UNIVERSITÀ DI PADOVA FACOLTÀ DI INGEGNERIA

TESI DI LAUREA

DESIGN, IMPLEMENTATION AND ANALYSIS OF A DISTRIBUTED 3D RECONSTRUCTION ALGORITHM FOR MARKER-BASED MOTION CAPTURE.

Laureando: Nicholas Felicini

Relatore: Chiar.mo Prof. Angelo Cenedese

Supervisori: Paul Smyth

Jacek Czyz

Corso di laurea Magistrale in Ingegneria Informatica

Data di laurea: 14 Marzo 2011

Anno Accademico: 2010 - 2011

Summary

1 Introduction

1.1 Motion Capture Definition

Motion capture is the process of recording a live motion event and translating it into usable mathematical terms by tracking a number of “key points” in space over time and combining them to obtain a single 3D representation of the performance [1].

The subject to be captured can be anything that exists in the real word and has motion. The “key points” are the areas that best represent the motion of the subject. The location of these points is identified by one or more sensors or markers.

Motion capture is used mostly for recording people moving and the data obtained can be used for different applications. The most popular are in the entertainment industry. In videogames it is used to animate athletes as in Figure 1.1, or to reproduce particular movement of the characters as to take a penalty kick in soccer, or to score a point in basketball.

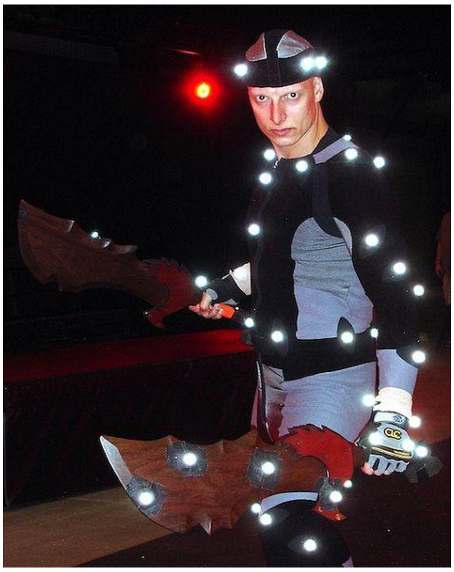


Figure 1.1. Joesph Gatt as the motion capture actor for 'KRATOS' from God Of War II & III on the mocap stage for SCEA in San Diego.

In filmmaking it is used to animate digital character models in 2D or 3D. The most famous film that used this technique is the “Lord of the ring” where the Gollum is animated by a real person which movements are captured using a passive marker system.

Commonly it is used in medical application for gait analysis, rehabilitation and to increase athletes’ performance. Clinical studies require accurate motion knowledge for the diagnosis of locomotion difficulties in patients. Sports people use motion capture systems to record themselves in order to diagnose potential improvements in their performance.

There are different systems to obtain this information from the “key points”:

- **Passive markers**:

  In these systems, on each person there are attached a certain number of markers, manually coated with stripes of retro reflective material. The density and the position of the markers may vary depending on the number of information to obtain. For example, if it is important to model facial expressions, up to 350 markers should be put on the face of the person. Otherwise, to capture the movement of a leg and the foot, three markers one the foot to recreate its orientation and position plus one marker one the knee and one on the hip are enough.

  Around the stage where people move, there will be a certain numbers of cameras, usually between 6 and 40. Special cameras are used, sensible only to infrared radiation that is generated near the camera’s lens (the red dot in Figure 1.1 is the infrared strobe of the camera). When two or more cameras see the same marker, using epipolar geometry and knowing the exact position of each camera, it is possible to reconstruct its 3D position.

  In the reconstruction, there is a marker swapping problem, because all markers are equals so all possible couple of points have to be tested to find the right match.

  In this work and by Vicon Motion Systems are used these passive markers system are used these passive markers system.

- **Active markers:**

  In these systems rather than reflecting back the light that is generated externally, the markers themselves are powered to emit their own light. If they alternately light up very quickly it is possible to triangulate their position without matching problem.

- **Markerless**:

  These systems don't use markers for the 3D reconstruction. The most common systems use cameras placed in different positions. Particular algorithms extract the 2D shape of the body and combine them trying to recreate the 3D model of the subject.

  These algorithms first extract the shape of the human body from the image and then they try to match the shape with a dynamic model of the human body with usually 33 degree of freedom [2].

1.2 Vicon

For this thesis, it has been chosen to do an internship in Vicon, the world's largest supplier of precision motion tracking systems, and match moving software.

Vicon is a subsidiary of OMG (Oxford Metrics Group), plc., a group of technology companies that produces image-understanding solutions for the Entertainment, Defense, Life Science and Engineering markets. Vicon and OMG global clients include: Life Science leaders University of Pennsylvania, the VA Hospitals, Shriners Hospitals for Children, Titleist Golf, The Andrews Institute; Engineering industry leaders Ford, BMW, Airbus, Lockheed, Pratt-Whitney, NASA, Caterpillar, International Truck, and Toyota; and Entertainment companies Sony Pictures Imageworks, Sony Computer Entertainment, Industrial Light and Magic, Sega, Nintendo, Ubisoft, Vivendi, Electronic Arts, Square Enix.

Vicon offers a complete range of products from 2D video analysis through to the high accuracy of the 3D digital optical systems. The main product is the Vicon Mx system whose major components are cameras, the controlling hardware module and the software to analyze and present the data. There are many cameras; the most commons are the “Bonita”, that offers the best price – performance mix with a frame rate of 240Hz and a VGA resolution. There are then the high resolution T-series cameras that go from 1 megapixel to the 16 megapixel motion capture camera capable of capturing 120 frames per second [3].

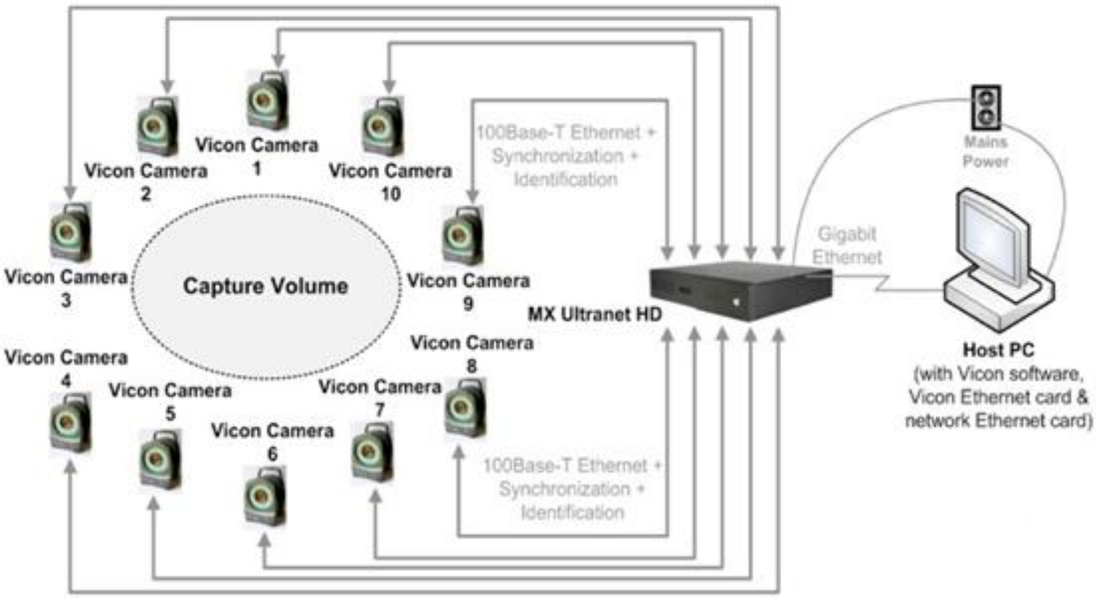


Figure 1.2. Typical Vicon 10 cameras set-up.

1.3 Centralized and Distributed Reconstruction

In collaboration with the University of Padova, Vicon and the FeedNetBack European research project on network control systems, the aim of this work is to create a distributed algorithm for the 3D reconstruction of the markers.

As described in chapter 2, Vicon uses a centralized algorithm; all data arriving from the cameras reach a single machine where they are processed. The performance of this solution although optimized, is highly dependent on the number of cameras and markers. Complex system with hundreds of cameras and thousand of markers cannot be analyzed in real time. In a centralized version it is difficult to increase the speed of the reconstruction because there is a trade-off between the time required and the accuracy.

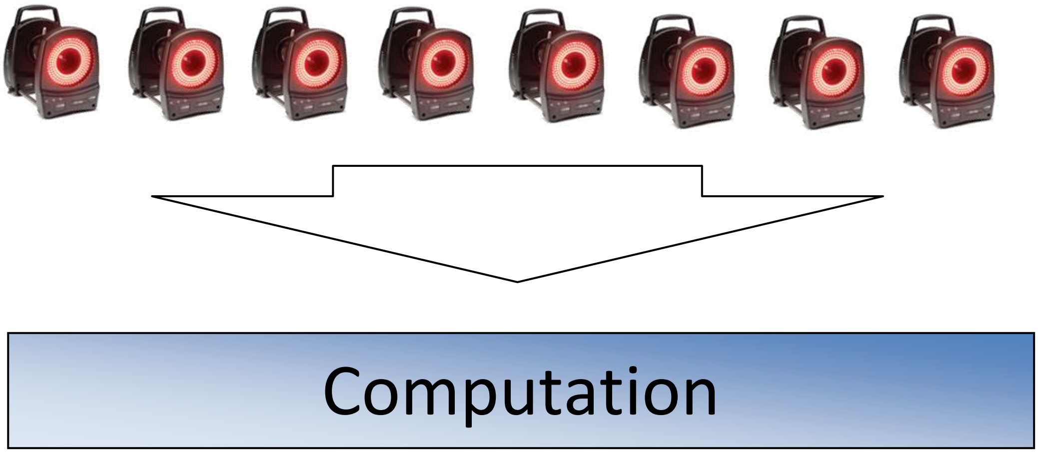


Figure 1.3. Centralized approach.

In chapter 3, a distributed system is tested to spread the calculations on different nodes to increase the performance. A binary tree is chosen, its first level has a number of nodes equal to half of the number of cameras. Each node tries to reconstruct all possible 3D points using the data coming from two cameras and send these results to the nodes in the second level of the tree. These nodes try to reconstruct all possible 3D points using the preprocessed data coming from the node in the first level and so on, until the root merge all the information.

The nodes in the first level of the tree process data from two cameras.

In chapter 4, it is tried to evaluate if the two cameras can be associated random cally or if there is a better way to choose them.

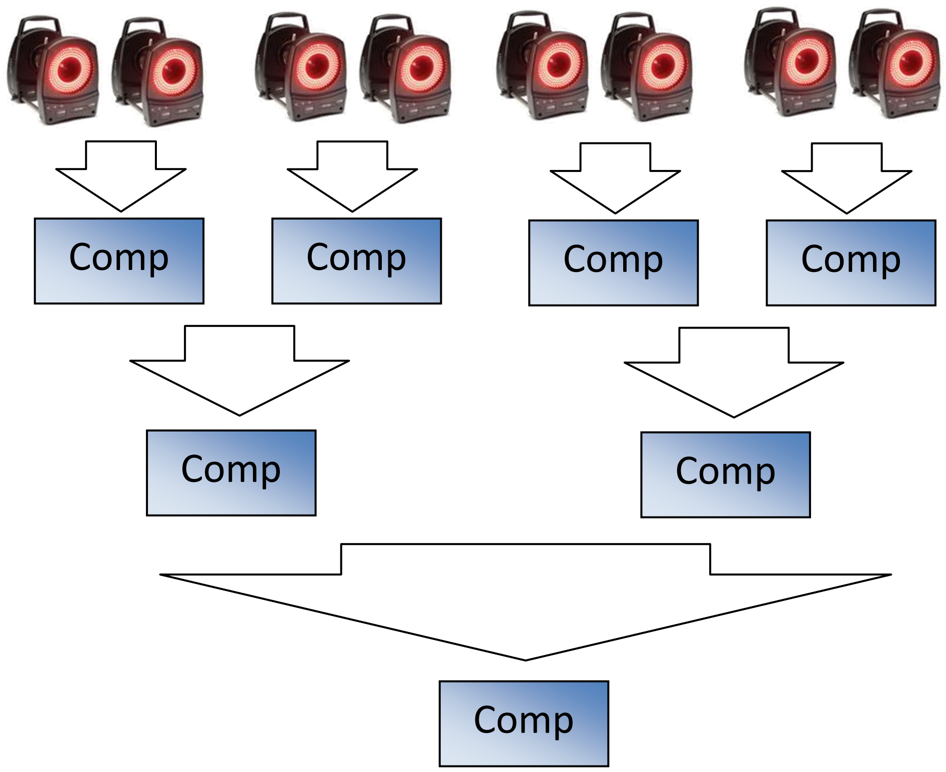


Figure 1.4. Distributed approach.

A cost functional is introduced to evaluate the association using as parameters the volume shared and the angle between the cameras. There is also an analysis on the number of markers reconstructed in each level of the tree to test if most of the reconstruction is done in the first levels of the tree.

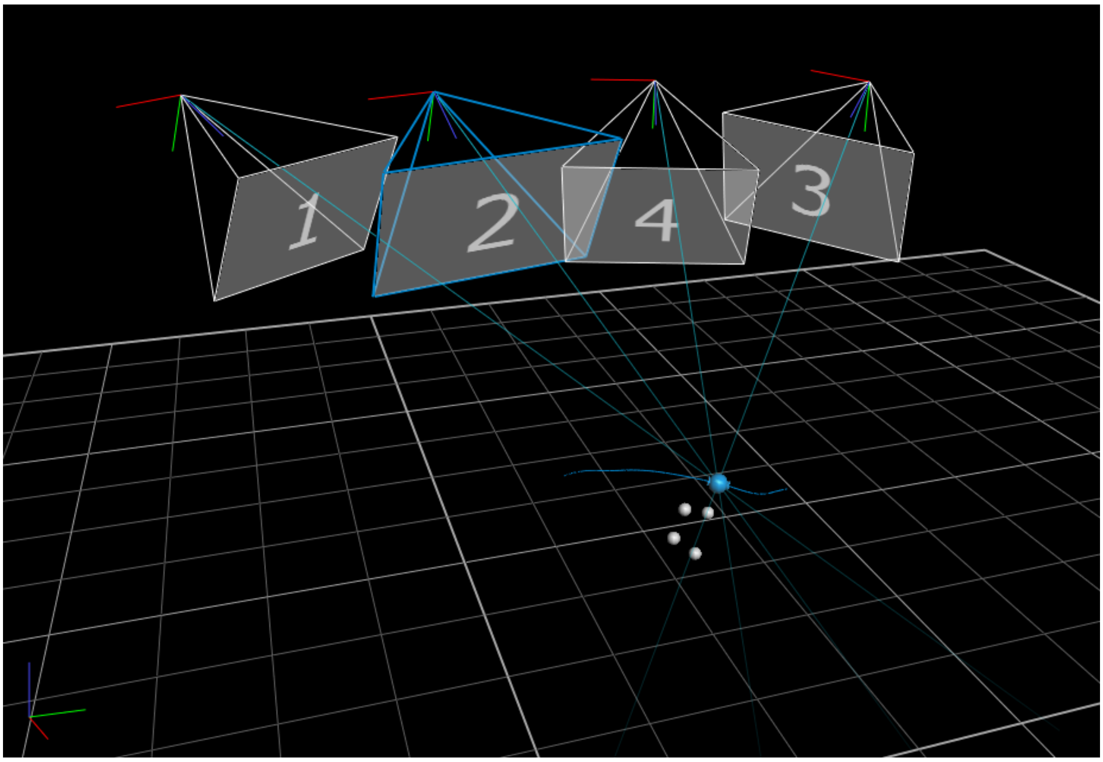


Figure 1.5. Marker reconstruction.

