate the relative position and orientation between the ToF camera and the RGB-camera [8].

Thanks to this calibration, the Monocular VO algorithm can transfer the ToF data to the camera reference system.
The principal elements of this Hybrid Monocular VO are:

- Map initialization.
- VO scaling.
- Local bundle adjustment.

2.2 Map Initialization.

At the start, the algorithm initializes an unscaled 3D map. It takes two consecutive frames, and detects their features. Then, it uses this information to find an unscaled transformation between the two frames. With this newfound transformation, it triangulates the features in an unscaled 3D cloud. At this point, it completes a full bundle adjustment to correct the two camera poses. This bundle adjustment optimizes the 3D cloud and the roto-translation matrix for the camera pair.

2.3 Map Scaling

To accurately scale the VO, the algorithm finds corresponding landmarks between the ToF camera and the RGB-camera. Chiodini et al describe two possible methods: an association on the image plane, and an association on the landmarks map space.

1) Image plane association

Firstly, the algorithm transfers the ToF 3D data from the ToF reference system to the reference frame of the first frame in the pair. To associate points from the ToF to the RGB-camera, we need them to be in the same plane. Therefore, the algorithm projects the points of the ToF to the image plane of the first frame. For every newfound point, it finds a matching point in the second frame. This is done by computing their descriptor and searching for matching points in the second frames only along corresponding epipolar lines. Then, it triangulates the matching points, and compares their three positions with their respective position from the ToF. From this comparison, it gets a scale factor. The algorithm then uses this scale factor to adjust the estimated transformation. Finally, it performs a local Bundle adjustment to optimize the camera poses obtained until that point.

2) Landmarks space association

The first step of this method is identical to the previous one. Again, the points from the ToF are projected to the image plane of their respective camera. Then, for every point, the algorithm finds three Speeded-up Robust Features (or SURF; we will expand the definition of this method later during the thesis) around it. It triangulates the position of these three features, obtaining a set of three points in the 3D space. These three points are used to define a plane in 3D space. If the estimate of the unscaled transformation was correct, then this plane would contain the 3D points of ToF camera. Therefore, the algorithm finds a scale factor from the intersection between the plane and the rays back-projecting from the ToF cloud to the camera.

2.4 Local Bundle Adjustment (BA)

At regular intervals, the algorithm performs a local bundle adjustment to improve the computed poses. Bundle adjustment is a global optimization method largely used in VO [9]. It utilizes correspondences between the data calculated at the various frames to optimize the camera poses. Therefore, it requires that the algorithm keeps track of the same landmarks as view by different frames. A global bundle adjustment can be highly time consuming. Furthermore, the hybrid Monocular VO algorithm does not perform loop closure, so it does not need to perform a global bundle adjustment.

The proposed algorithm performs a local bundle adjustment, optimizing only the poses for a fixed number of previous frames. The data the algorithm uses to compute the bundle adjustment is: the key-points, their relative 3D position, the cloud of the ToF camera, and the camera poses. Then, the optimization problem can be solved with the Levenberg-Marquardt optimization.

2.5 Hybrid MVO:Results

To evaluate the hybrid MVO algorithm, Chiodini et al. mounted the RGB-camera and the ToF-camera on a linear slide with a resolution of 1 mm. The vision system took pictures at fixed intervals of 50 mm; this measurement gave them the ground truth necessary to evaluate the results of the MVO.

They tested three different sequences: an indoor sequence, an outdoor sequence, and a non-planar sequence. In all the sequences, the hybrid MVO algorithm is able to retrieve the scene absolute scale with both scaling methods. The relative error is comparable with the standard stereo set-up. Therefore, we can state that the accuracy of the estimated poses is sufficient to achieve a correct dense reconstruction.

Chapter 3

Dense Monocular Algorithm

The following chapters will focus on the presentation of the Dense Monocular (DM) algorithm. We will start providing a general overview of the main issues and limits relative to the use of a monocular set-up instead of a stereo set-up. Secondly, we will present a general overlook of the algorithm itself. Then, the main elements of the DM algorithm will be explained in details. We will review how they operate in practice, and how they cooperate to complete the dense reconstruction.

3.1 Stereo/Mono Dense reconstruction

A stereo set-up is composed of two cameras in a fixed position between one another. The two optical axes are parallel, and the vector that connects the two camera centers, the baseline, is always perpendicular to the optical axis.

The stereo set-up takes two pictures at the same time, one from the left camera, the other from the right camera. From these pictures, it forms one pair. This pair is easily analyzed by a dense reconstruction algorithm because:

- The relative position between the cameras is fixed and know.
- The epipolar lines on the left and right pictures are always parallel and have the same direction of the baseline vector.
- The pictures require little to none adjustments to calculate a correct depth map.

All this greatly simplifies the process needed to complete the dense reconstruction.

In a monocular set-up, all the images are taken by only one camera. Therefore, we form a pair with pictures taken at subsequent times by the same camera. Furthermore, the camera pose can change significantly from one frame to another. Therefore, the orientation of the optical axis and the baseline can change significantly from one frame to the other.

Some of the issues regarding the monocular set-up are apparent:

- The optical axis on the two camera poses are not generally parallel one another, because the camera can move freely.
- The vector between the camera centers at subsequent times is not perpendicular to both camera axis.

Therefore, for every pair of images in a monocular set-up, the fundamental matrix changes. Furthermore, the epipolar geometry is not well defined as in a stereo set-up: epipolar lines are not horizontal, they are not parallel one another, and corresponding epipolar lines do not align.

To solve this problem we need to find the fundamental matrix between the two poses. Then, we can use this matrix to transform the pair in a rectified image pair. This pair will have the same epipolar geometry of an ideal stereo pair.

3.2 Algorithm Overview

The code for the Dense Monocular algorithm was written in C++ and largely used the OpenCV library [10] and the Point Cloud Library [11].

The algorithm requires three sets of inputs: a stream of images, the camera poses on these images, and the intrinsic parameters of the camera.

The main cycle starts by loading a pair of pictures and the corresponding camera poses. From the camera poses, the algorithm calculates the fundamental matrix and the projective matrices. After that, the algorithm checks if one of the epipolar points is near the image planes, to evaluate witch rectification method should be used.

The next step is the feature detection and matching process; here, the algorithm looks for features and matches them using SURF (Speeded-up Robust Features). Then, it filters the features to avoid mismatch.

Once the pictures are correctly rectified, the algorithm utilizes a block matching algorithm to find the disparity map. Then, the algorithm matches every pixel in one image to the corresponding one in the other image.

From that information, the algorithm is able to triangulate the position of the points of the picture in the 3D space, and build the dense points cloud. The features obtained previously are also triangulated to build a corresponding sparse point cloud.

At this point, the algorithm filters the dense cloud to remove noise. Both clouds are then transformed from the camera system to the world system using the camera poses loaded at the start of the cycle. The clouds are then pushed in two different vectors, one for the dense cloud and one for the sparse cloud. The main cycle starts again and repeats until all the subsequent pairs are analyzed.

Lastly, an optional evaluation step begins. The algorithm analyzes the dense point cloud to calculate the average distance between subsequent dense clouds. The dense clouds should partly overlap one with another; therefore, their average distance should be low. We use this parameter to judge the point clouds registration, namely how well the point clouds combine one with another. Another parameter used to judge the quality of the dense reconstruction is the distance between a dense point cloud and the corresponding sparse point cloud. This evaluation step, however, is optional and time-consuming. Therefore it should be done only during verification.

Only at this point, the algorithm access the dense cloud vector and it outputs a 3D visualization of a single cloud or of all the clouds in the vector.

The algorithm is summed up in Figure 3.1.

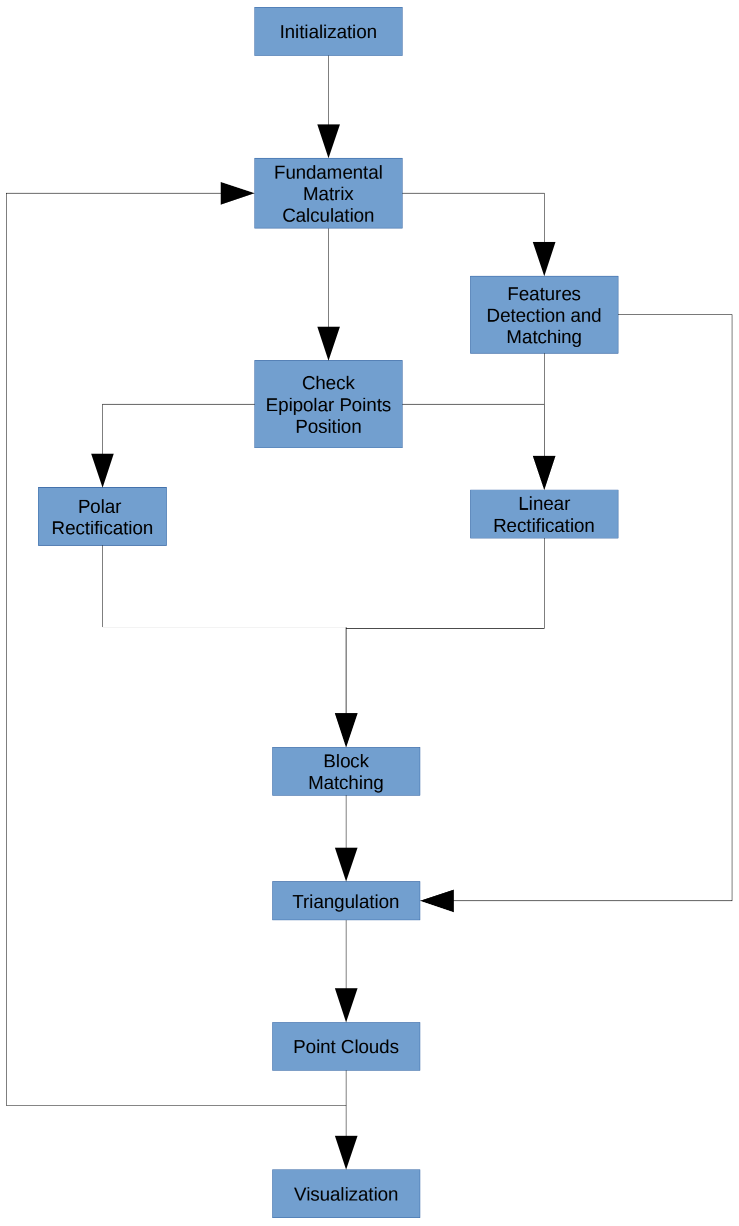


Figure 3.1: Simple diagram of the algorithm.

3.3 DM Algorithm:Fundamental Matrix

The first step of the main cycle is the calculation of the fundamental matrix.

The fundamental matrix, as mentioned before, is an algebraic representation of the epipolar geometry. For us, it is necessary to know this matrix due to the fact that block matching algorithms are written to work with a pair of pictures obtained with an ideal stereo set-up. In an ideal stereo set-up, an image pair has parallel, horizontal epipolar lines. Furthermore, correspondent epipolar lines in the two images align one another. Therefore, the block matching algorithm looks for a feature matching one on an image, only along the same row of the other image. Since the fundamental matrix encapsulates the epipolar geometry, it gives us the information necessary to transform the images in such a way that their resulting epipolar geometry matches the one of an ideal stereo pair.

We calculate the fundamental matrix with Equation 1.22, which requires as input the projective matrices of the images composing the pair. We calculate these matrices accessing the camera poses, the intrinsic parameters, and using Equation 1.13. From the fundamental matrix we compute the position of the epipolar points, and check if they are inside or near the image. We do this to evaluate witch rectification method the algorithm should use. This passage will be better clarified in the rectification section.

The fundamental matrix is extremely important, since the accuracy of the rectification process depends on it. Therefore, the more accurate is the calculation of the fundamental matrix, the more accurate the dense reconstruction will be.

3.4 Features

The DM algorithm requires matching features during rectification step, if the linear rectification method is used. Also, the algorithm compute the features to build a sparse point cloud to evaluate the DR. This point cloud contains fewer points than the dense counterpart. Nevertheless, the points it contains have a more accurate position. Thus, we can use it as a measure of the accuracy of the dense reconstruction.

Features detection and matching is an important part of many computer vision applications. Some common examples are camera calibration, motion estimation, and sparse reconstruction. Features are small patches of the image that distinguish themselves from their neighbors for shape, color, texture or intensity. There are different types of features detector; we can divide them into two categories, corner and blob:

- Corner detectors search the images for an intersection of edges or borders. They are fast, but they are not scaling invariant, nor affine invariant.
- Blob detectors use a more complex way to define a feature, which examines the texture, intensity or color of the patch. They are scaling invariant and affine invariant, but comparatively slower.

A feature detector should be chosen accordingly to the type of scene the camera is going to observe, eventual computational restraints, and motion of the vision system. Taking these parameters into account, we need to choose a feature detector that is accurate and have high feature repeatability, efficiency, robustness distinctiveness, and invariance to photometric and geometric changes. The algorithm works with a monocular set-up. Therefore, there can be huge variations of camera orientation and of the baseline from frame to frame. This leads to significant changes in the scale and perspective of the same features in different frames. Thus we use a blob detector, SURF by Bay, Tuytelaars and Van Gool [12].

3.4.1 Speed-up Robust Features (SURF)

SURF employs a fast Hessian detector to find features. It finds the Hessian matrix for any point **x** (x,y) in the image and for a given scale σ:

$$H(\boldsymbol{x},\sigma)=\begin{bmatrix} L_{xx}(\boldsymbol{x},\sigma) & L_{xy}(\boldsymbol{x},\sigma) \\ L_{yx}(\boldsymbol{x},\sigma) & L_{yy}(\boldsymbol{x},\sigma) \end{bmatrix} \tag{3.1}$$

Where $L_{ij}(\boldsymbol{x},\sigma)$ are the convolution of the Gaussian second order derivative on a point **x** in the image I: $L_{ij}(\boldsymbol{x},\sigma)=\partial^2 g(\sigma)/\partial i\partial j$. Then, SURF identifies a point of interest as a point where the determinant of the Hessian matrix is maximal.

The method is defined fast because it approximates all these values using a box filter and integral images which greatly speed up the process.
An integral image $I_{\sum}(\boldsymbol{x})$ for a point **x** is obtained by summing up all the values of the pixels in a square region, which is defined by the origin and the point **x**:

$$I_{\sum}(x,y)=\sum_{i=0}^{i=x}\sum_{j=0}^{j=y} I(x,y) \tag{3.2}$$

SURF filters the image starting with a box filter of size 9x9, which is an approximation of a Gaussian with σ 1.2. Then, it applies a box filter of bigger size to account for size variation of the features. Specifically, SURF scales the filter to 15x15 then 21 x21, 27x27 and so on. These scale variations correspond to different Gaussian scales. For example, a size of 27x27 is equivalent to a σ of 3*1,2.

Once the image is filtered, SURF applies a non-maximum suppression in a 3x3x3 neighborhood to find the points of interest in the image and interpolates the maximal of the Hessian matrix determinant in image space and scale.

At this point, SURF associates with every feature founded an orientation. This is done to achieve rotational invariance. SURF calculates the Haar-wavelet responses in the x and y directions in a circular neighborhood of the point. Then, it weights the responses with a Gaussian and it represents them as vectors with component dependent from the horizontal and vertical responses. From this information, it obtains an orientation of the feature.

Finally, SURF defines the descriptor of a feature as follows. Firstly, it extracts a square region centered on the point and oriented along the direction founded with the Haar-wavelet response. Secondly, it splits this region into 4x4 square sub-regions. Thirdly, for every sub-region, it calculates the Haar-wavelet responses in 5 equally distant sample points. Then, it weights them using a Gaussian. In the end, it sums up the weighted responses in both directions. Thus, for each region it generates the vector:

$$\mathbf{v} = \{ \sum d_x, \sum d_y, \sum |d_x|, \sum |d_y| \}$$

Therefore, SURF defines the final descriptor vector merging these vectors of 4 elements in a vector of 64 elements. Once SURF finds the features and their descriptor for a pair of pictures, we match features between the two images that have a similar descriptor.

3.4.2 Features Filtering

We filter the matched features to eliminate bad matches due to an erroneous association between features. In the Monocular Dense Reconstruction algorithm, we use two filters:

- Epipolar constraint filter
- Homography based filter

The epipolar constraint filter requires that inliers respect the epipolar constraint:

$$\boldsymbol{x}_2^{\boldsymbol{T}} F\, \boldsymbol{x}_1 = 0 \tag{3.3}$$

Due to unavoidable errors and noise in the measurement of the key points position, no points actually satisfies this constraint. For this reason, instead of requiring $\boldsymbol{x}_2^{\boldsymbol{T}} F\, \boldsymbol{x}_1$ to be null, we define a threshold and require for a given set of points that:

$$|\boldsymbol{x}_2^{\boldsymbol{T}} F\, \boldsymbol{x}_1| < \mathit{threshold} \tag{3.4}$$

The threshold is chosen accordingly with the dataset analyzed.

This filter only guarantees that matching points are going to lay on corresponding epipolar lines. A mismatch that lies on corresponding epipolar lines is thus not eliminated by this filter. To remove these mismatches, we utilize the homography filter.

Homographies are projective transformations that bring points from a 2D space to another set of points in the 2D space. In our case, the homography taken into account is used to transfer the key point from one image to another, and to check if they actually match in the new common space. This process helps us finding correspondences that matches for a pattern.

It must be noted that this homography is not actually applied to the image, but it is only used to check for inliners for the key points. This homography also differs from the rectification homography: in fact, this homography does not try to bring the epipolar point to infinity. Therefore, it can also be used when the epipolar point is inside the image, without generating infinity big images.

This filter was judged adequate for the sequences analyzed, because the images contain a great number of the same planes in different positions, mainly the walls of the corridor and the ground. Key points that do not lie on these planes or on other planes parallel to them will be filtered out, as will be seen in the outside sequences.

A homography can be found only using four points, as described in [13]. Therefore, we use an iterative method to estimate this homography: RANSAC. In this case, RANSAC operates following six consequent steps:

1. It selects four random key points points.
2. Then compute the homography from these points.
3. It checks for the number of inliners from the key points.
4. It repeats the process until all key points are sampled.
5. Then it keeps the most numerous inliers.
6. At the end, it calculates new homography from those inliers.

The inliers at every cycle are the point for which:

$$\| \mathbf{x}_1 - H\,\mathbf{x}_1 \| < threshold \tag{3.5}$$

3.5 Rectification

Once we find corresponding features, we pass the information to the rectification process. The goal of the rectification is to transform the images in the pair so that their epipolar geometry could transform and became comparable to an ideal stereo pair. We need to complete this step to accurately match every pixel in one image to the corresponding pixel in the other. The resulting rectified images should have the following epipolar proprieties:

- The epipolar points have to be at infinity.
- The epipolar lines have to be horizontal and parallel one another.
- Corresponding epipolar lines in the two rectified pictures have to align one another.

We present two possible methods of rectification: a linear one based on Hartley rectification [14], and the polar rectification by Polifileris [2].
Hartley rectification tries to find a 2D projective transformation, or homography, to rectify the pair. The polar rectification method, instead transforms the image pair using polar coordinates around their epipolar points. The algorithm chooses which rectification method to use checking the position of the epipolar points. If one of these points is far away from the image plane then we use the linear rectification. Otherwise, we use the polar rectification.

3.5.1 Linear Rectification Method

The linear rectification computes a pair of homography transformation to rectify the image pair loaded in the cycle. The homographies bring the images from their image plane to a common virtual rectified plane parallel to the baseline. This process requires three steps:

1. We find a pair of homographies H_{h1}, H_{h2} with the Hartley rectification method and apply the transformation to the images.
2. To compensate for eventual unwanted distortion, we compute a shearing transformation for both images: S_1, S_2. Then we apply S_1 and S_2 to the images transformed in step 1.
3. Finally, we center the newly found images applying a translation transformation T_1, T_2 along the epipolar lines.

The final homographies that map points from the original un-rectified planes to a virtual rectified plane are:

$$H_{f1} = T_1 S_1 H_{h1} \qquad H_{f2} = T_2 S_2 H_{h2} \tag{3.6}$$

We use linear rectification only when the epipolar geometry is almost ideal. Therefore, we use this rectification method only to compute small, but vital, improvement in the epipolar geometry.

3.5.1.1 Hartley Rectification

This method of rectification was invented by Richard I. Hartley [14] and is the cardinal part of the linear rectification process.

The first objective is to find a projective transformation able to render the epipolar lines of one of the images horizontal. Therefore, the algorithm computes a transformation that brings the epipolar point to infinity. Then, it rotates the epipolar lines around the epipolar point until they are horizontal.

Now we need a matching transformation for the other image. There is more than just one possible transformations at this point, and for this reason our solution must be constrained. As suggested by Hartley, the algorithm impose a minimization of the least-square distances between matching features.

Once the algorithm finds both projective transformations, it applies it to the images. However, it should be noted that Hartley's rectification may lead to some unwanted distortions along the horizontal axis (Figure 3.2).

Now, the rectified pair has matching horizontal epipolar lines. However, Hartley's rectification distorts the left image. The algorithm calculates the homography for the left picture with a minimization of the least-square distances between matching features. In order for this process to work, it requires that the features cover the majority of the images. The features must have a good uniform coverage in all the picture area. However, for some sequences, I was unable to find features with adequate coverage on the images.

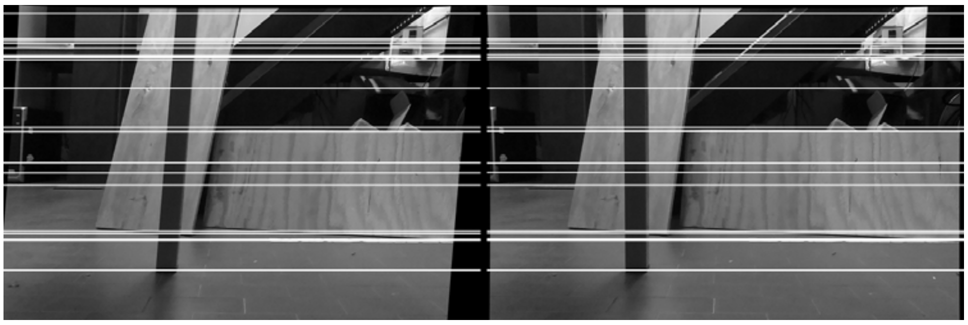


Figure 3.2: Images rectified only using Hartley's rectification

I tried relaxing the filter's parameters, to suppress the homography based filter and to increase the number of features detected by SURF relaxing its parameters. None of these operations improved the coverage of the features in the images, though. Therefore, I had to resort to another solution. This solution came from a paper by Loop and Zhang [15], where they propose a further projective transformation to correct distortions similar to the ones founded.

3.5.1.2 Loop and Zhang Sharing Transformation

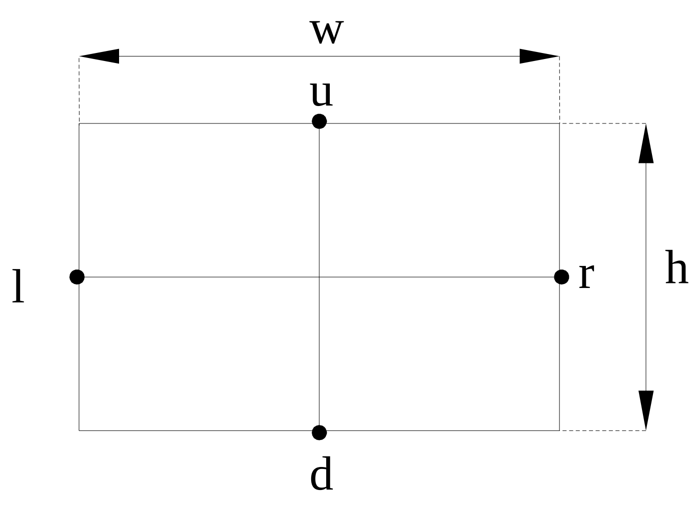
Loop and Zhang suggest to use a sharing transformation:

$$S = \begin{bmatrix} a & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

Let us consider the homogeneous coordinates of the middle points on the borders of the un-rectified images:

$$\boldsymbol{u} = [\frac{w-1}{2}, 0, 1]^T \quad \boldsymbol{r} = [w-1, \frac{h-1}{2}, 1]^T \quad \boldsymbol{d} = [\frac{w-1}{2}, h-1, 1]^T \quad \boldsymbol{l} = [0, \frac{h-1}{2}, 1]^T$$

Where h and w are the height and width of the image in pixels. Thus, the aim of the sharing transformation S is to maintain the aspect ratio and perpendicularity of the lines $\overline{\boldsymbol{lr}}$ and $\overline{\boldsymbol{ud}}$ between transformations.



The algorithm transforms **u**, **r**, **d** and **l** with the Hartley homography H_h :

$$\hat{\boldsymbol{u}}=H_h\boldsymbol{u} \qquad \hat{\boldsymbol{r}}=H_h\boldsymbol{r} \qquad \hat{\boldsymbol{d}}=H_h\boldsymbol{d} \qquad \hat{\boldsymbol{l}}=H_h\boldsymbol{l} \tag{3.8}$$

From this, the algorithm finds **x** and **y**:

$$\boldsymbol{x}(x_u, x_v, 0)=\hat{\boldsymbol{l}}-\hat{\boldsymbol{r}} \qquad \boldsymbol{y}(y_u, y_v, 0)=\hat{\boldsymbol{d}}-\hat{\boldsymbol{u}} \tag{3.9}$$

Then, the transformation S preserve perpendicularity when:

$$(S\boldsymbol{x})^T(S\boldsymbol{y})=0 \tag{3.10}$$

And the aspect ratio when:

$$\frac{(S\boldsymbol{x})^T(S\boldsymbol{x})}{(S\boldsymbol{y})^T(S\boldsymbol{y})}=\frac{w}{h} \tag{3.11}$$

The algorithm then solves this quadratic polynomial up to sign for a and b:

$$a=\frac{h^2 x_v^2+w^2 y_v^2}{hw(x_v y_u - x_u y_v)} \qquad b=\frac{h^2 x_v x_u+w^2 y_v y_u}{hw(x_u y_v - x_v y_u)} \tag{3.12}$$

We prefer a solution with a positive *a*. Therefore, if a is negative then the algorithm inverts the signs of *a* and *b*.

3.5.1.3 Centering The Pictures

The algorithm transforms the images with the following homography: $H_p = S H_h$. Then, for every point in the left border of the original image, it finds the corresponding point in the transformed image. The leftmost of these points should have a horizontal pixel coordinate of 0. However, this is not always the case, as we show in Figure 3.3.

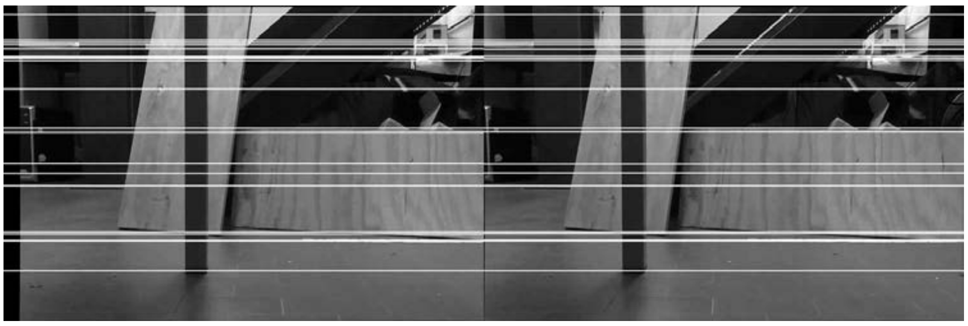


Figure 3.3: Images rectified at the second step of linear rectification.

To account for this discrepancy we need to correctly position the images in the rectified virtual plane. Let Δ be the coordinate of the leftmost point. So, we need to translate the image horizontally of a quantity Δ. The algorithm finds a translation transformation T:

$$T=\begin{bmatrix}1 & 0 & -\Delta\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{3.13}$$

The algorithm finds T for both images, and then it transforms them.

A similar process can be done for a vertical translation, but in this case, the transformation must be the same for both pictures in order to preserve the epipolar geometry.

3.5.2 Polar Rectification Method

The linear rectification method fails when the epipolar points are near or inside one of the image plane. In these conditions, the virtual plane and the image planes forms near perpendicular angles and the resulting image have huge distortion and is infinitely large.

When the epipolar points are near the images, we use a different approach: the polar rectification [2]. This type of rectification is able to find a rectified couple for any type of motion. Thus, it is the ideal form of rectification for a monocular set-up.

We can divide the polar rectification process in two main operative steps:

1. We determinate the common region.
2. We sample the area in the common region, and remap the points inside this area in the rectified space.

Before we explain how this method works, we need to introduce two new concepts related to the epipolar geometry:

- Epipolar line transfer.
- Epipolar line orientation.

Epipolar line transfer:

Epipolar line transfer states that it exists a homography able to map epipolar lines on one image to the corresponding ones in the other image [2]:

$$\boldsymbol{l}_2 \sim H^{-T}\boldsymbol{l}_1 \qquad \text{or} \qquad \boldsymbol{l}_1 \sim H^{T}\boldsymbol{l}_2 \tag{3.14}$$

We define H using the fundamental matrix:

$$H=[\boldsymbol{e}_2]_\wedge F+\boldsymbol{e}_2\boldsymbol{a}^T \tag{3.15}$$

Where **a** is an arbitrary vector for which the determinant of *H* is not null so that *H* is invertible.

Epipolar line orientation:

Epipolar line orientation reduces the matching ambiguity to half the epipolar line when the epipolar point is inside the image. Points located on the right side of an epipolar plane in one image should still be on the right side of that epipolar plane in the other image. We can guarantee this by imposing that the homography calculated in the previous step maintain the orientation when transferring the epipolar lines.
Let us consider a couple of matching points in the two images. The product between these two points and the epipolar lines define the following factors:

$$\begin{aligned} f_1(\boldsymbol{x}_1)&=\boldsymbol{l}_1\boldsymbol{x}_1\\ f_2(\boldsymbol{x}_2)&=\boldsymbol{l}_2\boldsymbol{x}_2 \end{aligned} \tag{3.16}$$

Where we obtain $\boldsymbol{l}_2$ with Equation 3.14. Then if the two factors shares the same sign the homography already preserves the orientation of the epipolar line, otherwise we need to change the sign of *H*.

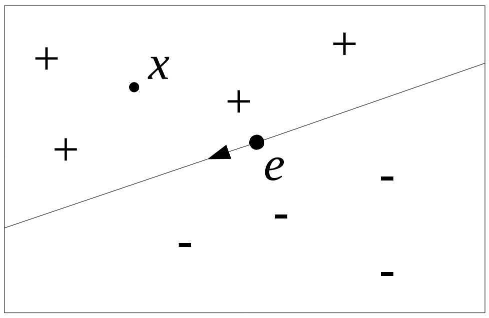


Figure 3.4: Convection for the epipolar line direction

The convection for the direction of the epipolar line is that it is positive if the matching points are on the right side of the vector. This side of the image is also called the positive side.

3.5.2.1 Determining The Common Region

Firstly, we determine the corner lines for both images. These are the lines that pass through the epipolar point, and one of four corners of the image. So, using homogeneous coordinates, if **c** is a corner in the image, then the corresponding corner line would be:

$$\boldsymbol{l}_c = \boldsymbol{e} \wedge \boldsymbol{c} \tag{3.17}$$

We call corner lines that do not pass through the images, but only touch the corners, extremal lines. Therefore, for every image, there are two extremal lines.

Once we found the extremal lines of the first image, we transfer them from one image plane to the other. We do this using the epipolar line transfer. If one of the transferred lines pass through the image, it defines one limit of the common region. We repeat the same procedure from the second image to the first. Thus, we find the two lines delimiting the common region. An extremal line whose transformed line does not touch the other image at any point can never be chosen as a limit line.

Figure 3.5 represents the three cases that can be encountered determining the common region when both epipolar points are outside the image.

If one of the epipolar points is inside the image, then we define the common region only with the extremal line of the other image. If both epipolar points are inside, then the common region is all the images.

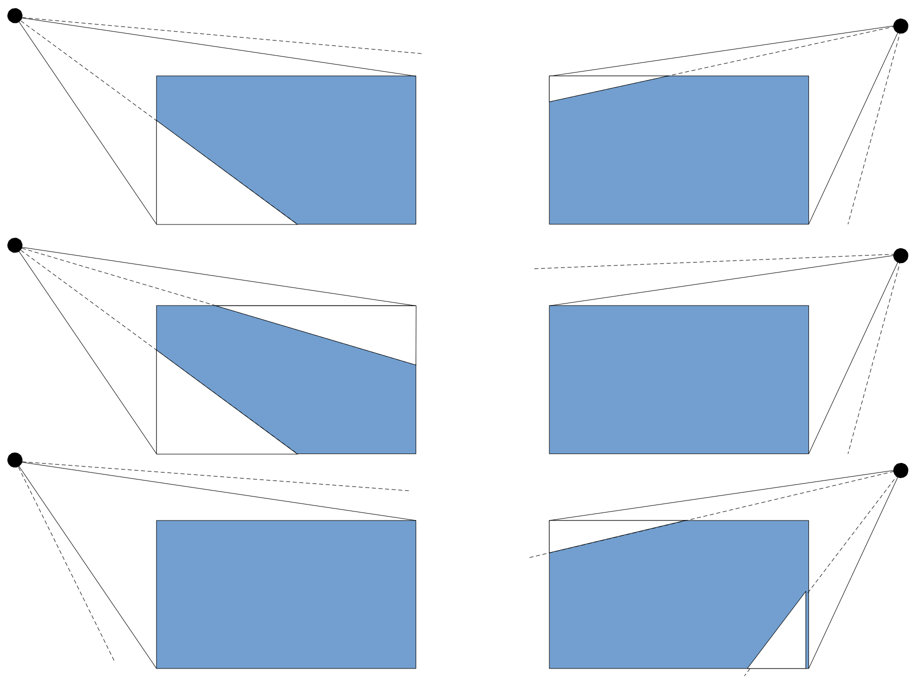


Figure 3.5: The three cases that can be encountered when determining the common region for images with epipolar points outside them. The dashed lines are the re-projection of the extremal lines from the other image in the pair. The colored area is the common region.

3.5.2.2 Sweeping and Resampling

To build the rectified pictures, we sweep the images rotating the half epipolar lines around the epipolar point. From the upper limit epipolar line to the other we sample and remap the area between subsequent half epipolar lines to the rectified space. The distance between two subsequent half epipolar lines must be at maximum equal to 1 pixel, in order to avoid information loss.

At the start of the process, we identify the upper limit line of the first image as $\boldsymbol{l}_1^1$ and its projection in the second image $\boldsymbol{l}_1^2$. Therefore, the rectification for the first row of the rectified image results as follow:

1. We find the intersections between $\boldsymbol{l}_1^1$ and the image border of the first image. Let $\boldsymbol{a}_1^1$ be the point of intersection nearest to the epipolar point $\boldsymbol{e}_1$, and $\boldsymbol{b}_1^1$ be the other.

2. Similarly, we found the points $\boldsymbol{a}_1^2$ and $\boldsymbol{b}_1^2$ on the second image.
3. We search for the next half epipolar line on the two images $\boldsymbol{l}_c^i$ as the half epipolar line with a maximum distance of 1 pixel from $\boldsymbol{b}_1^i$. Where i = 1, 2.
4. We project $\boldsymbol{l}_c^2$ the first image plane, let us call this newfound line $\boldsymbol{l}_c^{2,1}$.
5. We chose the line among $\boldsymbol{l}_c^1$ and $\boldsymbol{l}_c^{2,1}$ that is closer to $\boldsymbol{b}_1^1$ as the next epipolar line and we call it $\boldsymbol{l}_2^1$.
6. For every pixel in the area between $\boldsymbol{l}_1^1$ and $\boldsymbol{l}_2^1$, we calculate its distance from $\boldsymbol{a}_1^1$ along the epipolar line $\boldsymbol{l}_1^1$.
7. That distance, in the end, results to be the horizontal coordinate of the pixel on the rectified row of the rectified image.

Then we repeat this process with a new $\boldsymbol{l}_1^1 = \boldsymbol{l}_2^1$. The rectification ends when all the common region is resampled in the rectified image.