In Figure 1.5 there is an example on how it is possible to reconstruct a 3D marker. From each camera a ray is traced between the optical centre of the camera and the 2d point impressed on its image plane from the 3D point.

The intersection of the rays coming out from each camera forms a marker (highlighted in blue).

Explanation of common terms (Refer to Figure 1.6 and Figure 1.7):

2d point, *centroid* and *ray* are referred to the projection of a marker in the image plane of a camera. I

Marked (or Grabbed) 2D point is referred to a 2D point associated with a big 3D point. Marked 2D points cannot be used to reconstruct other markers respect the one they are associated with.

2d points associated to a 3D point are referred to the 2D points that are used to reconstruct a 3D point.

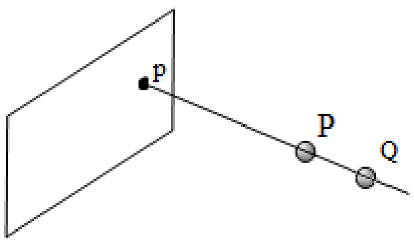
3d point and *Marker* are referred to a 3D point in the space. It can be real or reconstructed by the algorithm.

Big / Small 3D point is referred to a 3D point that is associated with a number of rays larger or equal / smaller than a certain threshold.

Ghost 3D point is referred to a 3D point reconstructed by the algorithm but it does not exist for real.

Missed 3D point is referred to a 3D point that is not reconstructed by the algorithm but it exists for real.

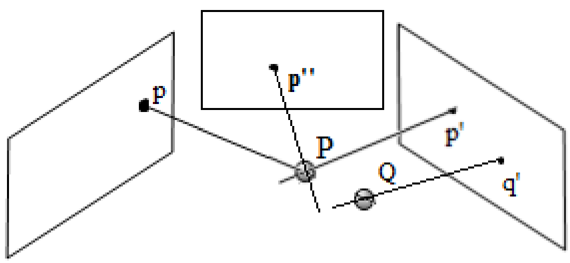
Associated cameras is referred to the cameras that are analysed in the same node.



P and *Q* are 3D points.

p is a 2D point associated with both

Figure 1.6. 2d point association.



P is a **big** 3D point while *Q* is a **SMALL** 3D point.
The 2D points *p*, *p'* and *p''* are associated only with P

Figure 1.7. 2d points marked.

2 Study of Vicon's routine

For the 3D reconstruction of the markers, Vicon uses a centralised approach. The algorithm used is non disclosable. Below there is a brief explanation.

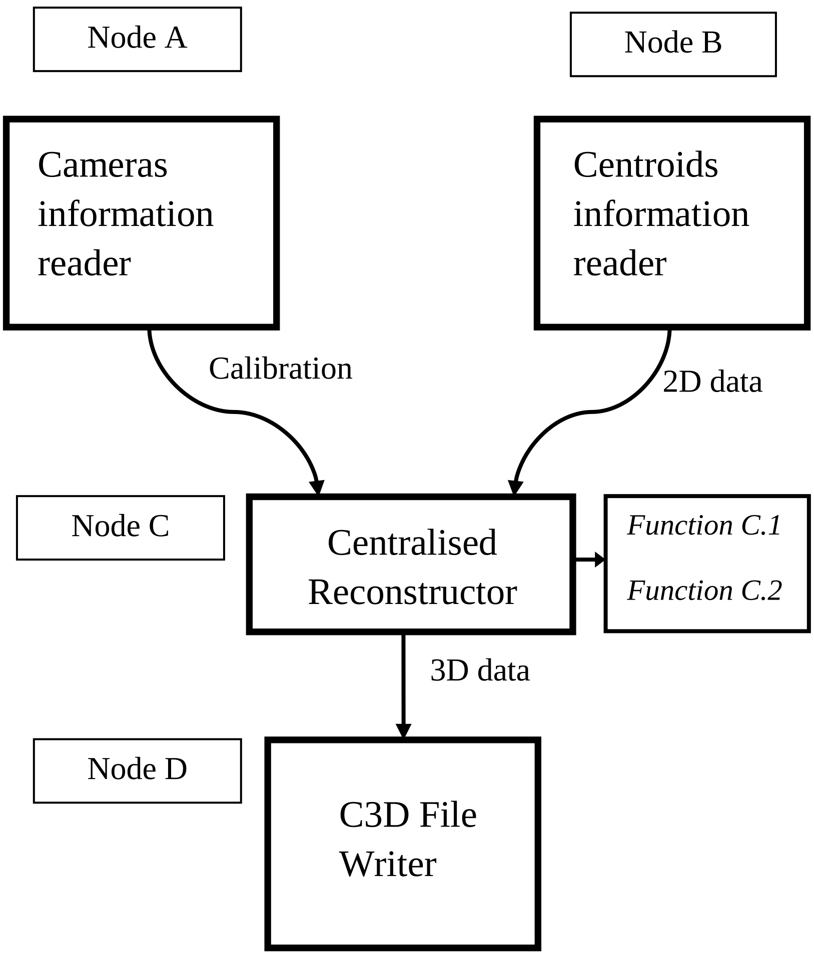


Figure 2.1. Scheme of the centralized algorithm.

The complex structures of the algorithm consist of different nodes for diverse tasks that can run in parallel to speed up the algorithm. For example while Node D is writing the 3D points of frame 23, Node C can do the reconstruction of frame 24 and Node A and B can read the information of frame 25.

Node A reads the cameras calibration information that contains all the intrinsic and extrinsic parameters of the camera. Node B reads the 2D centroids position that each camera creates in the image plane.
The core algorithm is contained in Node C that does:

1) It corrects the radial distortion of the centroid, using the hardware information,
2) It calls two *functions* one after the other:
    a. The first reconstructs all possible 3D points and stores them in a queue,
    b. The second analyses the queue to extract only the valid 3D points. A 3D point is valid only if it is associated with a number of rays bigger than a certain threshold, usually equal to three[1] but in case the number of cameras is high (>60) the threshold can be higher.

Function C.1 for each two cameras, exploiting the epipolar geometry, tries to match every couple of centroids. Using the fundamental matrix that correlates the two cameras is it possible to verify whether two 2D points can create a SMALL 3D point and if this is true, the pair of points is stored in the queue. Each time it finishes to analyse two cameras, it analyses the queue and using the information of the position of the 2D points and the Projection matrix of the cameras, calculates the 3D position of the marker. Subsequently this 3D point is back projected on all other cameras image planes looking for new correspondences. This is done looking for the closest 2D point with respect to the 3D back projection and if the distance between the two is under a certain threshold, the 2D point can be used to reconstruct the 3D point. After examining all other image planes, the 3D position is re-estimated using the new 2D correspondences.

When at least three cameras see a 3D point it becomes BIG therefore all 2D points used to reconstruct it, are marked and they are not analysed again to create other 3D points, speeding up the algorithm. With the 3D position, a score is estimated based

[1] Two cameras are needed and usually enough to correctly reconstruct a 3D point, but sometimes more than two cameras are used to avoid ghost points.

on the quality of reconstruction. The easiest way to calculate the score is to count the number of rays that are associated with the 3D marker.

Function C.2 analyses the 3D points before the output. First the queue is ordered based on the score of the 3D reconstruction. Each point is extracted from the queue starting from the one with the highest score. If the point is BIG (or it has more rays than a certain threshold), it is straightaway put into the output data structure; otherwise the algorithm checks that all the rays that form it are not used by other BIG 3D points. If no ray is already used, the point goes to the output list. If a point has less rays than before there are two possibilities. If the rays are less than two, the 3D point is deleted because a 3D point needs at least two cameras that can see it. Otherwise the position of the point and its score are calculated again, using only the current numbers of rays. Then the point is reinserted in the ordered queue.
The output list containing the valid 3D points is sent to Node D where they are saved in a c3d file [4].

3 Distributed Reconstruction

The main aim of a distributed system is to have the largest number of different tasks running in parallel, so that the algorithm can increase it performance in time, simply increasing the number of cores.

In the specific case of 3D distributed reconstruction, the distributed algorithm is realised using a binary tree. The leaves receive in input the information coming from the cameras. The information is processed and transmitted to the internal nodes. In each internal node of the tree, the algorithm combines the information arriving from the two nodes of the lower level. The root node output contains the final reconstruction of the markers.

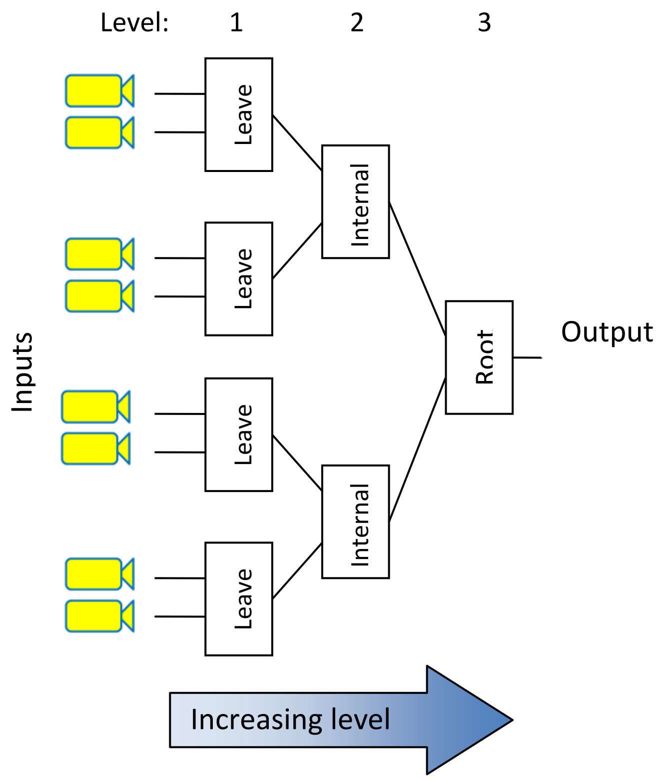


Figure 3.1. Tree structure with 8 cameras.

The total number of levels is $\log_2(\#cameras)$ and using a binary tree structure, the total number of cores is *#cameras – 1*.

3.1 Implementation details

The aim is to make most of the reconstruction at the low levels of the tree, where there are lots of cores working in parallel. Intuitively if most calculations are made at high level, the distributed algorithm requires a longer time than a centralised one because of its complexity.

Two different programming languages are chosen to develop the algorithm: Matlab® and c++.

Matlab is used for the first phase of the procedure:

- to create the synthetic markers;
- to create the synthetic cameras;
- to calculate the affinity function;
- to create the tree;
- to evaluate the number of 3D markers reconstructed at each level of the tree;
- to simulate a camera that projects the 3D position of the markers on a 2D image plane;
- to add a Gaussian error to the centroid;
- to store the centroid position information in a random or a predefined way;
- to store the cameras position information in a random or a predefined way.

The second phase of the procedure is related to the proper reconstruction algorithm. This is written in c++ with Microsoft Visual Studio®. The reasons behind this choice are that the first phase doesn't need to be in real time and Matlab even if it is slower than c++, from the workspace, permits to assess easily to the content of the variables. Part of the Matlab's code is taken from report[5].

The reconstruction algorithm is written in c++ for two reasons:

1) It is a compiled language so it is faster and more efficient than Matlab.
2) It is easy to integrate in Vicon algorithms that are written in c++ too.

3.2 Algorithm description

Two version of the *reconstructor* class are written.

The *first* reads the synthetic centroids and the cameras information from a text file realized in Matlab. Then it simulates a distributed algorithm to reconstructs the 3D position of the markers and saves them into a text file (Figure 3.2).

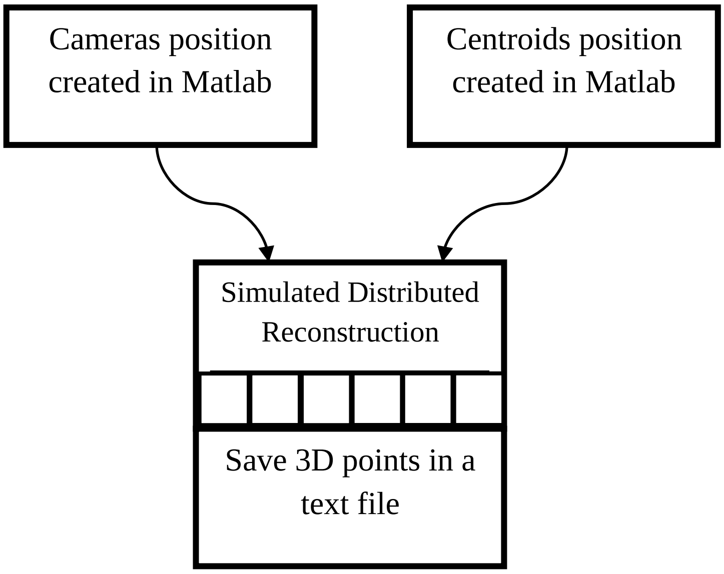


Figure 3.2. Simpler distributed *reconstructor*.

The *second* version is more complex and is integrated in the Vicon reconstruction algorithms (Figure 3.3). It reads real data and stores them in a *c3d* file, which can be opened with Vicon Nexus® (The main aim of Vicon Nexus is to reconstruct the 2D information coming from the cameras and visualize them in a virtual 3D space) [3]. This software can display the 3D reconstruction

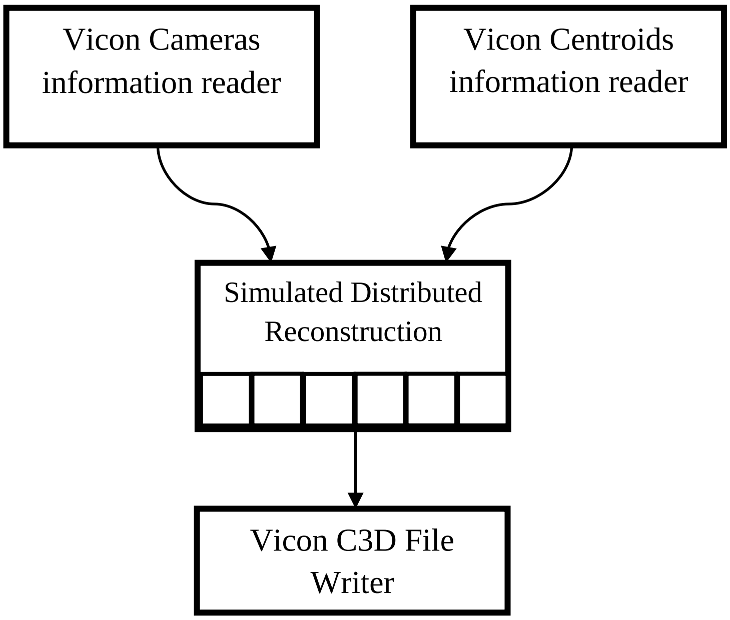


Figure 3.3. Full integrated distributed *reconstructor*.

The creation of a real distributed algorithm is complex because it requires a large number of CPU. For example, with 512 cameras the number of nodes in the first level of the tree is 256. The total number of node in the binary tree is equal to the number of cameras minus one so in take case 511. In this work for simplicity a simulated distributed algorithm is implemented and is run in only one core.