Once the rectification process is completed we pass the rectified images to the block matching algorithm, which in return give us the disparity map.

3.6 Disparity Calculation

In dense reconstruction, we want to match every pixel of one image with the corresponding pixel in the other image. In order to do this, we use a block matching algorithm. This algorithm computes the distance along the epipolar lines between matching pixels, otherwise noted as disparity.

In order to do this the block matching algorithm requires as inputs two rectified images: one is regarded as the left picture of a stereo pair, the other as the right picture. Starting from the first row and column of the left image the algorithm considers a portion of the left picture, which is determined by the block size. Then, it looks for a match on the same row of the corresponding right picture. At this point, the algorithm chooses the best matching block on the right image by minimization of a cost function. This cost function depends on the intensity and the entropy of the pixels inside the block.

Once a matching block is found, the algorithm calculates the disparity pixel by pixel and then moves the block left. Once all the pixels in a row are matched the block is moved to the following column. At this point, the algorithm just repeats itself until all the pixels in the picture are scanned. The disparity information is saved in a picture, called disparity map, where at every pixel of the leftmost image is assign the value of the disparity.

A disparity map is a useful tool. We use it to check the noise level of the disparity to better chose the parameters of the stereo blocking algorithm. Then, we use the resulting disparity to find corresponding points in the pair and triangulate their position in the 3D space.

3.7 DM Algorithm: Triangulation

Triangulation requires as input a set of matching points in the image plane.

In our case, the points are matched using the disparity map that is defined in the rectified virtual plane. Therefore, points matched with the disparity map must be transferred from the rectified plane to the image plane.

For any point **u** (u,v) in pixel coordinates on the disparity map we find the matching points:

$$\boldsymbol{u_1}(u,v) \qquad \text{and} \qquad \boldsymbol{u_2}(u+d,v) \tag{3.18}$$

Where d is the value of the disparity on the point **u**, $\boldsymbol{u_1}$ is the point in the first rectified image and $\boldsymbol{u_2}$ is the corresponding point in the second rectified image. We then transfer $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ from the rectified plane to the image plane inverting the rectification process on those points. Therefore, we obtain the points $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ on the image plane:

$$\boldsymbol{u_1} \rightarrow \boldsymbol{x_1} \qquad \text{and} \qquad \boldsymbol{u_2} \rightarrow \boldsymbol{x_2} \tag{3.19}$$

We then check if $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ lie on their image plane. Also, in case we rectified the image with the polar rectification we investigate if $\boldsymbol{x_1}$ or $\boldsymbol{x_2}$ is in a neighborhood of the epipolar points. If this is the case, we reject the point pair: this because near the epipolar points the rectified images are severely distorted [2][3]. After all the valid points pairs are found in the image planes the triangulation process can begin.

In not ideal conditions errors in the measured points make the ray back-projecting from the image points skewed, so they do not actually intersect. In other words, the two points can not actually satisfy the epipolar constraint.

3.7.1 Linear Triangulation

Firstly the scale factor is eliminated from $\lambda_1 \boldsymbol{x_1} = Pj_1 \boldsymbol{X}$ using the cross product: $\boldsymbol{x_1} \wedge Pj_1 \boldsymbol{X}$. This product is null since $\boldsymbol{x_1}$ and $Pj_1 \boldsymbol{X}$ are parallel. Therefore, we obtain the following equations linear in **X**:

$$\begin{gathered} x_{1,x}(Pj_{1,3}\boldsymbol{X}) - Pj_{1,1}\boldsymbol{X} = 0 \\ x_{1,y}(Pj_{1,3}\boldsymbol{X}) - Pj_{1,2}\boldsymbol{X} = 0 \\ x_{1,x}(Pj_{1,2}\boldsymbol{X}) - x_{1,y}(Pj_{1,2}\boldsymbol{X}) = 0 \end{gathered} \tag{3.20}$$

where $Pj_{1,i}$ is the i -th row of Pj_1 and $\boldsymbol{x_1}(x_{1,x}, x_{1,y}, 1)$.

We do the same also for $\lambda_2 \boldsymbol{x_2} = Pj_2 \boldsymbol{X}$:

$$\begin{gathered} x_{2,x}(Pj_{2,3}\boldsymbol{X}) - Pj_{2,1}\boldsymbol{X} = 0 \\ x_{2,y}(Pj_{2,3}\boldsymbol{X}) - Pj_{2,2}\boldsymbol{X} = 0 \\ x_{2,x}(Pj_{2,2}\boldsymbol{X}) - x_{2,y}(Pj_{2,2}\boldsymbol{X}) = 0 \end{gathered} \tag{3.21}$$

Then the first two Equation of 3.20 and 3.21 are used to combine $\lambda_1 \boldsymbol{x_1} = Pj_1 \boldsymbol{X}$ and $\lambda_2 \boldsymbol{x_2} = Pj_2 \boldsymbol{X}$ in a single linear equation A**X** = 0:

$$A\,\boldsymbol{X} = \begin{bmatrix} x_{1,x}(Pj_{1,3}) - Pj_{1,1} \\ x_{1,y}(Pj_{1,3}) - Pj_{1,2} \\ x_{2,x}(Pj_{2,3}) - Pj_{2,1} \\ x_{2,y}(Pj_{2,3}) - Pj_{2,2} \end{bmatrix} \boldsymbol{X} = 0 \tag{3.22}$$

However, this linear equations system is not exactly solved by our points. The position of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ is not accurate due to unavoidable noise. Therefore, we find an approximate solution by requiring a minimization of the norm ||A**X** ||. The easiest way to do so is to use the singular value decomposition [16] $A = UDV^T$. Where *U* and *V* are orthogonal matrices, and *D* is a diagonal matrix with non negative values. Then **X** is equal to the last column of *V*. At this point, we check if **X** is inside the field of view of the camera if: this is the case we push the point **X** inside its point cloud.

3.7.2 Ideal Set-Up

In an ideal set up the rectified space and virtual space are the same. Therefore, we can easily triangulate the points using Equations 1.29 and the disparity information:

$$X_p = b\frac{u_1 - c_x}{d}$$

$$Y_p = b\frac{k_x}{k_y}\frac{v_1 - c_y}{d} \tag{3.23}$$

$$Z_p = b\,k_x\frac{f}{d}$$

Where *d* is the disparity value on the point $\boldsymbol{u}(u_{1,}v_1)$ in pixel coordinates of the disparity map.

3.8 Point Cloud

A point cloud is a set of points in the 3D space. They usually are the result of a LiDAR, a Time of Flight camera or a 3D range sensor. In our case, we generate the point cloud using the 3D coordinates obtained during the triangulation step. We create two clouds at every main cycle: a dense point cloud, and a sparse point cloud.

We build the dense point cloud using the 3D coordinates of every pixel of the image pair. Initially, the dense point cloud is composed of the points triangulated using the disparity map. We assign a color to every point dependent from its corresponding position on the image plane of an RGB version of the first image. Then, we build the sparse point cloud using the 3D coordinates of the key points obtained during the feature detection and matching. At this point, we filter the dense point cloud to remove noises with a radius outlier filter. Before pushing the two clouds inside their vectors, we transform them from the camera system to the world system.

3.8.1 Radius Outlier Filter

The radius outlier filter is a very simple filter that let us eliminate isolated points that generate noise. For every point **P** in the point cloud, the radius outlier filter searches for points in a spherical neighborhood defined by the radius *r*. If the number of points inside the sphere is less than a defined threshold, N, we consider the point **P** as an outlier.

Essentially, the radius outlier filter requires the points in a cloud have a minimum density:

$$D_{min}=\frac{N}{4\pi r^2/3} \tag{3.24}$$

The parameters of the filter must be adjusted accordingly with the dataset considered.

Chapter 4

Results

In this chapter, we present the results of the Monocular Dense Reconstruction algorithm. We have elaborated five sequences, that we have divided into two groups. We decided to divide the results in two parts, because every part has a series of sequences that have similar characteristics. Furthermore, the results differentiate themselves for how the poses were estimated and how their rectification and triangulation were computed.

In the first set of sequences, the camera was mounted on a linear slide, and moved along it with a fixed displacement. These two sequences correspond to the indoor and the outdoor non-planar sequence that we mentioned in Chapter 3. Then, the camera poses are retrieved via the hybrid algorithm of Chiodini et al. The optical axis of the camera and the baseline are always semi-perpendicular. Therefore, we used linear rectification and ideal triangulation to obtain the dense reconstruction.

The second set is formed by three sequences. In each of these sequences, we mounted the camera with a different orientation, compared to the direction of motion of the rover. Then, the rover moved along a corridor and took pictures at fixed time intervals. The movement of the rover was not constant: therefore, the virtual baseline from one camera pose to the subsequent one is variable. The poses are obtained by a LiDAR mounted on the rover. Thus, the pose estimation is especially accurate. In this set of sequences, we computed the rectification through the polar method, and we constructed the point clouds using the coordinates we obtained via general triangulation.

We structured this chapter as follow: firstly, we present and discuss the results of the first set of sequences in a singular section; then, we report and discuss the results of the second set, divided by the orientation of the camera into different sections.

Unless otherwise noted, the numerical results and the point cloud figures in this chapter refers to clouds down-sampled to 10% of their actual size. This was done to improve the data handling and the computational times of the algorithm.

The point clouds density are calculated as the average number of points in a spherical volume of radius 3 cm form every point of the cloud.

We evaluated the distance between the sparse features cloud and the dense cloud as the average of the distances between every point of the dense cloud to the nearest point of the features cloud.

The minimum distance between the sparse LiDAR cloud and the dense cloud was evaluated with the minimum of the average distances between a point of the dense cloud and the ten closest points on the LiDAR cloud. The LiDAR produces a sparse planar cloud on the XY plane. Therefore, we evaluated the average distance from the LiDAR as the distance on that plane from every point in the dense cloud to the nearest point of the LiDAR cloud.

4.1 Results: Linear Rectification

The sequences we present in this section were the first we analyzed. We used them to test the functionality of the algorithm in an easy to solve the condition, namely semi-perpendicularity of the camera optical axis and the virtual baseline.

The camera parameters are summed in following camera matrix and distortion vector:

$$K=\begin{bmatrix} k_x f & k_s f & c_x \\ 0 & k_y f & c_y \\ 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} 1270.6 & -2.4 & 592.5 \\ 0 & 1257.9 & 474.3 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad dist=\begin{bmatrix} k_1 \\ k_2 \\ p_1 \\ p_2 \end{bmatrix}=\begin{bmatrix} 0.0811 \\ -0.2884 \\ 0.0061 \\ 0.0030 \end{bmatrix}$$

The camera poses we used to analyze these sequences were evaluated by the monocular VO algorithm of Chiodini et al. Both sequences were taken with the same camera while it was moving horizontally with a semi-fixed direction of the optical axis. Moreover, the system captured the pictures at fixed intervals of motion, around 0.05 m. Therefore, the camera poses between every pair are highly comparable to an ideal stereo pair. Thus, we treated these cases as an ideal stereo set-up: we rectified the pairs only using linear rectification, and we triangulated the points with the ideal triangulation method.

For these sequences, the camera axis and the baseline are almost perpendicular for every pair. Thus, we expect a fundamental matrix comparable to an ideal stereo pair. The fundamental matrix calculated by the algorithm matches the theoretical one:

$$F_{ideal}=\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{bmatrix} \quad F_{code}=\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{bmatrix}$$

The first sequence is composed by a series of images taken in an office, the second to a sequence of images taken outside. We will refer to them as Office sequence, and Outside sequence.

Robust and repeatable features detection and matching is not the main goal of the algorithm. However, it is still an important part of it. Thus, it is relevant to show the results the algorithm obtains at this step. The algorithm tries to provide accurate feature estimation without compromising their coverage on the images. As we show in the Figures from 4.1 to 4.9, the filters were able to remove the majority of the mismatches. Furthermore, the homography filter detects and remove mismatches undetected by the epipolar constraint filter (Figure 4.5).

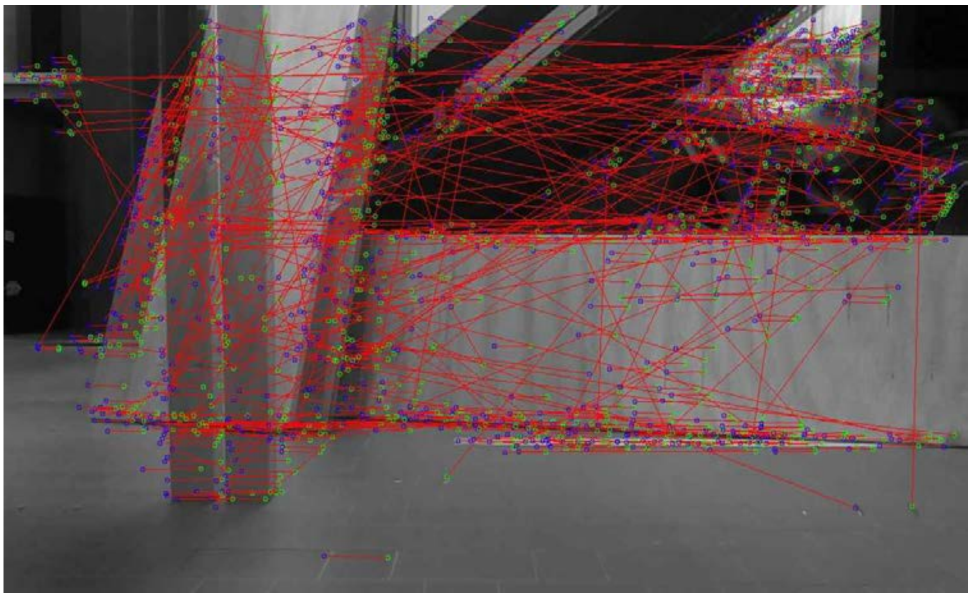


Figure 4.1: Unfiltered matches for a frame of the first sequence.

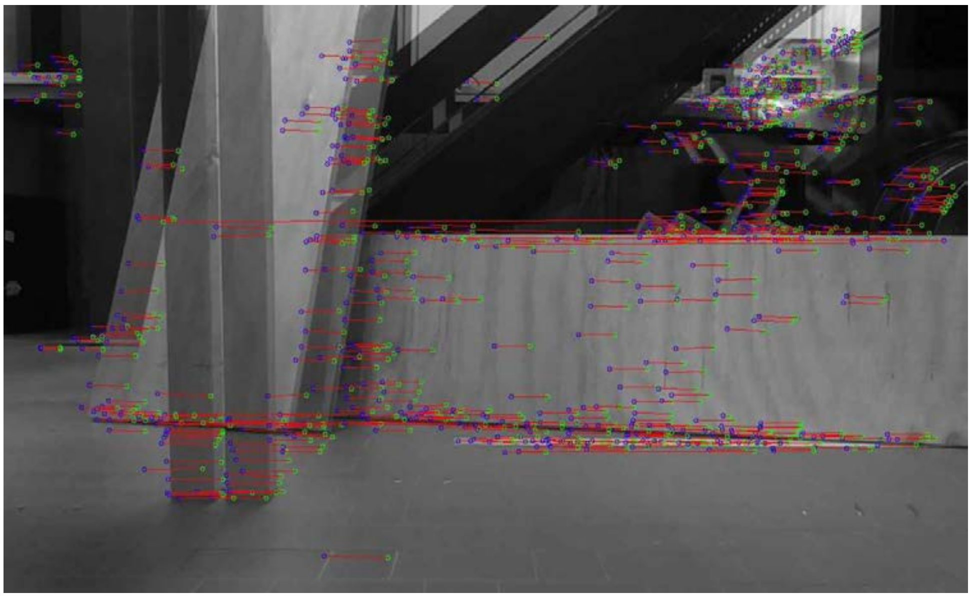


Figure 4.2: Matches filtered only with the epipolar constraint filter for a frame of the office sequence.

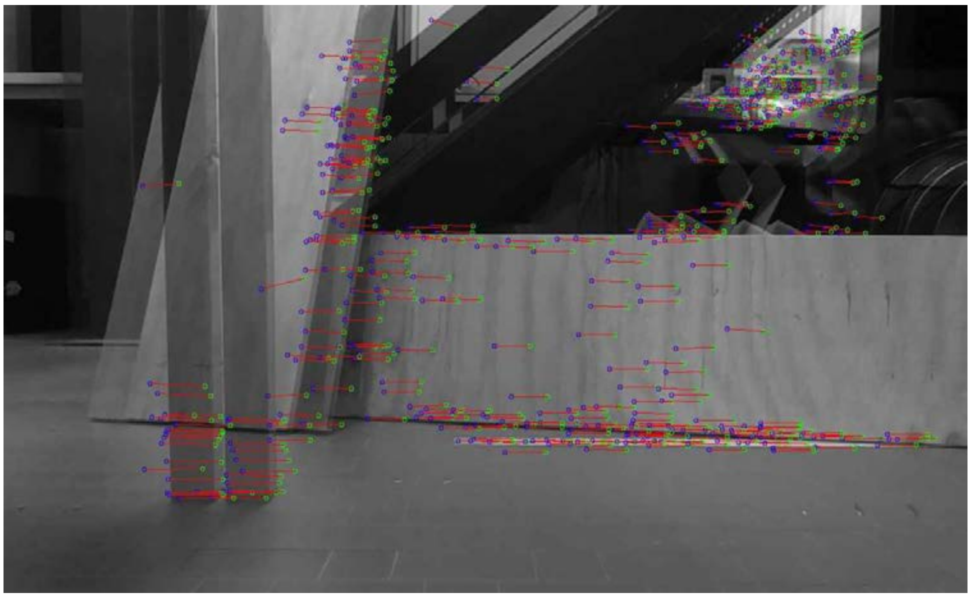


Figure 4.3: Matches filtered only with the homography filter for a frame of the office sequence.

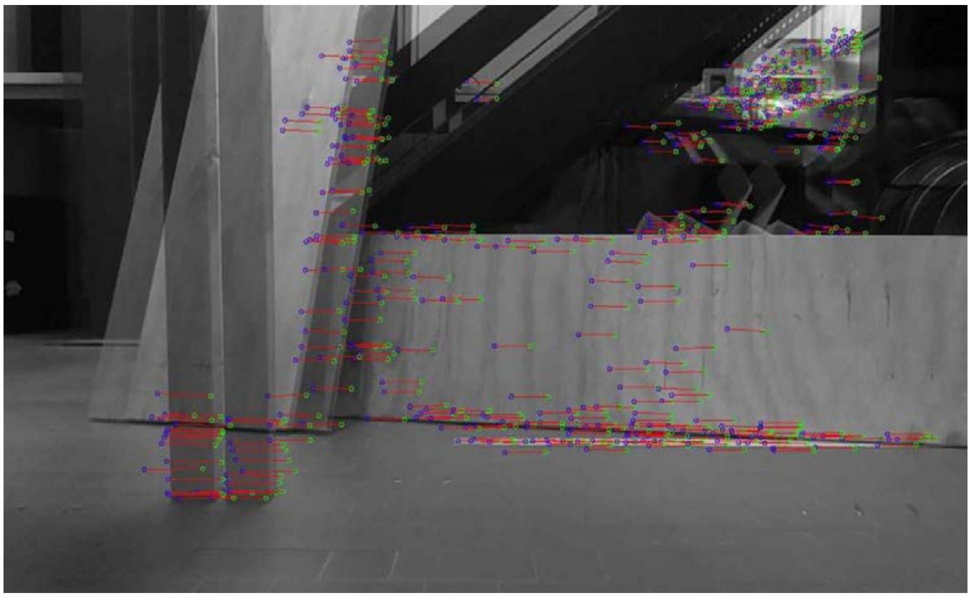


Figure 4.4: Matches filtered with both filters for a frame of the first sequence.

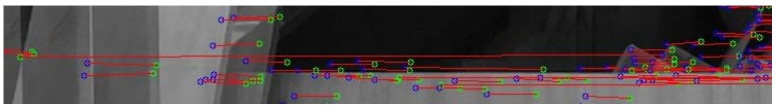


Figure 4.5: Detail showing a mismatch that passed through the epipolar constraint filter.

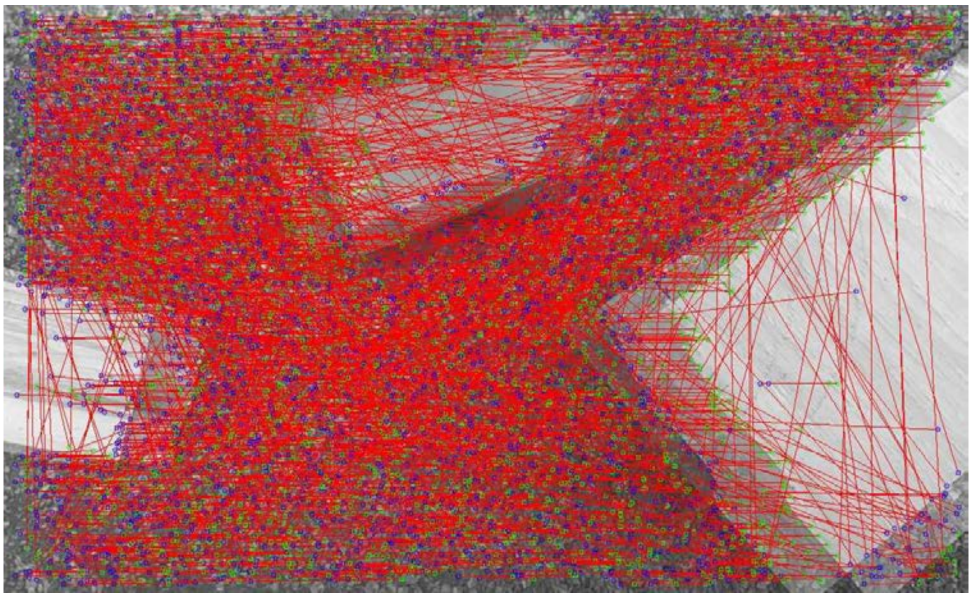


Figure 4.6: Unfiltered matches for a frame of the second sequence

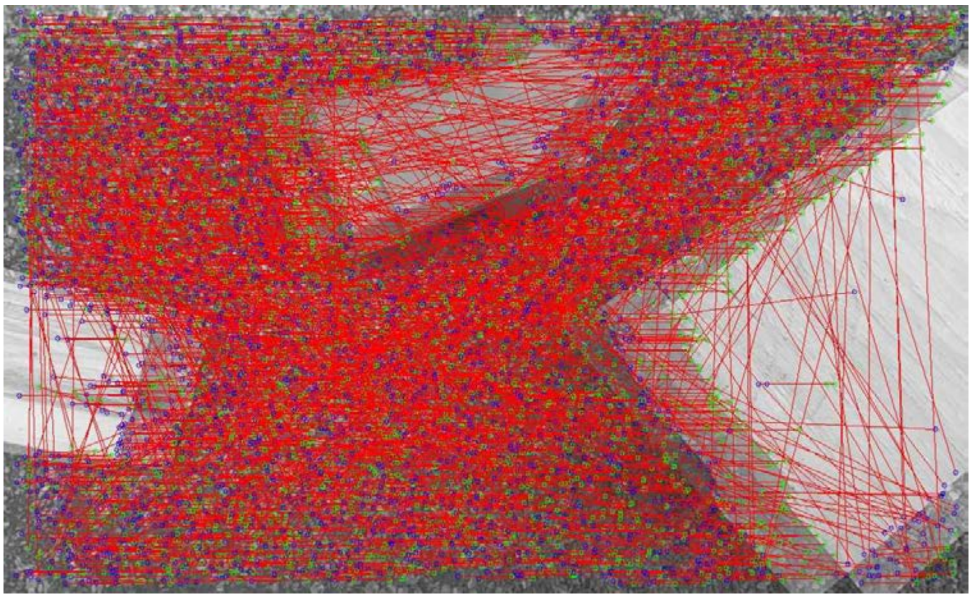


Figure 4.7: Matches filtered only with the homography filter for a frame of the second sequence

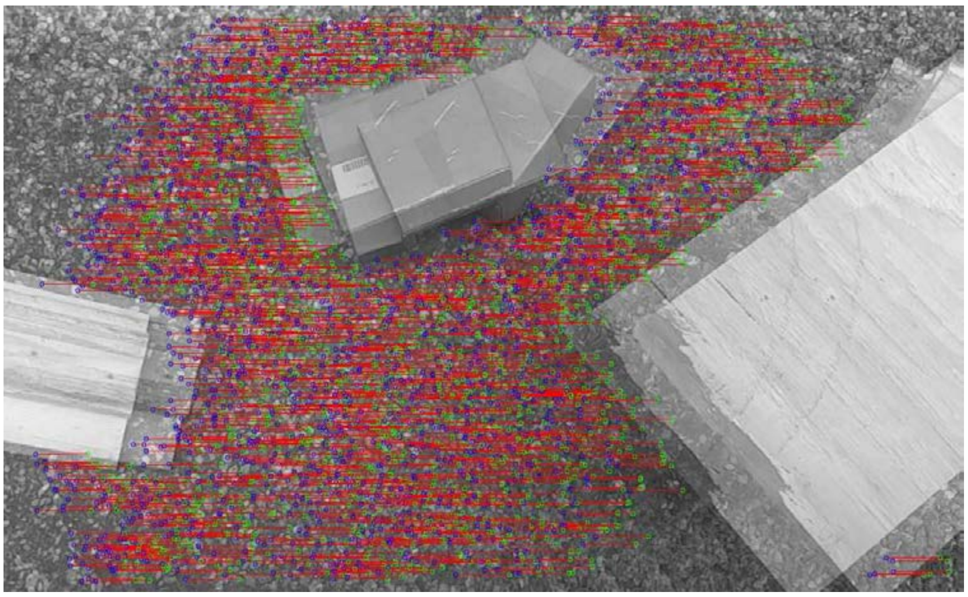


Figure 4.8: Matches on the outside sequence filtered only with the homography based method.

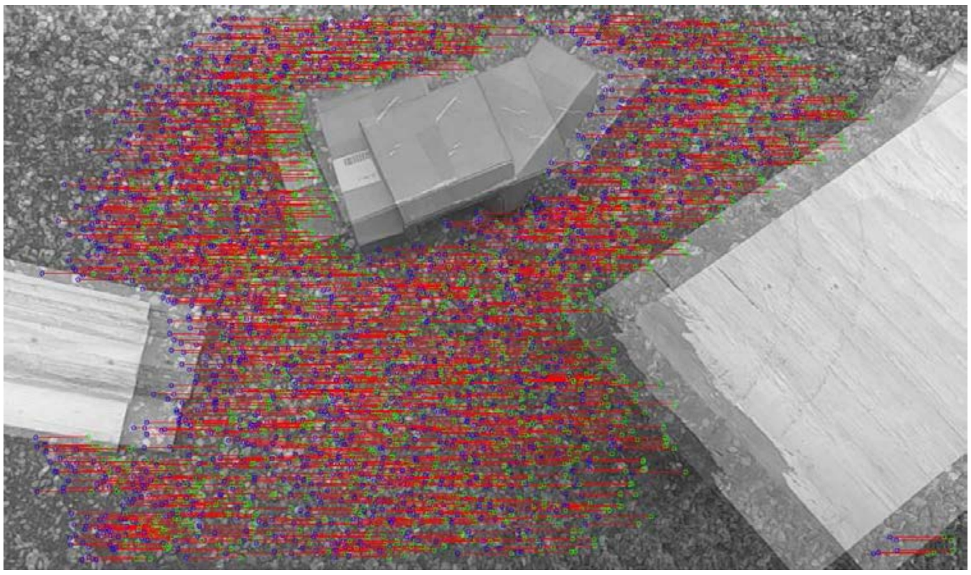


Figure 4.9: Matches filtered with both filters for a frame of the second sequence.

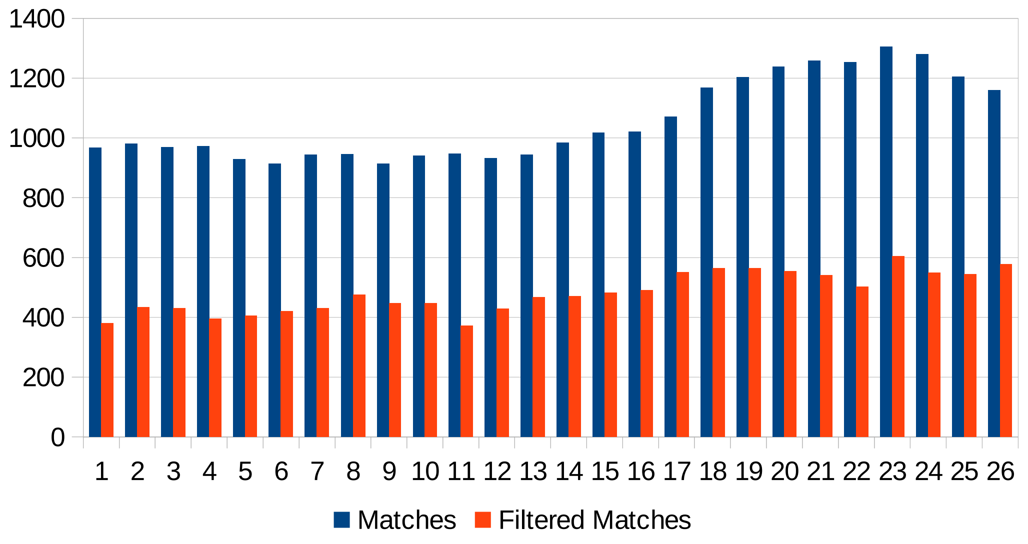


Figure 4.10: Histogram with the number of matched feature in the office sequence before and after filtering.

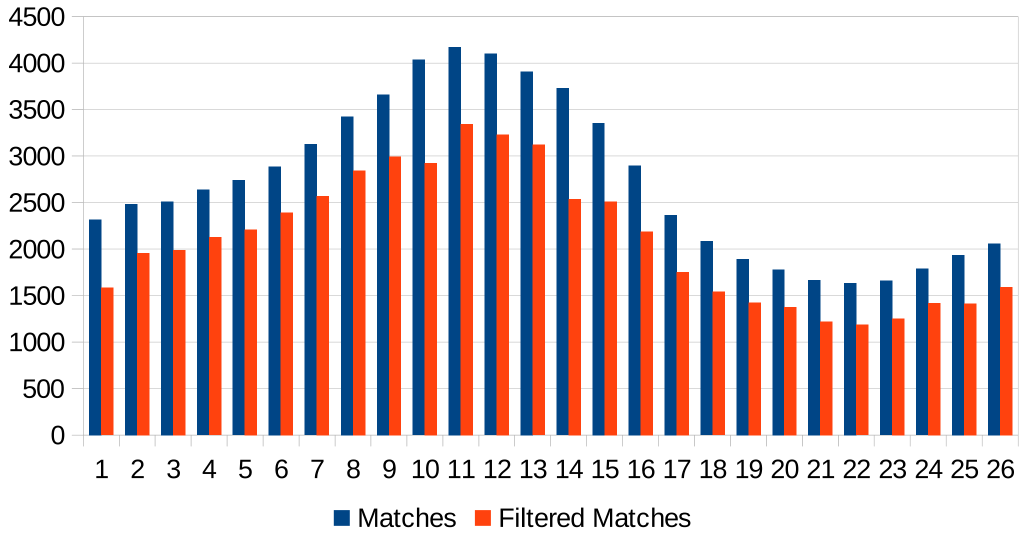


Figure 4.11: Histogram with the number of matched feature in the outside sequence before and after filtering.

From Figure 4.10, we can see that the number of features in the Office sequence is almost the same at different frames. The Outside sequence is extremely rich in features as we show in Figures from 4.7 to 4.9. However, the ground features are too dominant. Therefore, the homography filter removes correct matches that do not lay on the ground plane. However, this does not negatively impact the rectification process. The features are still numerous, and they cover the majority of the pictures.

As we show in Figure 4.11, the number of features in the second dataset increases, with a peek at the eleventh frame. From the starting frame to the eleventh frame the ground becomes more and more prominent in the images (Figures 4.12 and 4.13). Therefore, due to the homography filter, the number of features increases accordingly.

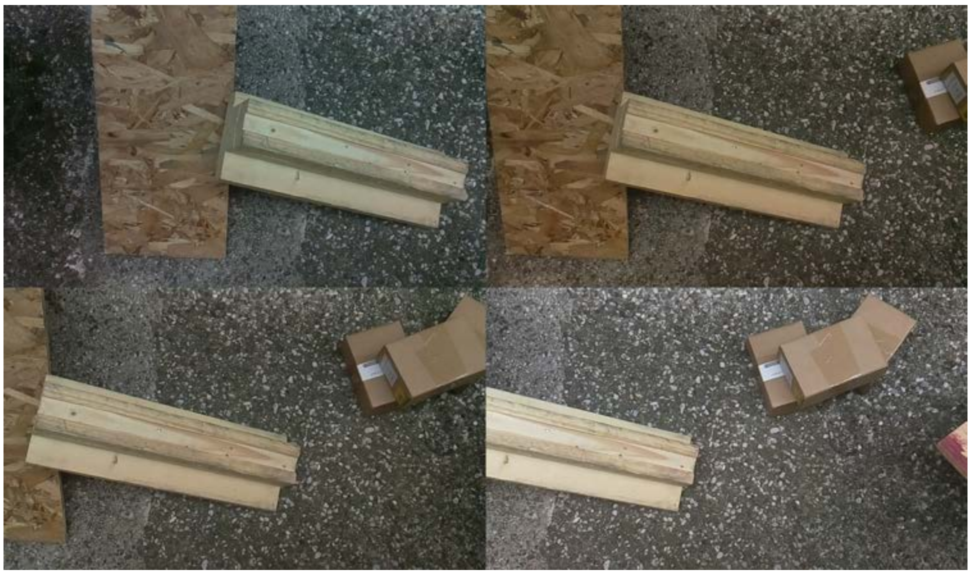


Figure 4.12: Frames 1, 3 , 6 and 9 of the outside sequence. The ground becomes more and more prominent.

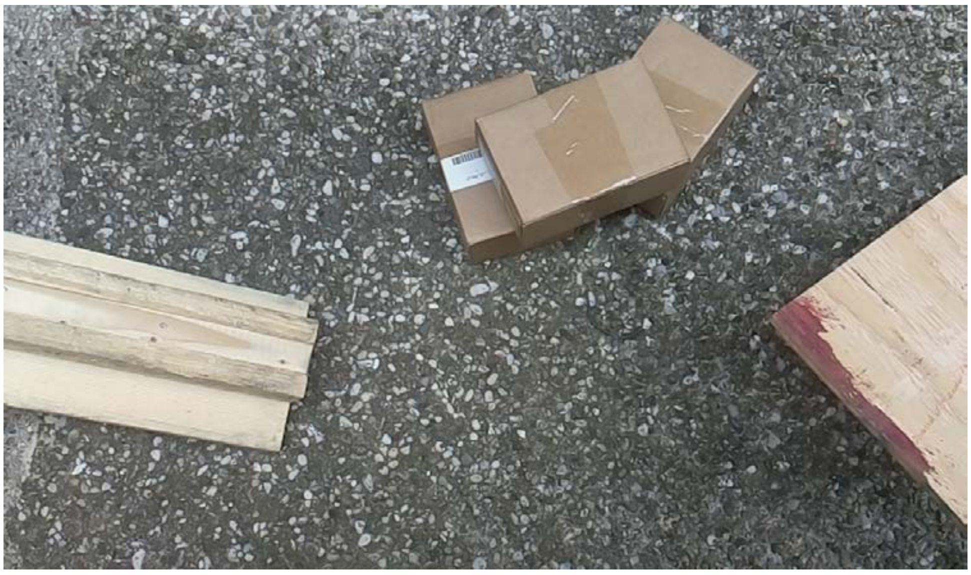


Figure 4.13: Frame 11 of the outside sequence.

Even if well matched, the features on the first dataset are not well distributed in the images. Thus, the Hartley homography the algorithm calculates at the first step of the linear rectification is not well estimated. However, the corrections we implemented in the algorithm at step two and three of the linear rectification are sufficient to correct the homography.
Figures from 4.14 to 4.16 show some images of the first sequence at different steps of the linear rectification. The resulting rectified images are undistorted, and their epipolar geometry matches the ideal case.

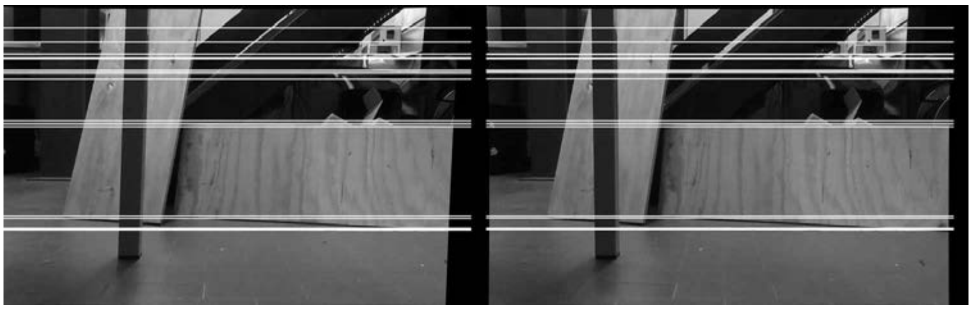


Figure 4.14: Images of the office sequence with epipolar lines, rectified via Hartley without corrections.

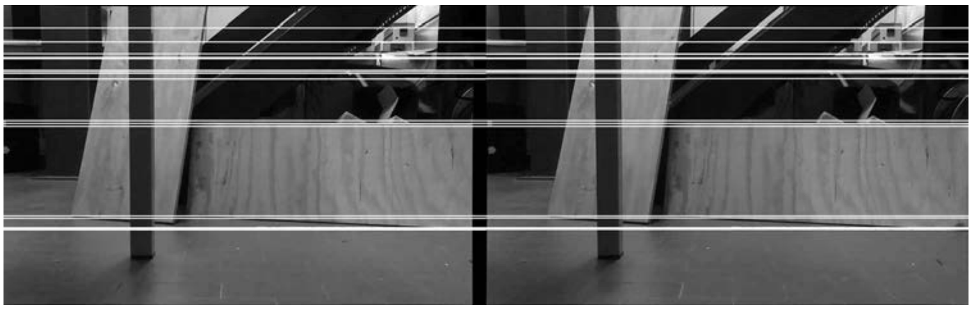


Figure 4.15: Images of the office sequence with epipolar lines, rectified via Hartley corrected with Loop and Zhang method

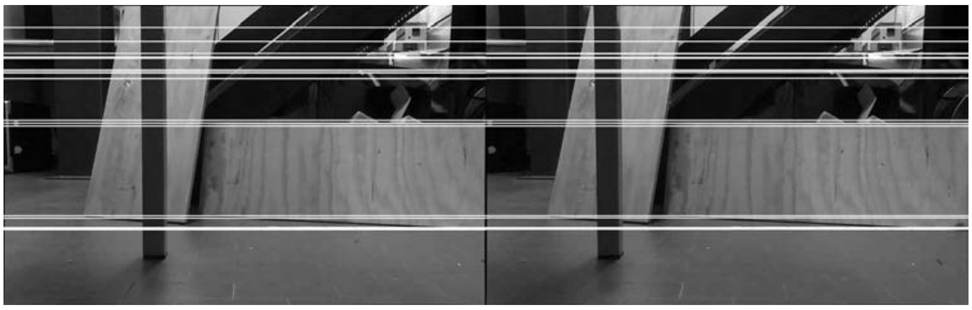


Figure 4.16: Images of the office sequence with epipolar lines, rectified and correctly centered.

The second sequence suffers less from homography distortion. The algorithm is easily able to compensate them, and to correctly center the images without compromising the epipolar geometry.

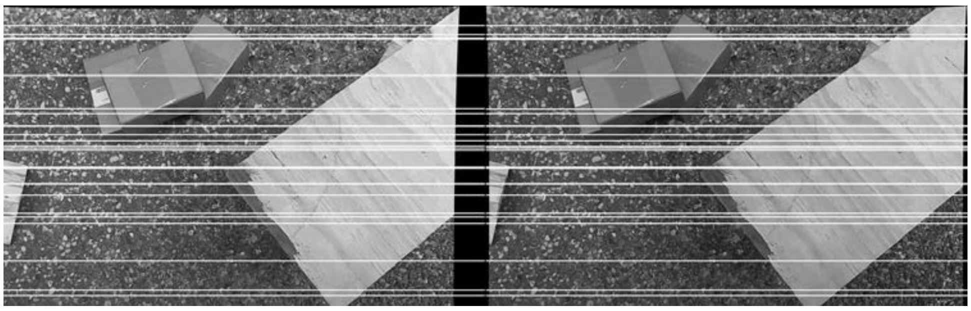


Figure 4.17: Images of the outside sequence with epipolar lines, rectified via Hartley without corrections.

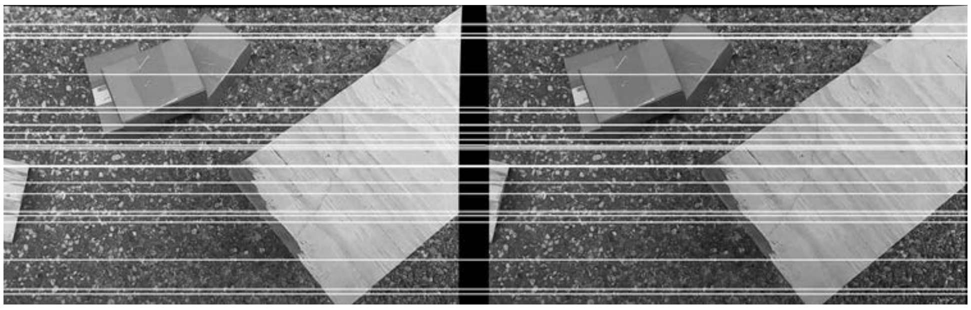


Figure 4.18: Images of the outside sequence with epipolar lines, rectified via Hartley corrected with Loop and Zhang method

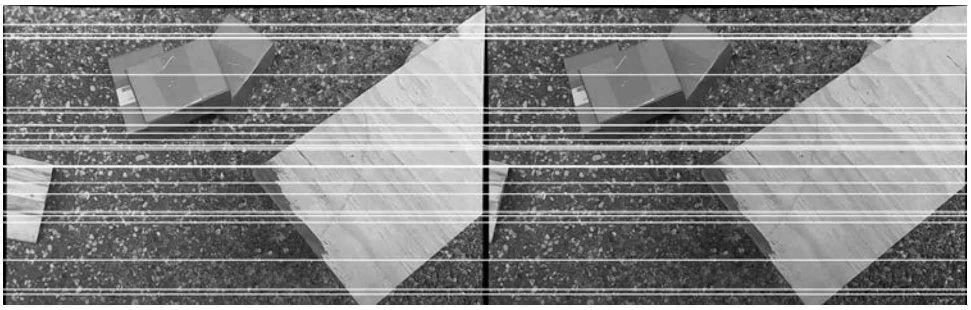


Figure 4.19: Images of the outside sequence with epipolar lines, rectified and correctly centered.

We tested different parameters of the block matching algorithm. Figures from 4.20 to 4.25 shows disparity maps for the two sequences obtained with different parameters. Two of these parameters were significant in determining the disparity maps: block size and maximum disparity. A small block size produces a more defined disparity map. However, the resulting map can also be noisier around the edges of objects and more sparse. A larger block size produces a smoother and denser disparity map. However, the resulting map has fewer details. The block size does not significantly impact the computational time of the block matching algorithm.

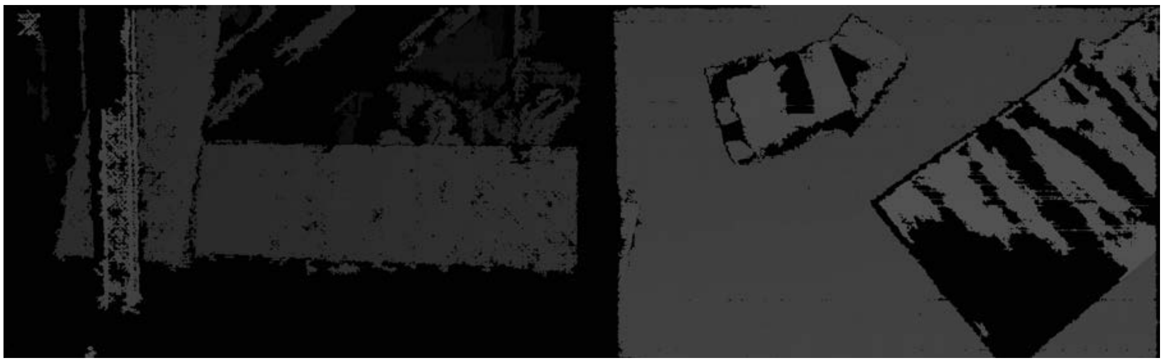


Figure 4.20: Frames densely matched with a block size of 1

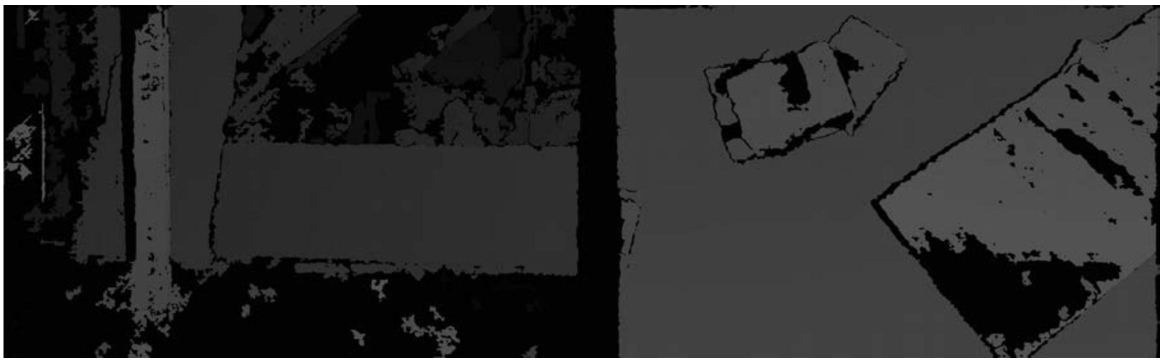


Figure 4.21: Frames densely matched with a block size of 7

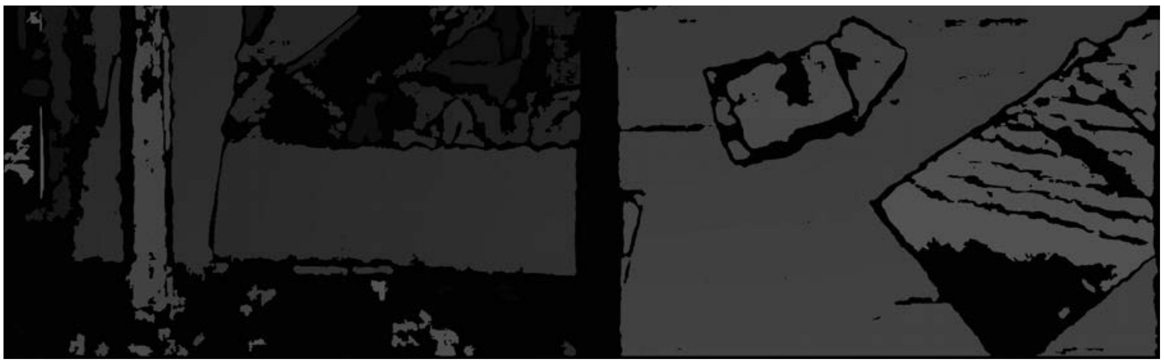


Figure 4.22: Frames densely matched with a block size of 15

As for the block size, we tested different values of maximum disparity. The maximum disparity is a parameter usually linked to the maximum depth of the image and the magnitude of the baseline. In our cases, the depth of the scene is pretty low, as is the baseline. Therefore, a big maximum disparity does not improve the resulting disparity map. Increasing it over 16*5 did not improve the quality of the map. However, it negatively affected the computational speed.

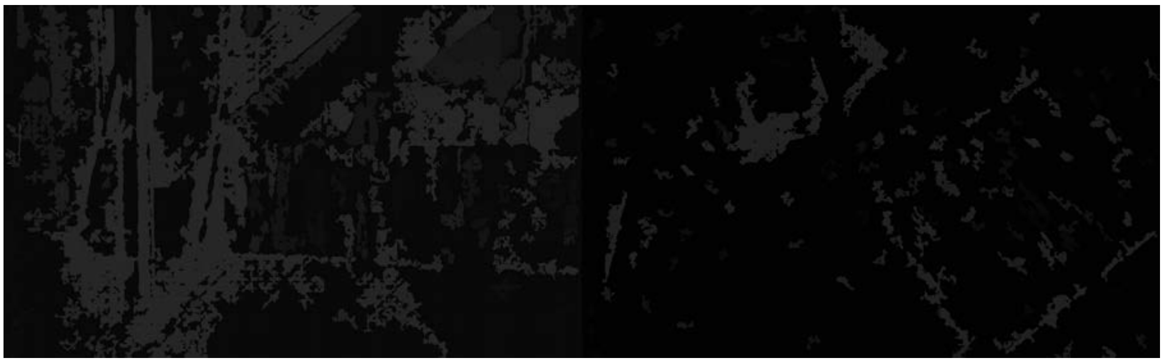


Figure 4.23: Frames densely matched with a maximum disparity of 16*2

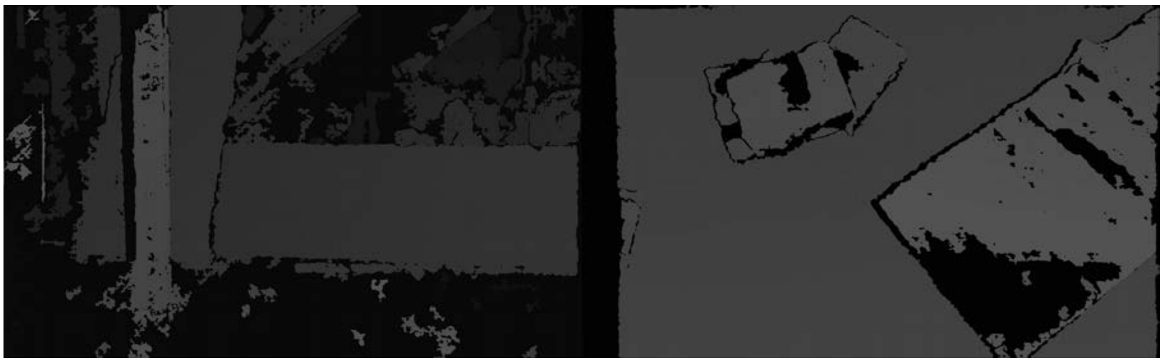


Figure 4.24: Frames densely matched with a maximum disparity of 16*5

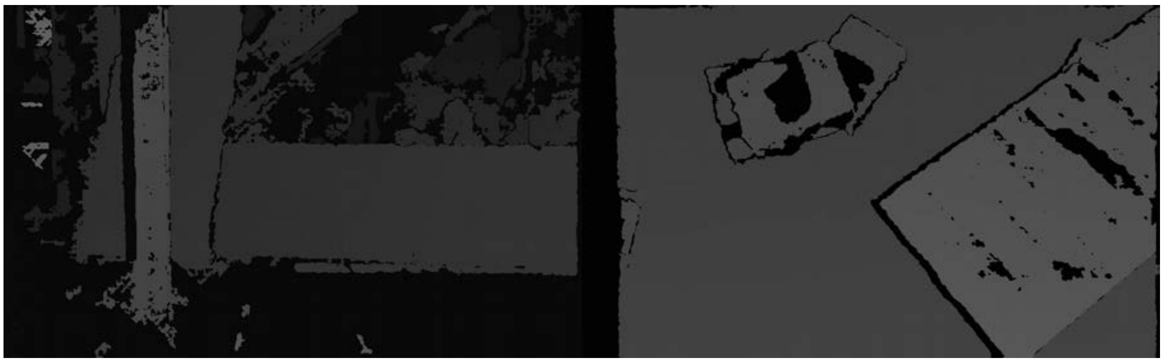


Figure 4.25: Frames densely matched with a maximum disparity of 16*30

	Computational time of Block Matching per cycle (s)		
Max. Disparity	16x2	16x5	16x30
Office sequence	0.33	0.65	3.81
Outside sequence	0.35	0.68	4.12

Table 4.1: Time required for the block matching

At this point, the triangulation process is completed, provided we assume an ideal stereo set-up. Therefore, the process is fast and does not lead to particular issues. At every cycle of the algorithm, we make sure that the 3D points are mapped inside the FoV of the camera, and then we use the information to build a point cloud.

The radius outlier filter is able to remove most of the noise caused by points scattered around the densest part of the clouds. As can be seen in Figure 4.26, however, the densest noise patches are still present.

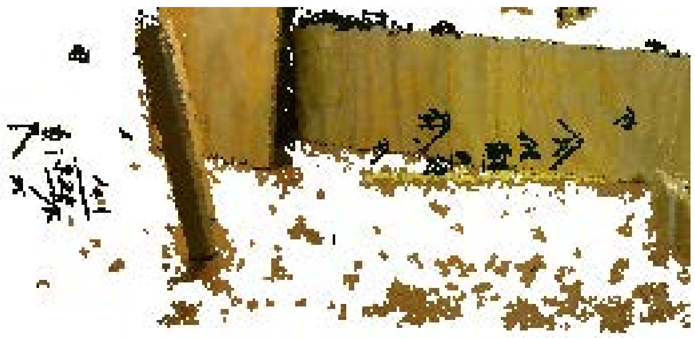


Figure 4.26:Detail of dense reconstruction for the office sequence. Some patches of noise are visible

We assemble the cloud obtained for different pairs in a point cloud vector. These clouds partially overlap in common regions. We use this overlapping to check the relative proximity of the two clouds. We do this to check for consistency in the dense reconstruction. The value we obtain are reported in Figure 4.27. As we can see, the relative distance between subsequent clouds it is not particularly high. The first sequence seems to suffer more of dis-alignment. However the resulting dense point cloud seems to reconstruct pretty well the environment as seen by the camera (Figure 4.28 and 4.29).

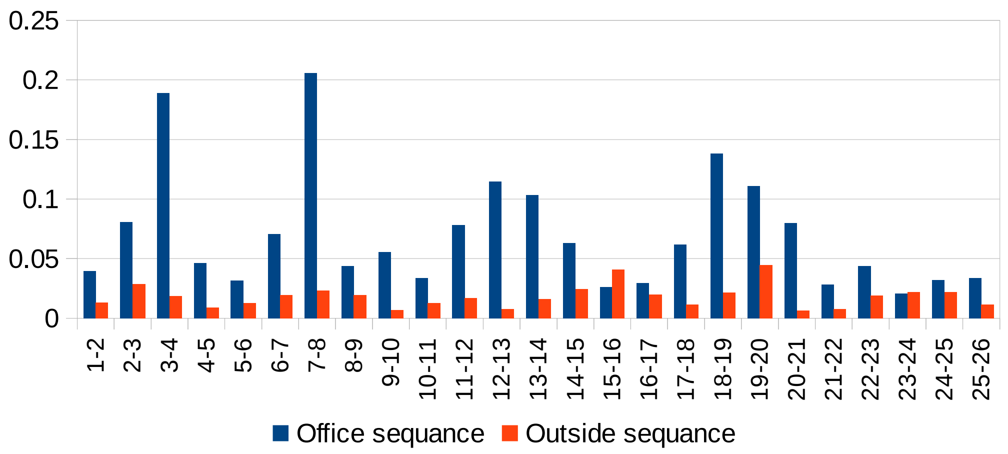


Figure 4.27: Average distance between subsequents clouds (cm).

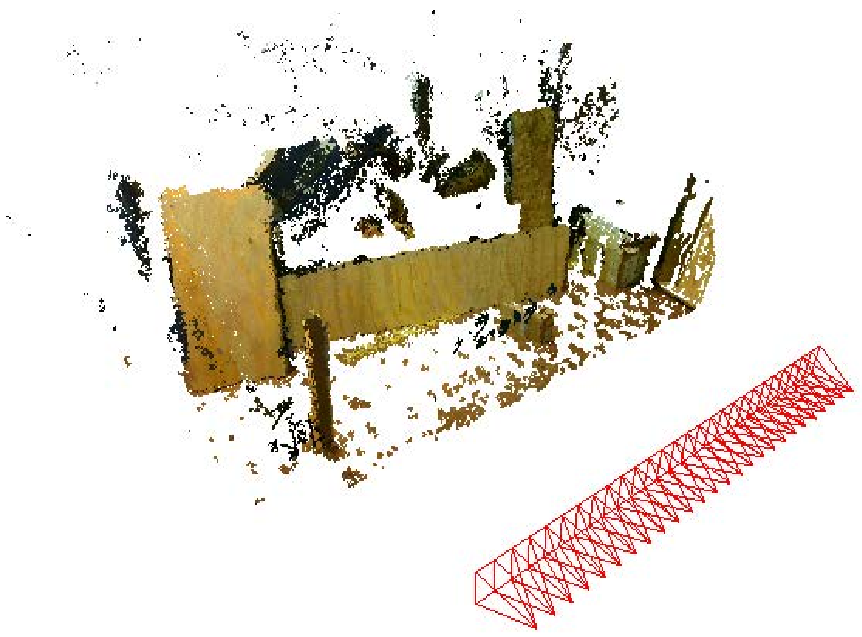


Figure 4.28: Dense point cloud for the office sequence.

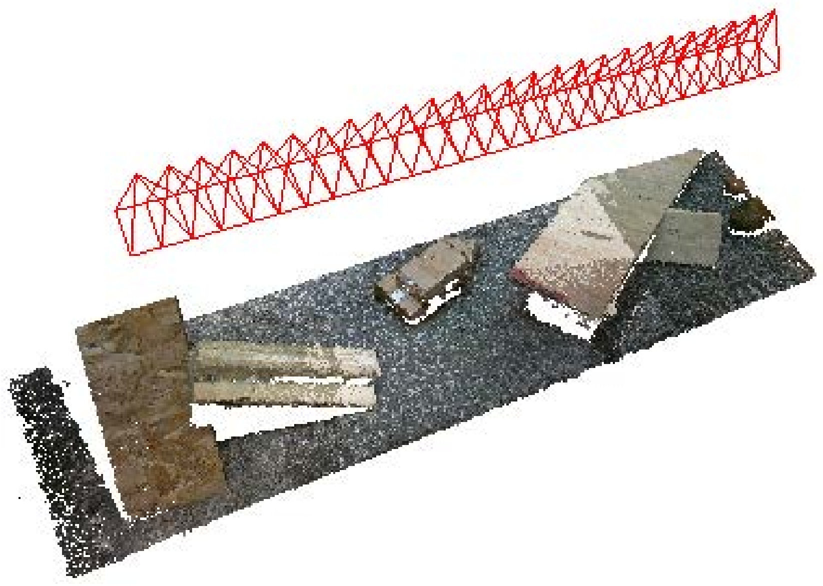


Figure 4.29: Dense point cloud for the outside sequence.

The point cloud obtained from the key point skips the filtering process; points in these point clouds, while sparser, should have a more accurate position. This information can also be used to check for the accuracy of the Dense Point cloud. We check the relative distance between a dense point cloud and the corresponding sparse point cloud for every point. Then, we average the results for every point exterminated. Figure 4.30 reports the results.

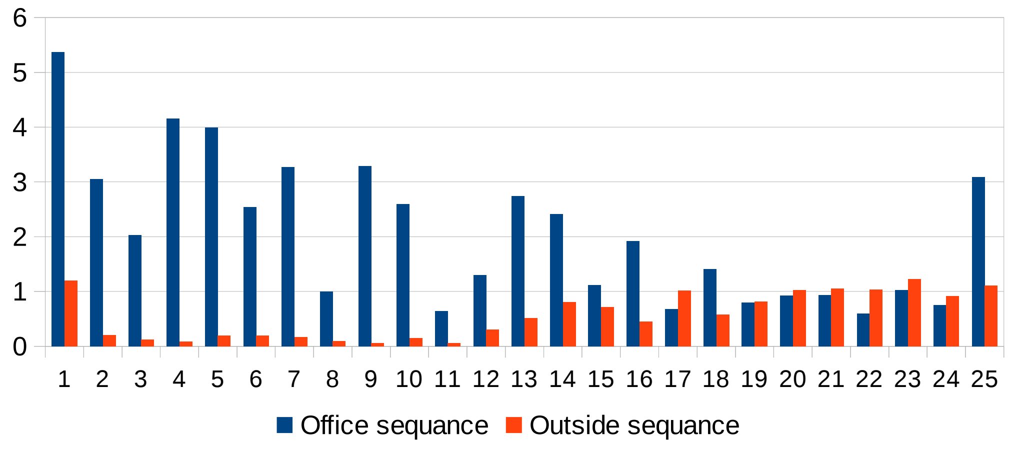


Figure 4.30: Average distance between a dense cloud and the corresponding sparse cloud (cm).

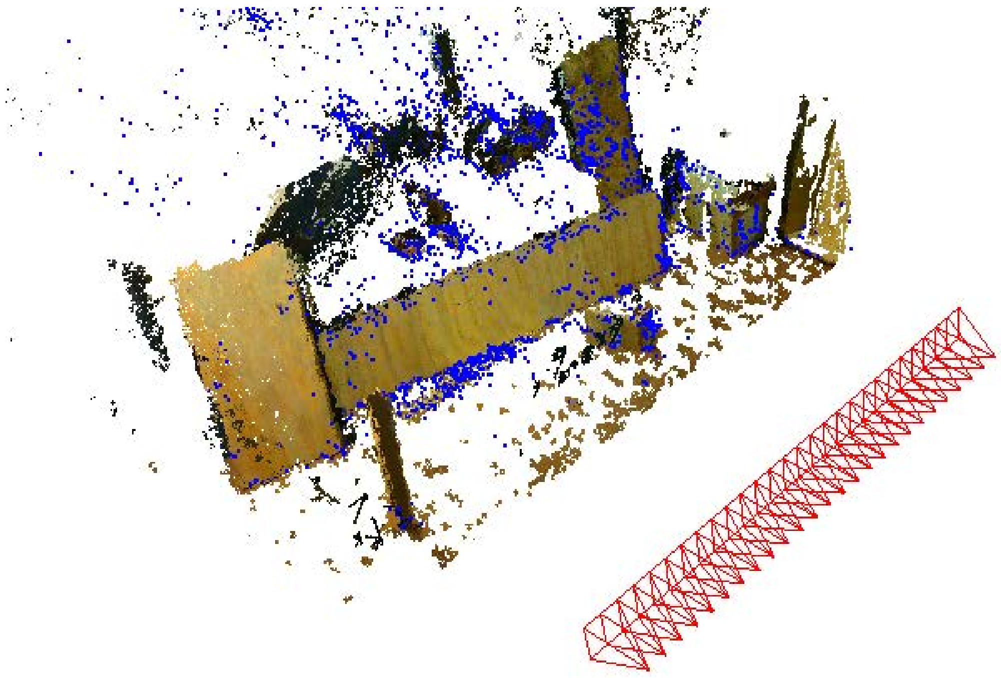


Figure 4.31: Dense point cloud and sparse point cloud for the office sequence.

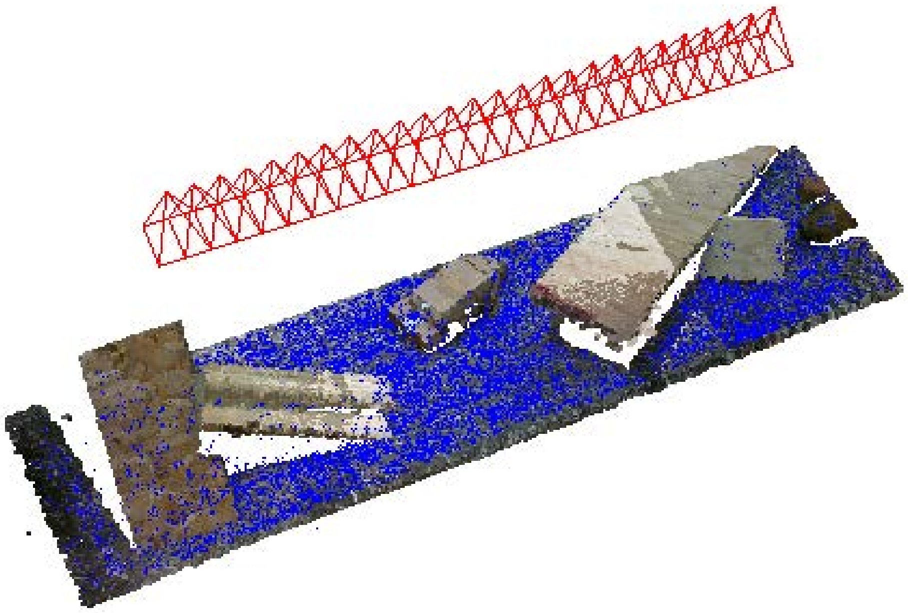


Figure 4.32: Dense point cloud and sparse point cloud for the outside sequence.

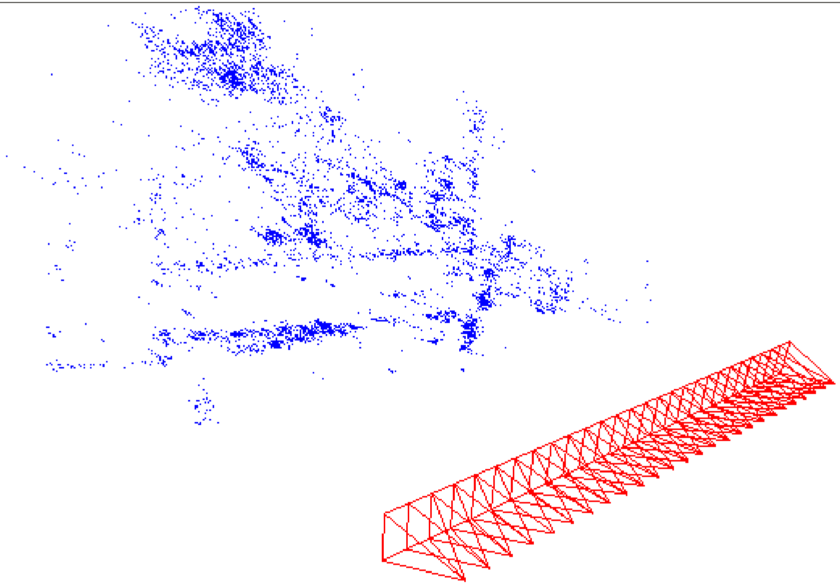


Figure 4.33: Sparse point cloud for the office sequence.

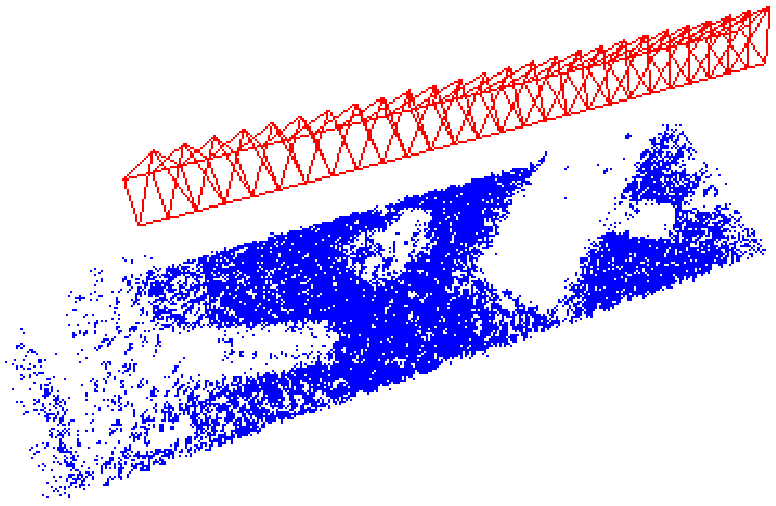


Figure 4.34: Dense point cloud and sparse point cloud for the outside sequence.

Lastly we check the density of the dense point clouds obtained. This measurement is done by considering the amount of points in a sphere of radius 3 cm. The resulting densities seem to be consistent between different clouds and between the two sequences.

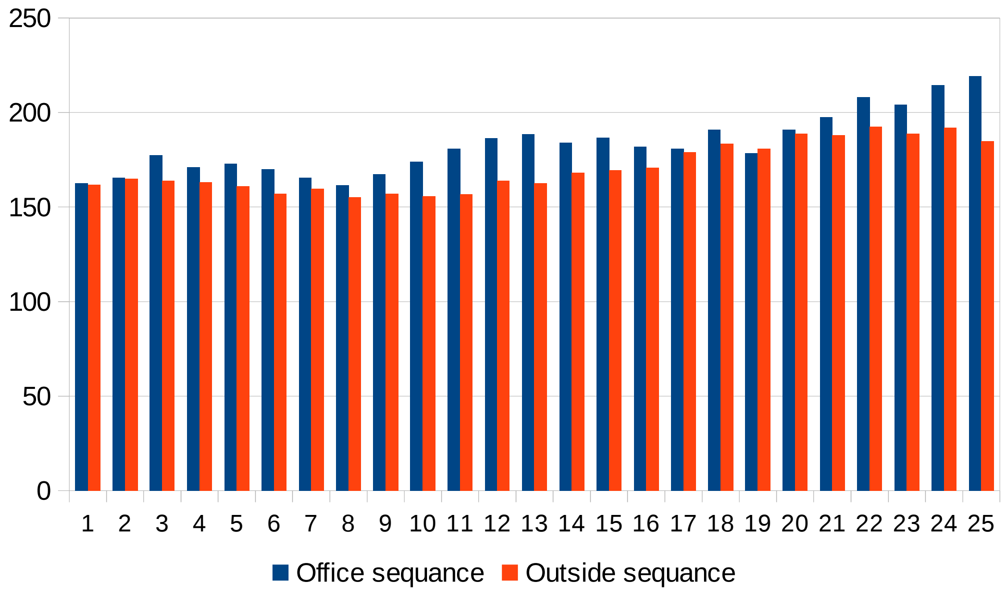


Figure 4.35: Density of the dense clouds.

4.2 Results: Polar Rectification

In this section, we present the results obtained with the polar rectification method for three different sequences. Each one of these sequences is composed of images taken from a camera mounted on a rover. The sequences differ one another for the orientation of the camera. All the sequences are composed by a series of images taken in an indoor environment: to be more precise, a corridor. Therefore, we name these sequences as “corridor sequences” and divide them by the angle θ (Figure 4.36). For all these sequences the rover moved forward with a small adjustment in its direction.

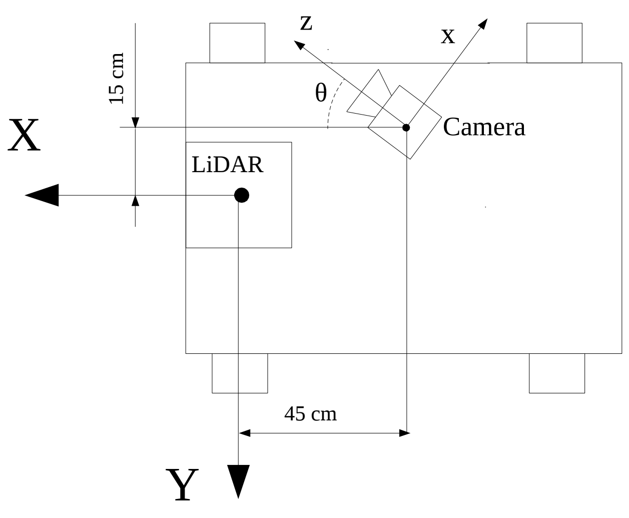


Figure 4.36: A simple outline of the set-up. Also, the camera is 50 cm above the LiDAR.

The three sequences we are going to present are:

1. Corridor 60°.
2. Corridor 45°
3. Corridor 0°.

All the images are taken with the same camera.