Each node of the tree should be able to manage an input coming from 2 sources. These inputs are the 2D points arriving from one or more cameras at lowest level and the 3D points created in other levels of the tree. Each node has an output which consists of 2D points and 3D points and can access to a shared memory containing the cameras information (Figure 3.4).
A node does the elaborations that are needed to correlate all possible 2D points with all possible 3D points. When the number of rays associated with a 3D point exceed a certain threshold, usually three, it is possible to delete or mark the 2D points for further analysis.

Consideration about the deletion of rays used and BIG 3D

A consideration should be done on the exact definition of marking a ray.
In the centralised version, when a ray is marked, there is a support data structure that contains a boolean information about all the rays. The algorithm uses a ray only if it is not marked.
In this distributed algorithm, marking a 2D point means to delete it.
In the distributed version, it is chosen to delete them rather than to mark because every node of the distributed system has to transmit some data to the following node. The less information is transmitted the better. By deleting the 2D points and not having a support data structure for the marked point, the amount of information to send between the nodes is reduced. As it is possible to see in chapter 4, the amount of 2D points sent in the first level is 7588, in the third 3436 and in the seventh 492. Every two levels the number is halved and the communication time is lower.
When the algorithm associates new single 2D point with a 3D point, its position cannot be recalculated. But in a big set-up, this problem is negligible, because the high number of cameras should guarantee that each marker is seen by many cameras, and so with the 3D+3D phase is possible to improve the precision of the reconstruction.

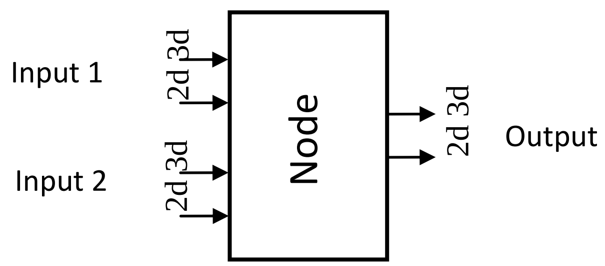


Figure 3.4. Inputs and Outputs of a node.

As it is possible to see in Figure 3.4, these are the inputs:

1. 2d_input1
2. 3d_input1
3. 2d_input2
4. 3d_input2

There are three possible combinations to elaborate the inputs:

1. 2d_input1 + 2d_input2
2. 3d_input1 + 2d_input2 (and vice versa)
3. 3d_input1 + 3d_input2

It is very important the sequence of these operations in order to reduce the number of centroids to analyse and to avoid the creation of multiple-equal 3D points. The best sequence starts with the ***3d_input1 + 3d_input2***, then the ***2d_input1 + 3d_input2*** *(and vice versa)* and finally the ***2d_input1 + 2d_input2*** because it guarantees the diminishing of most of the 2D point as soon as possible. In Figure 3.5 this sequences are expressed by the phase **3D+3D**, then the **3D+2D** and finally **2D+2D**.

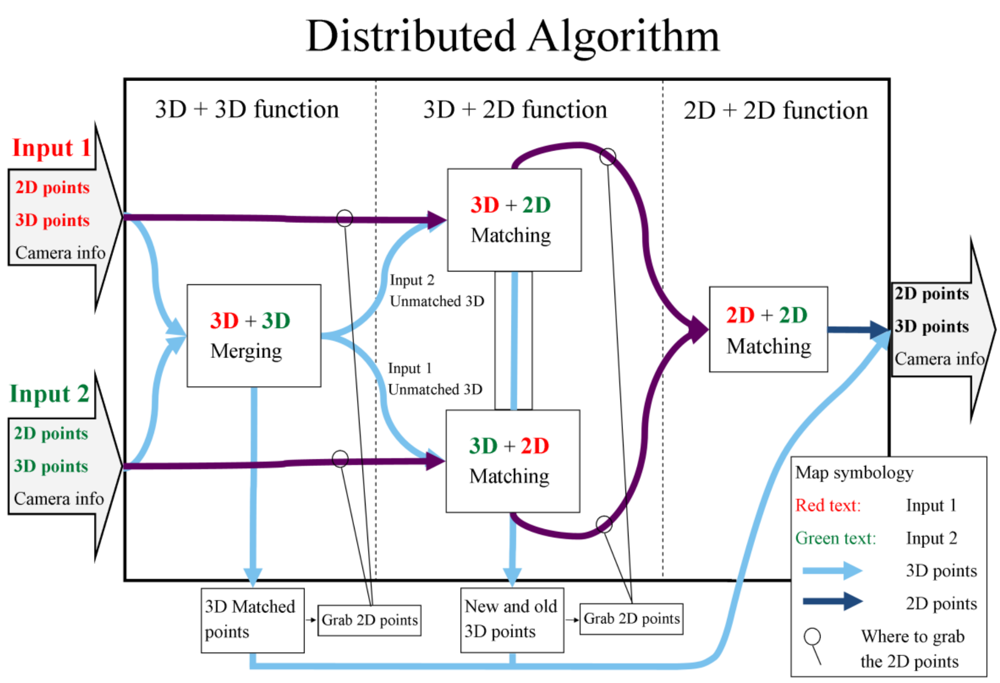


Figure 3.5. Scheme of the distributed algorithm.

3.2.1 3D + 3D function

At the beginning, the algorithm merges the 2 inputs of 3D points. In a distributed approach, in fact, it is very easy that different nodes reconstruct the same 3D points. To avoid the problem of having multiple copies, each node checks how each 3D point in input1 is far from every 3D point in input2. If their distance is under a certain threshold, they can be considered as a unique point. In this case, the algorithm creates a new 3D point and calculates its position doing a weighted mean of the 3D points previous position, based on the number of rays that from them.

*New3Dposition = (3Dpoint1_position/totalray*ray1) + (3Dpoint2_position/totalray*ray2).*

Then, the algorithm associates all 2D points matched with the previous 3D points to the new one. The 3D point just created can be put directly in the output list of the node, because it never needs to be compared with any other 2D points. This is because if a 3D point is obtained by merging two 3D points coming from the 2 inputs, it is already associated with the 2D points of both inputs. The two 3D points that create it are deleted.

If the new 3D point is BIG, all rays that form it must be marked as used, so that no other 3D points can be associated with it.
There are different cases depending on the status of the preceding 3D points. If both were BIG, all rays forming the new 3D points are already marked; if one was not BIG, the algorithm marks its rays; if both were not BIG, all their rays are marked.

The pseudo code 3.1 of this function can be found at the end of this chapter.

Input of this function

In this function are analysed the lists of 3D points coming from the two input.

Output of this function

The new 3D points go directly to the output of the node, while the 3D points that didn't matched remain in the same input data structure and they are analysed by the next function. If a 3D point becomes BIG, its 2D associated points are marked and so the 2D inputs are changed.

3.2.2 3D + 2D function

In this function the algorithm looks for new correlation between the 3D points of input1 and the 2D points of input2 and vice versa. One possible solution to this problem is to use a similar way to Vicon approach. Every time a 3D point is found, it is back-projected on all the other cameras plane to look for correspondences.

Theorem 1:
In a distributed approach, considering a node that performs the 3D+2D function.
If:

(1) A node does all possible associations that it can;
(2) A 3D point becomes BIG when it is associated with at least 3 rays.

Then a 3D point from input1 can be associated with no more than 1 not marked point from input2.

Proof: by Contradiction:

Supposing that input1 comes from node A and input2 from node B. A 3D point coming from node A is associated with two (not marked) points from node B. In node B those 2D points are already associated to form a SMALL 3D point for the first hypothesis. But this SMALL 3D point in the 3D+3D phase in the current node is merged with the 3D point coming from node A and creates a BIG 3D point for the second hypothesis. The consequence is that all associated 2D points are marked, but this is in contrast with the assumption that both points were not marked.

If the new 3D point is BIG, all rays that form it are marked.

The pseudo code 3.2 of this function can be found at the end of this chapter.

Input of this function

In this function are analysed the 3D point not matched in the previous function and the 2D information coming from the inputs.

Output of this function

All the 3D points analysed here go to the output list of the node because they are not used in the 2D + 2D function. In the 2D inputs there are some changes due to possible creation of BIG 3D points that mark the associated 2D points.

3.2.3 2D + 2D function

In this function all possible new 3D points are created starting from the 2D information.

Theorem 2:

In a distributed approach, considering a node that performs the 2D+2D function.
If:

(1) *A node does all possible associations that it can;*
(2) *A 3D point becomes BIG when it is associated with at least 3 rays.*

Then the new 3D points are created from no more than two 2D (not marked) points.
Proof: by Contradiction.

Supposing that input1 comes from node A and input2 from node B, it is possible to create a 3D point from three 2D points, two of them must arrive from the same input for example node A, the other arrives from node B. But for hypothesis 1, from the same node there must be the 3D point associated with those two 2D points. In this node in the 3D + 2D function the algorithm associates that 3D point with the single 2D point from node B and for the second hypothesis it becomes BIG because three rays form it, which are marked as used. In this 2D+2D function all three 2D points are marked and this is in contrast with the assumption that the three points were not marked.

For creating a 3D point starting from two 2D points, it is used the same approach as Vicon. All cameras from input1 are compared with all cameras from input2. For each couple of cameras using the fundamental matrix, it is possible to check if they can create a 3D reconstruction and it is true, the position of the 3D point is calculated and is associated with the 2D points.

The pseudo code 3.3 of this function can be found at the end of this chapter.

Input of this function

Here are analysed the cameras information coming from the two inputs.

Output of this function

All the 3D points created here go to the 3D output list of the node.

3.3 Data structures

It is not trivial to choose the data structure to contain the 2D and 3D information. Based on Vicon's code, for the 3D points the class `VReconHypothesis` is chosen, while for the 2D points the class `VCentroidSet` is chosen.

`VReconHypothesis` permits to store lots of information. The most important are the x, y, z position of the point, a Boolean variable that says if the point is BIG or SMALL, the number of rays associated with the point and a vector with a reference to the associated 2D points. The reference is a *pair* that contains the number of the camera and the number of the centroid. This information is very important because, when a point becomes BIG, the algorithm must mark the points that create it.

The class `VCentroidSet` contains a vector that stores the position of the 2D points for every image plane.

The algorithm, to simulate a real distributed situation, creates a more complex data-structure. In particular:

```
std::vector< std::vector< std::vector< const VCentroidSet *> > >
std::vector< std::vector< std::vector< VReconHypothesis > > >
```

The first vector has the same size of the tree level number.

The second vector has the same size of the nodes number in that level of the tree.

The third vector has the same size of the cameras number in that node.

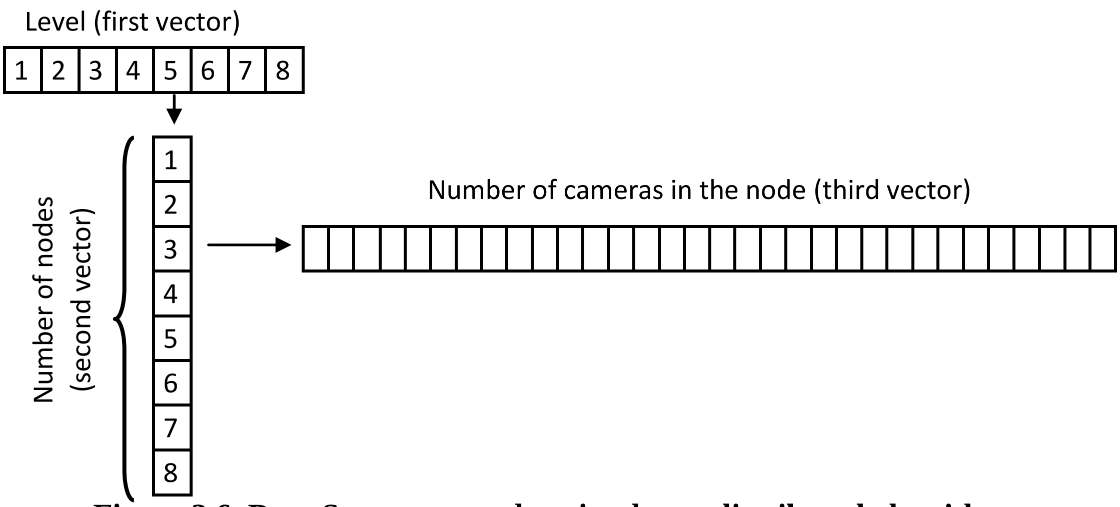


Figure 3.6. Data Structure used to simulate a distributed algorithm.

When the algorithm starts, it reads the number of cameras and resizes all vectors inserting the centroids information of the cameras. The total number of levels is = $\log_2(\#\, cameras)$; the number of nodes in level i is = $\frac{\#\, cameras}{2^i}$; the number of cameras in each node of level i is = 2^i.

Then it calculates the projection matrix of every camera and for every couple of cameras it calculates the Fundamental matrix.

To simulate the distributed system, the algorithm starts a series of cycles, one cycle for each tree levels with inside a cycle for every node using the complex data-structure described before.

When all cycles are finished, the output of the last node contains the markers 3D reconstruction.

In each node, a total of 5 functions are called, the 3D+3D, 3D+2D and 2D+2D plus other two functions for preparing the output results.

The performance in time of the algorithm is calculated summing for each level the time taken by the slowest node.

$$TotalTime = \sum_{i=1}^{\#Levels} \max_{1 \le k \le \#\, nodes\ \ in\ level\ \ i} (t_k)\,,\ \ t_k\ time\ taken\ in\ node\ k \tag{3.1}$$

This time is a good approximation of the real elaboration time of a distributed system. The main difference is that all times due to communication between different nodes are not taken in consideration, because those times depends on a lots of parameters like the connection speed between nodes, the lag introduced by the communication protocol and the fact that nowadays most processors contains 4 or 6 cores with very fast shared memory. The total amount of 2D and 3D data transmitted is saved for each experiment.

3.4 Parameters analysis

In this part, the intention is to correlate the parameters used and the performance of the algorithm.

For the reconstruction three parameters are needed.
The *first* is called *"SampsonImageError"* and is used in the 2D+2D phase to evaluate if two 2D points can be used to reconstruct a 3D point. The Sampson Error, in particular gives a first order approximation of the error between the back-projection *p* and *p'* of the reconstructed 3D point *P* and the two original 2D points used to create *P* (Figure 3.7) [6] [7].

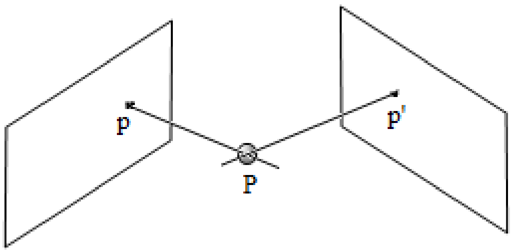


Figure 3.7. SampsonImageError.

The *second* is called *"distance2d3d"* and is used in the 3D+2D phase to evaluate if an already existing 3D point can be associated with a 2D point from a new image plane. As it is possible to see in Figure 3.8, if the distance between *p* and *q* is under the *distance2d3d* threshold, the 2D point *q* is associated with point *P*.

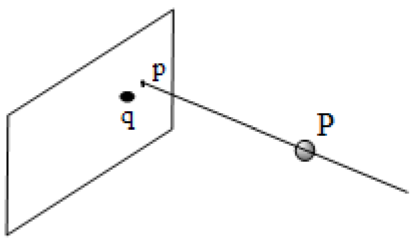


Figure 3.8. Distance2d3d.