The camera parameters are summed in following camera matrix and distortion vector:

$$K=\begin{bmatrix} k_x f & k_s f & c_x \\ 0 & k_y f & c_y \\ 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} 696.3 & 0 & 698.1 \\ 0 & 696.2 & 357.9 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad dist=\begin{bmatrix} k_1 \\ k_2 \\ p_1 \\ p_2 \end{bmatrix}=\begin{bmatrix} -0.1708 \\ 0.0231 \\ -0.0003 \\ 0.0017 \end{bmatrix}$$

The aim of these sequences was to test the polar rectification method with a well-known VO. Therefore, we decided to use a LiDAR to obtain the camera poses. Furthermore, the LiDAR give us an ideal sparse cloud to use as a reference for our dense point clouds. Thus, for these sets of sequences, we also compute the relative distance between the dense point cloud and the LiDAR point cloud.

In some preliminaries tests, we tried to rectify the images of the 45° sequence and 60° sequence using the linear rectification. However, the resulting rectified images were too big and deformed to be used during the dense match step of the algorithm. As we can see in Figure 4.38, some portion of the image is still usable. However, to identify this portion of the image requires a very precise method. In this case, the method we employed for centering rectified images is not able to accurately detect and center the best part of the pictures.

For this set of results, we do not have a fundamental to use as a reference. The orientation and the virtual baseline between the images change from pair to pair. Therefore, the fundamental matrix changes accordingly.

The polar rectification method does not require matching features to resolve the epipolar geometry. However, we compute them to build a sparse point cloud to use as evaluation in addition to the LiDAR point cloud. The same consideration regarding the features we discussed in the previous set of results is still valid.

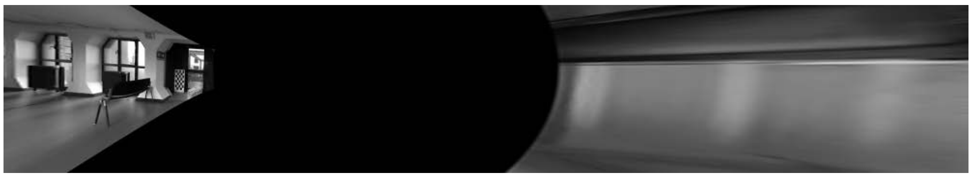


Figure 4.37: Linear rectification for an image with the epipolar point near it.

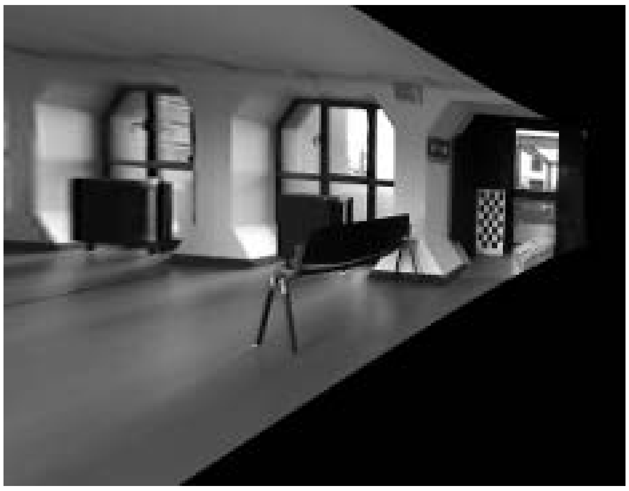


Figure 4.38: Detail of Figure 4.37.

4.2.1 Polar Rectification: Corridor 60°

The features match well and, as we are going to see later, they are numerous enough to build an accurate sparse cloud.

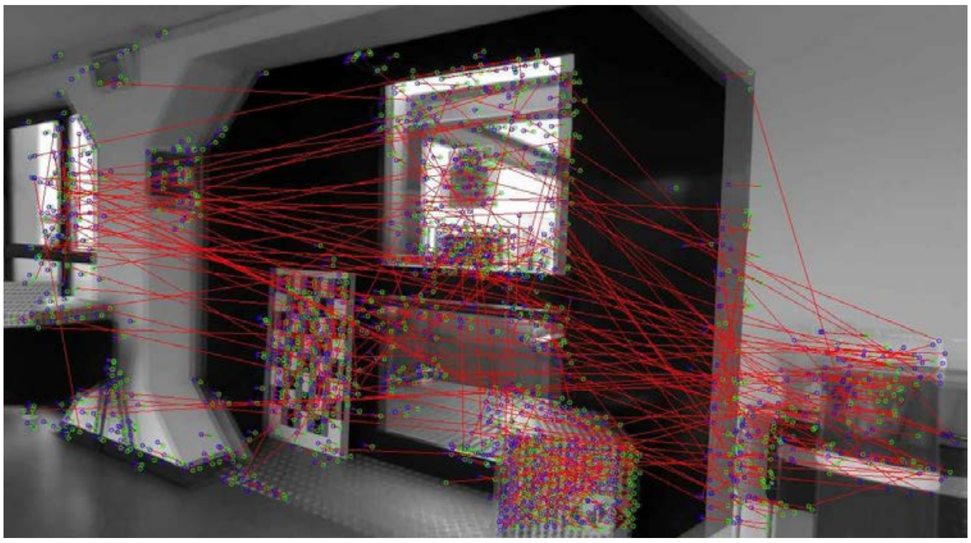


Figure 4.39: Corridor 60° unfiltered matched features.

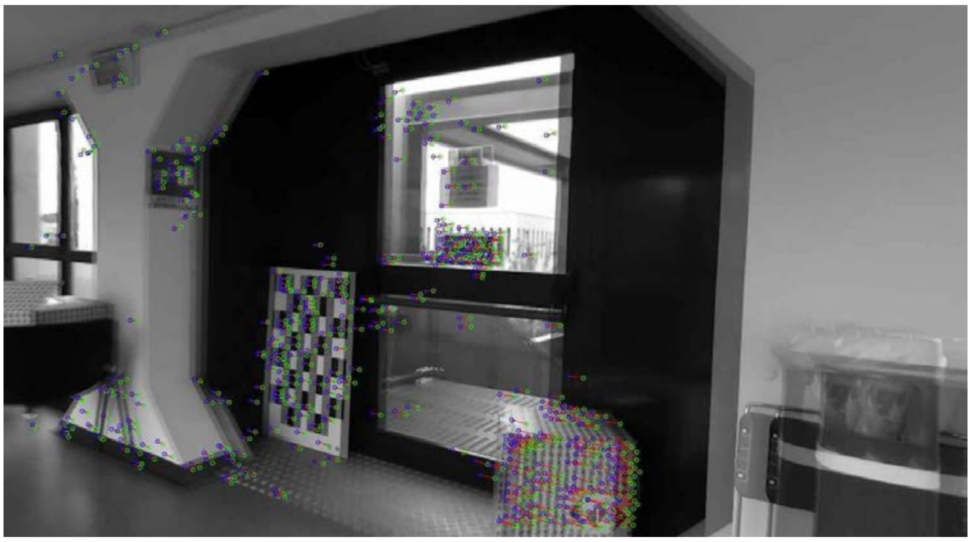


Figure 4.40: Corridor 60° filtered matched features.

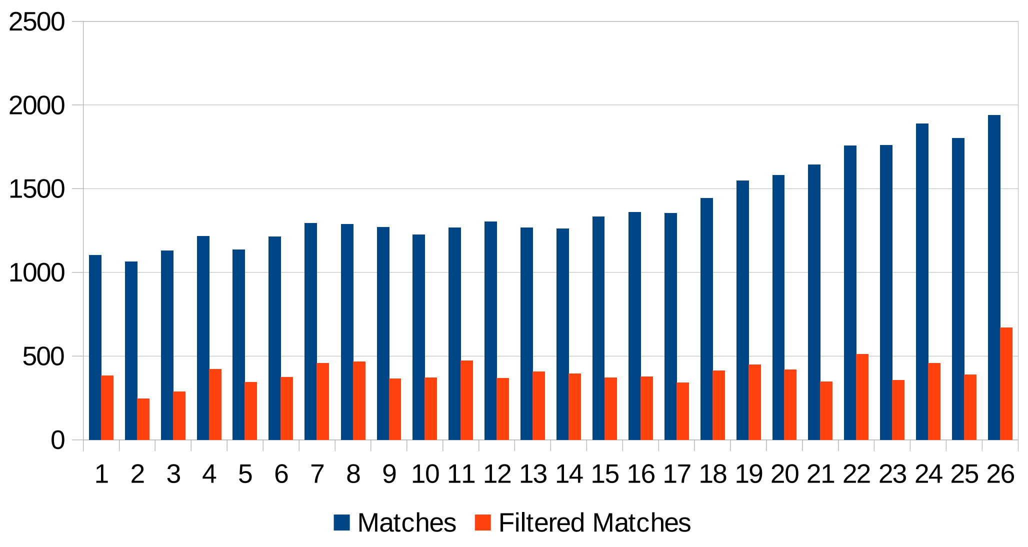


Figure 4.41: Corridor 60° Number of features in the elaborated pair.

As we show in Figures 4.42 and 4.43, the polar rectification method correctly adjusts the epipolar geometry. However, it also introduces unwanted distortions, that grow more severe the closer the pixel is to the epipolar point in the un-rectified images. Therefore, during triangulation, we have to check if the pixel is near the epipolar point. If this is the case, we avoid triangulating it. Another possible solution is to directly avoid the leftmost area in the rectified images, which correspond to the nearest points to the epipolar point. For example, S. Cavegn et al. [4] apply this method in their six camera set-up when matching images from a pair of cameras with a baseline in the direction of movement. Furthermore, the polar rectification method maps one point in the un-rectified image to more than one position on the rectified image. Therefore, around the same point on the 3D space, we may encounter points with slightly different coordinates.

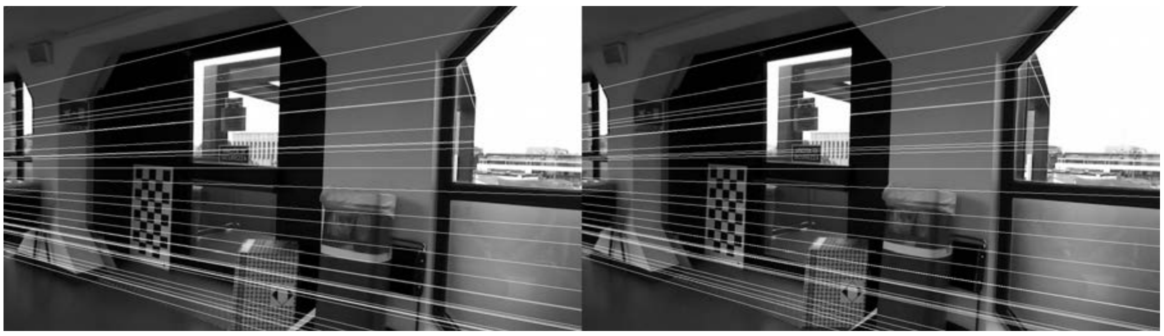


Figure 4.42: Pair of images with their epipolar lines.

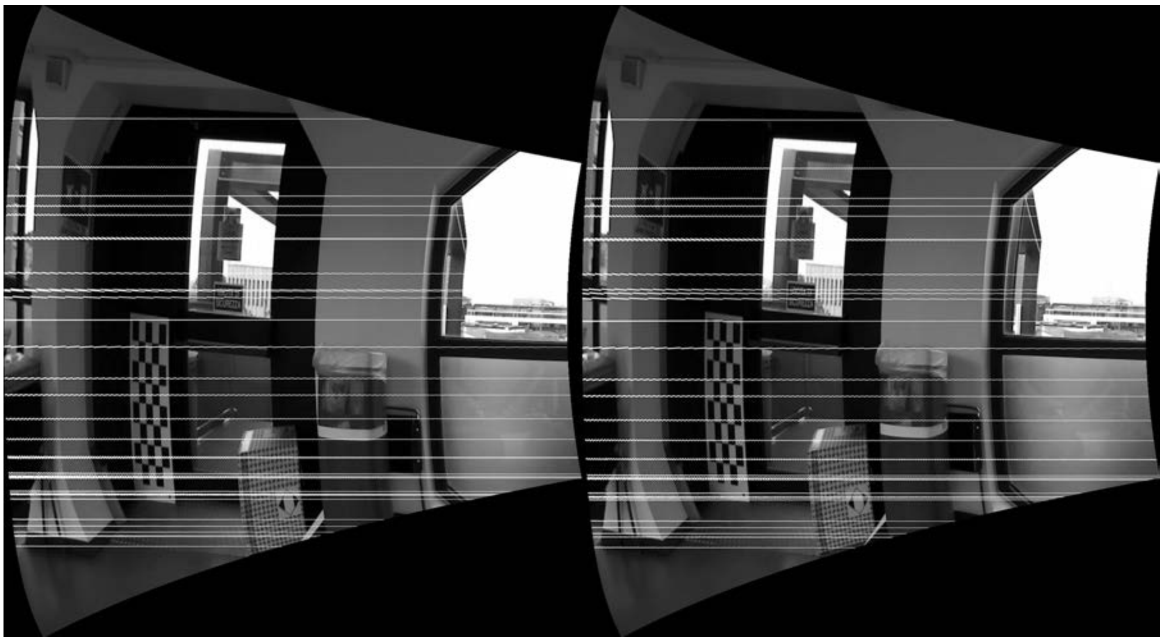


Figure 4.43: The same pair of Figure 4.42 rectified. The epipolar lines are horizontal.

As in the previous sequences, we tested different values of block size and maximum disparity. Figures from 4.44 to 4.51 shows some of the resulting disparity maps. In this particular sequence, the depth of the scene is low. Therefore, we can use a small maximum disparity value to detect the disparities and to build the disparity map.

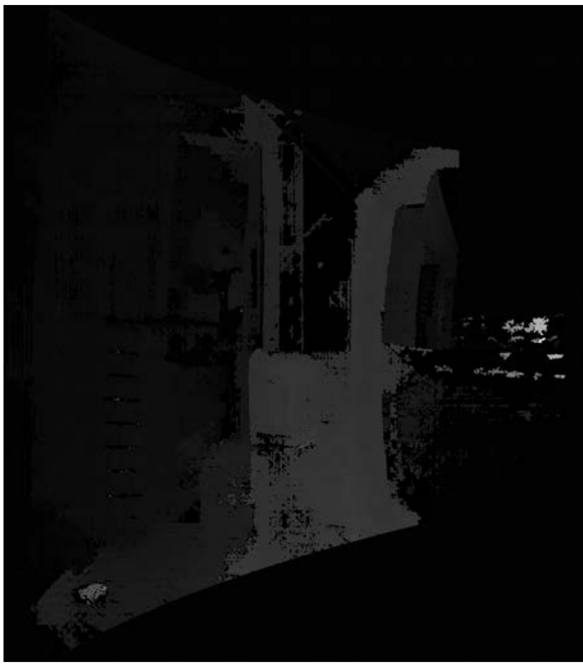


Figure 4.44: Block size of 3.

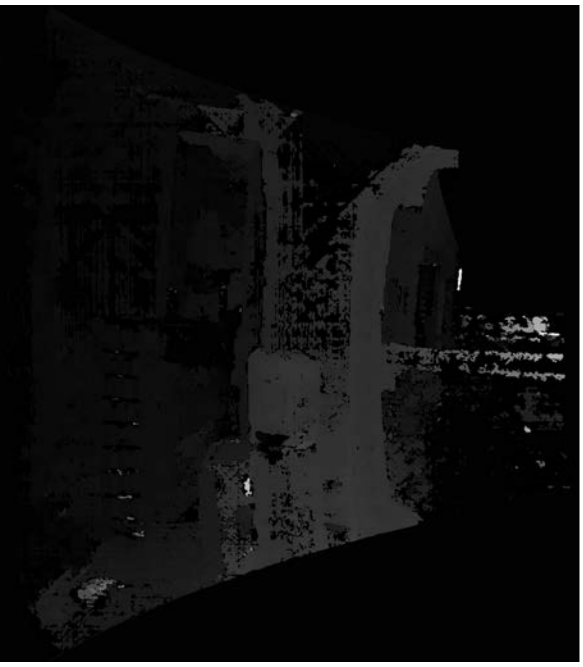


Figure 4.45: Block size of 5.

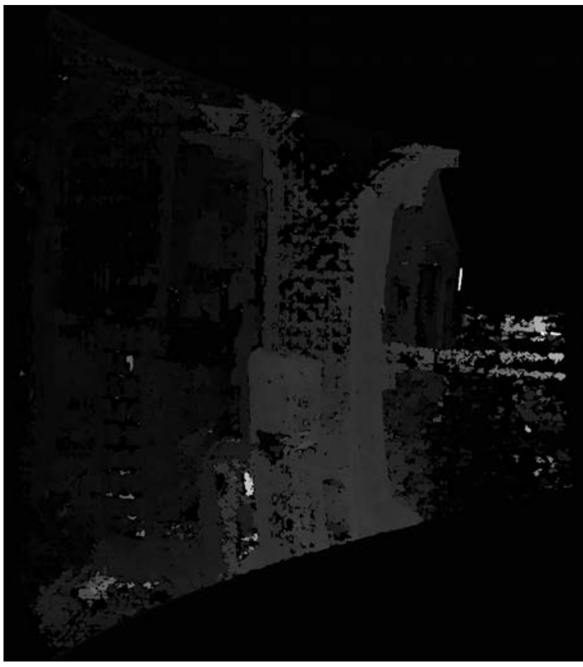


Figure 4.46: Block size of 7.

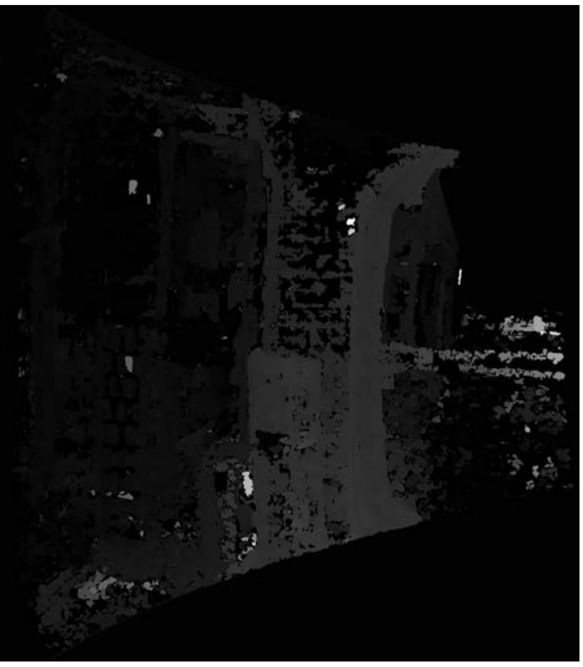


Figure 4.47: Block size of 9.

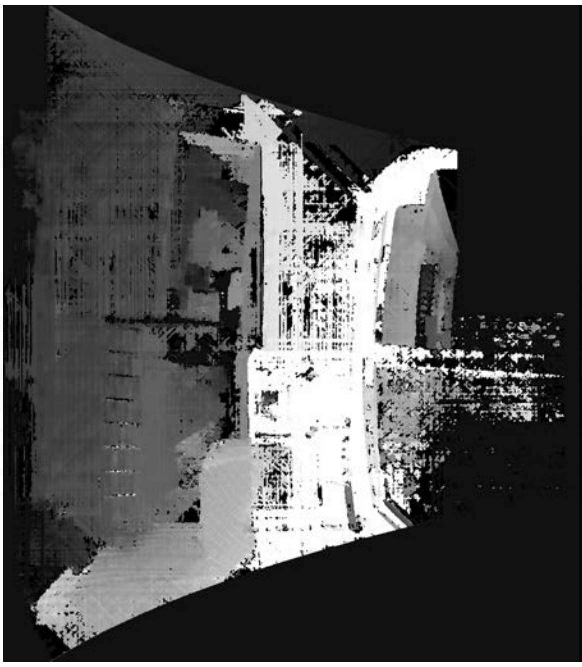


Figure 4.48: Maximum disparity equal to 16*1

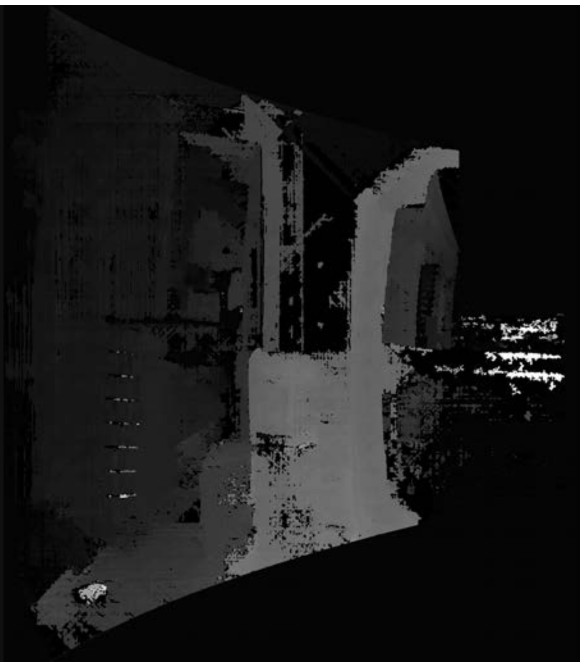


Figure 4.49: Maximum disparity equal to 16*3

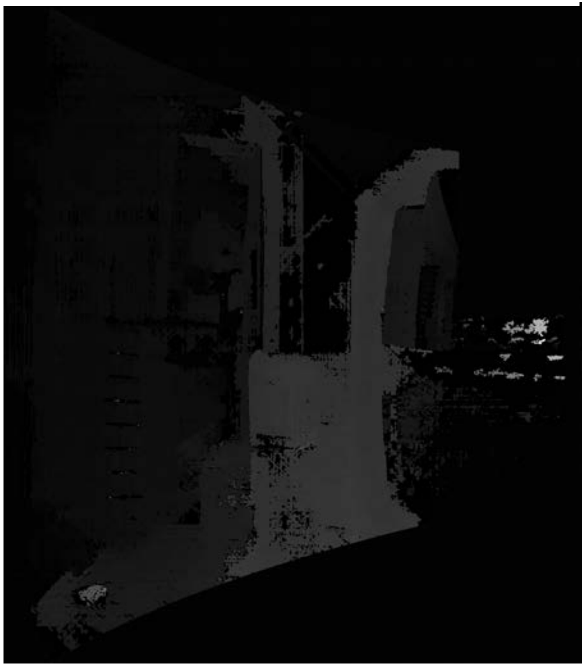


Figure 4.50: Maximum disparity equal to 16*9

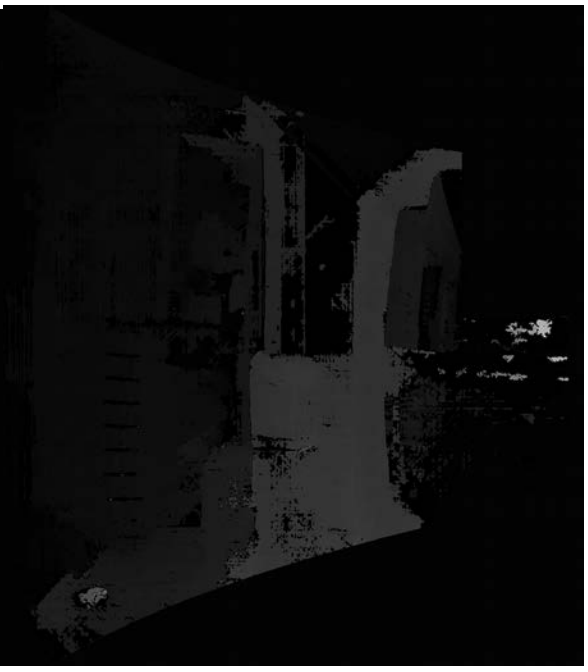


Figure 4.51: Maximum disparity equal to 16*27

The quality of the disparity map seems to improve for low block sizes. As mentioned before, the maximum disparity value can be set to the relatively low value of 16*9. Increasing the maximum disparity over this value does not improve the quality of the map.

	Computational time of Block Matching per cycle (s)			
Max. Disparity	16x1	16x3	16x9	16x27
Corridor 45 ° sequence	0.38	0.76	1.87	5.33

Table 4.2: Time required for the block matching

When seen from a favorable point of view, the clouds seem to be well recontacted and well registered (Figures from 4.54 to 4.61). Furthermore, the sparse and dense clouds seem to overlap well as such as the dense clouds and the LiDAR clouds. However, when considered from the front or from a higher position (Figures from 4.62 to 4.69), we can notice that a great amount of noise is still present. Furthermore, the clouds do not overlap correctly. The same object is mapped more times in a cloud at different positions from the camera. Nevertheless, the reconstructed clouds seem to correctly identify the position of the box and the column present in the FoV of the camera (Figure 4.53).

The computed results confirm these assumptions (Figures from 4.70 to 4.73). The average relative distance between subsequent clouds is much higher than in the linear rectification case, as such as the distance between the key point cloud and the dense cloud, and the distance between the LiDAR cloud and the dense cloud. We can identify two reasons motivating these results: the distortion introduced by the polar rectification, and a point cloud registration process not robust enough.

Furthermore, the block matching algorithm we employed is not adequate in situations where the images are particularly distorted. This algorithm uses a fixed block size within which it tries to find corresponding objects in the images forming a pair. If the objects are big, a larger block size produces better results. On the contrary, when the objects are small, a smaller block size is preferable. In our case, the distortion on the image is not uniform. Therefore, objects near the epipolar point assume bigger sizes and objects far away smaller ones. A dense matching method with a variable block size, or that does not depend on block size, would be more suitable to analyze the images rectified with the polar rectification.

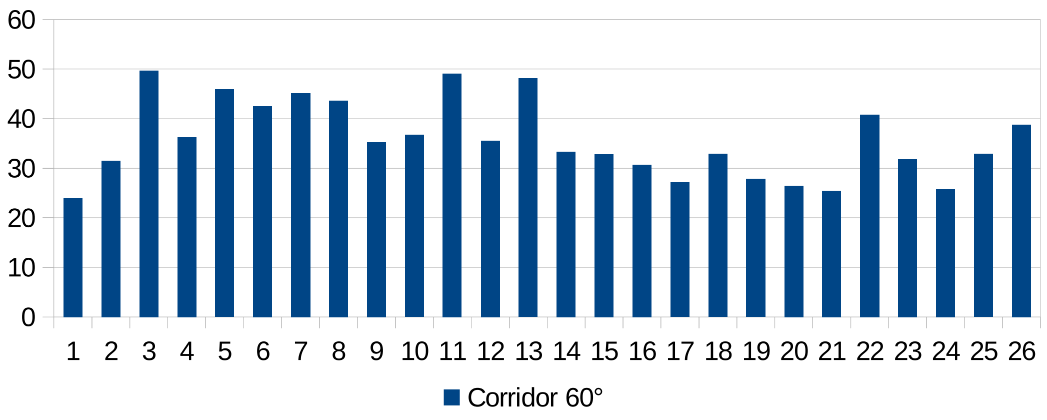


Figure 4.52: Corridor 60° density of dense clouds.

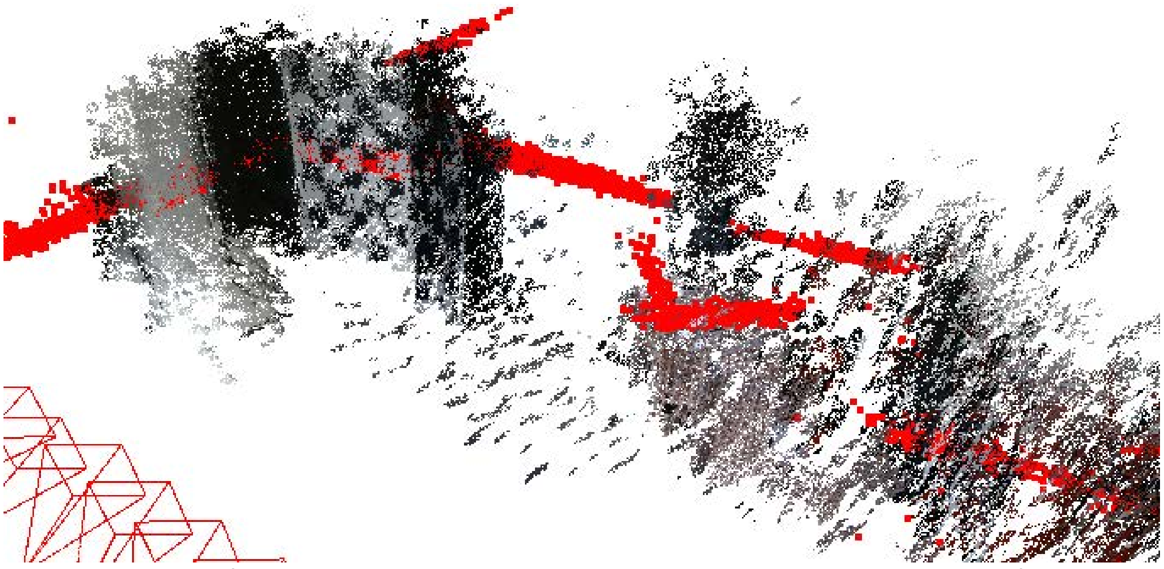


Figure 4.53: Detail of the overlapping between the LiDAR and the dense cloud. The position of a box and of the column on the left side of the image are correctly identify by the dense clouds.

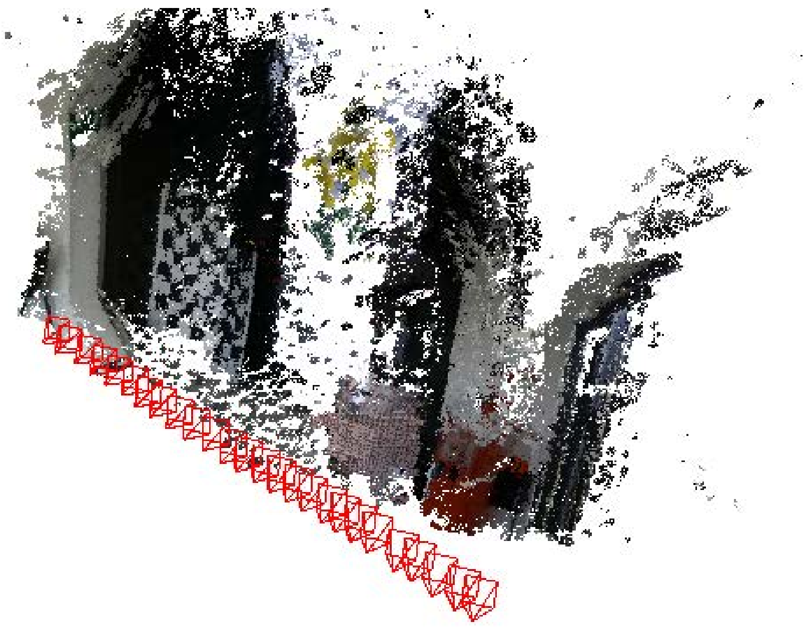


Figure 4.54: Corridor 60° dense cloud. View 1.

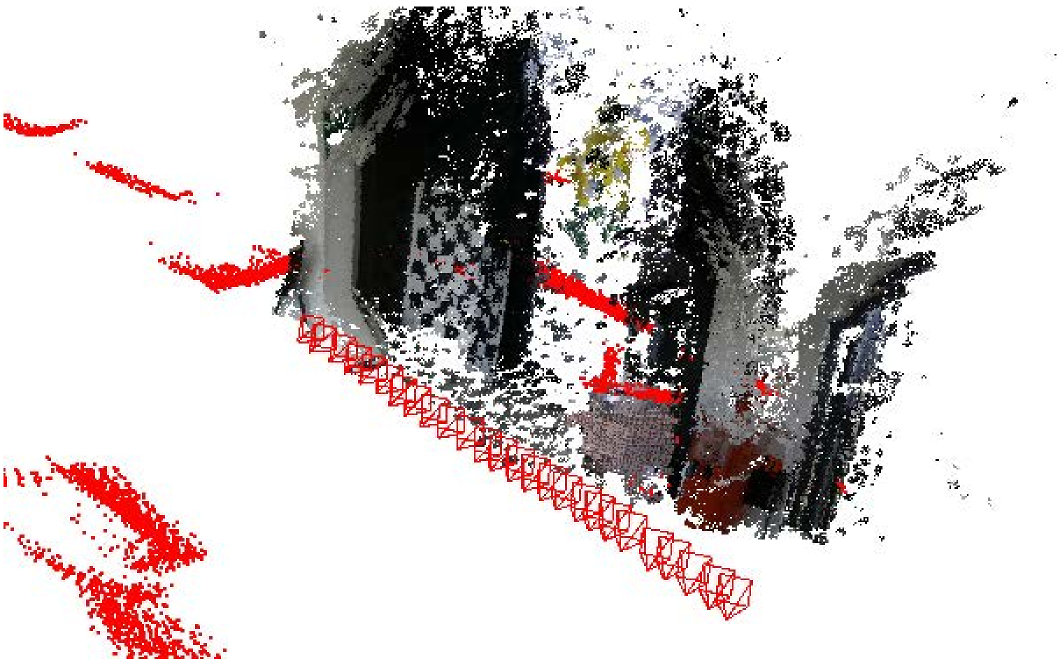


Figure 4.55: Corridor 60° dense cloud and LiDAR cloud. The sparse LiDAR cloud is in red. View 1.

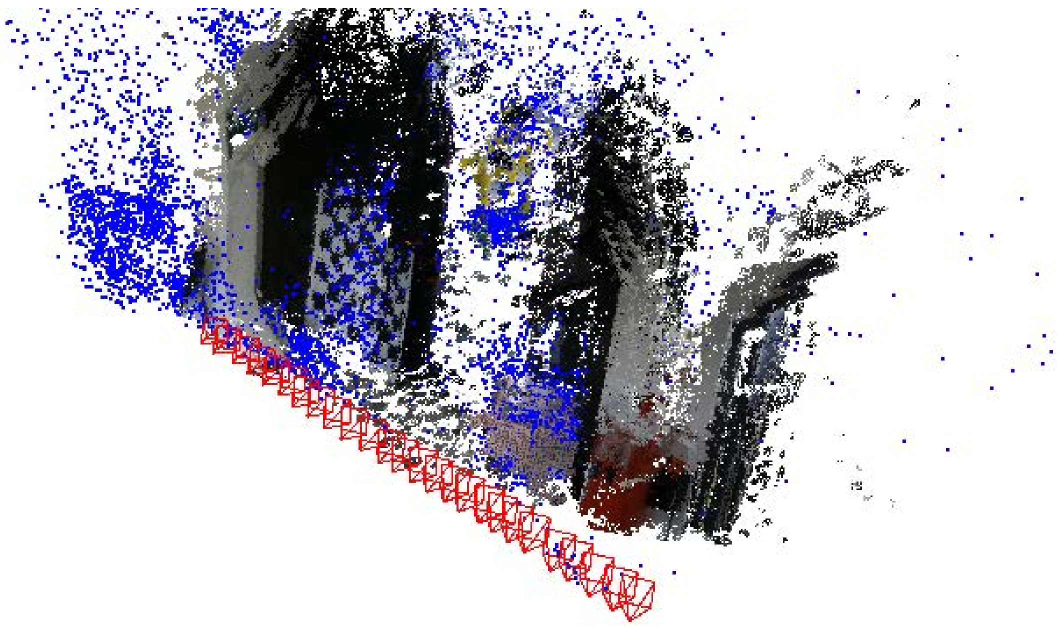


Figure 4.56: Corridor 60° dense cloud and sparse cloud from the key points. The key point cloud is in blue. View 1.