The *third* is called *"distance3D"* and is used in the 3D+3D phase to decide if two 3D points represent the same or not. As it is possible to see in Figure 3.9, if the distance between point Q and P is under a certain threshold, they are merged to form a unique 3D point.

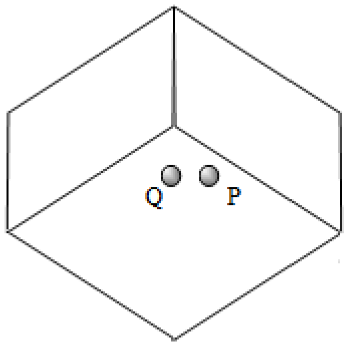
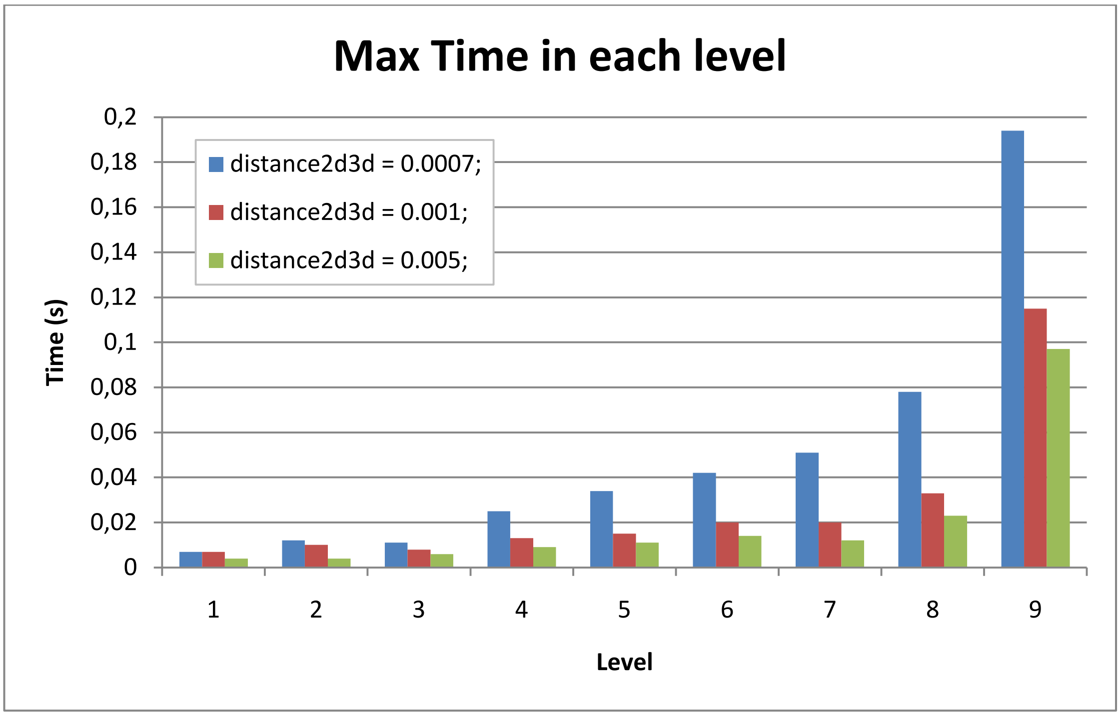


Figure 3.9. Distance3D.

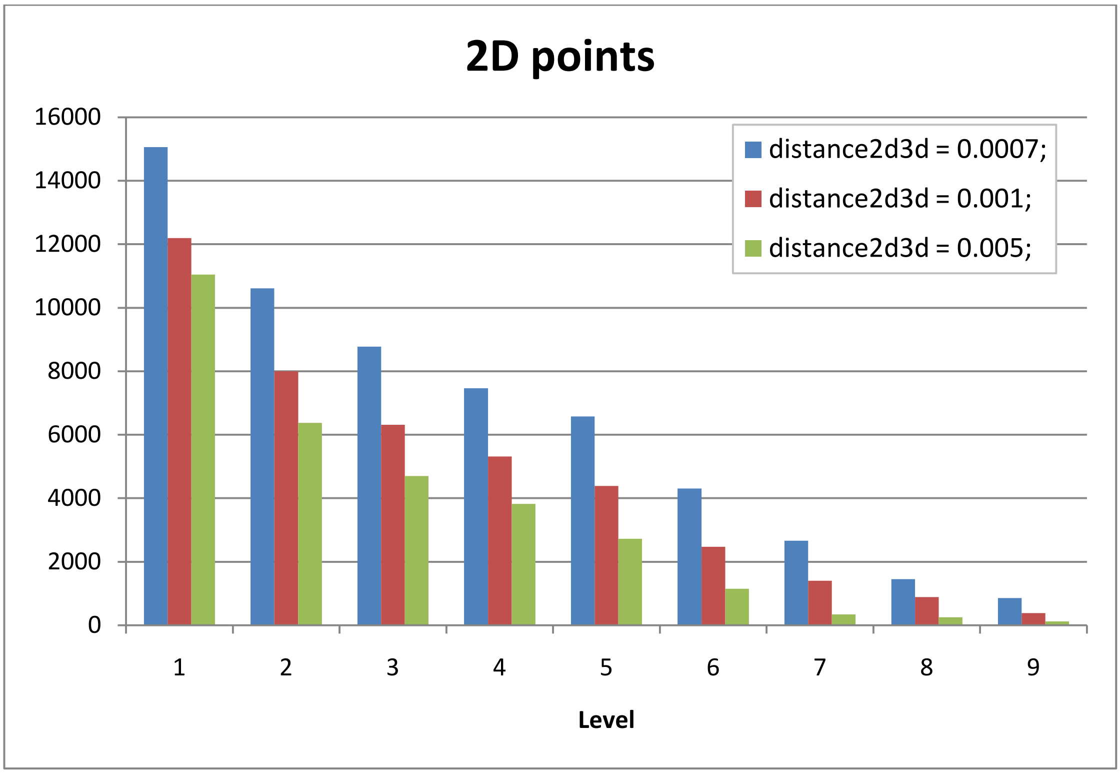
The first and the second parameters are very important and are usually three times bigger than the *"CameraImageError"* that is an error calculated directly by each camera it is usually equal to 0.2 pixel. When synthetic data are used they have to be chosen manually.
The third is chosen depending on the size of the marker, a typical value is 7 mm.

Every time the numbers of makers and cameras are changed, also the parameters required to be changed to obtain a better result in term of performance and accuracy. An example is presented in the Graphs 3.1, 3.2 and 3.3 created with 512 cameras and 846 markers.

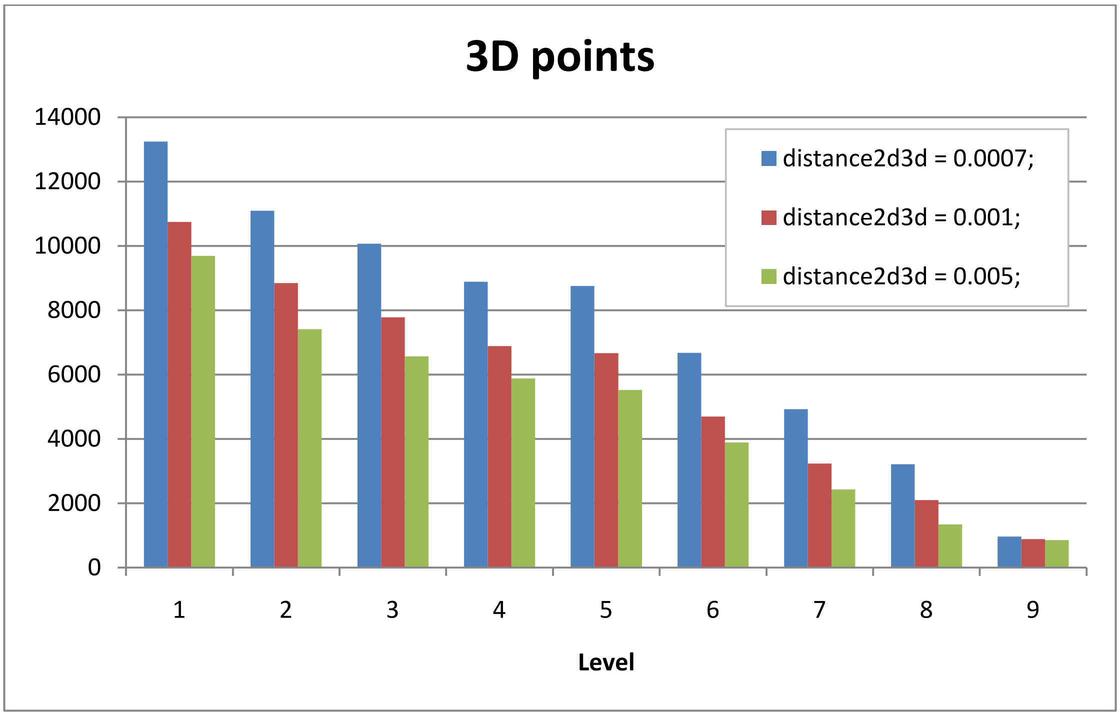


Graph 3.1. Max Time in each level.

The only parameter modified in the different trials is the *distance2d3d*. This parameter is used in the 2D + 3D phase of the distributed algorithm: increase this parameter means to be less strict looking for new rays to associate to existing 3D points. As soon as the 3D points become BIG, all 2D points are deleted. If the *distance2d3d* parameter is increased too much, the performance in terms of accuracy starts decreasing because some 3D points are associated with the wrong 2D points.



Graph 3.2. 2D points.



Graph 3.3. 3D points.

Pseudo code 3.1: 3D+3D function

```
Input:   3Dinput1, 3Dinput2.
Merge3Dpoint()
{
      for  i = 0 : 3Dinput1.size // for each 3D point from input1
          for  j = 0 : 3Dinput2.size // for each 3D point from input2
              {
                dist = euclidean_distance( 3Dinput1[i] , 3Dinput2[j] );
                if ( dist > distance3D threshold) continue;
                create 3Dnewpoint;
                3Dnewpoint.rays.pushback (3Dinput1[i].rays);
                3Dnewpoint.rays.pushback (3Dinput2[j].rays);
                ray1 = 3Dinput1[i].rays.size;
                ray2 = 3Dinput2[j].rays.size;
                totalray = ray1 + ray2;
                //new weighted position
                3Dnewpoint.m_X = (3Dinput1[i].m_X/totalray*ray1) + (3Dinput2[j].m_X/totalray*ray2);
                3Dnewpoint.m_Y = (3Dinput1[i].m_Y/totalray*ray1) + (3Dinput2[j].m_Y/totalray*ray2);
                3Dnewpoint.m_Z = (3Dinput1[i].m_Z/totalray*ray1) + (3Dinput2[j].m_Z/totalray*ray2);
                if ( 3Dinput1[i].isBig && 3Dinput2[j].isBig )
                {
                  delete  3Dinput1[i] and 3Dinput2[j];
                  continue;
                }
                if ( 3Dinput1[i].isBig)
                {
                  marks all rays that forms 3Dinput2[j]
                  delete  3Dinput1[i] and 3Dinput2[j];
                  continue;
                } // do the same as above for the other input
                if ( 3Dnewpoint.isBig )
                {
                  marks all rays that forms 3Dinput1[i] and 3Dinput2[j]
                  delete  3Dinput1[i] and 3Dinput2[j];
                  continue;
                }
                Add 3Dnewpoint to the 3D output list.
              }
          end for
      end for
}
```

Pseudo code 3.2: 3D+2D function

```
Input:   3Dinput1, 3Dinput2, 2Dinput1, 2Dinput2.
Find3Dcorrespondances()
{
      for  i = 0 : 3Dinput1.size // for each 3D point from input1
          for  j = 0 : 2Dinput2.size // for each camera from input2
              {
                2Dbackproj = back-projection of the 3D point i in the 2D camera image plane j
                for k = 0 : 2Dinput2[j].size // for each centroid in camera j
                find the nearest centroid in camera image plane j in respect of the 2Dbackproj and stores
                      it in nearest2Dpoint, saves its position in the 2Dinput2[j] vector in k_near
                end for
                dist = euclidean distance between 2Dbackproj and nearest2Dpoint
                if ( dist < distance2d3d threshold) // the new 2D point forms the 3D point
                3Dinput1.rays.pushback ( pair(j,k_near) );
                if ( 3Dinput1[i].isBig)
                {
                  marks all rays that forms 3Dinput1[i]
                }
              }
          end for
      end for
} // do  the same function reversing the two inputs.
```

Pseudo code 3.3: 2D+2D function

```
Input:   2Dinput1, 2Dinput2.
Matching2D()
{
      for  i = 0 : 2Dinput1.size // for each camera from input1
          for  j = 0 : 2Dinput2.size // for each camera from input2
              for k = 0 : 2Dinput2[j].size // for each centroid in camera j
                  for w = 0 : 2Dinput1[i].size // for each centroid in camera i
                  {
                      2Dpoint1 = 2Dinput1[i][w]; // centroid number w in camera i
                      2Dpoint2 = 2Dinput2[j][k];  // centroid number k in camera j
                      if ( both centroids are not marked)
                      {
                        if ( the two 2D points can create a 3D point)
                        {
                          creates and stores it in 3Dnewpoint
                          add 3Dnewpoint to the 3D output list
                        }
                      }
                  }
                  end for
              end for
          end for
      end for
}
```

4 Analysis of the functionals cost

In a distributed cameras system for motion capture reconstruction, the most vital aspect is the way in which the cameras associate.

A group of cameras is associated if they are analysed in the same node of the tree.

If the cameras look at different directions, it is possible that, if they are associated in a random way, some of them don't share any marker.

The node that contains those cameras is not able to reconstruct any 3D point, so no 2D points are deleted in the first tree levels.

The best way to evaluate the association between two cameras is to estimate the total volume that they share: the larger the volume, the higher the probability they see the same markers.

Moreover, it is very important to estimate the position of the markers with a good approximation. For this reason, the angle between the optical rays of the two cameras should be as close as possible to $\pi/2$.

In the next paragraphs, two functionals costs are introduced. They evaluate the volume and the angle between the cameras. For each couple of cameras, the functional value is placed in a matrix, in which it is possible to find the best associations.

4.1 Study of the angle between cameras

For a more accurate reconstruction, the 2 cameras used to reconstruct the 3D point, should have an angle of $\pi/2$ between their normal. With this angle, the uncertainty in the reconstruction of the rays' intersection is at its minimum (Figure 4.1).

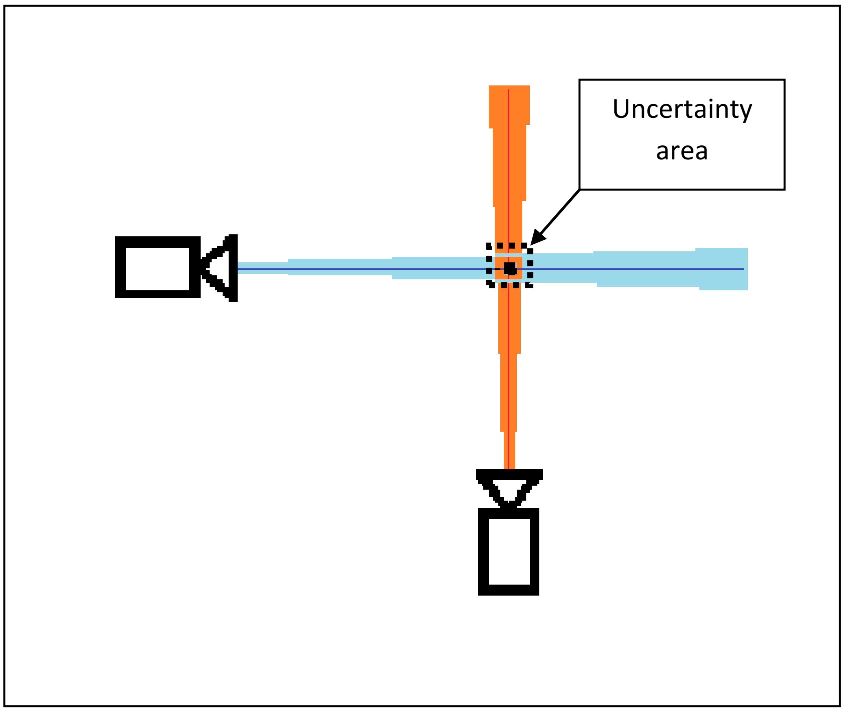


Figure 4.1. Uncertainty area with 90° cameras.

Instead, if the angle is near *0* or π the uncertainty is at its maximum (Figure 4.2).

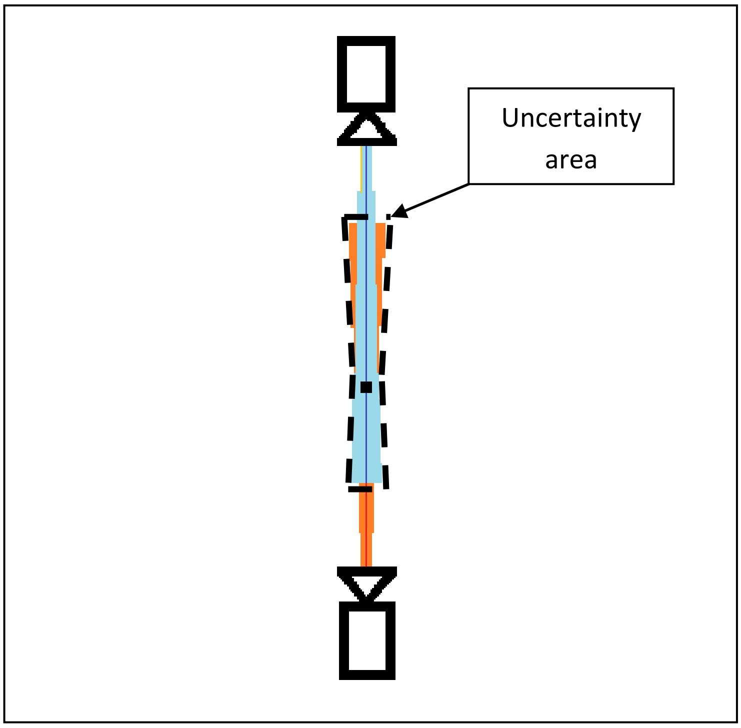
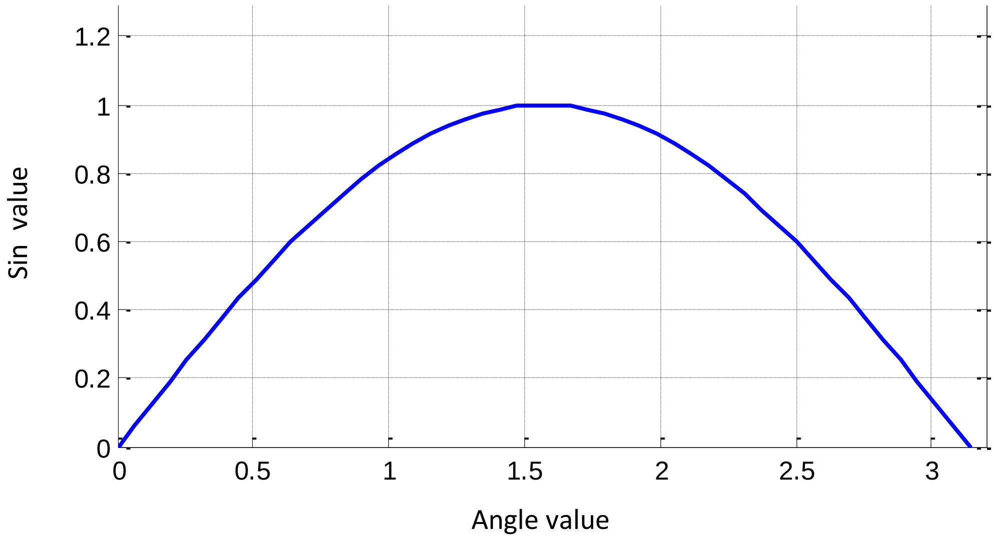


Figure 4.2. Uncertainty area with 180° cameras.

A function that well approximates this situation is the sin function



Graph 4.1. Sin function.

In the graph, near *0* and *π* the function is close to zero. In *π/2* the function is close to one.

The *sin* function does not take into consideration practical problems that might happen while working with markers on solid objects. For example, a marker on a hand is seen perfectly by all cameras with an angle < *π/2* with respect to the normal of a hand. All cameras with an angle > *π/2* don't see that particular marker, because it is hidden by the hand itself. In Figure 4.3 is possible to see an example of an occlusion. Marker p_1 is perfectly seen by camera C_1 but camera C_2 cannot see it because it is obstructed by the solid object.

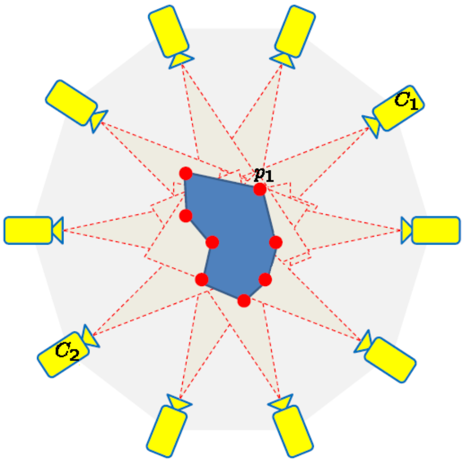


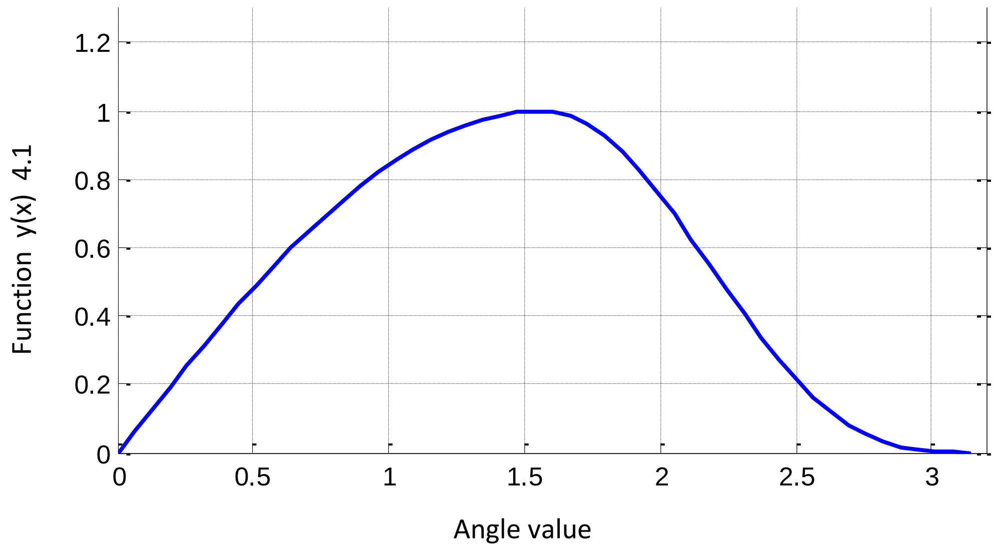
Figure 4.3. Occlusion example.

This situation can be modelled with a more complex function considering that the probability to have an occlusion is higher when there is a large angle between the two cameras.

A good approximation based on experimental evaluation, use the traditional *sin* function between *0* and *π/2* and the sin^3 function between *π/2* and *π*.

$$y(x) = \begin{cases} \sin x & 0 \leq x \leq \frac{\pi}{2} \\ \sin x^3 & \frac{\pi}{2} < x \leq \pi \end{cases} \quad (4.1)$$

For every two cameras the value of *y(x)* is estimated and is inserted into a symmetric matrix (Figure 4.4). On the rows and columns the matrix has the numbers of the cameras as indexes.



Graph 4.2. y(x) function.

The angle between the two cameras *j* and *k* is calculated as follows:

$$\theta_{jk} = \arcos\left(u_{zj} * u_{zk}\right) \tag{4.2}$$

With u_{zi} normal {unit} vector of camera's *i* image plane. The value in the angle matrix for each couple j,k is:

$$costAngle_{(j,k)} = y\left(\theta_{jk}\right) \tag{4.3}$$

With $y(\theta_{jk})$ the function (4.1).

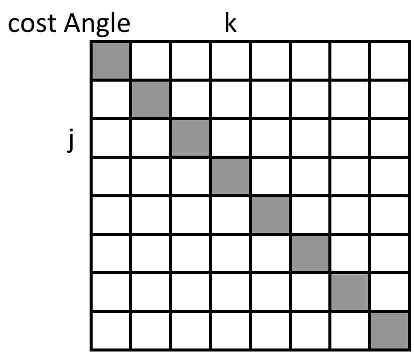


Figure 4.4. Angle cost matrix.

4.2 Study of the volume in common

To evaluate how much space is shared between two cameras, the best way is to partition the volume of the space in voxels (Figure 4.5). Their barycentres are then calculated and projected onto every camera. The number of voxels shared by each couple of cameras is calculated and inserted into a symmetric matrix that on the rows and columns has the numbers of the cameras as indexes.

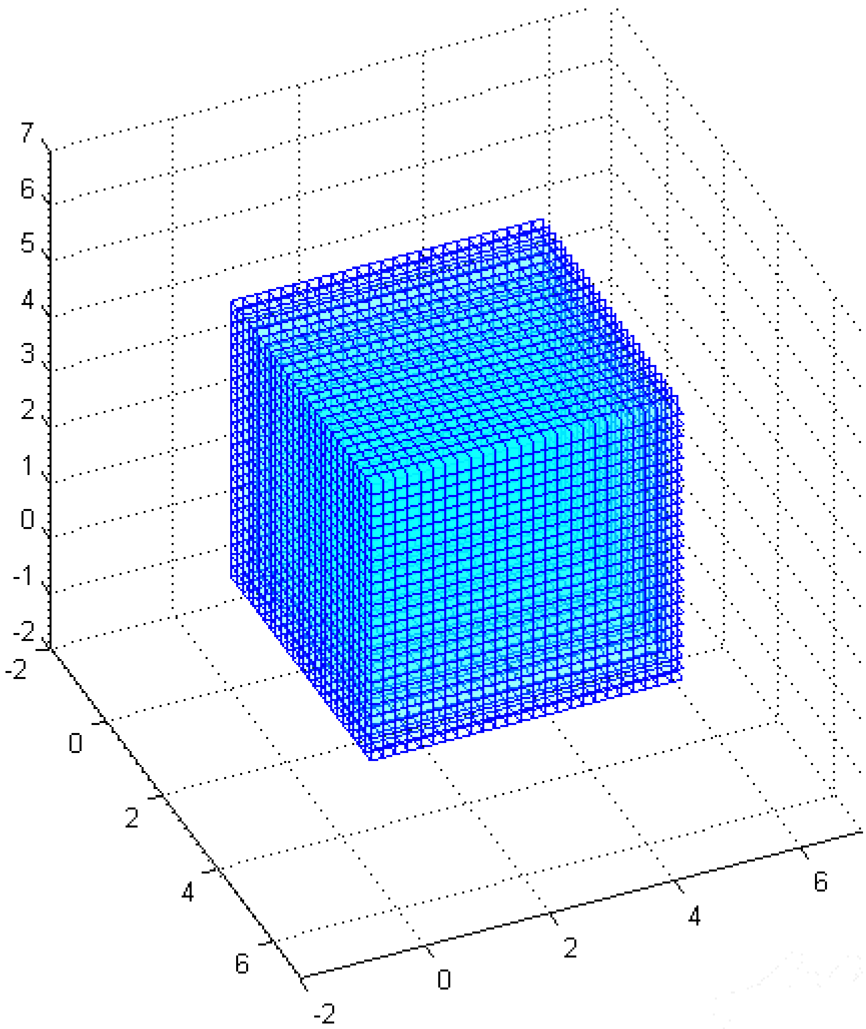


Figure 4.5. Partitioned Volume.

The value of the volume matrix for each couple j, k of cameras is:

$$costVolume_{(j,k)} = |\left(voxel_j \ \cap voxel_k\right)| \quad \textbf{(4.4)}$$

Where $voxel_i$ is the set of voxels seen by camera *i*.

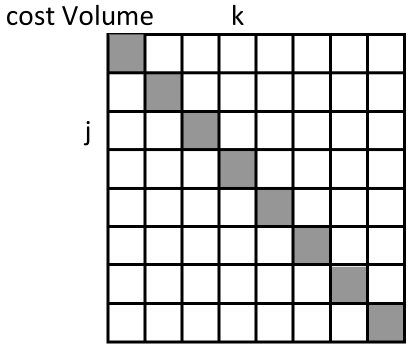


Figure 4.6. Volume cost matrix.

4.3 Final cost function

It is very important to combine the two cost matrixes that contain the cost values.
There are different ways to do that. For example, it is possible to combine the two values together but the angle is expressed by a sin function and is comprised between 0 and 1, while the number of voxels that two cameras share is usually very high, so the sum of the two values will hide the value of the angle.

A better way to combine them is to normalize the matrix of the voxels from zero to one.
The two matrixes now have comparable values, but to combine them it is important to choose the weight of the angle matrix and the weight of the voxels matrix.
The aim of the distributed approach is to reconstruct most of the markers at the first tree levels.
The volume is directly correlated with the number of markers that both cameras can see, so it is more likely that giving at the volume a larger weight, the reconstruction will become faster. After several trials, a good compromise is to use a 75% for the voxels and 25% for the angle value.

$$totalCost(j,k) = a * costAngle(j,k) + b * costVolume(j,k) \tag{4.5}$$

$$a = 0.25 \;\; ; \;\; b = 0.75$$

To find the couple of the cameras to associate, the algorithm scans the upper right part of the matrix (because the matrix is symmetric) looking for the biggest value (red in Figure 4.7). The j and k indexes of the cell represent the first cameras to couple. Then to avoid the use of those cameras again that row and column are deleted (Figure 4.8) and the algorithm will scan recursively the matrix looking for the new biggest value until all couples are found.

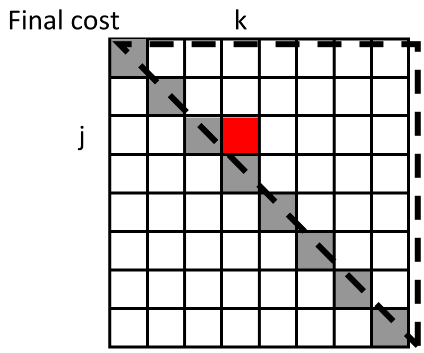


Figure 4.7. Final cost matrix (biggest value).