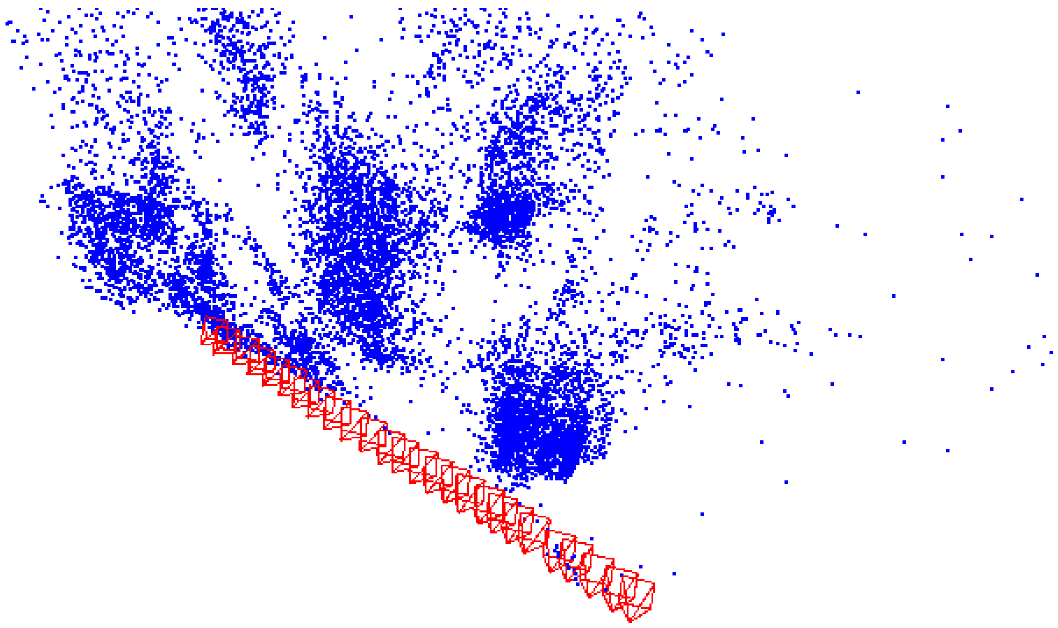


Figure 4.57: Corridor 60° sparse cloud built from the key points. View 1.

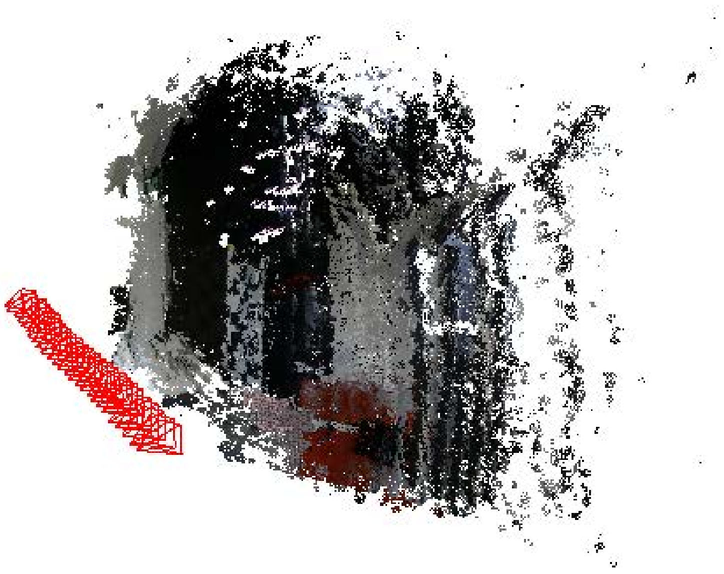


Figure 4.58: Corridor 60° dense cloud, another favorable view. View 2.

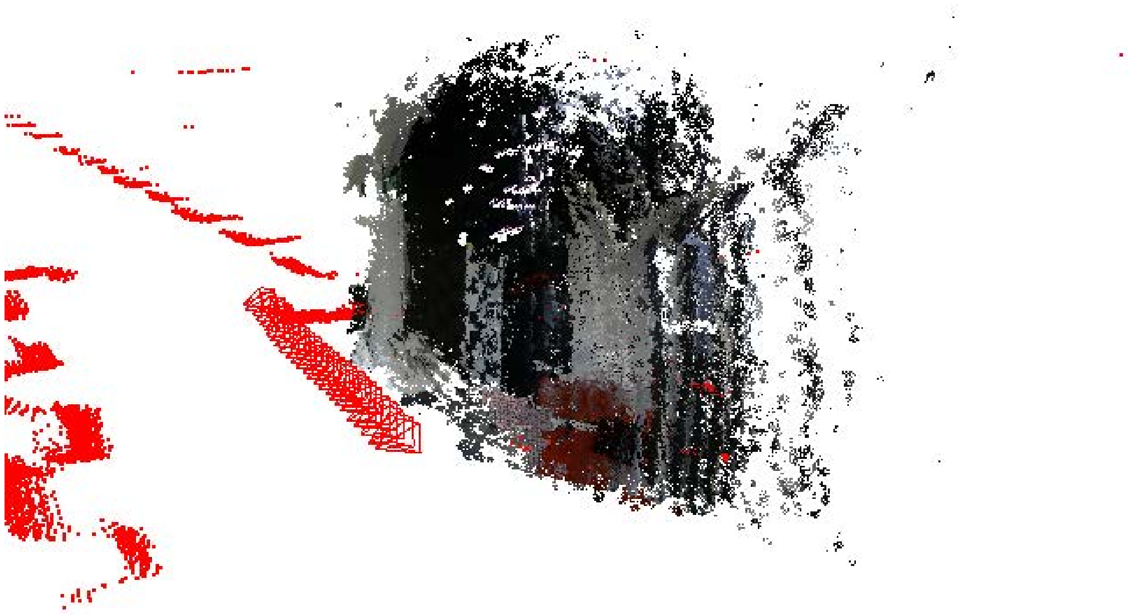


Figure 4.59: Corridor 60° dense cloud and LiDAR cloud. The sparse LiDAR cloud is in red. View 2.

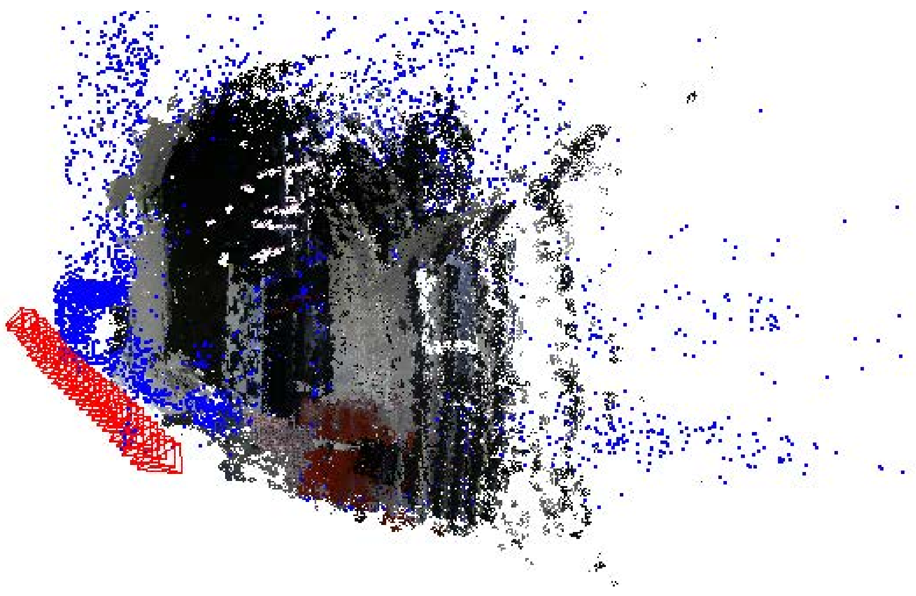


Figure 4.60: Corridor 60° dense cloud and sparse cloud from the key points. The key point cloud is in blue. View 2.

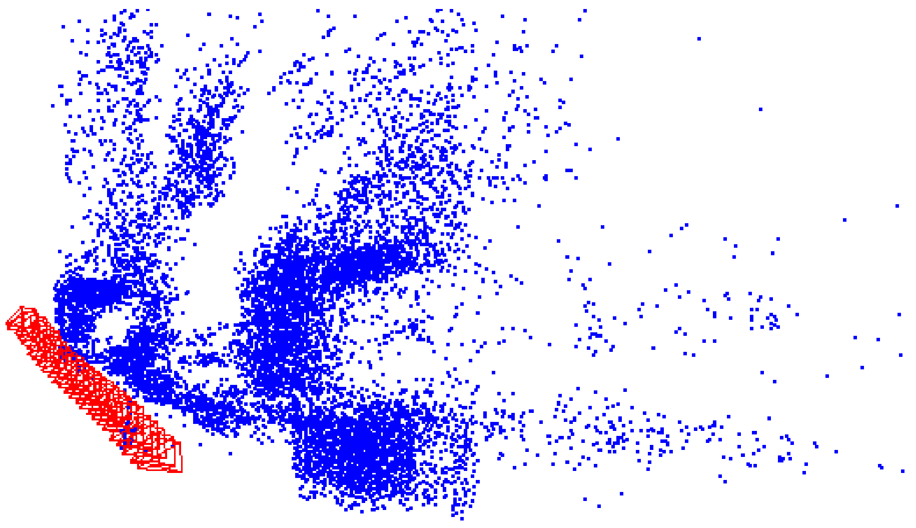


Figure 4.61: Corridor 60° sparse cloud built from the key points. View 2.

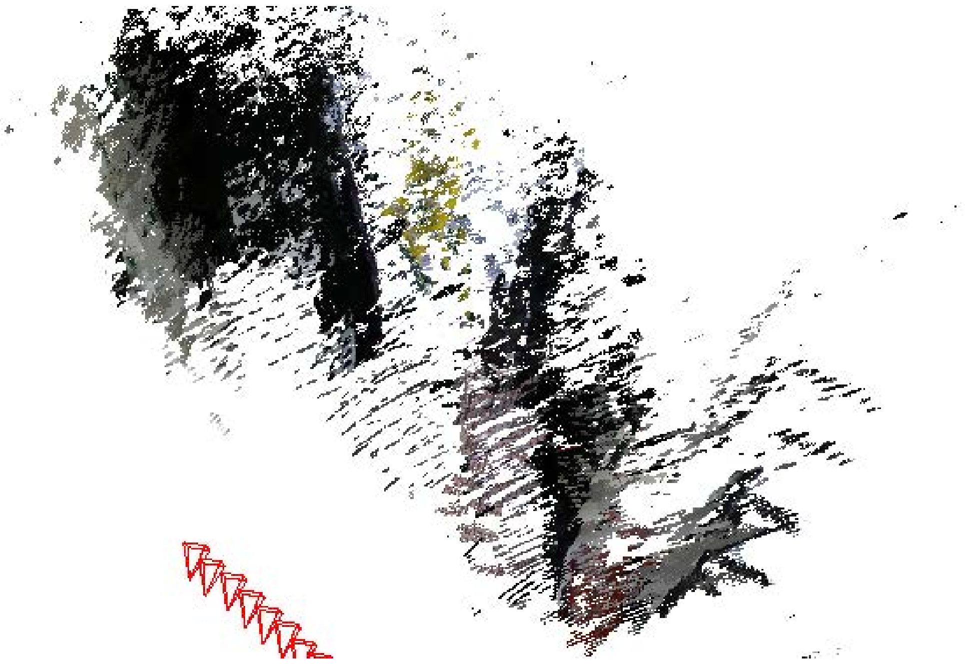


Figure 4.62: Corridor 60° dense cloud. Seen from an higher position. View 3.

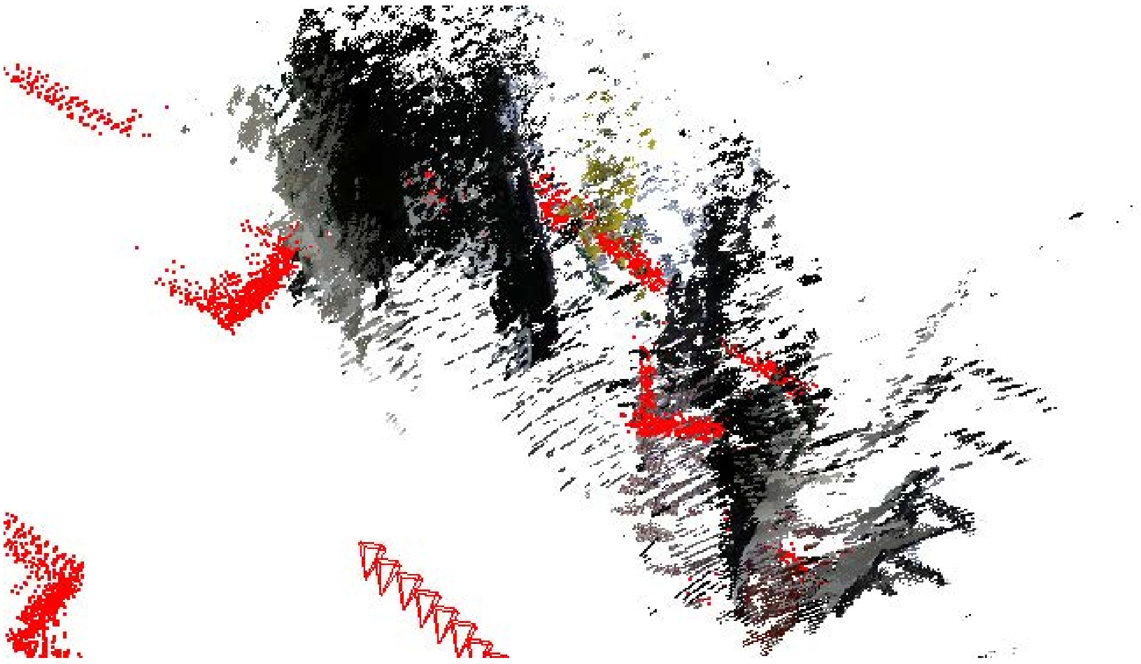


Figure 4.63: Corridor 60° dense cloud and LiDAR cloud. The sparse LiDAR cloud is in red. View 3.

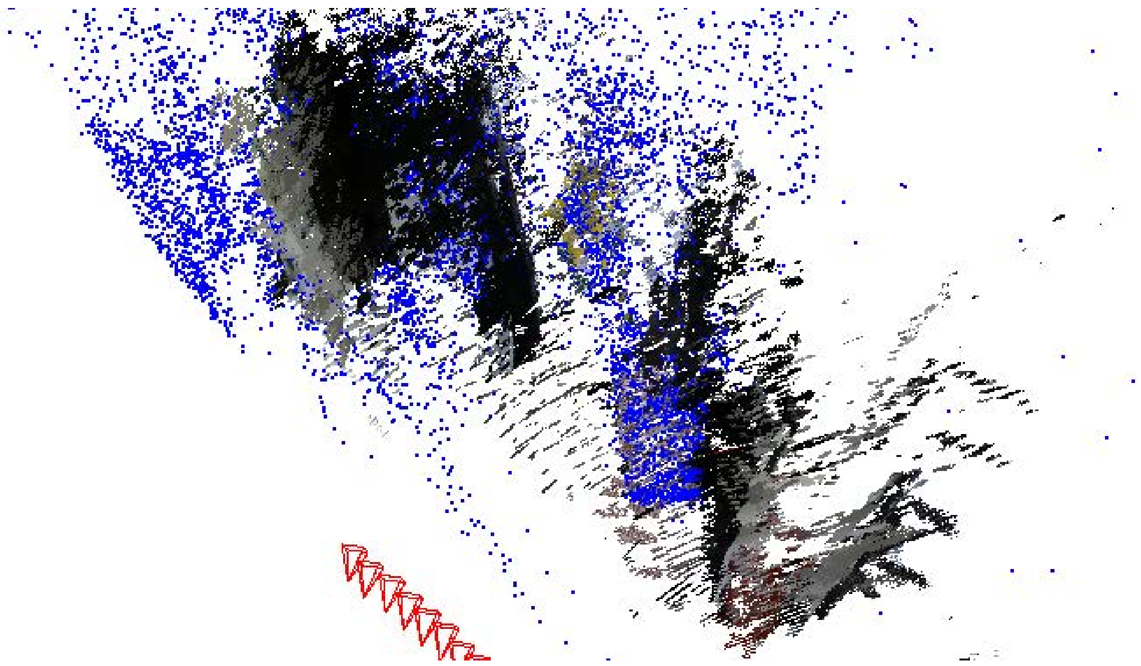


Figure 4.64: Corridor 60° dense cloud and sparse cloud from the key points. The key point cloud is in blue. View 3.

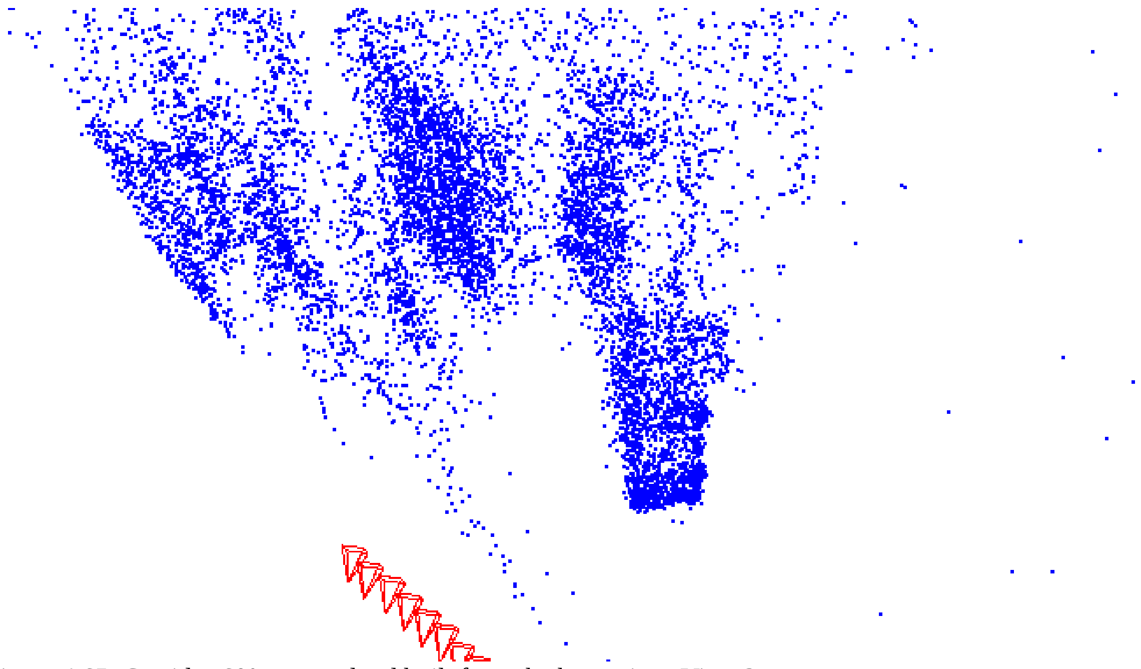


Figure 4.65: Corridor 60° sparse cloud built from the key points. View 3.

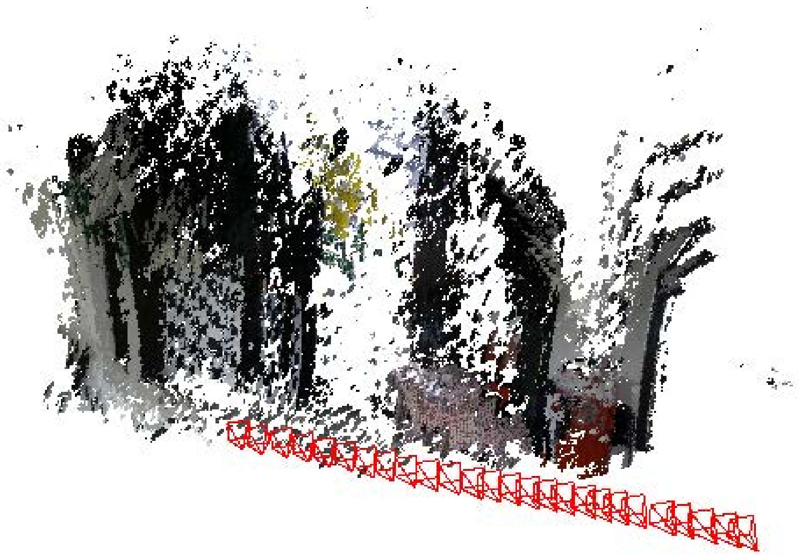


Figure 4.66: Corridor 60° dense cloud. Seen from the side. View 4.

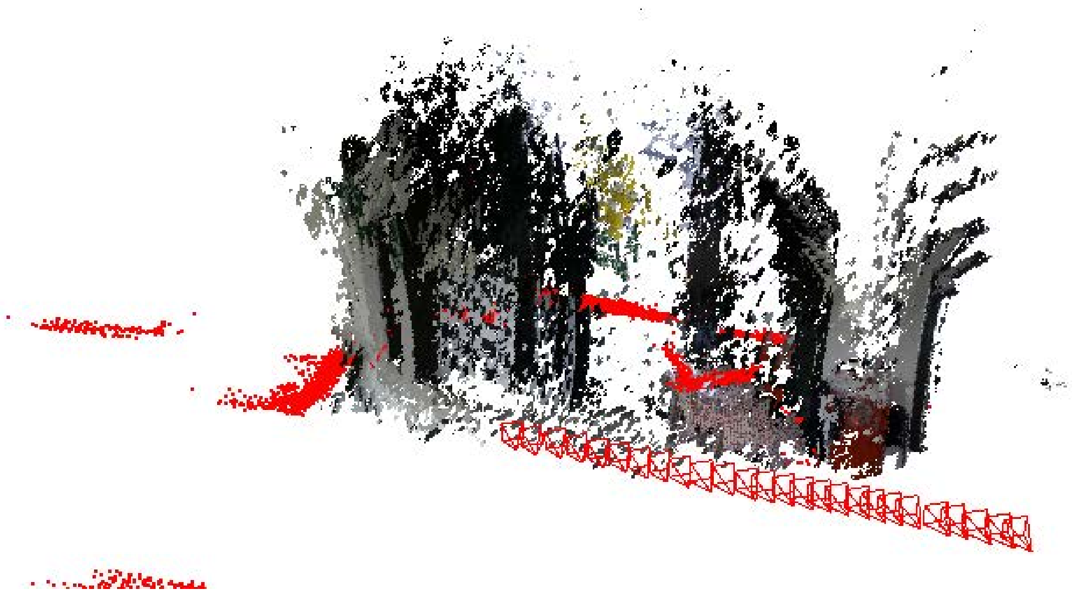


Figure 4.67: Corridor 60° dense cloud and LiDAR cloud. The sparse LiDAR cloud is in red. View 4.

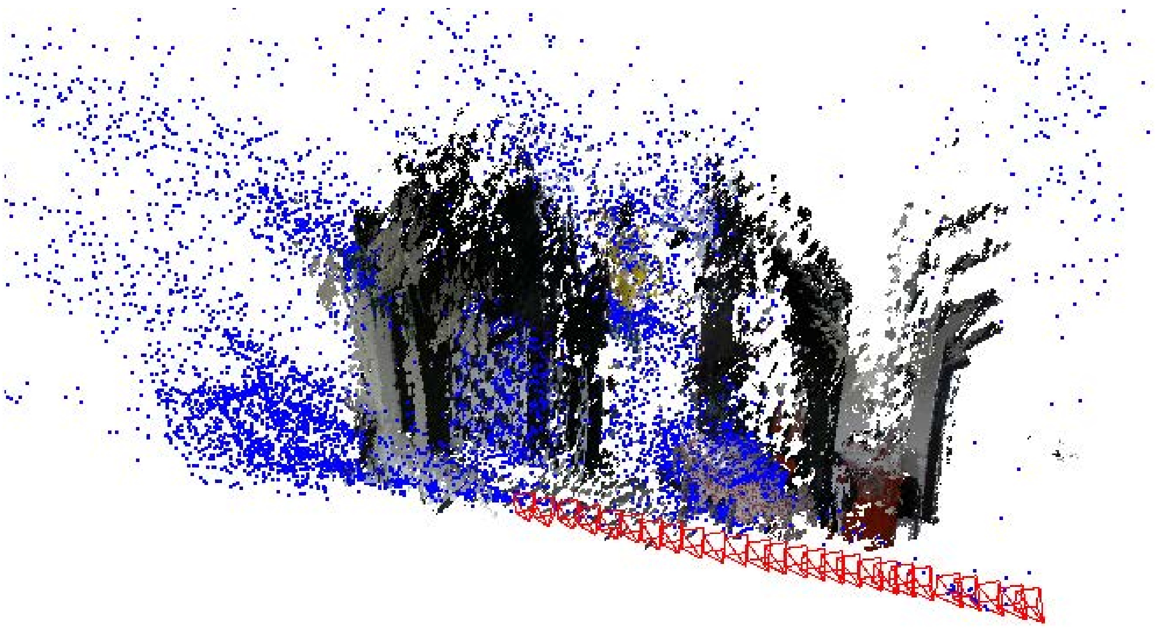


Figure 4.68: Corridor 60° dense cloud and sparse cloud from the key points. The key point cloud is in blue. View 4.

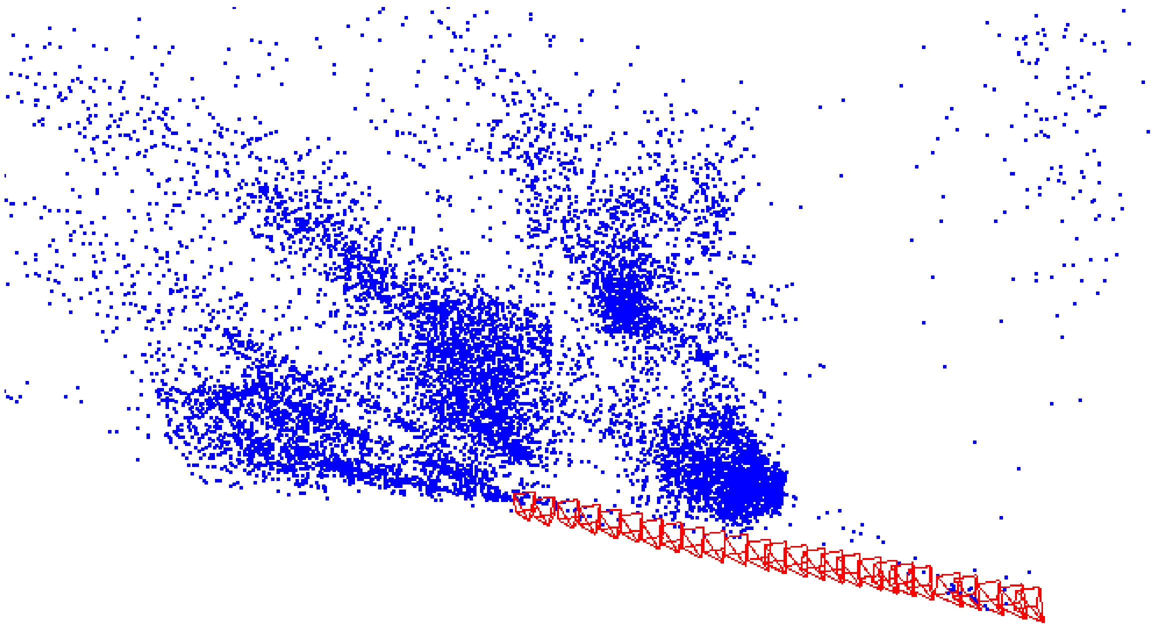


Figure 4.69: Corridor 60° sparse cloud built from the key points. View 4.

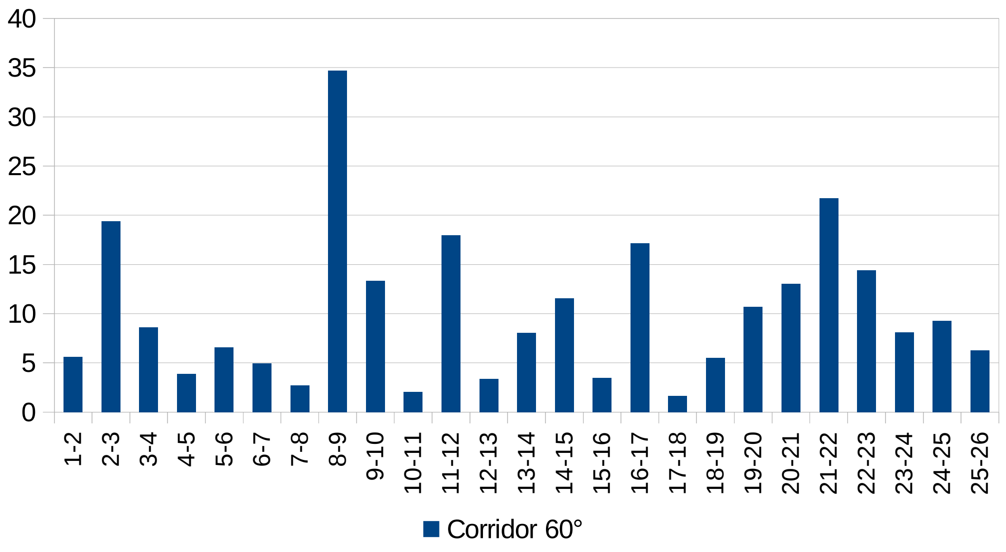


Figure 4.70: Corridor distance between subsequents point clouds (cm).

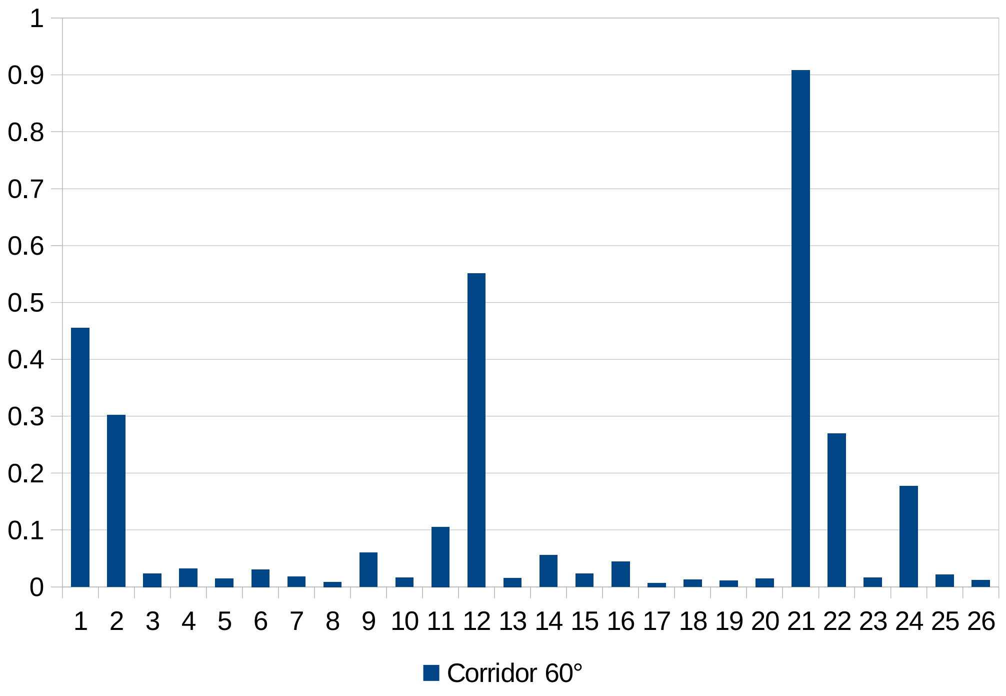


Figure 4.71: Corridor 60° minimum distance between LiDAR and dense clouds (cm).

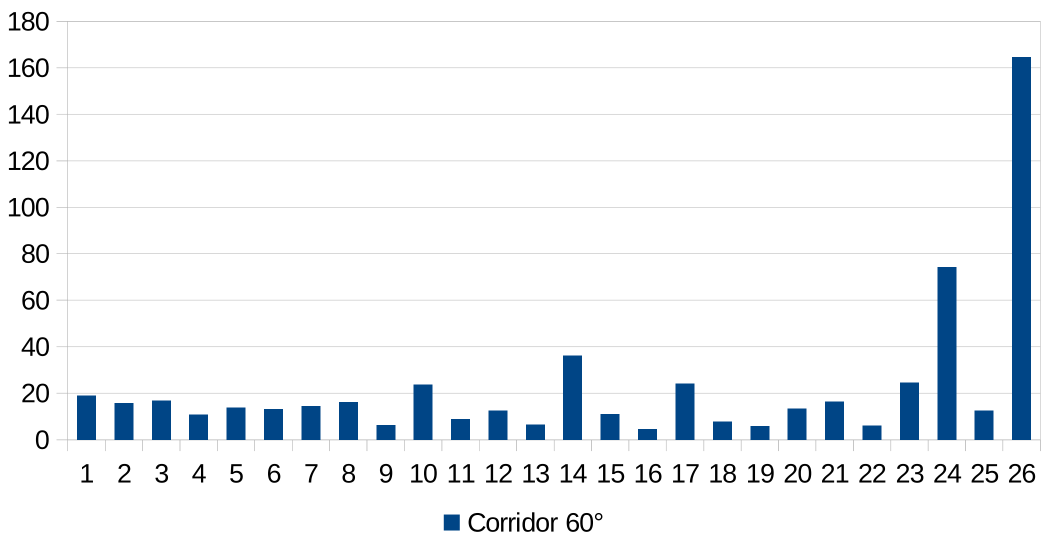


Figure 4.72: Corridor 60° average distance between LiDAR and dense clouds (cm).

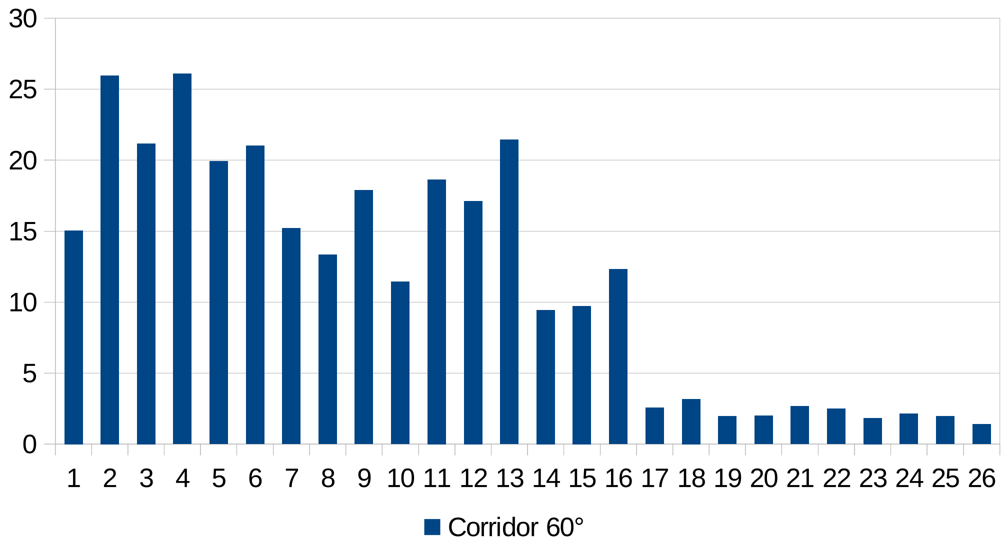


Figure 4.73: Corridor 60° distance between dense and sparse KP cloud (cm).

The twenty sixth cloud has a particularity high average distance. We checked the single dense cloud and found out that it has a extremely low number of points positioned far away from the LiDAR cloud. Therefore, we suppressed the radius outliers filter for that cloud and verified that the resulting cloud had a low density and were not well recontacted.

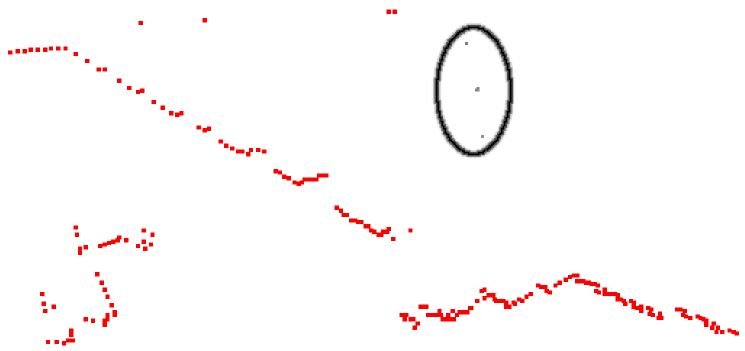


Figure 4.74: Detail of the 26th cloud. We circled the points remaining in the cloud after the filter.