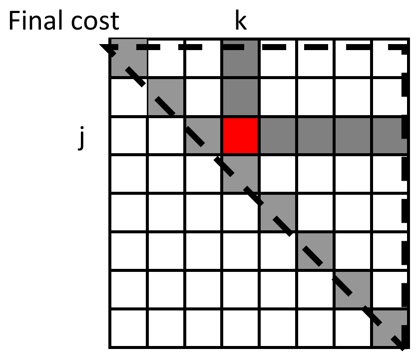


Figure 4.8. Final cost matrix (cameras value deleted).

4.4 High levels of the tree

After creating the first level, it is more difficult to decide how to group cameras in the other levels of the tree. It is important to choose how to calculate the new matrix of the costs between the groups of cameras.

The new matrix has half size of the previous one and the position *d(i,j)* becomes equals the costs of the cameras in the group *i* compared to group *j*. To calculate this value a sum is done, between the costs of each camera in group *i* and group *j*, based on the initial matrix.

For example:

If there are two groups of cameras: j= [1 2] and k= [3 4].

To evaluate the cost function of the group [1 2 3 4], the algorithm sums the cost of camera 1 compared to 3 and 4 plus the cost of camera 2 compared to 3 and 4.

$$d(j,k) = c(j_{1,}k_1) + c(j_{1,}k_2) + c(j_{2,}k_1) + c(j_{2,}k_2) \tag{4.6}$$

To find the best association, the algorithm uses the same strategy of the first level; it scans the matrix recursively, looking for the biggest value until all groups are created.

The number of cameras is a power of two, so when only two groups remain, the algorithm stops and unifies them to obtain the root of the tree.

4.5 Analysis of the functionals cost

The performances of the algorithm are tested using a code written in Matlab.
A simplified scenario is simulated where:

1. the cameras can be placed on the “ceiling” or on the “wall” of the volume;
2. the markers can be placed randomly or their position can be load from a file.

The outputs of the algorithm are three graphs referred for example to the first camera:

1. In the first graph, in X axis there is the representation of the values in the **angle** matrix; the largest number is 1, because it is a sinusoidal function. In Y axis there are the effective numbers of markers that the two cameras can see.

2. In the second graph, in X axis there is the representation of the values in the **volume** matrix. In Y axis there are the effective numbers of markers that the two cameras can see.

3. In the third graph, in X axis there are the values in the **final cost** matrix. The biggest number is 1, because the values are normalized. In Y axis there are the effective numbers of markers that the two cameras can see.

The camera with the biggest X value is chosen by the algorithm to associate with camera 1, depending on the affinity function. The aim of this algorithm is to couple cameras that share a large number of markers. A good algorithm should create a graph similar to an increasing monotone function with respect to the final cost.

Case 1

- The first example is done with **64 cameras and 200 random** points in a volume of *5 x 5 x 5* meters^3.

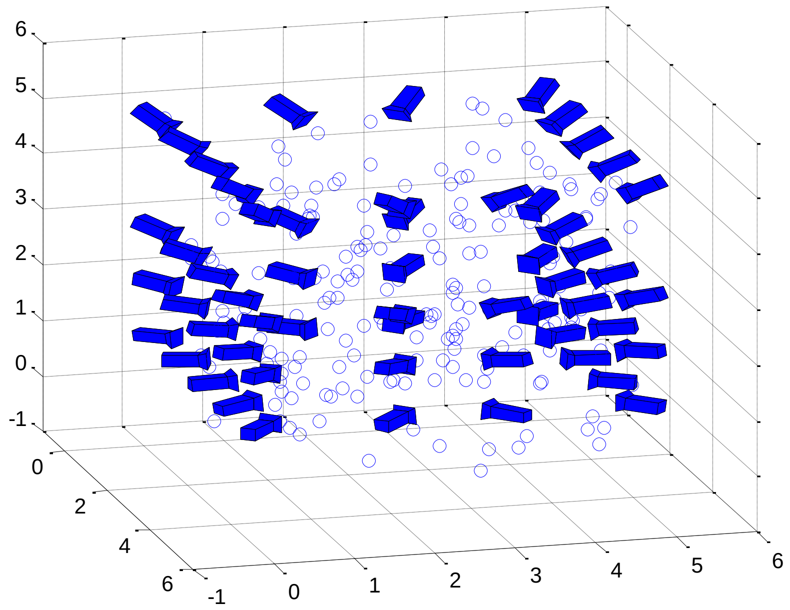
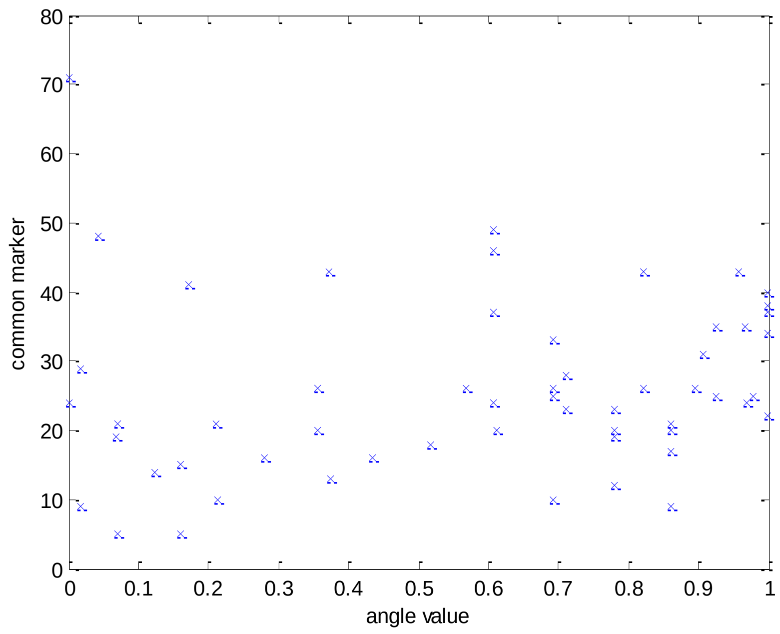
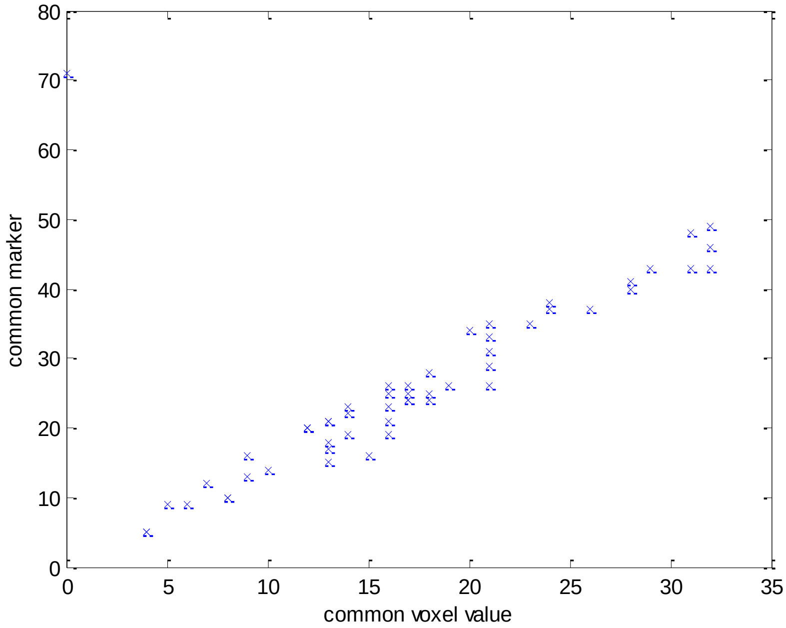


Figure 4.9. 200 random points and 64 cameras.

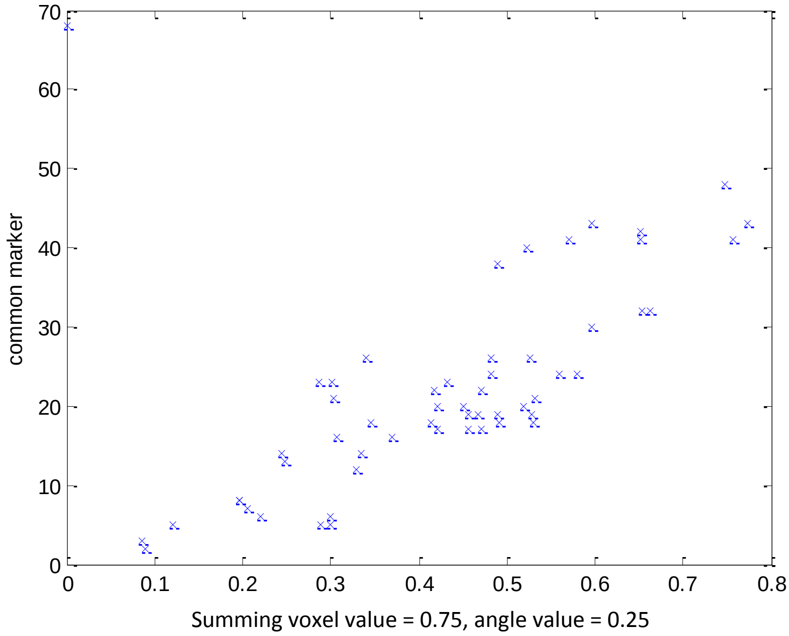
The three graphs obtained are:



Graph 4.3. Random angle value.



Graph 4.4 Random common voxel value.



Graph 4.5 Random total value.

In the Graph 4.3, the number of markers shared is not correlated with the angle between the cameras. In fact, this value is used for a more accurate reconstruction. In Graph 4.4 as expected the number of markers shared between two cameras is strictly correlated with the number of voxels they share. The bigger the volume they share, the larger the probability that they see the same markers. Graph 4.5 maintains a good correlation between the number of markers and the number of voxels shared.

These results should guarantee a good performance of the algorithm in a generic set-up.

Case 2

- In this second example, the algorithm tests a set-up with **18 cameras and 42 markers** put on a person using real data for the position of the markers.

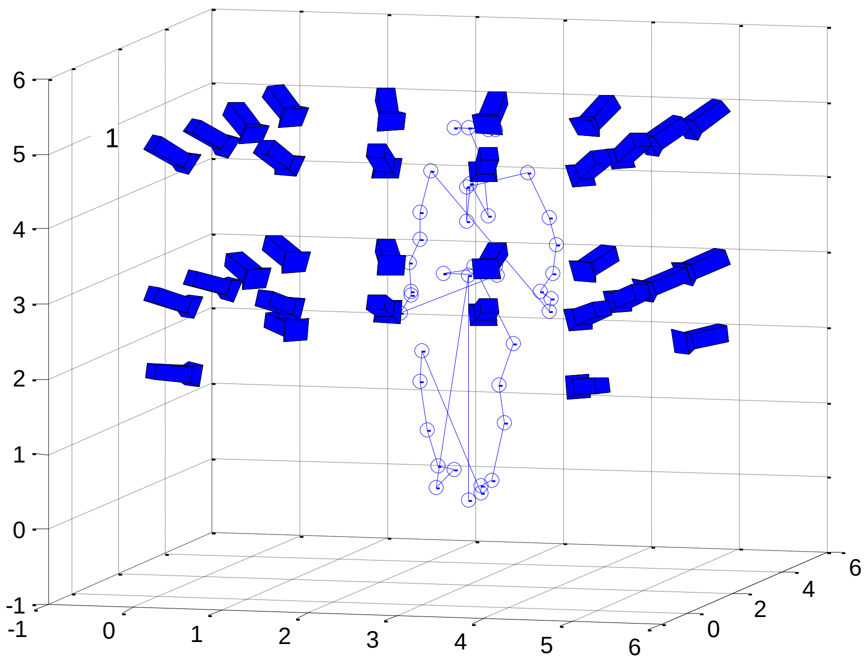
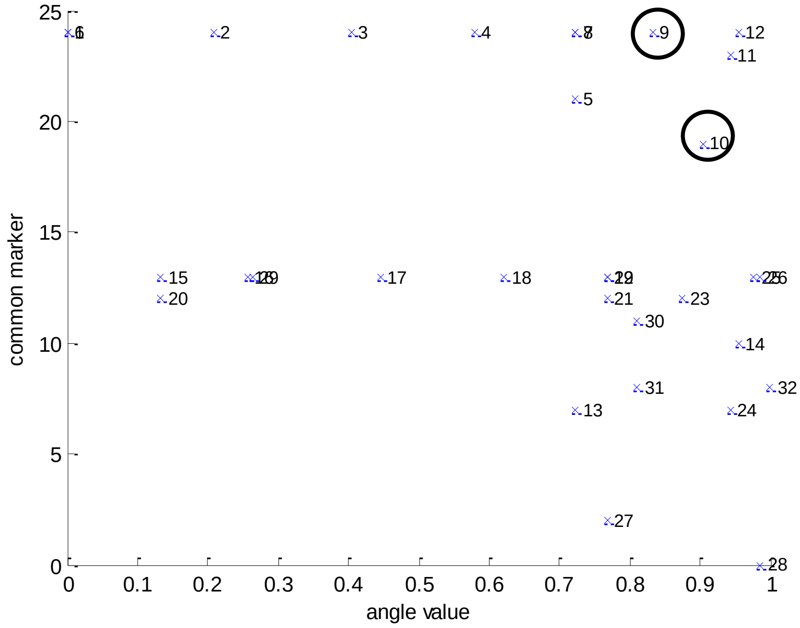
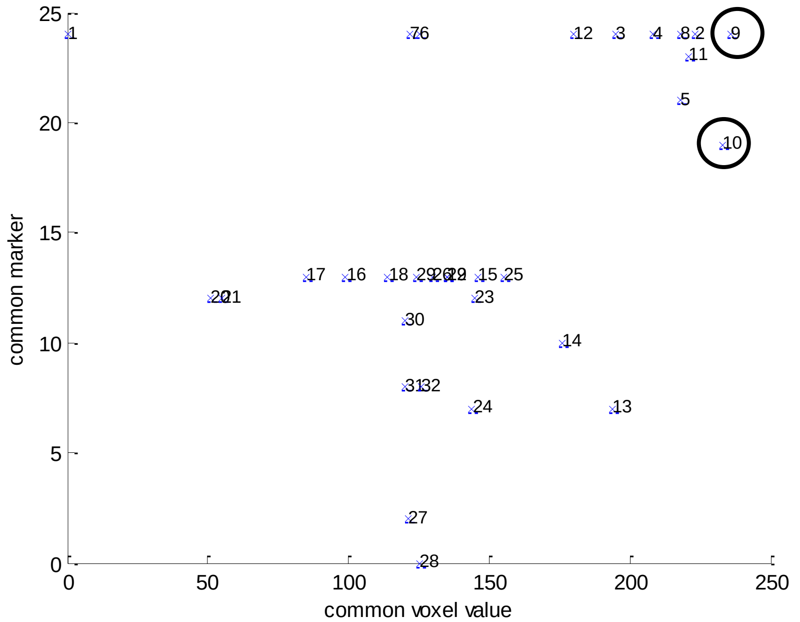


Figure 4.10. 42 points on a person and 18 cameras.

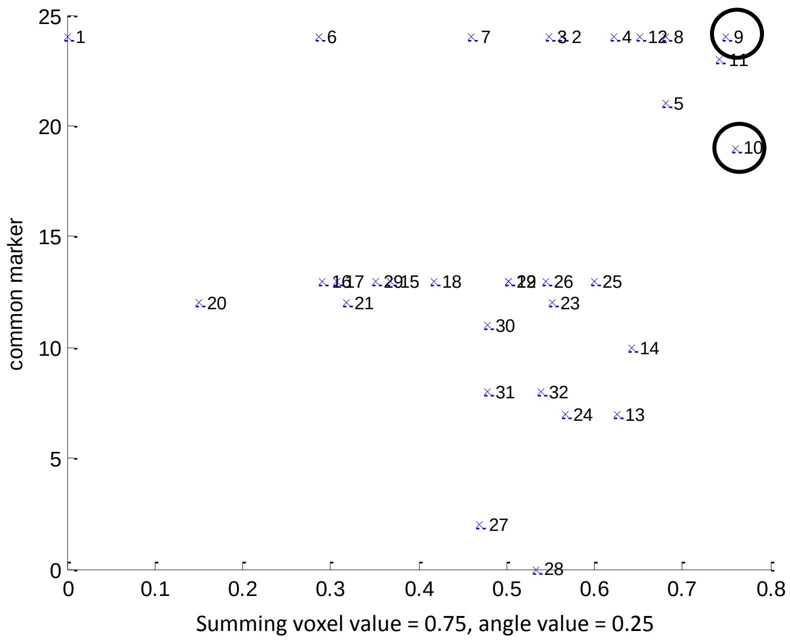
The resulting graphs are (always referring to camera1) Graph 4.6, Graph 4.7 Graph 4.8:



Graph 4.6. Ordered angle value.



Graph 4.7. Ordered common voxel value.



Graph 4.8. Ordered total value.

Analysing for example cameras 9 and 10, from Graph 4.6 it is possible to see that camera 10 has a better angle with camera 1, while from Graph 4.7, camera 9 shares few more voxels with camera 1. In Graph 4.8, the algorithm prefers to associate camera 1 with the camera 10 and this should guarantee a better accuracy in the reconstruction.

Case 3

- In the third example, a more complex set-up is tested. It has **512 cameras and 282** points on 6 people hugging. The dimensions of the stage are still *5x5x5* m and the positions of the markers are read from a real data file.

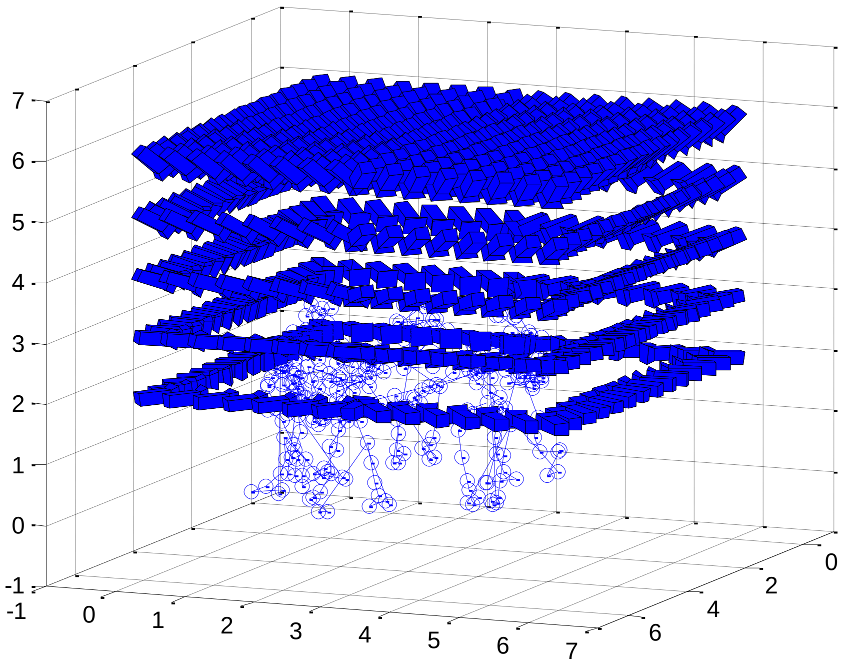
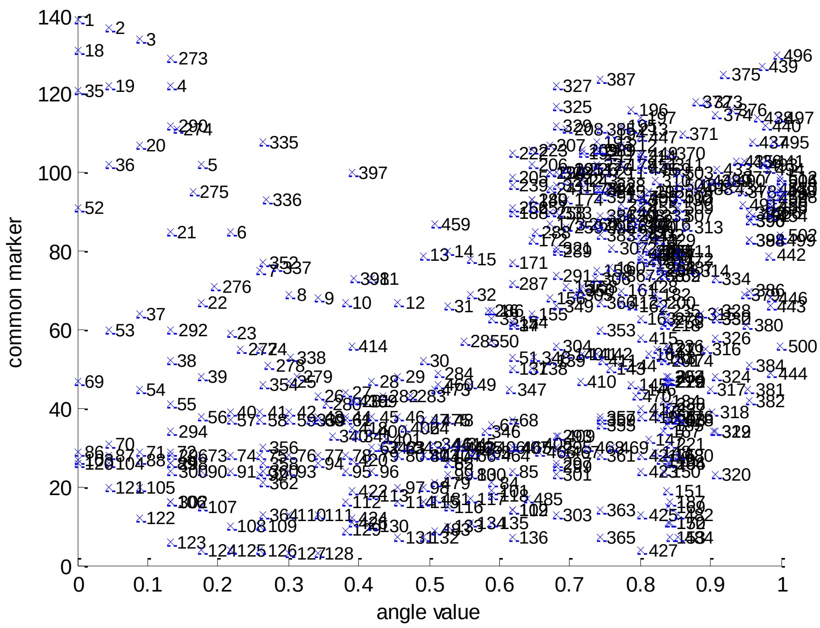
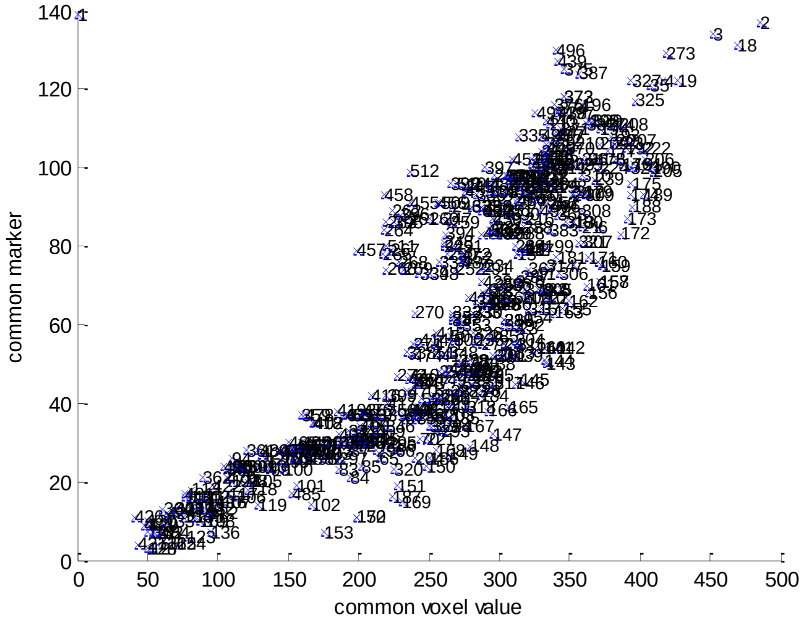


Figure 4.11. 282 points on 6 people and 512 cameras.

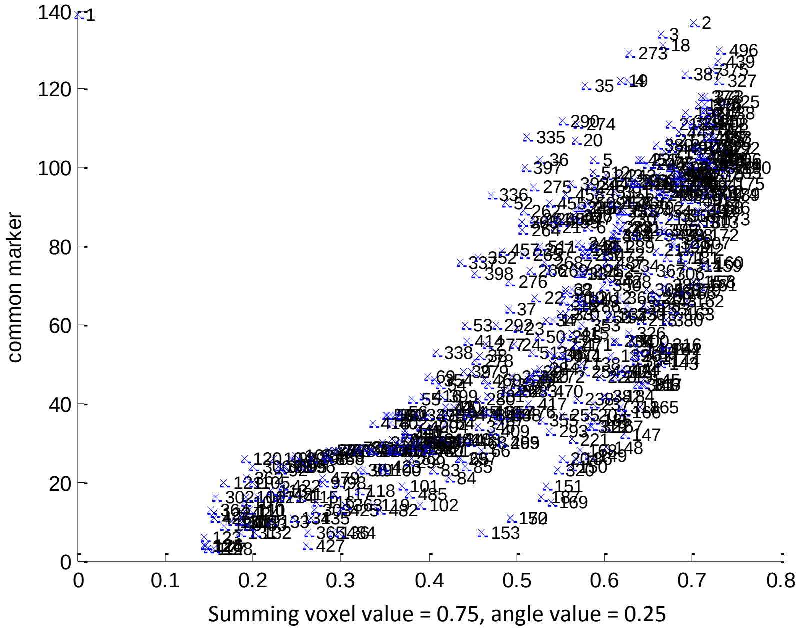
The resulting graphs are as follows:



Graph 4.9. 512 cameras angle value.



Graph 4.10. 512 cameras common voxel value.



Graph 4.11. 512 cameras, summing voxel value = 0.75, angle value = 0.25.

The Graph 4.11 shows that even with a high number of cameras and markers, there is a correlation between the number of markers and the volume shared between the cameras.

4.6 Analysis of the tree reconstruction strategy

EXP. 1

The following data are very important to evaluate the performance of the algorithm for the 256 cameras and 282 markers set-up.

In each level of the tree, the total number of 3D points reconstructed by at least three cameras is calculated. If a point is reconstructed more than once in different nodes, only the first reconstruction is considered.

These tests are of fundamental importance because they show how fast the algorithm is able to reconstruct all the markers. If most of the points are reconstructed in the last levels of the tree, a distributed solution is not on advantage with respect to a centralized version.

Reconstructed points:

These are the reconstructed 3D points in each level

Levels 1 through 8

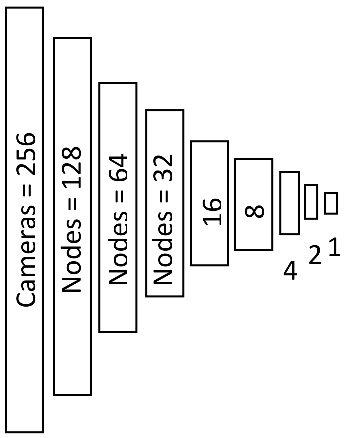
[0] [257] [20] [3] [2] [0] [0] [0]

Remaining points:

These are the remaining points to be reconstructed after each level

Levels 1 through 8

[282] [25] [5] [2] [0] [0] [0] [0]



These results are very interesting because most of the points are reconstructed at the second level of the tree, when groups of 4 cameras are analysed. In this level, the calculations are spread between 64 different nodes.

Tree analysis in a big space 40x40x5 meters^3

EXP. 2

A different situation is tested: the space is now of *40x40x5* meters^3 and the number of cameras is 256. The scope is to test the behaviour of the algorithm in the case of many cameras that do not see the markers.

The markers are positioned on 18 people and are in total 846 as Figure 4.12.

Starting from the whole set of 256 cameras (EXP 2A), the number of cameras is then reduced progressively (2B, 2C, 2D, 2E) to test more difficult situations, and to see how the performance of the algorithm changes.

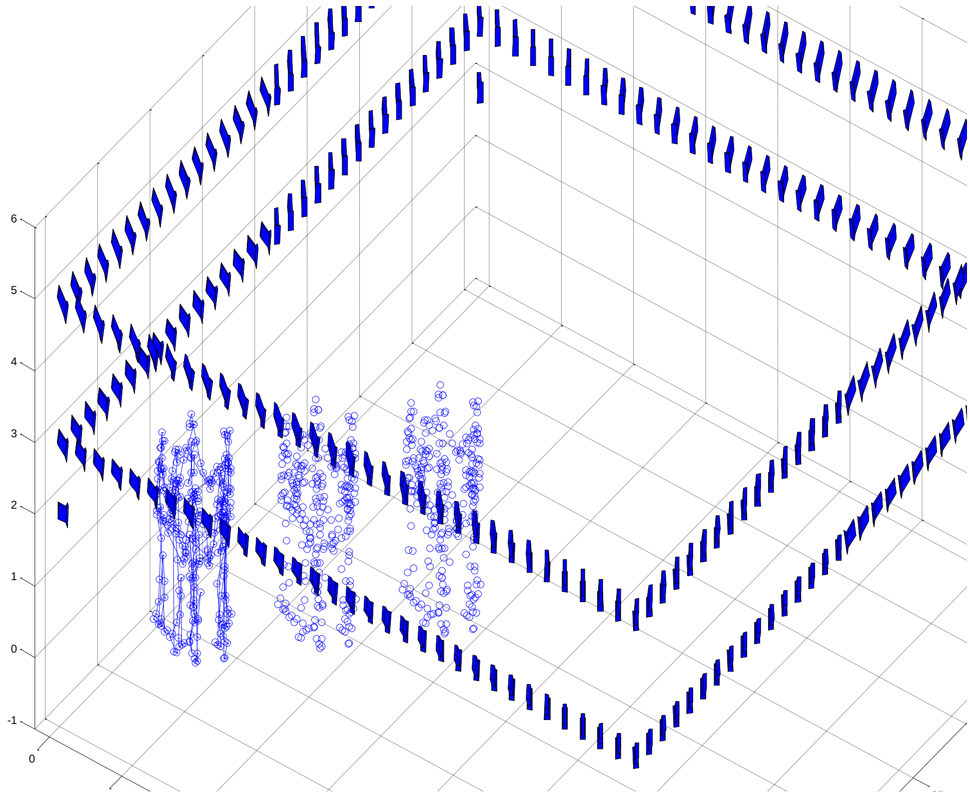
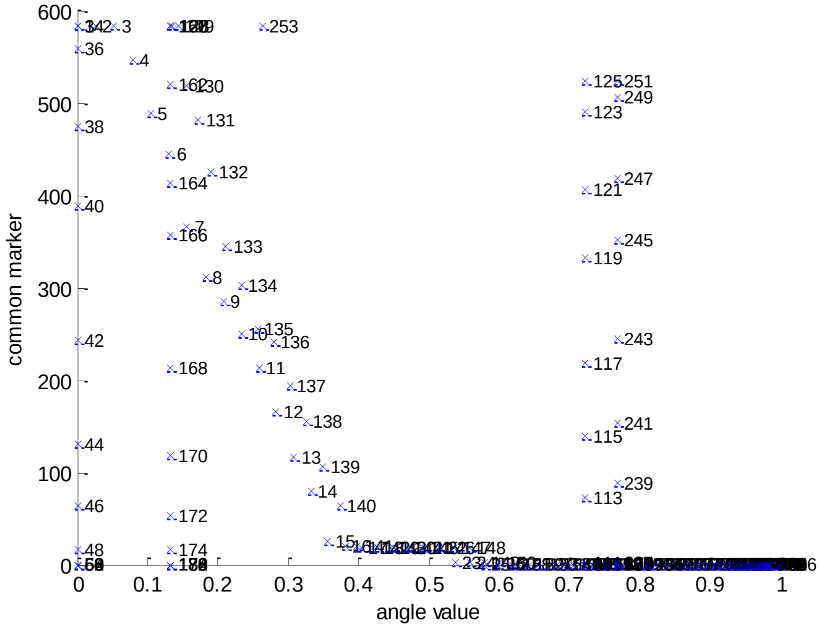
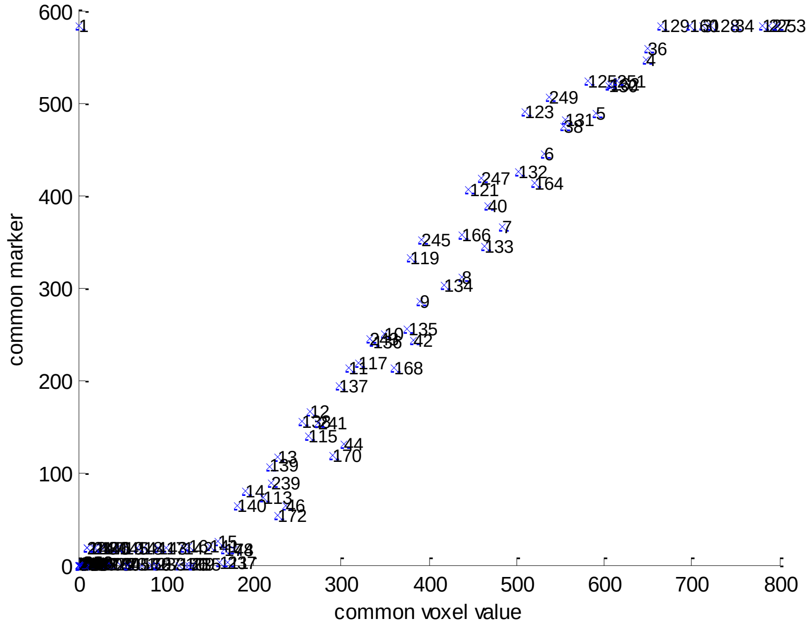