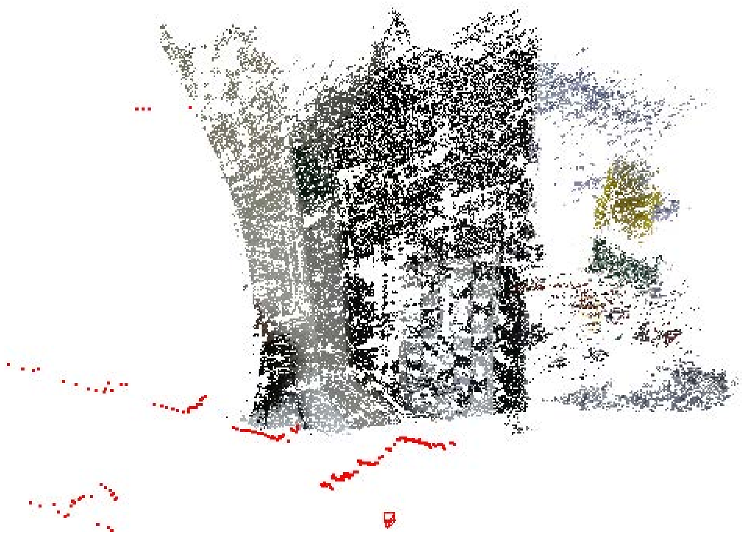


Figure 4.75: 26th Cloud without filter.

As we show in Figure 4.76, the un-rectified disparity map of the twenty sixth frame is too uniform. Usually, maps like this one are produced when the maximum disparity value is too high compared to the movement of the rover.

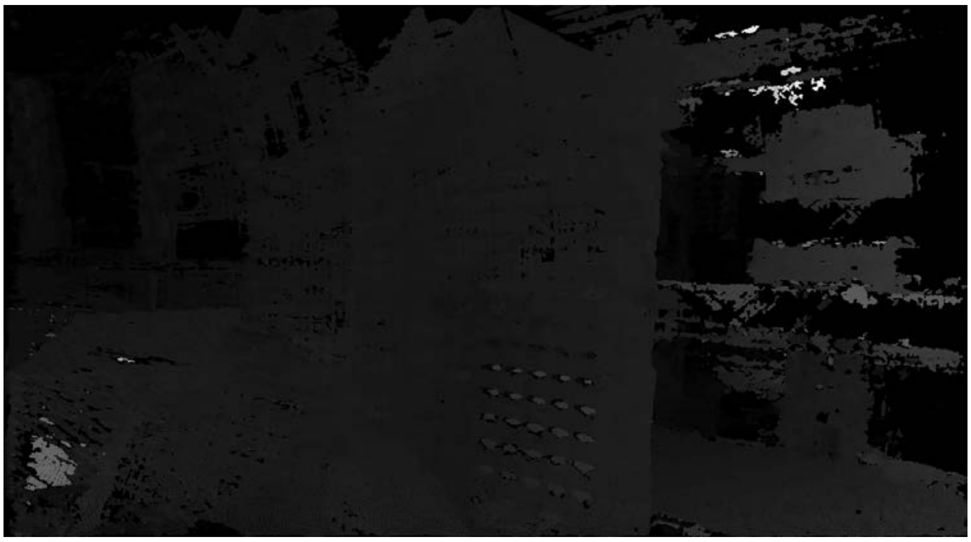


Figure 4.76: Un-rectified disparity map for the 26th cloud.

As in the linear rectification results, the high values of distance between the features clouds and the dense clouds, are due to the dense and the features cloud not mapping the same portion of the images in the 3D environment (Figure 4.77).

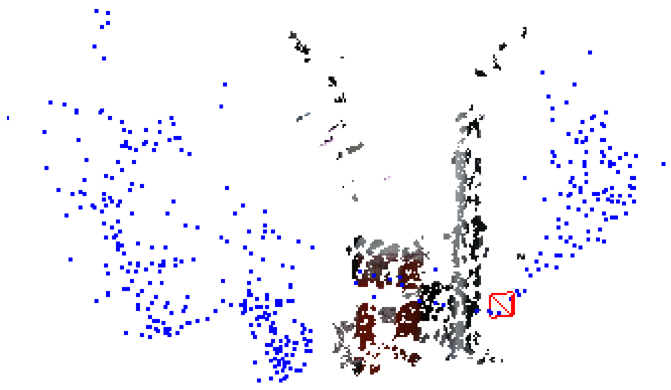


Figure 4.77: Detail of the feature and dense cloud not mapping the same points in the 3D environment.

4.2.2 Polar Rectification: Corridor 45°

The features are correctly matched and filtered (Figures from 4.78 to 4.80).

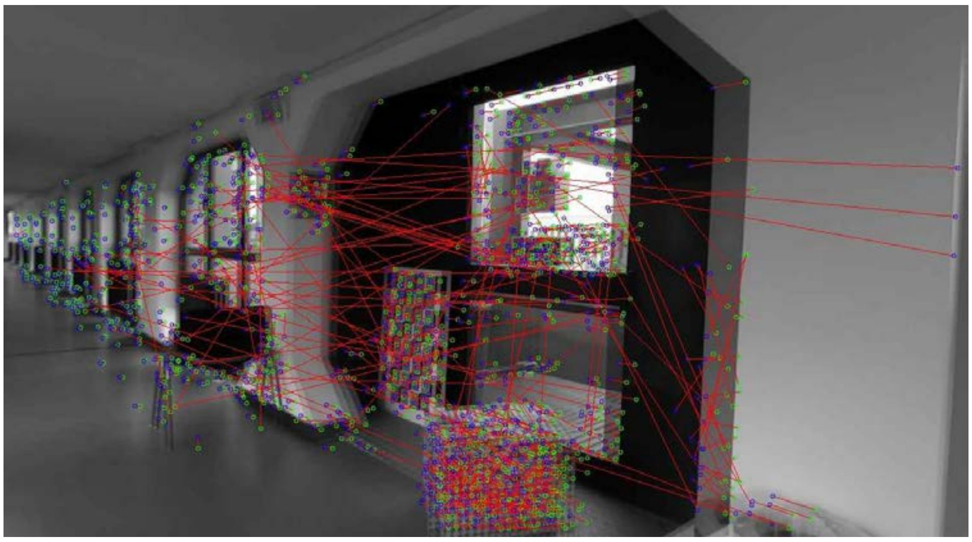


Figure 4.78: Corridor 45° unfiltered matches.

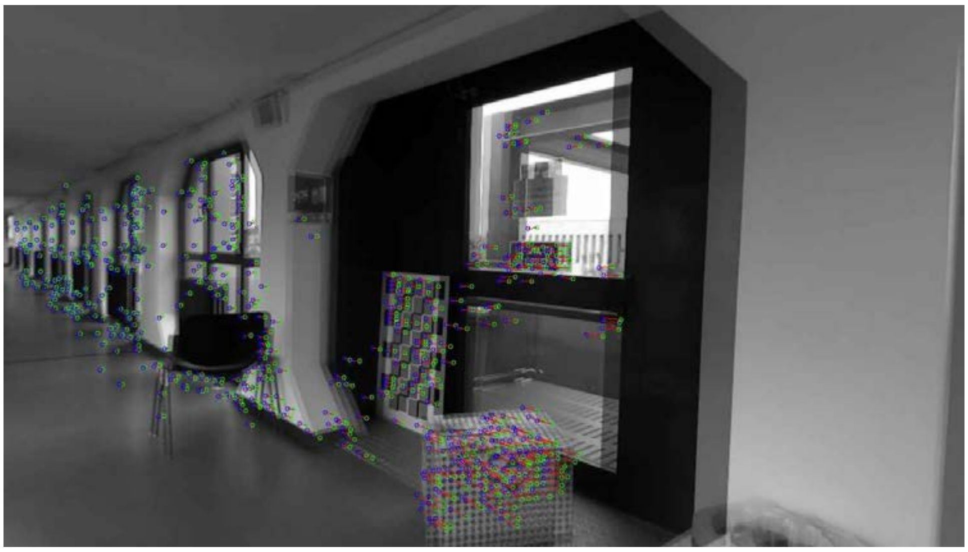


Figure 4.79: Corridor 45° filtered matches.

The number of features is lower than the previous case, but they are still numerous and accurate enough to build a sparse cloud from the key points.

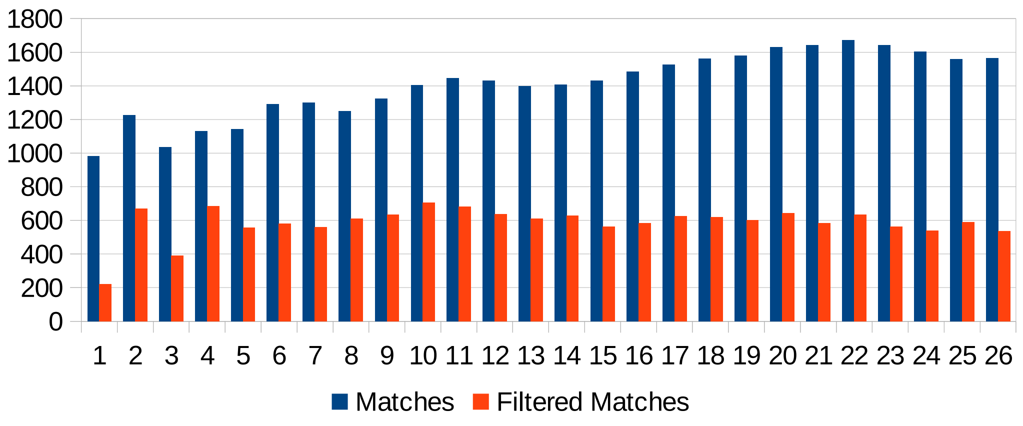


Figure 4.80: Corridor 45° number of features before and after the filters are applied.

This sequence is pretty similar to the 60° one. However, the different orientation of the camera produces two noticeable effects: the component of movement in the optical axis direction is more pronounced, and the depth of the scene is higher than the previous case.

The higher is the forward moving of the camera, the nearer is the epipolar point to the image will be. Therefore, in some of the pairs analyzed in this sequence, the epipolar points are inside the pictures. In those cases, the deformation of the rectified pairs is more pronounced, as we can see in Figure 4.83. The algorithm is still able to correct their epipolar geometry, but the dense point clouds suffer from the distortion.

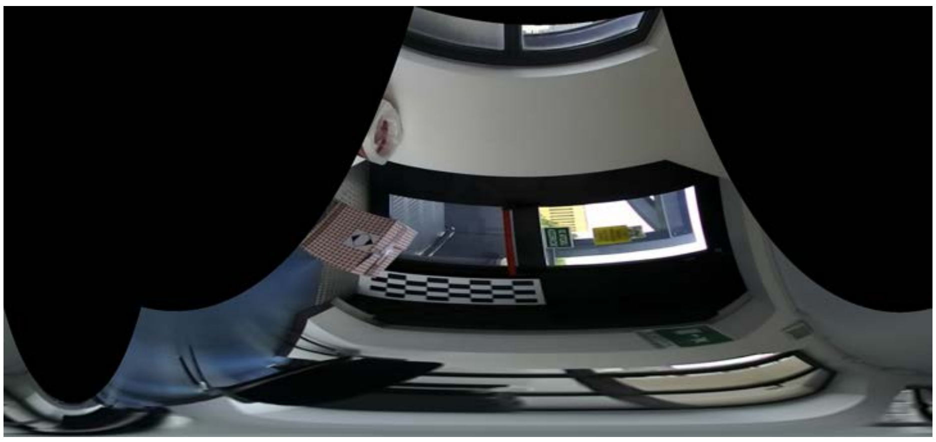


Figure 4.81: Frame deformed by polar rectification. Image rotated by 90°.

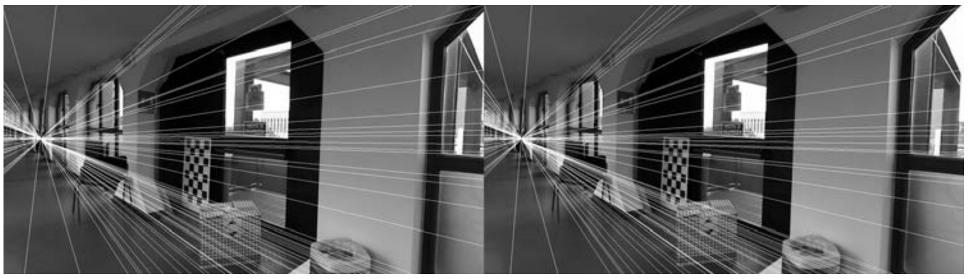


Figure 4.82: Corridor 45° image pair with epipolar points inside the images.

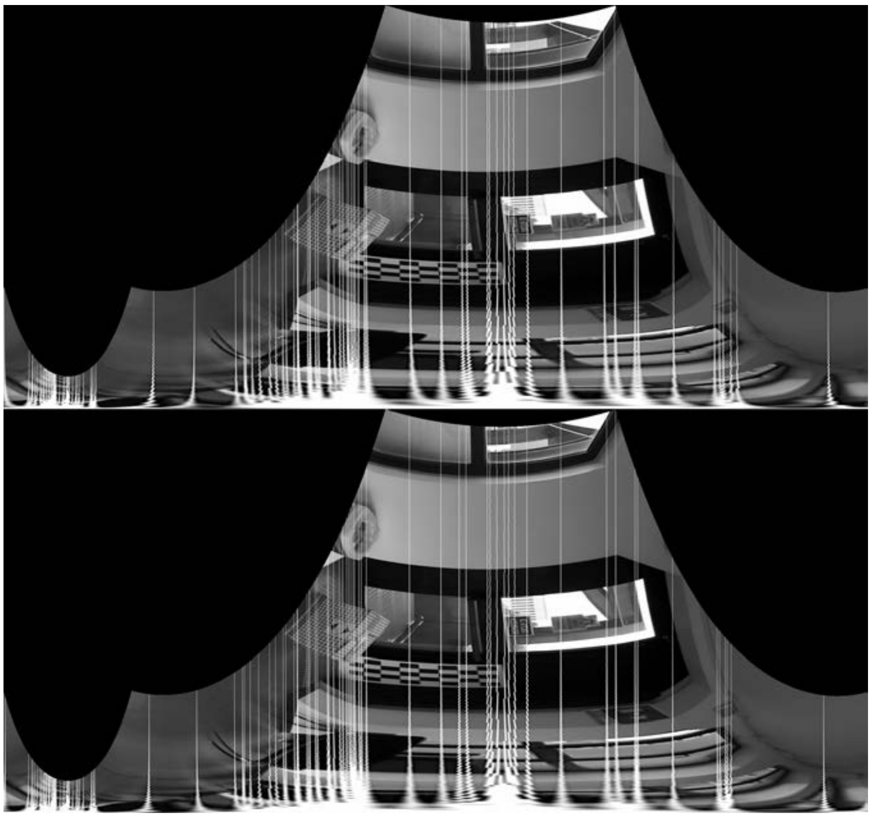


Figure 4.83: Corridor 45° the same pair rectified. The epipolar lines are distorted in the above picture due to the polar deformation. Image rotated by 90°.

As we already discussed, the deep of the scene affects the maximum disparity parameter of the block matching algorithm. Therefore, a high maximum disparity is required to find an accurate disparity map.

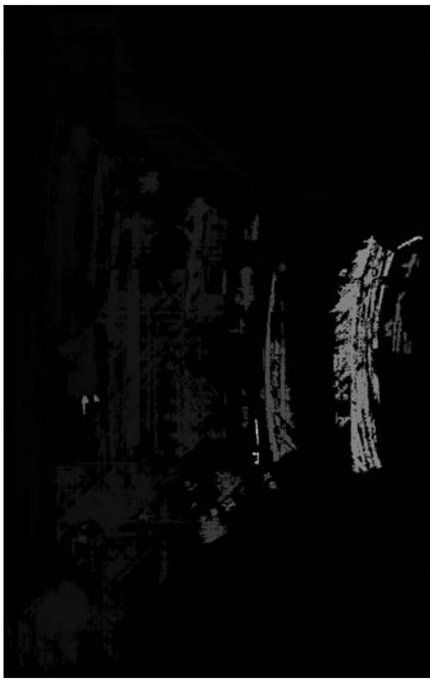


Figure 4.84: Block size of 1

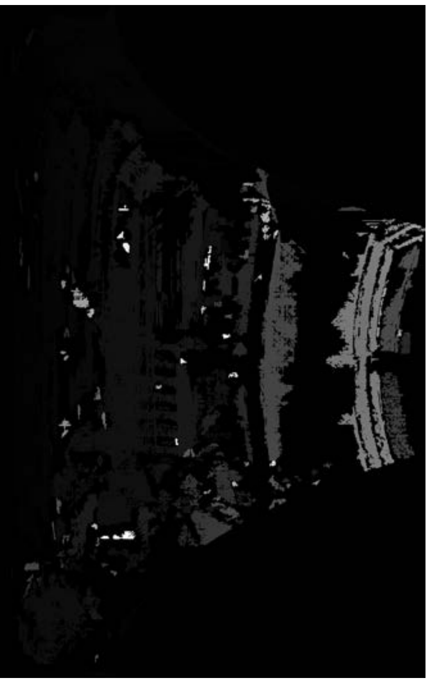


Figure 4.85: Block size of 3.

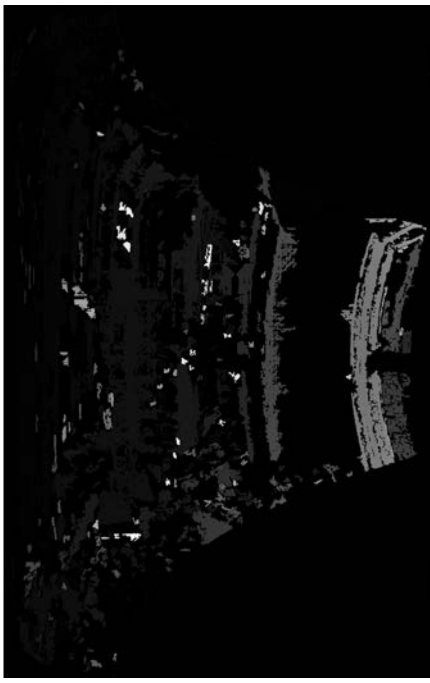


Figure 4.86: Block size of 5.

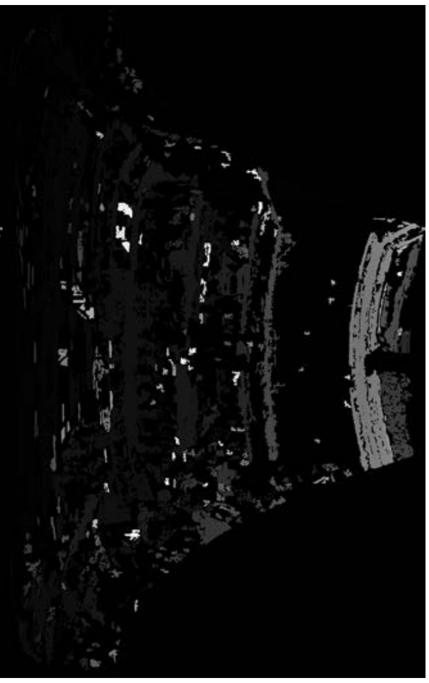


Figure 4.87: Block size of 7.

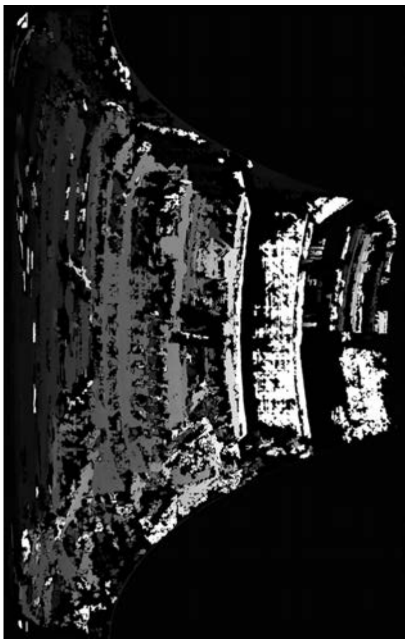


Figure 4.88: Maximum disparity of 16*5.

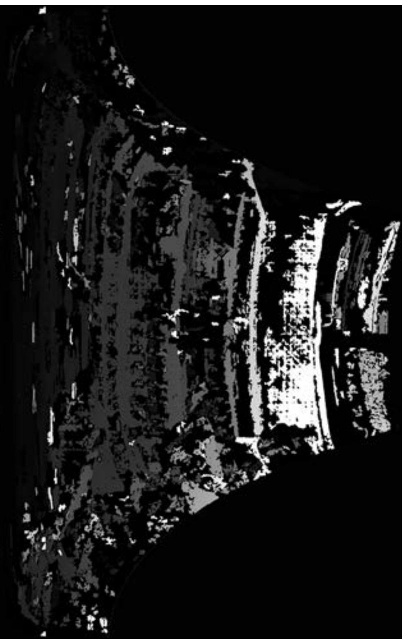


Figure 4.89: Maximum disparity of 16*9.

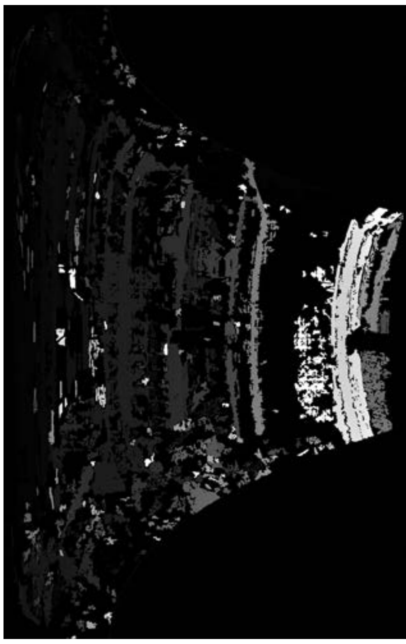


Figure 4.90: Maximum disparity of 16*15.

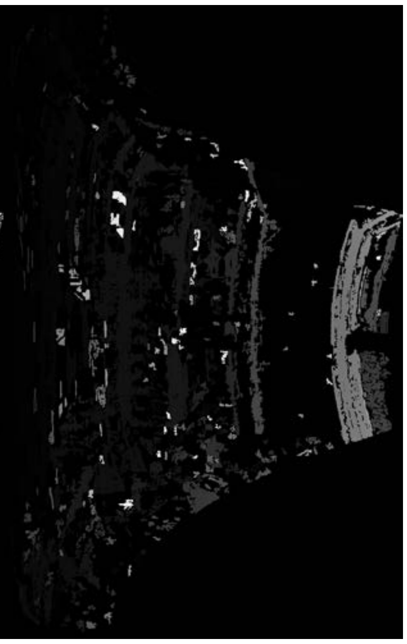


Figure 4.91: Maximum disparity of 16*27.

	Computational time of Block Matching per cycle (s)			
Max. Disparity	16x5	16x9	16x15	16x27
Corridor 60° sequence	1.09	2.56	3.74	6.72

Table 4.3: Time required for the block matching

While still struggling to find matching textures in the images, the algorithm is able to correctly and densely match enough of the images to build the dense clouds. These clouds suffer from the same problems we already identified in the previous case. However, in this sequence, they became more significant due to an epipolar geometry more difficult to solve. Therefore, the resulting dense clouds are more distorted than the previous case with bigger distances between subsequent clouds and from the reference clouds.

As in the 60° sequence, the clouds seems well reconstructed and well registered from a favorable point of view (Figure from 4.94 to 4.101). The dense and sparse clouds overlap pretty well. However, when seen from the side (Figures from 4.106 to 4.109) or from a higher position (Figures from 4.102 to 4.105), the distortion becames apparent. Nevertheless, the dense clouds correctly position the box, and the column as can be seen in Figure 4.92.

The distances between clouds suffer from the same problems we discussed in previous sequences:

- A high distance between the dense cloud and the feature cloud is due to frames where the features do not map the same objects in the 3D environment.
- The average distance between the LiDAR and dense cloud is high when the dense cloud contain few points away from the LiDAR cloud. This clouds corresponds to frames where the disparity is calculated with an incorrect maximum disparity value.
- The deformations that the polar method introduces in the images generate distortions and bad reconstruction. This negatively impact the clouds registration. Therefore, the clouds do not overlap properly.

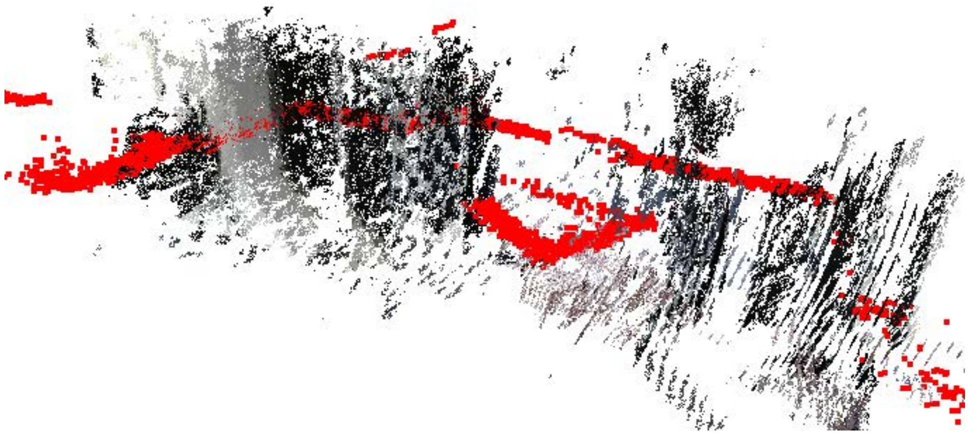


Figure 4.92: Corridor 45° detail showing the column and the box.

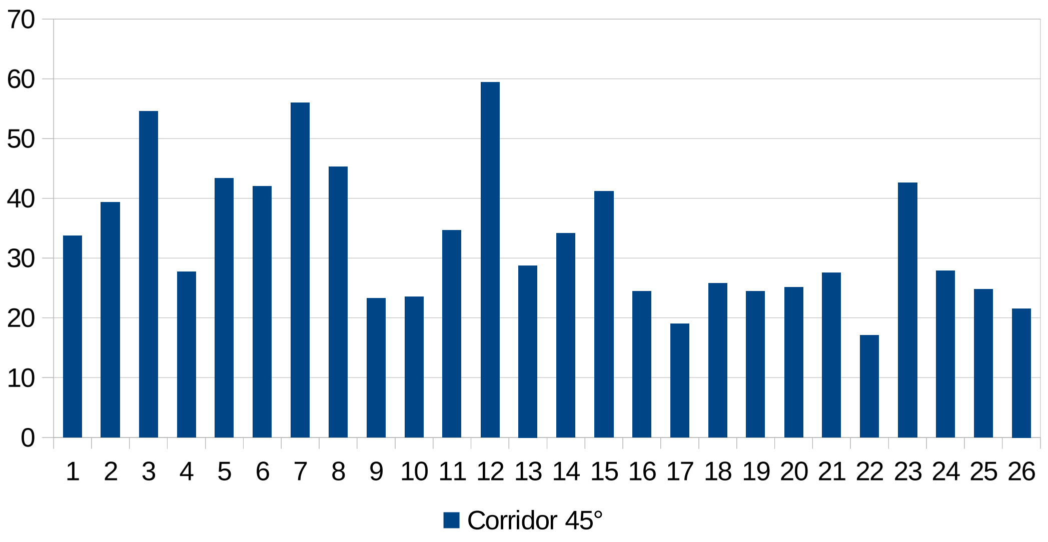


Figure 4.93: Corridor 45° Density of the dense point clouds

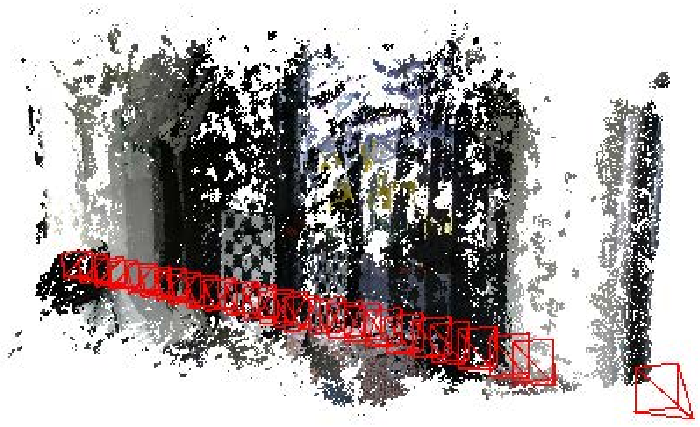


Figure 4.94: Corridor 45° dense clouds. View 1.

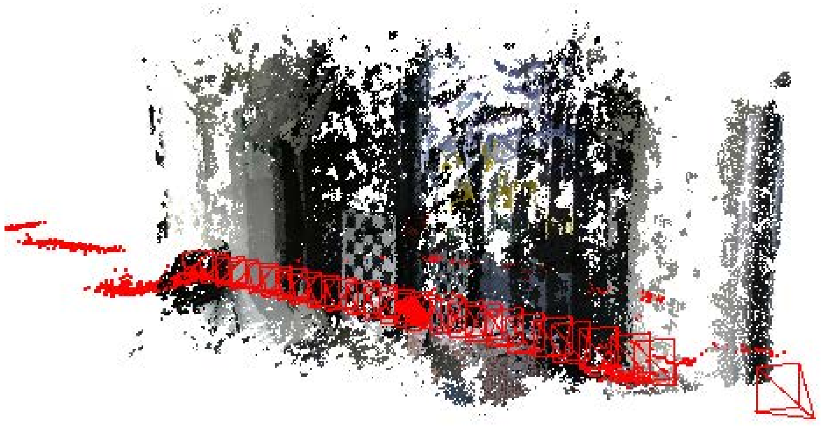


Figure 4.95: Corridor 45° dense clouds and LiDAR cloud. View 1.

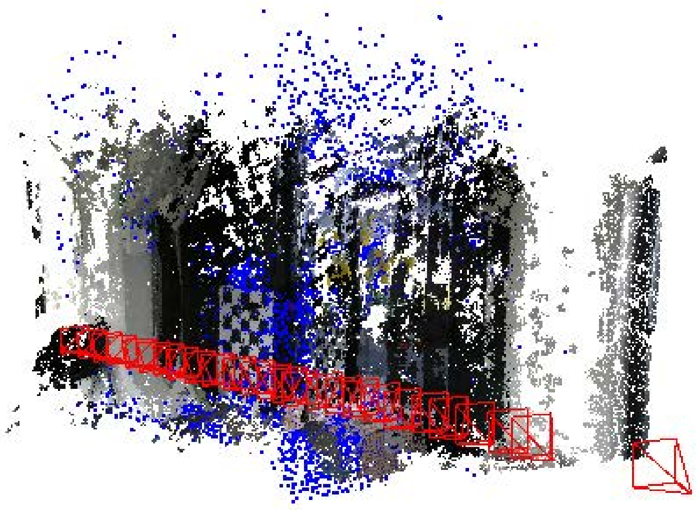


Figure 4.96: Corridor 45° dense clouds and features cloud. View 1.

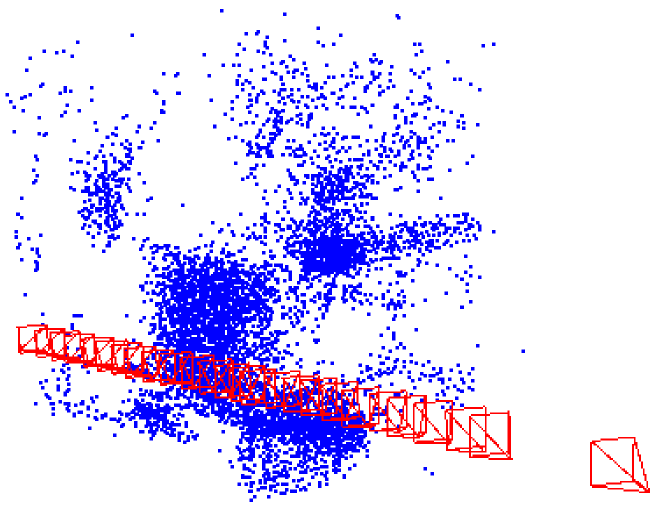


Figure 4.97: Corridor 45° Features cloud. View 1.

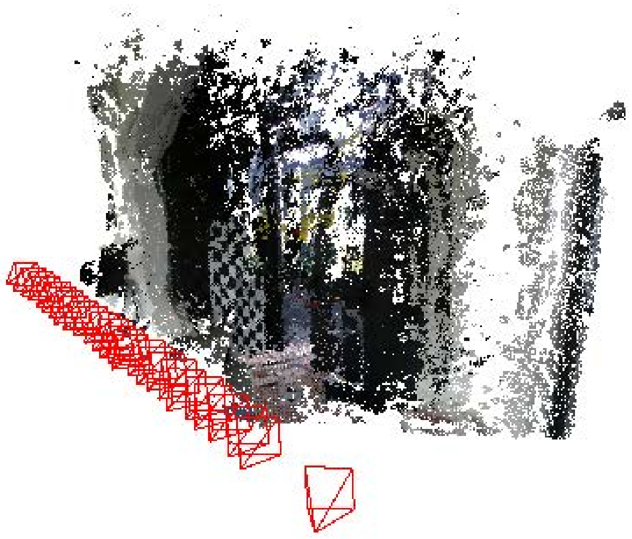


Figure 4.98: Corridor 45° dense clouds. View 2.

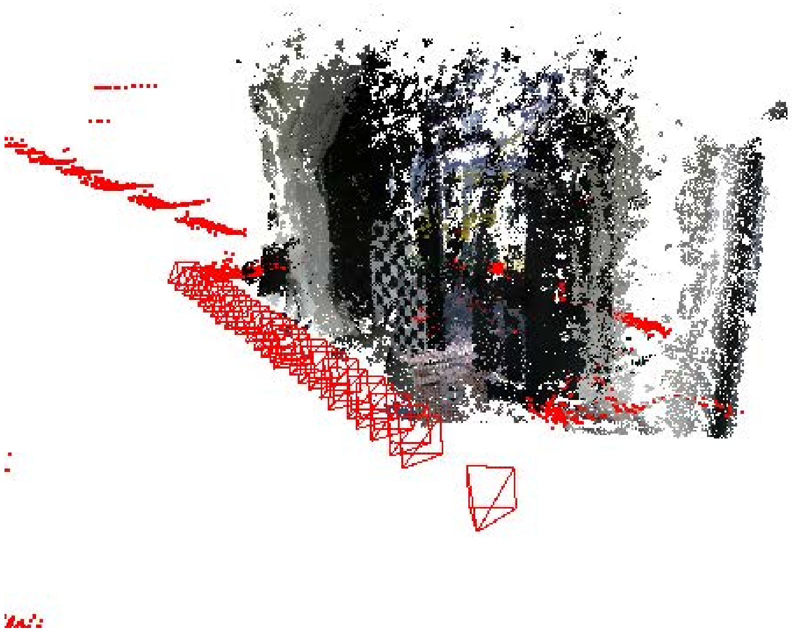


Figure 4.99: Corridor 45° dense clouds and LiDAR cloud. View 2.

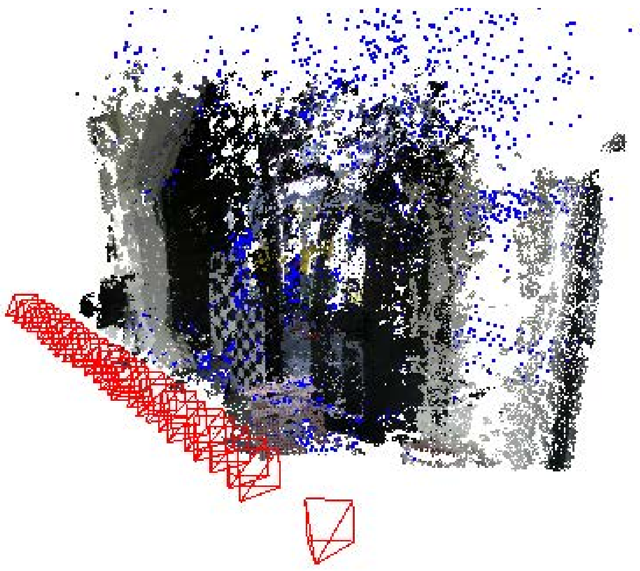


Figure 4.100: Corridor 45° dense clouds and Features cloud. View 2.

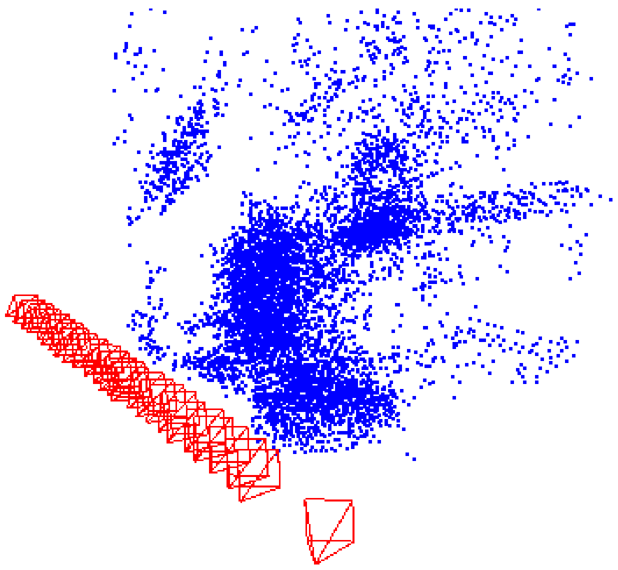


Figure 4.101: Corridor 45° Features cloud. View 2.

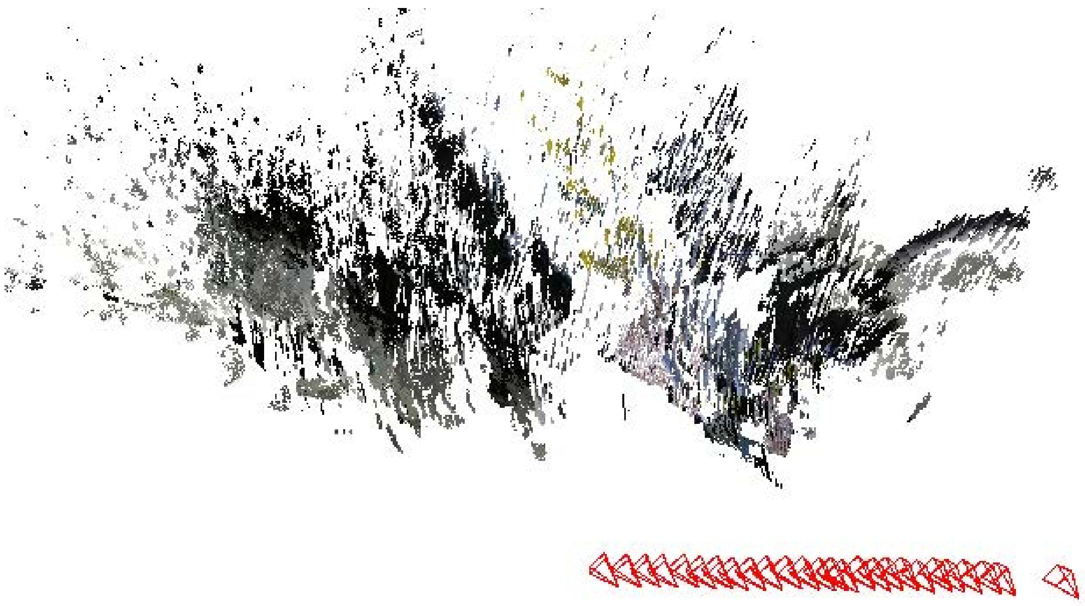


Figure 4.102: Corridor 45° dense clouds seen from above. View 3.

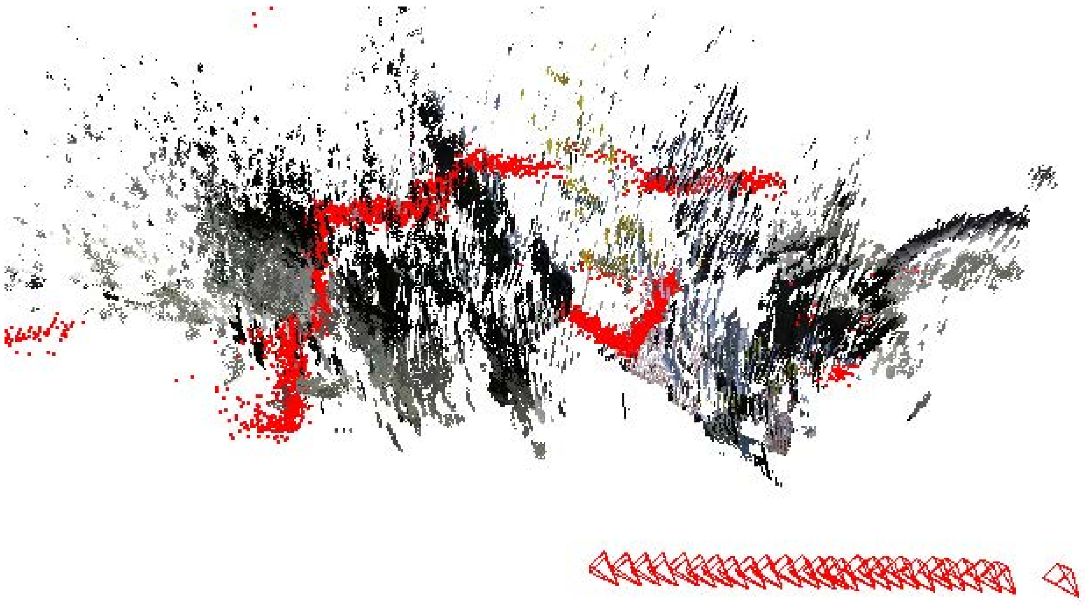


Figure 4.103: Corridor 45° dense clouds and LiDAR cloud seen from above. View 3.

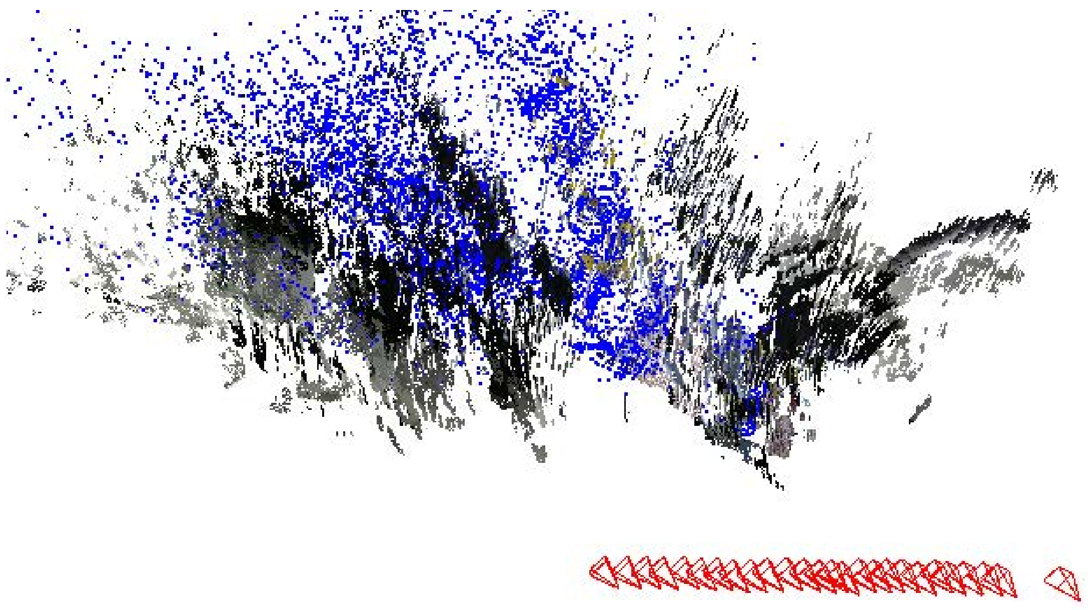


Figure 4.104: Corridor 45° dense clouds and Features cloud seen from above. View 3.

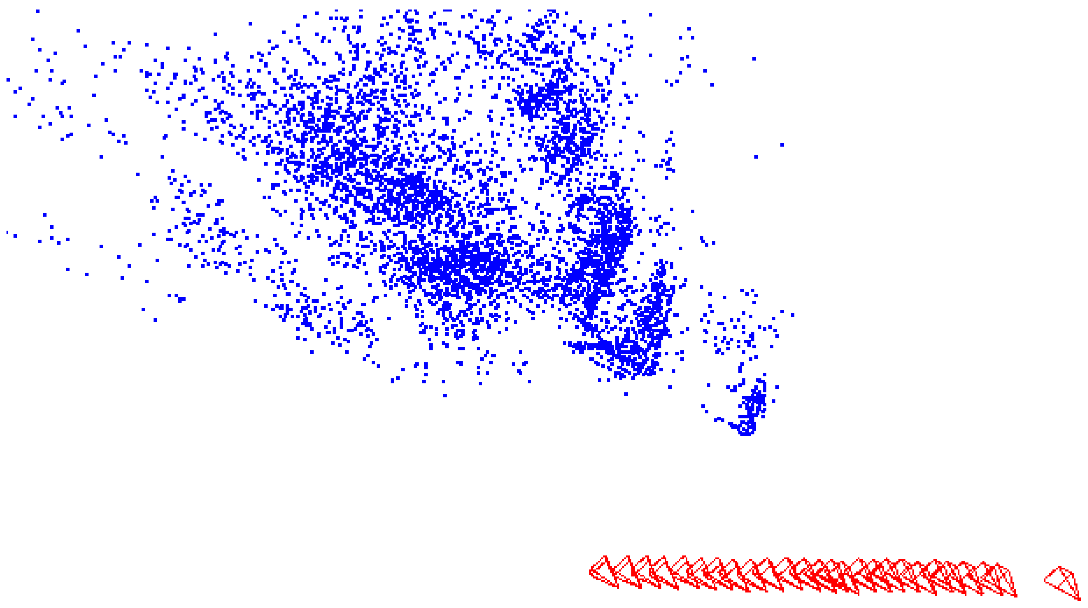


Figure 4.105: Corridor 45° Features cloud seen from above. View 3.

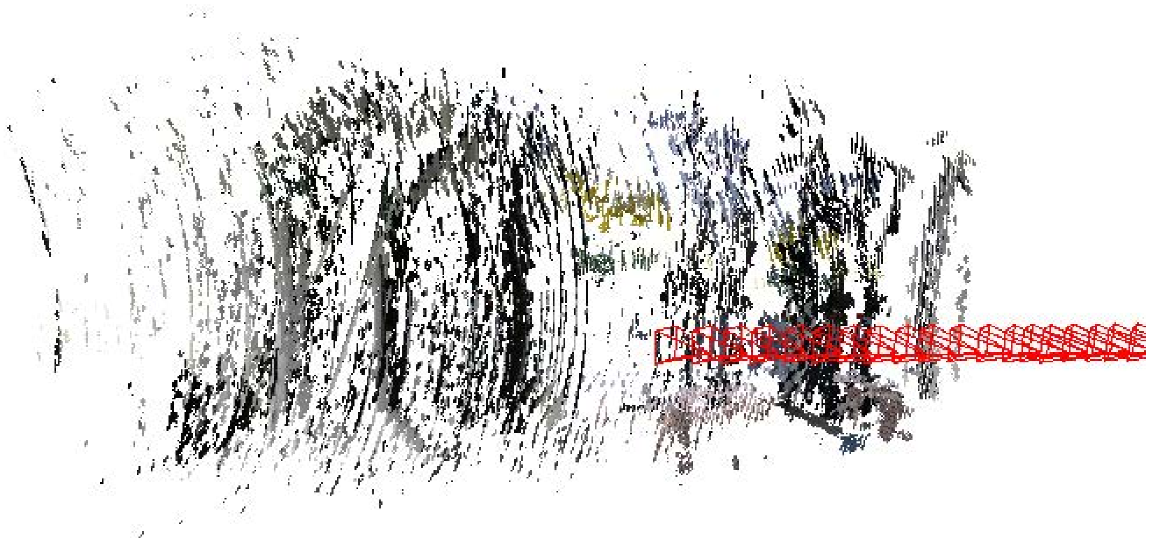


Figure 4.106: Corridor 45° dense clouds seen from the side. View 4.

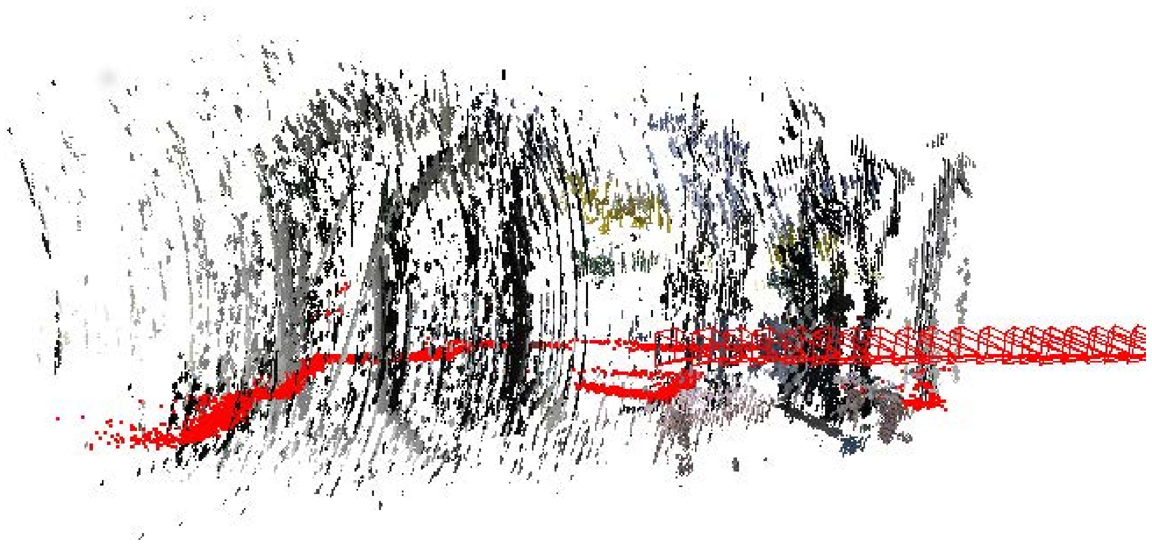


Figure 4.107: Corridor 45° dense clouds and LiDAR cloud seen from the side. View 4.

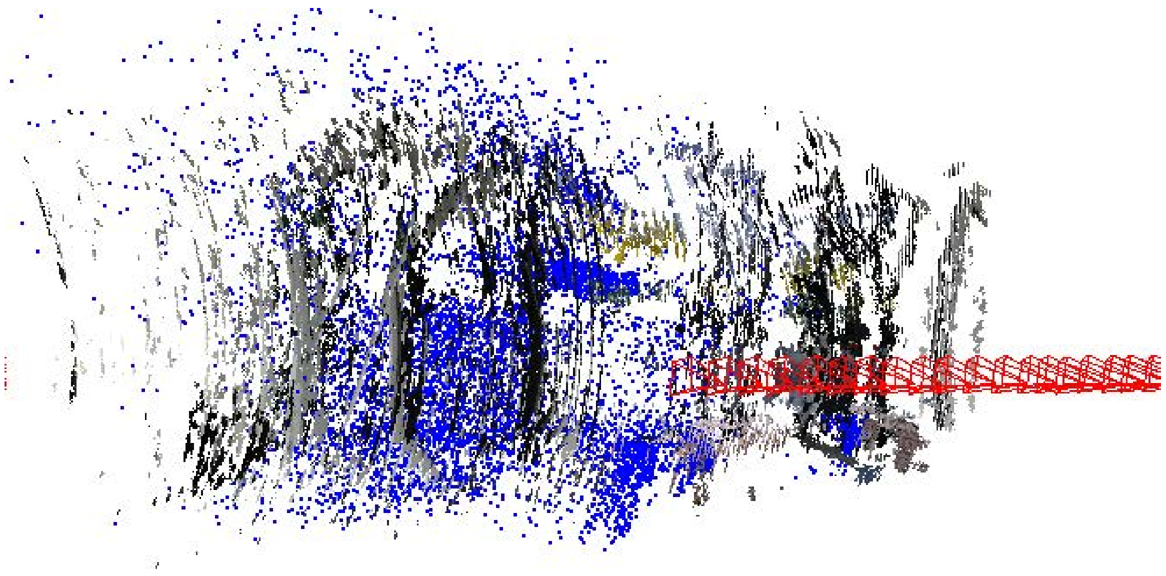


Figure 4.108: Corridor 45° dense clouds and features cloud seen from the side. View 4.

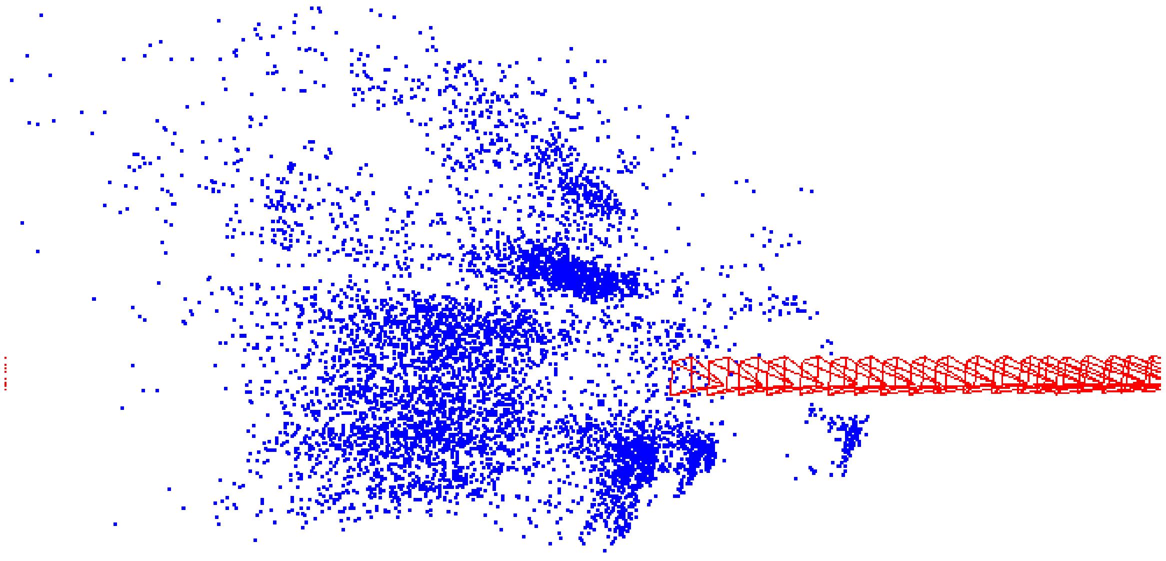


Figure 4.109: Corridor 45° Features cloud seen from the side. View 4.

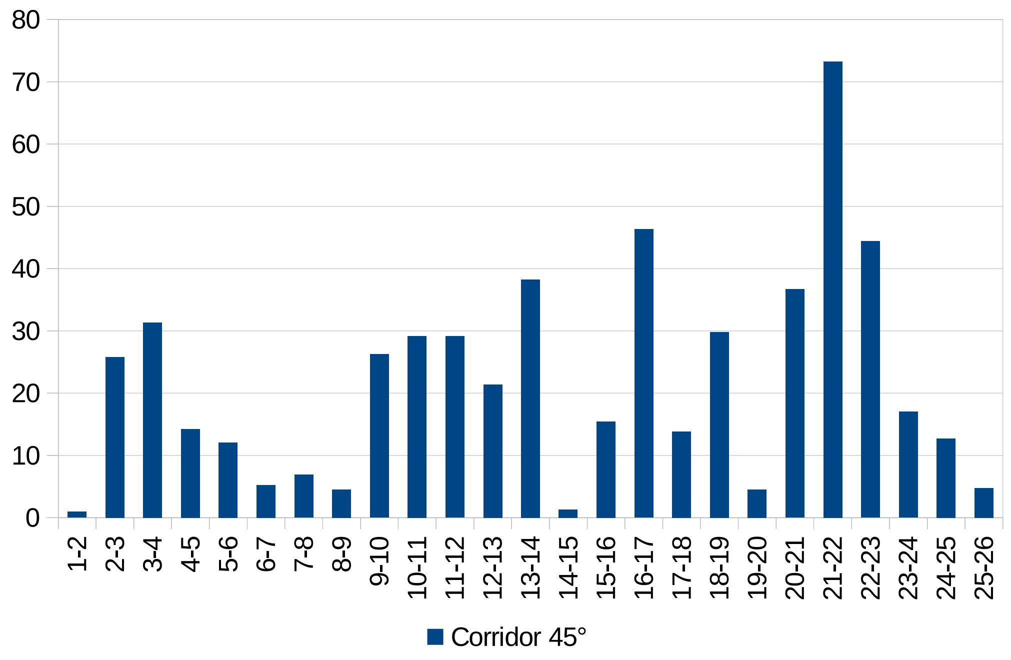


Figure 4.110: Corridor 45° distance between subsequents dense clouds (cm).

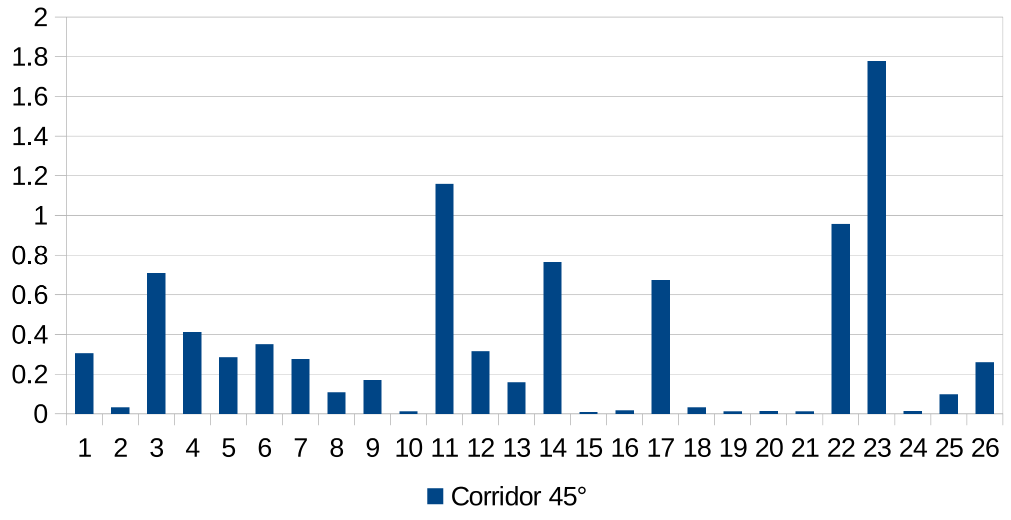


Figure 4.111: Corridor 45° minimum distance between dense clouds and LiDAR cloud (cm).

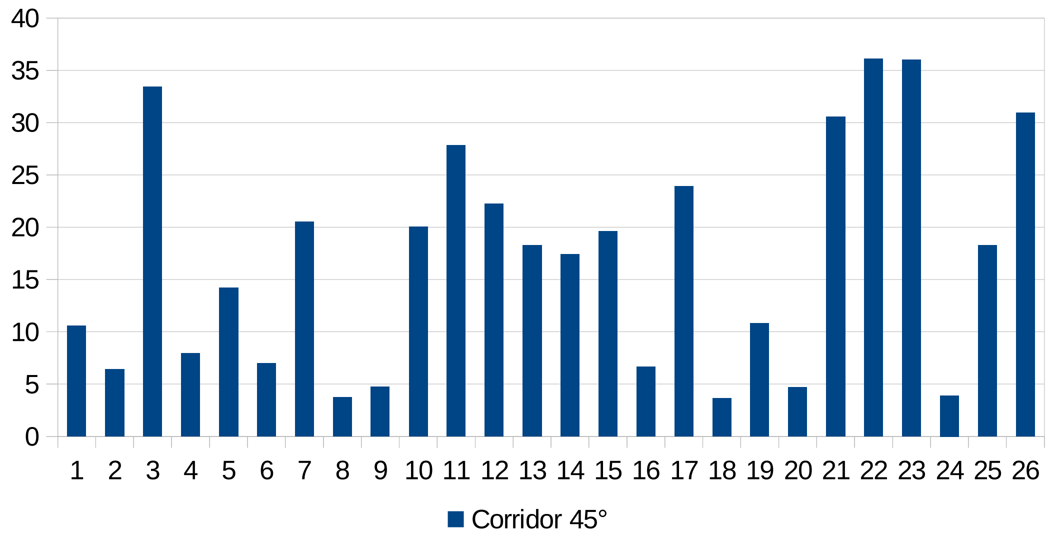


Figure 4.112: Corridor 45° average distance between dense cloud and LiDAR cloud (cm).

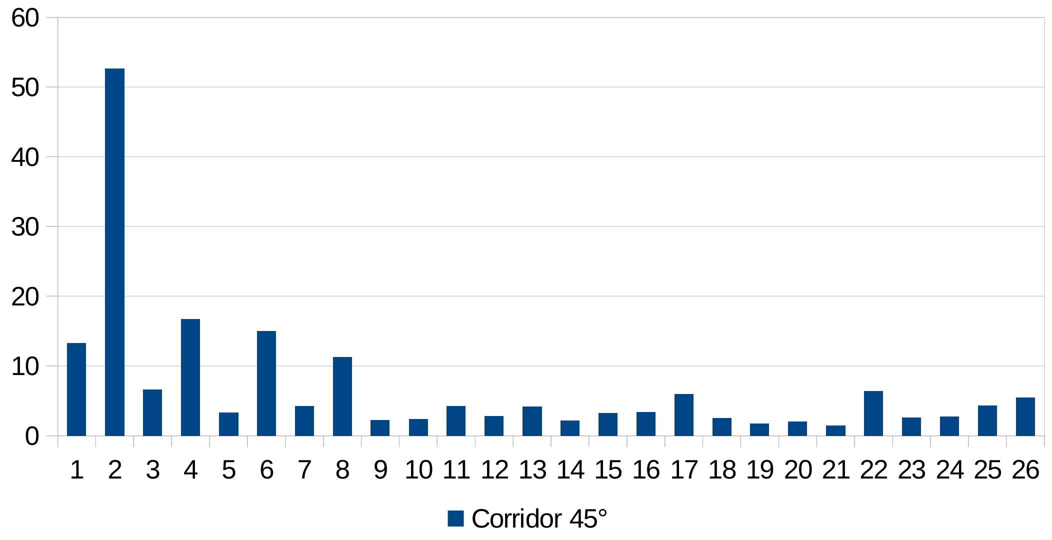


Figure 4.113: Corridor 45° distance between dense clouds and Features cloud (cm).

4.2.3 Polar Rectification Corridor 0°

This sequence was gathered by a camera with the optical axis in the direction of movement of the rover. Therefore, the epipolar geometry is particularly hard to solve, with epipolar points inside the frames for every pair. Moreover, the sequence was gathered in another corridor, with less distinguishable texture on its walls.

Obviously, the features detection and matching is not impacted by this. The features found and matched are still accurate (Figure 4.114 to 4.116).

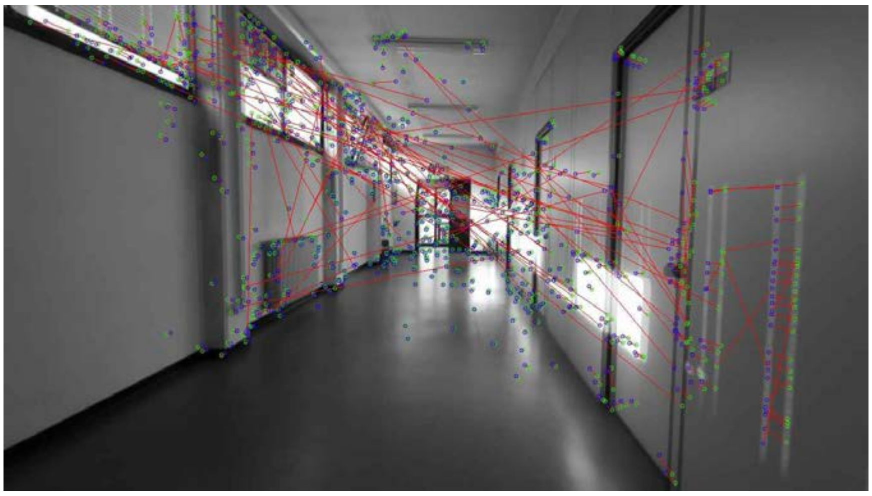


Figure 4.114: Corridor 0° Unfiltered features matches.

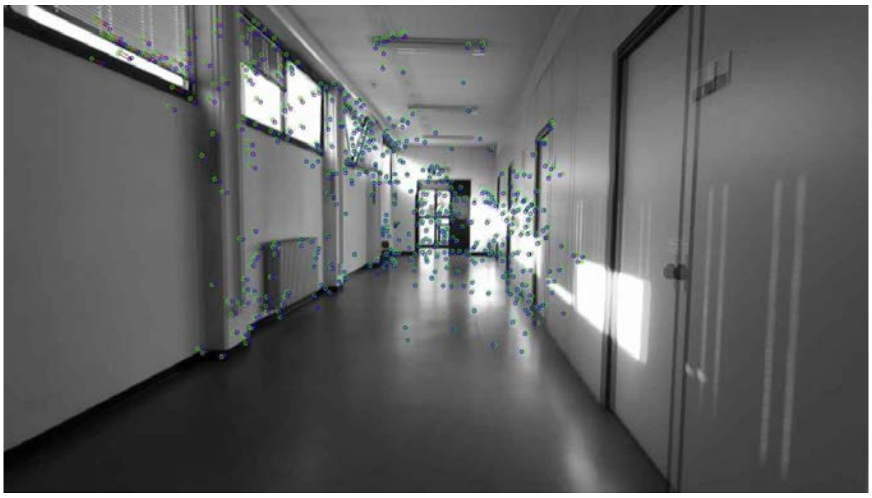


Figure 4.115: Corridor 0° filtered features.

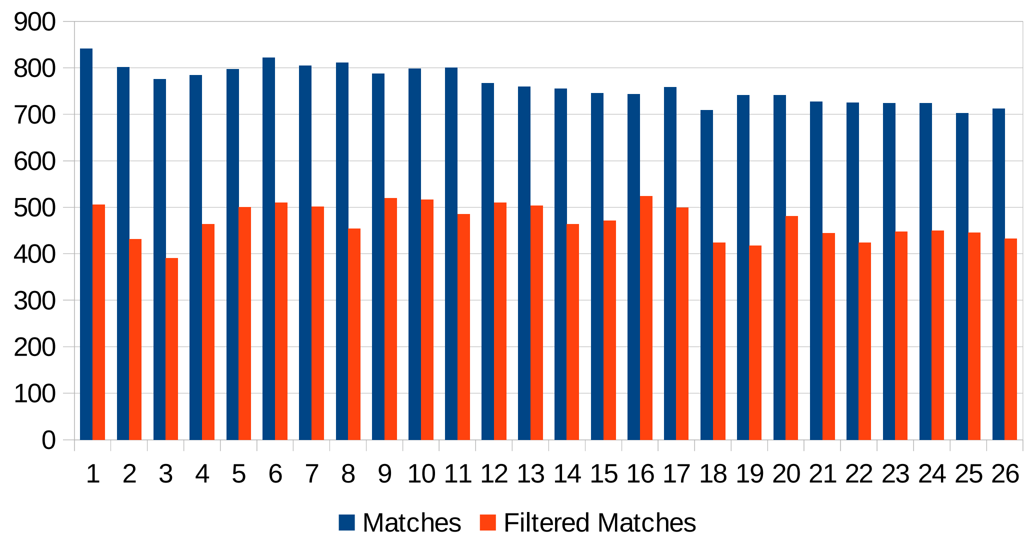


Figure 4.116: Corridor 0° number of features before and after the filters are applied.

The rectification process is deeply impacted by epipolar configuration. The resulting images are extremely distorted as we show in Figures 4.118 and 4.119.

The subsequent block matching process struggles to find correct matching texture in the pairs. This issue is emphasized by the lack of distinctive textures on the walls. Therefore, the dense clouds are sparse and not well positioned in the environment. In order to compute adequate results, we had to remove the down-sampling, thus the following data is obtained from cloud at 100% density.

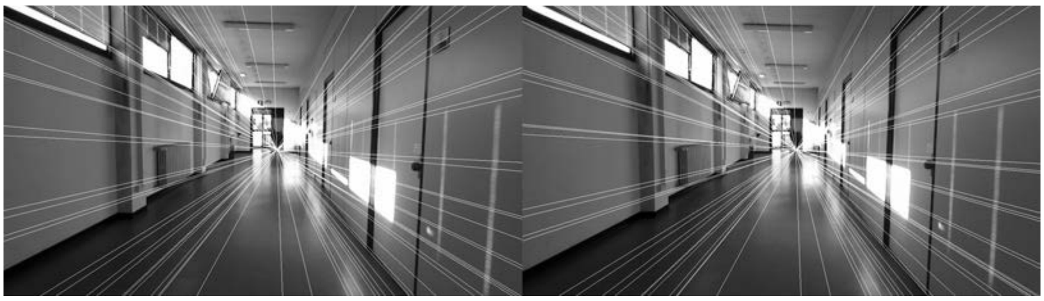


Figure 4.117: Corridor 0° Epipolar lines.

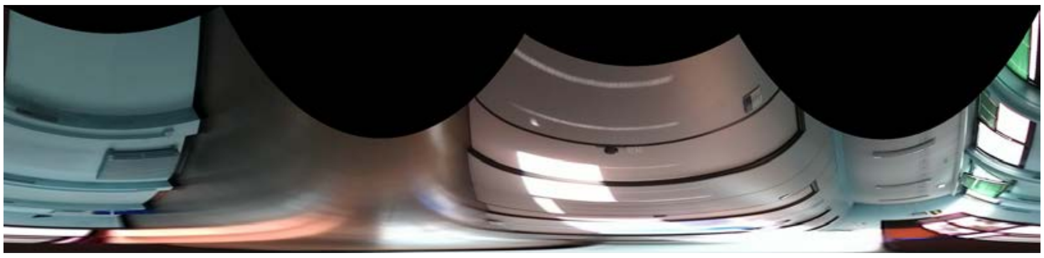


Figure 4.118: Corridor 0° image after rectification. Rotated 90°.

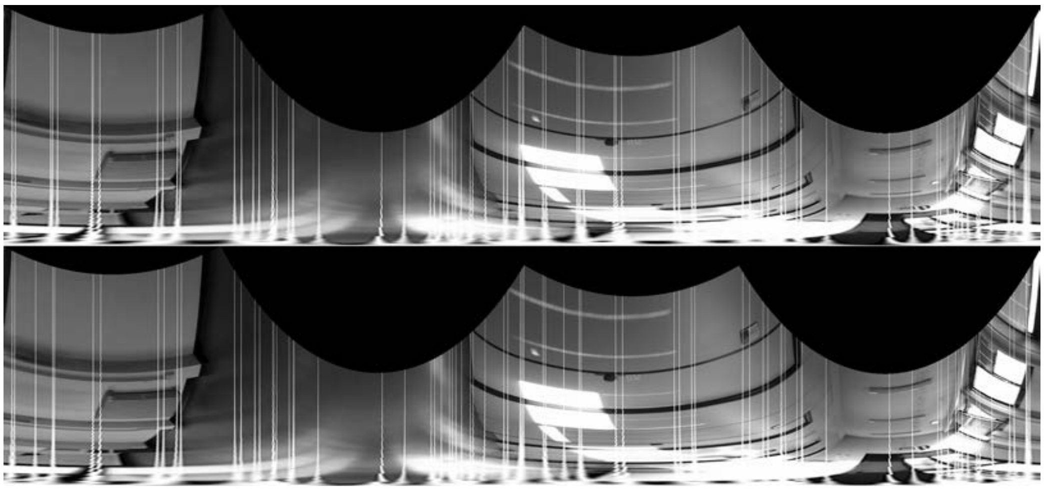


Figure 4.119: Corridor 0° Epipolar lines in rectified pair. Rotated 0°.

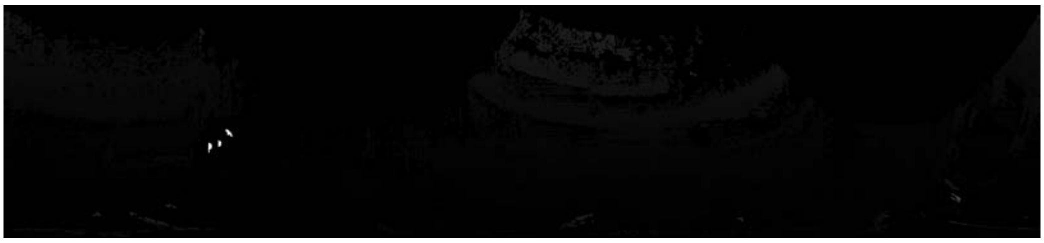


Figure 4.120: Corridor 0° disparity map for the same pair.

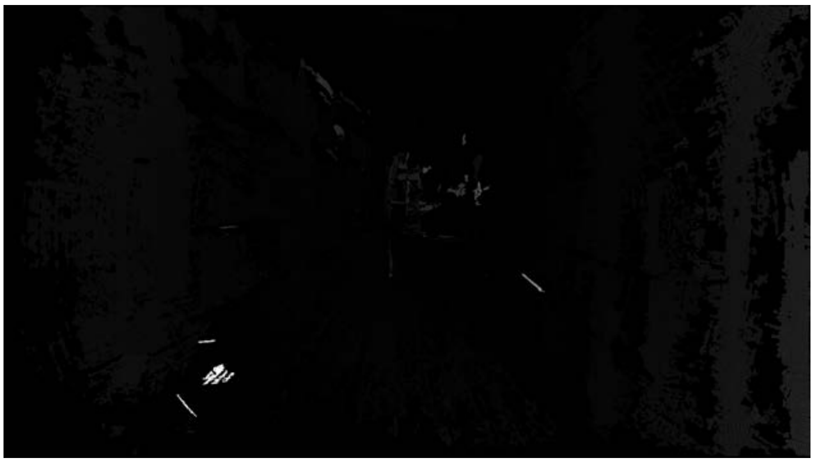


Figure 4.121: Corridor 0° un-rectified disparity map.

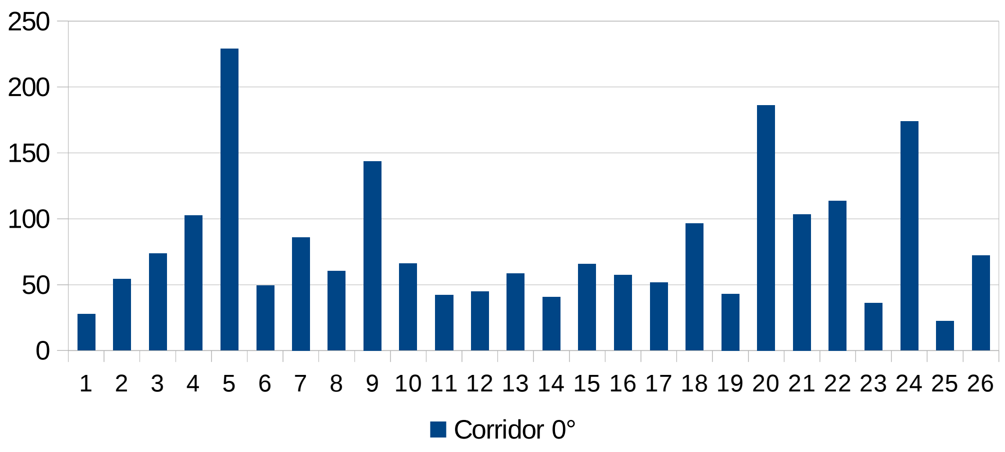


Figure 4.122: Corridor 0° dense cloud density. The clouds are not down-sample in this case.

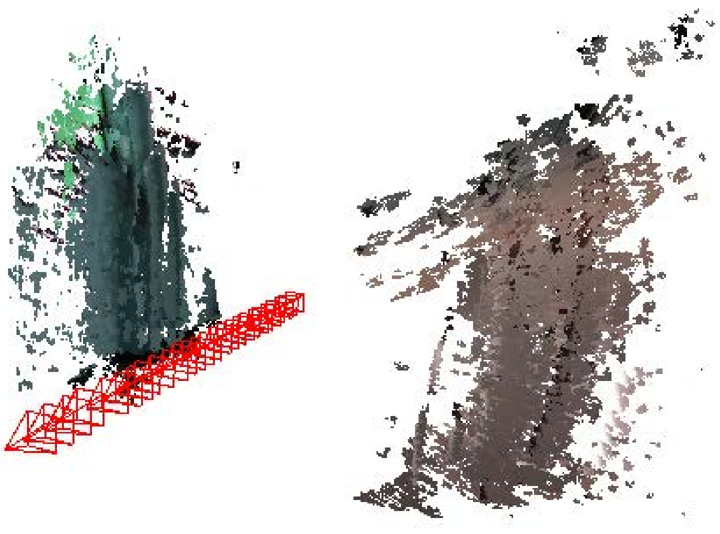


Figure 4.123: Corridor 0° dense cloud. View 1. The clouds are not down-sample in this case.

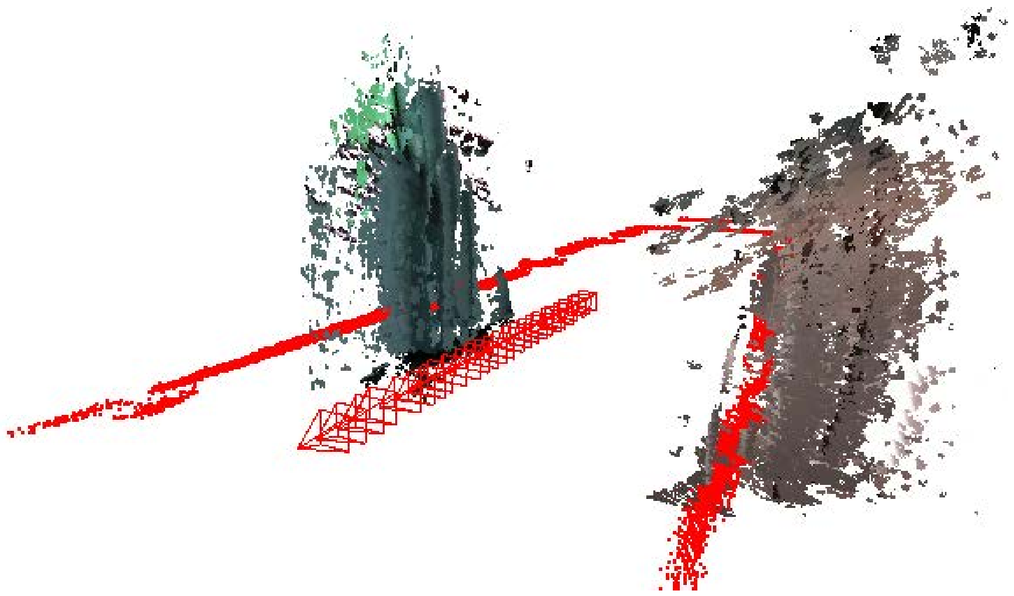


Figure 4.124: Corridor 0° dense cloud and LiDAR cloud. View 1. The clouds are not down-sample in this case.

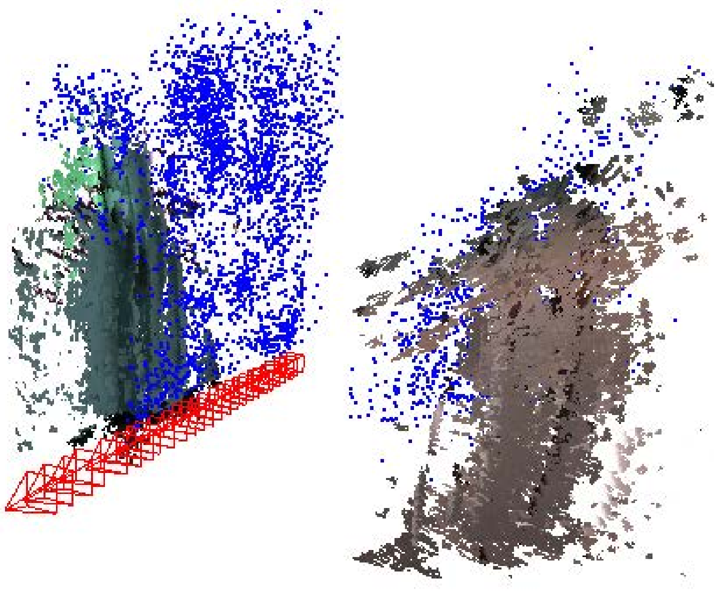


Figure 4.125: Corridor 0° dense cloud and features cloud. View 1. The clouds are not down-sample in this case.

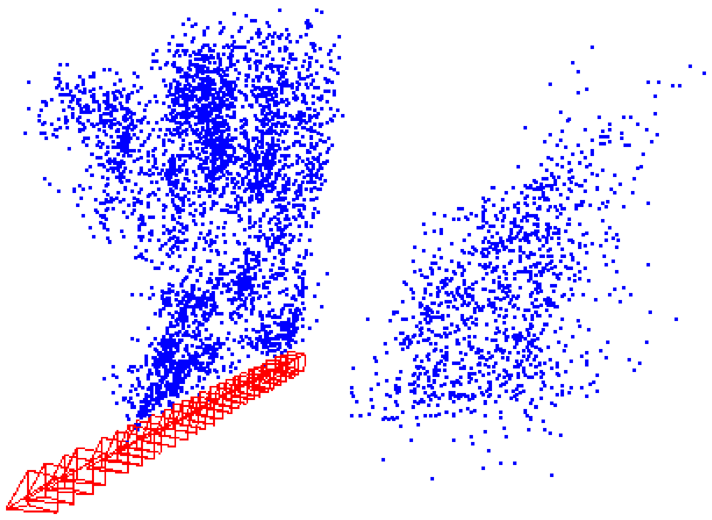


Figure 4.126: Corridor 0° features cloud. View 1. The clouds are not down-sample in this case.

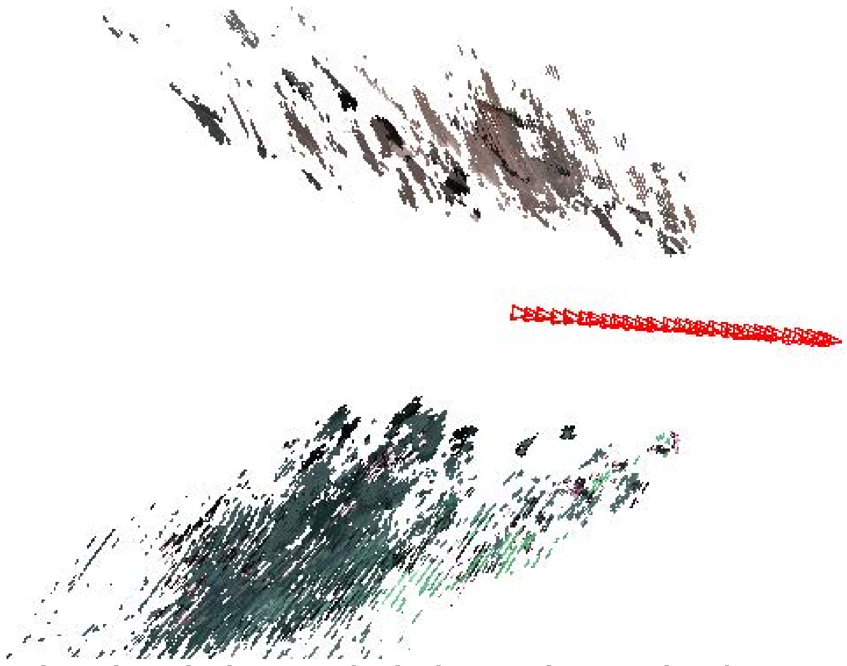


Figure 4.127: Corridor 0° dense cloud. View 2. The clouds are not down-sample in this case.

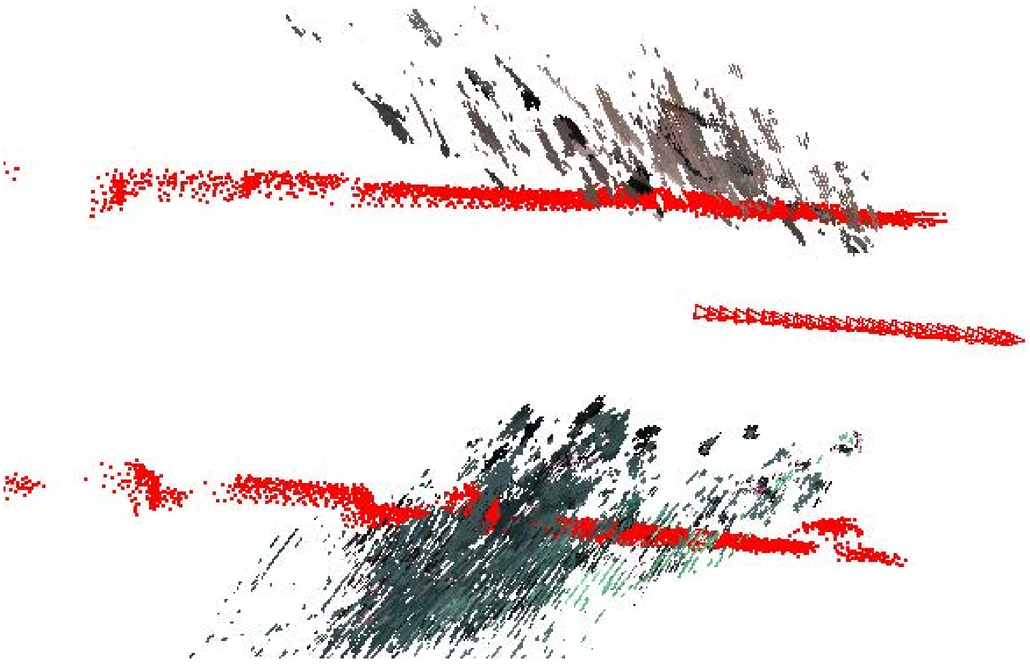


Figure 4.128: Corridor 0° dense cloud and LiDAR cloud. View 2. The clouds are not down-sample in this case.

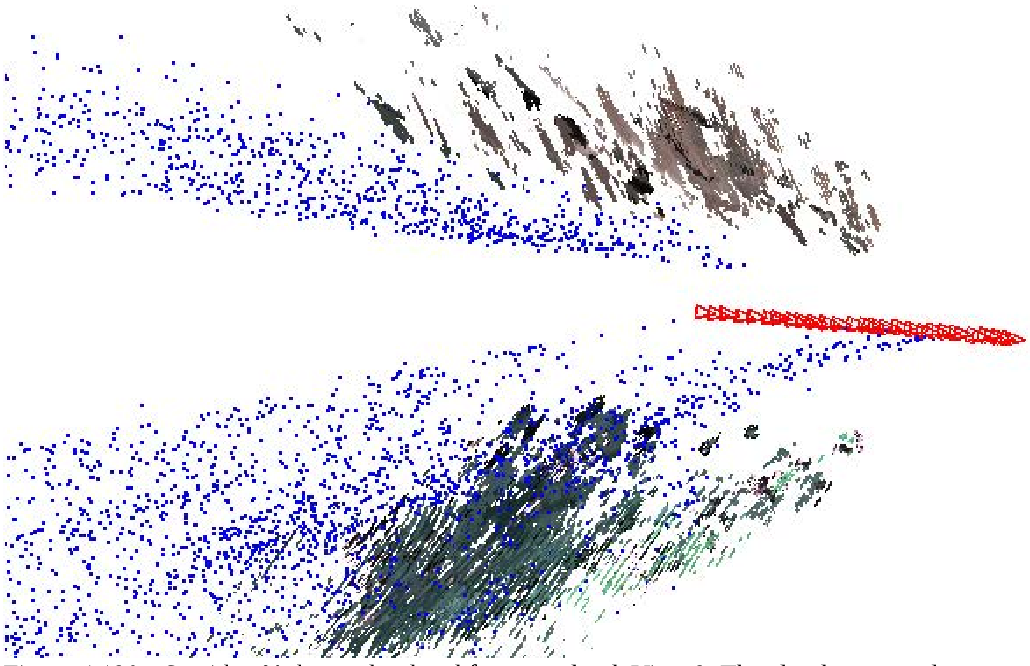


Figure 4.129: Corridor 0° dense cloud and features cloud. View 2. The clouds are not down-sample in this case.

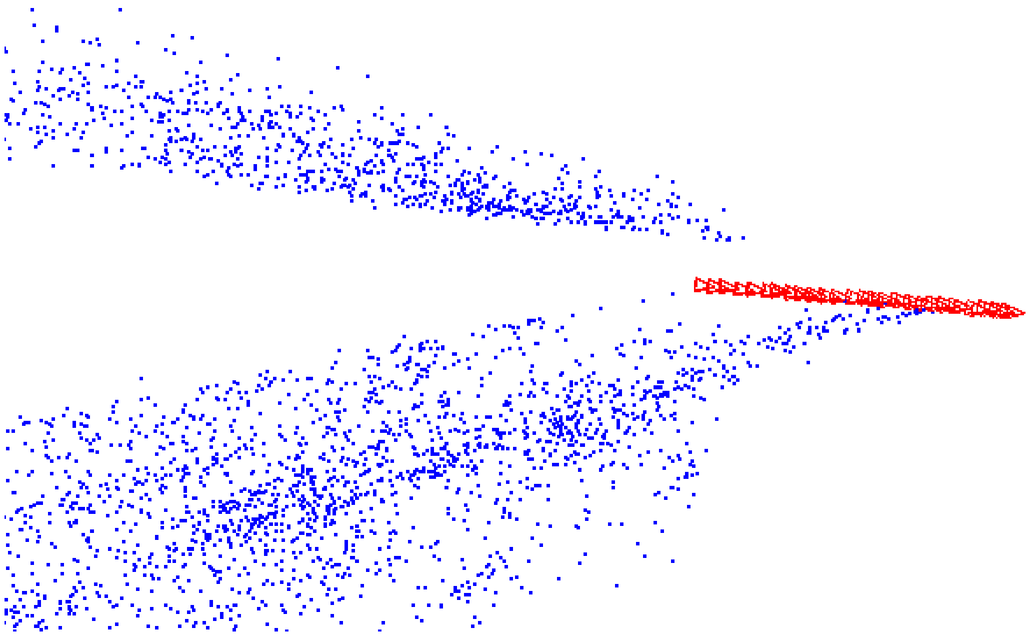


Figure 4.130: Corridor 0° features cloud. View 2. The clouds are not down-sample in this case.

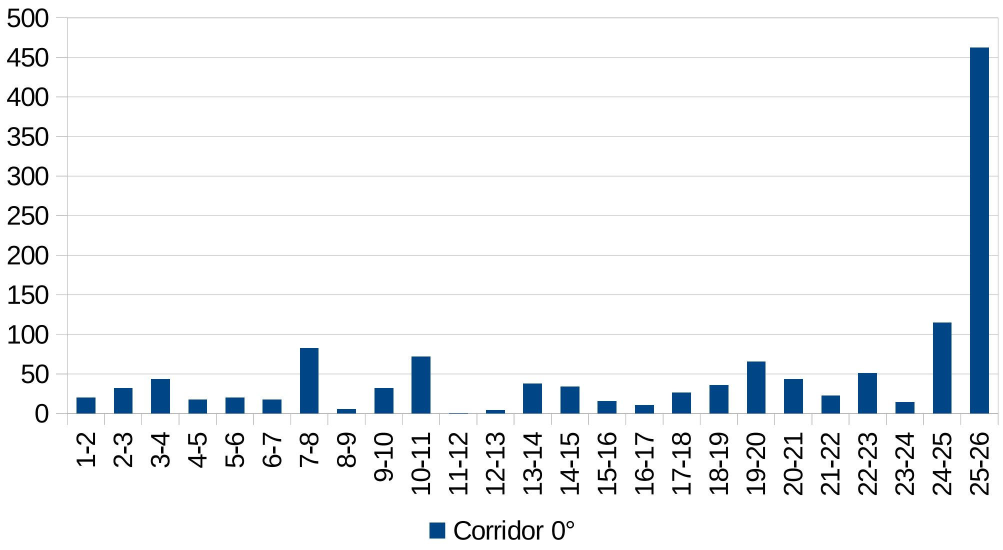


Figure 4.131: Corridor 0° distance between subsequents dense clouds (cm). The clouds are not down-sample in this case.

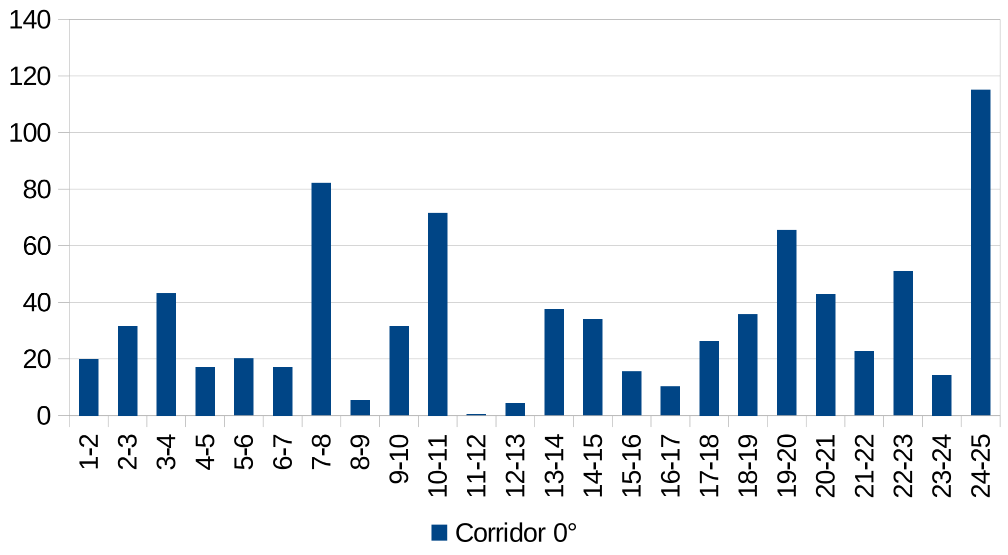


Figure 4.132: Corridor 0° same distances of Figure 4.131 the data from the last pair of clouds was removed to improve the readability of the rest of the data (cm). The clouds are not down-sample in this case.

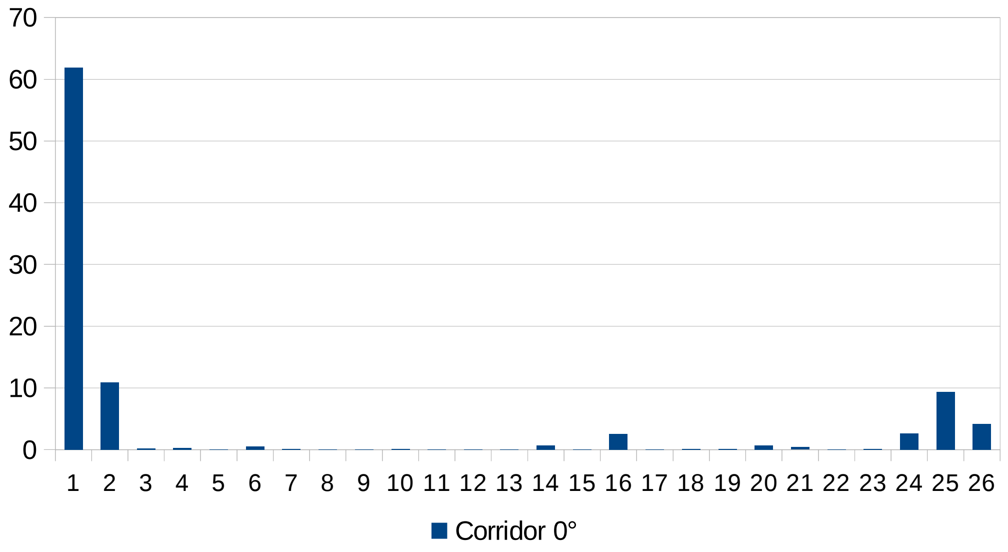


Figure 4.133: Corridor 0° minimum distance between dense clouds and LiDAR cloud (cm). The clouds are not down-sample in this case.

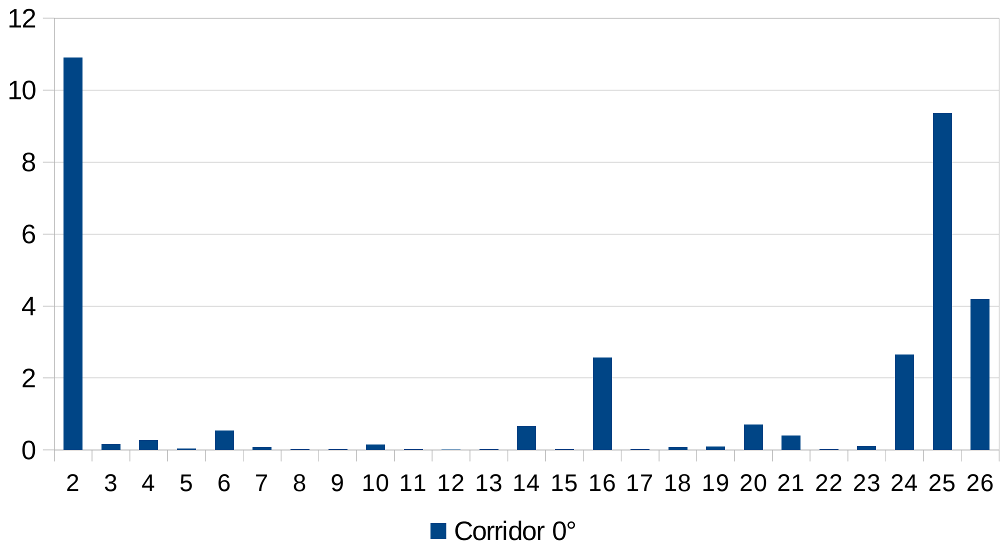


Figure 4.134: Corridor 0° same distances of Figure 4.133, the data from the first cloud was removed to improve the readability of the rest of the data (cm). The clouds are not down-sample in this case.

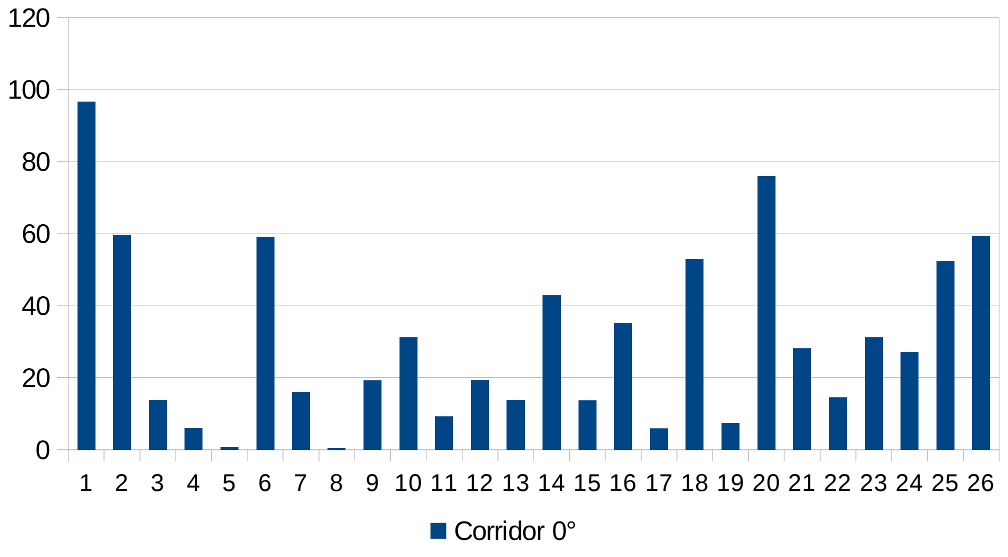


Figure 4.135: Corridor 0° average distance between dense cloud and LiDAR cloud (cm). The clouds are not down-sample in this case.

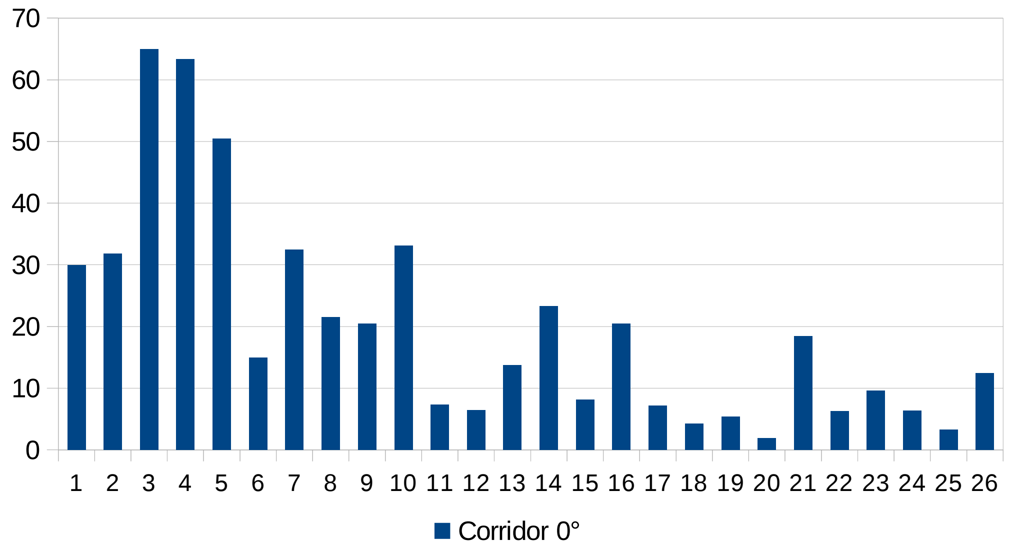


Figure 4.136: Corridor 0° average distance between dense cloud and features cloud (cm). The clouds are not down-sample in this case.

Figures from 4.123 to 4.130 confirm want previously stated. The clouds are sparse, not adequately reconstructed nor adequately registered. Even at full density, we were unable to find a viewpoint from which the clouds seem well reconstructed.

The epipolar geometry in this sequence is too difficult to solve for an adequate reconstruction with the Monocular Dense Reconstruction algorithm.

Conclusions

The Monocular Dense Reconstruction algorithm has been able to successfully reconstruct clouds from sequences, but only where their epipolar geometry was not exceedingly complex to solve. As we have seen in the results concerning the indoor and the outdoor sequences, the algorithm does not have any difficulty in building accurate clouds for cases, when we are in a semi-ideal situation of epipolar geometry.

However, it starts to struggle when the epipolar grow increasingly near to the image planes. In that condition to solve the epipolar geometry, we used the polar rectification method, which is able to rectify the images for any given epipolar geometry. Unfortunately, the resulting rectified images can result too distorted for the block matching algorithm to find adequate matching texture. If this be the case, a matching algorithm with fixed block sizes and parameters does not result adequate.

This distortions heavily impacts the dense reconstruction. The most preeminent example of this negative outcome is exemplified by the corridor 0° sequence, where the epipolar points are almost at the center of the images. In these conditions, the algorithm fails to compute an accurate reconstruction.

Nevertheless, when the angle between the direction of motion and the camera is grater then 0° the algorithm is able to compute more precise clouds. While not absolutely accurate, these clouds represent the environment well enough if we take into consideration a favorable point of view, namely the one of the camera. In particular, these more accurate clouds we have defined are able to reconstruct the position and form of the different the objects seen by the camera.

Another glaring issue in the computed Dense Reconstruction is the lack of a robust registration algorithm. This negatively impacts the distances between subsequent clouds, and in general, the whole reconstruction algorithm.

All things considered, despite being able to densely reconstruct the environment in different possible configurations, this algorithm offers a good number of possible improvements. We can mainly suggest two of them: a more suitable matching algorithm to compute the disparity, and a registration algorithm able to use the computed data from the entire sequence -or a limited number of sequences at least- to correct the position of the clouds.

Regarding the reconstruction problem we identify three possible methods:

1. Using the information from the computed dense clouds.
2. Following corresponding matching points in the images planes.
3. Computing a corrective transformation using the information from the Feature clouds.

From the dense cloud.

The concept is to detect features directly from the dense clouds, and find a corrective transformation between them. The Point Cloud Library proposes a possible pipeline [17]. Some preliminary tests were conducted following this pipeline. However, the implementation at the moment has not been sufficiently developed, and it was not able to provide accurate results.

From the image plane.

Alternatively, the Dense Monocular Reconstruction algorithm should be developed in a similar way of a VO algorithm. The Dense Monocular Reconstruction algorithm should try to follow points in the image plane that map to the same landmark through the sequence and progressively correct its 3D position using the information gathered from more than one image pair.

From the feature clouds.

Considering the feature clouds' perspective, finally, the algorithm should try to triangulate densely matched points in the position where the features are accurately triangulated. Then, the relation between these two set of triangulated points should be used to correct the position of the dense cloud. However, this method would be inaccurate for images where the features and the densely matched points do not correspond.

Bibliography

[1] Chiodini, S., Riccardo Giubilato, M. Pertile and Stefano Debei. “Monocular visual odometry aided by a low resolution time of flight camera.” *2017 IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)* (2017): 239-244.
[2] Pollefeys, Marc, Reinhard Koch and Luc Van Gool. “A Simple and Efficient Rectification Method for General Motion.” *ICCV* (1999).
[3] Pollefeys, Marc and Sudipta N. Sinha. “Iso-disparity Surfaces for General Stereo Configurations.” *ECCV* (2004).
[4] Cavegn, Stefan & Haala, Norbert & Nebiker, Stephan & Rothermel, Mathias & Zwölfer, Thomas, “Evaluation of Matching Strategies for Image-Based Mobile Mapping.” *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences. II-3/W5. 10.5194/isprsannals-II-3-W5-361-2015* (2015).
[5] Pizzoli, Matia, Christian Forster and Davide Scaramuzza. “REMODE: Probabilistic, monocular dense reconstruction in real time.” *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014): 2609-2616.
[6] Greene, W. Nicholas, Kyel Ok, Peter Lommel and Nicholas Roy. “Multi-level mapping: Real-time dense monocular SLAM.” *2016 IEEE International Conference on Robotics and Automation (ICRA).*
[7] Gang Xu and Zhengyou Zhang. *Epipolar Geometry in Stereo, Motion, and Object Recognition: A Unified Approach*. Kluwer Academic Publishers, Norwell, MA, USA. (1996).
[8] M. Pertile, S. Chiodini, R. Giubilato and S. Debei, "Calibration of extrinsic parameters of a hybrid vision system for navigation comprising a very low resolution Time-of-Flight camera," *2017 IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, Padua, 2017, pp. 391-396.
[9] Triggs, Bill, Philip F. McLauchlan, Richard I. Hartley and Andrew W. Fitzgibbon. “Bundle Adjustment - A Modern Synthesis.” *Workshop on Vision Algorithms* (1999).
[10] The OpenCV library. Available at: https://opencv.org/

[11] The Point Cloud Library. Availabe at: https://pointclouds.org/

[12] Bay, Herbert, Andreas Ess, Tinne Tuytelaars and Luc Van Gool. “Speeded-Up Robust Features (SURF).” *Computer Vision and Image Understanding 110 (*2008): 346-359.

[13] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision* (2 ed.). Cambridge University Press, New York, NY, USA (2003): 92.

[14] Hartley, Richard I.. “Theory and Practice of Projective Rectification.” *International Journal of Computer Vision 35* (1999): 115-127.

[15] Loop, Charles T. and Zhengyou Zhang. “Computing Rectifying Homographies for Stereo Vision.” *CVPR* (1999).

[16] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision* (2 ed.). Cambridge University Press, New York, NY, USA (2003): 592-593.

[17] PCL registration pipeline, from the Point Cloud Library documentation: http://pointclouds.org/documentation/tutorials/registration_api.php#registration-api