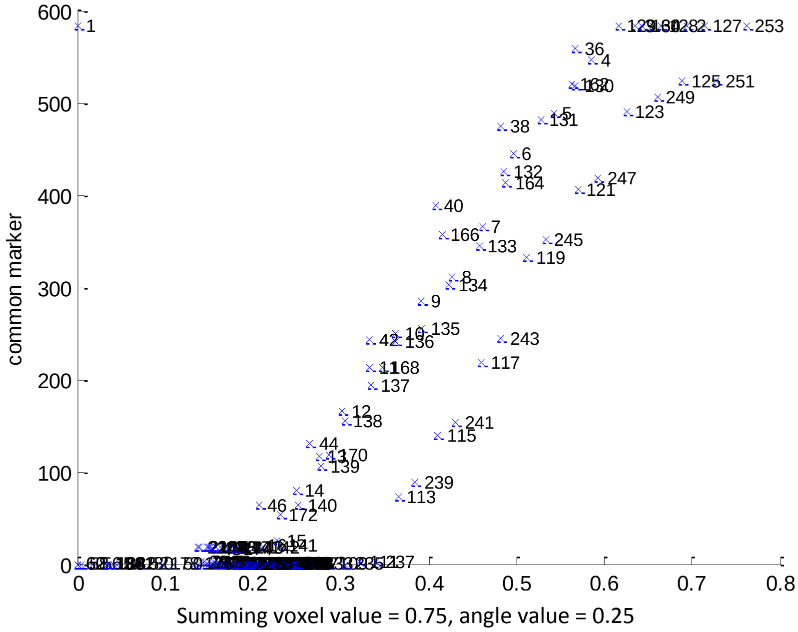
Figure 4.12. 846 points on 18 people and 256 cameras (EXP. 2A).



Graph 4.12. Big space Angle value.

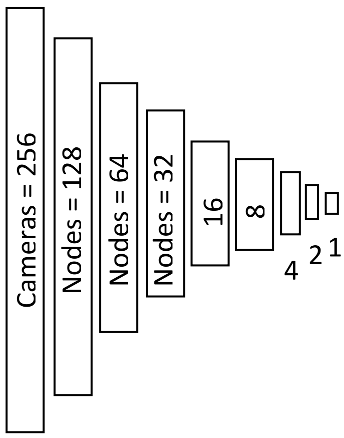


Graph 4.13. Big space common voxel value.



Graph 4.14. Big space summing voxel value = 0.75, angle value = 0.25.

Some markers are near the centre of the room so it is necessary to extend the visibility range of the cameras from 6 meters to 20 meters. The Graph 4.14 shows that lots of cameras don't share points with camera 1; in any case the "curve" is similar to a monotone function, so there is a good correlation between the real common markers and the affinity function calculated by the algorithm. In this set-up, every marker is seen by many cameras and all markers are correctly reconstructed at the second level of the tree.



Reconstructed points:

[0] [846] [0] [0] [0] [0] [0] [0]

Remaining points:

[846] [0] [0] [0] [0] [0] [0] [0]

EXP. 2B: 128 cameras trial

In the next example, the number of cameras is 128 but even with this configuration, all markers are reconstructed at the second level of the tree.

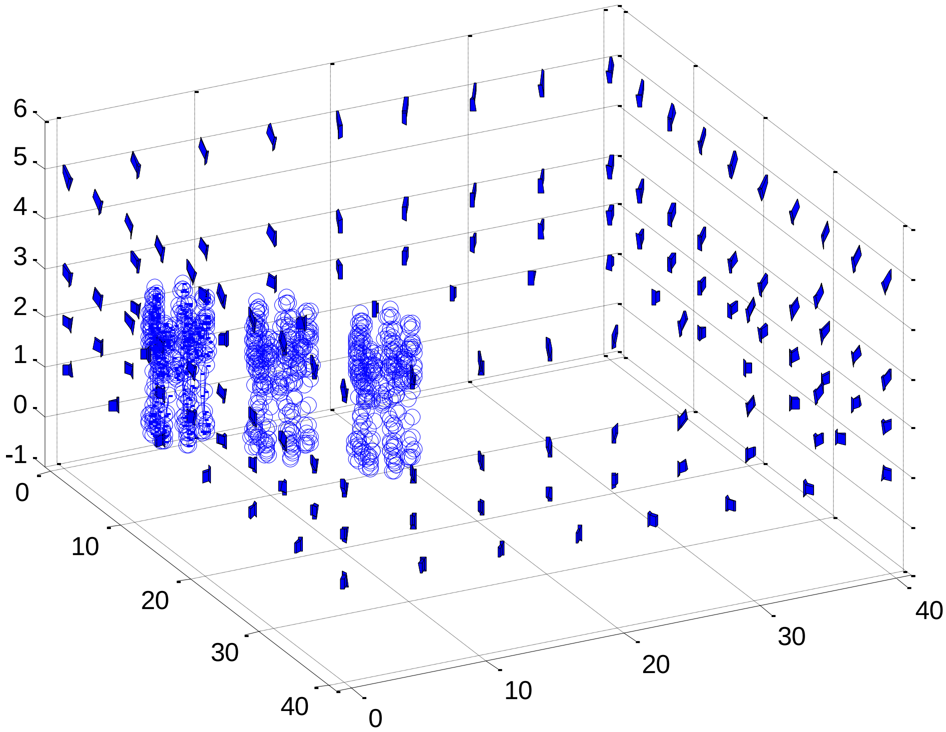


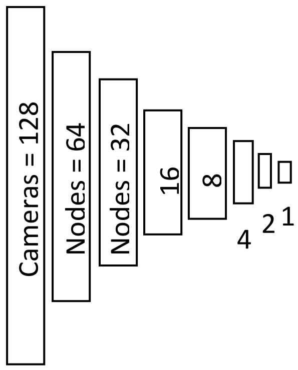
Figure 4.13. 846 points on 18 people and 128 cameras (EXP. 2B).

Reconstructed points:

[0] [846] [0] [0] [0] [0]

Remaining points:

[846] [0] [0] [0] [0] [0]



EXP. 2C: 64 cameras trial

In this experiment, the number of cameras is 64. As it is possible to see in the results, not all markers are reconstructed in the second level of the tree.

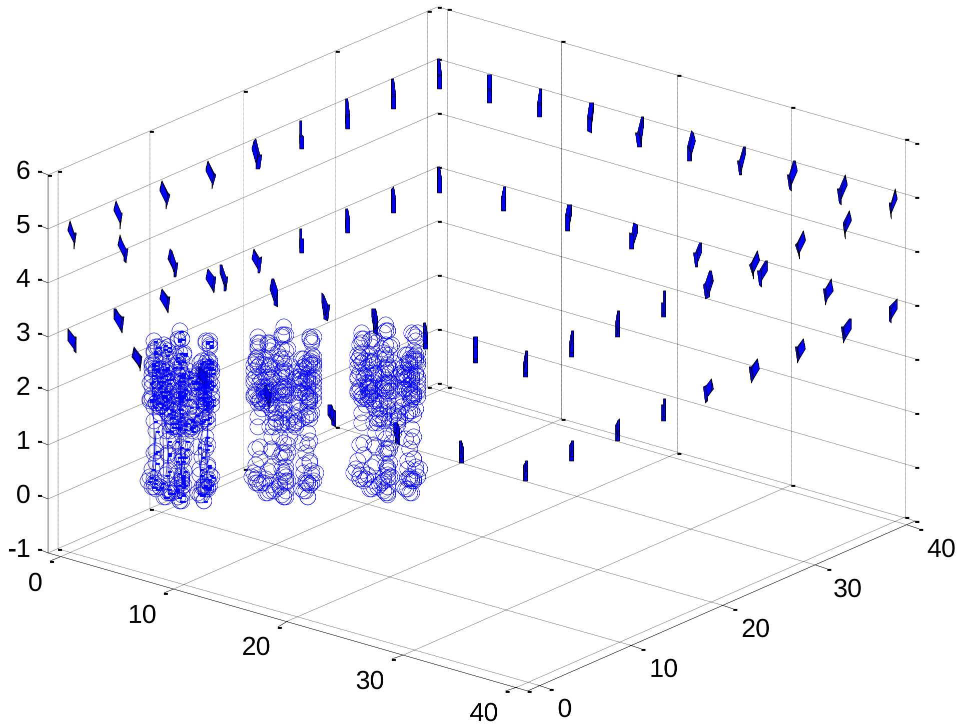


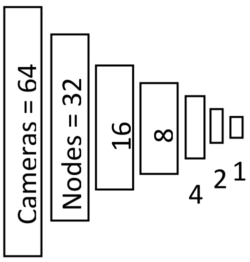
Figure 4.14. 846 points on 18 people and 64 cameras (EXP. 2C).

Reconstructed points:

[0] [779] [67] [0] [0]

Remaining points:

[846] [67] [0] [0] [0]



EXP. 2D: 32 cameras trial

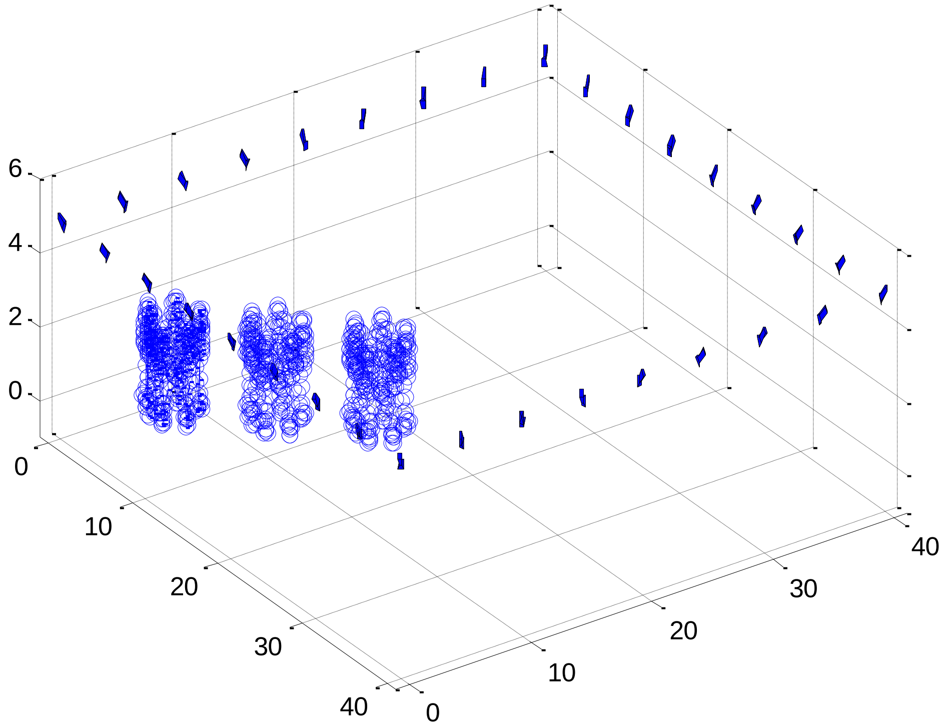


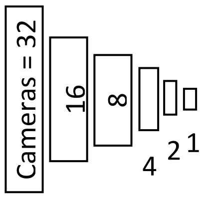
Figure 4.15. 846 points on 18 people and 32 cameras (EXP. 2D).

Reconstructed points:

[0] [632] [192] [22]

Remaining points:

[846] [214] [22] [0]



Even with 32 cameras all markers are reconstructed.

EXP. 2E: 16 cameras trial

The last test is done with 16 cameras. In this situation, 281 markers out of 846 are not seen by at least 3 cameras; most of the other markers are reconstructed at the second level of the tree.

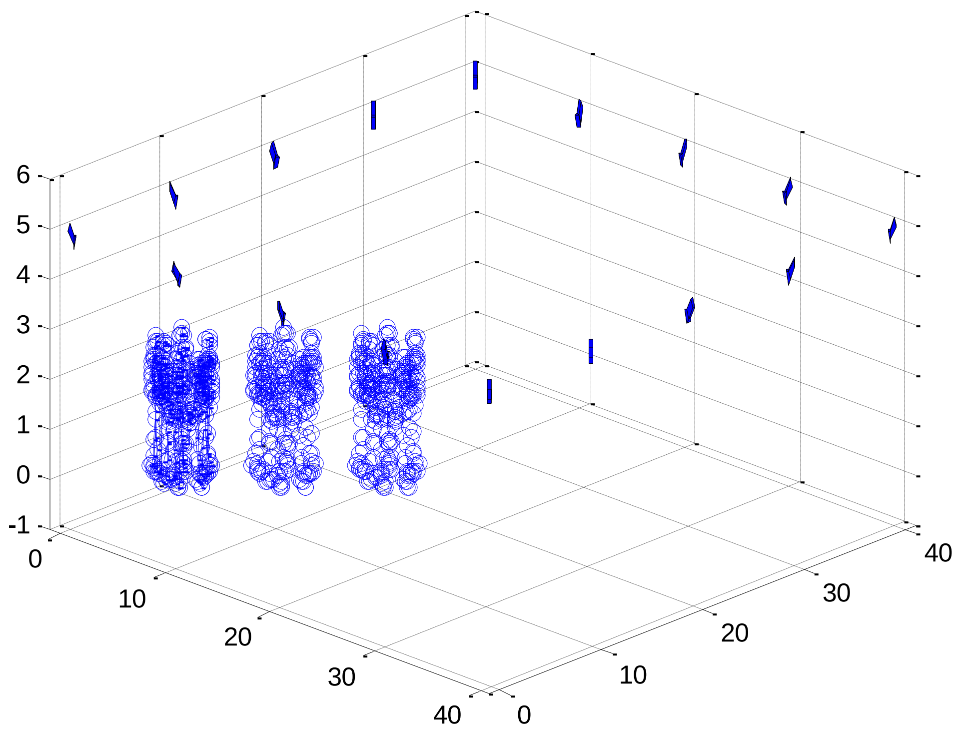
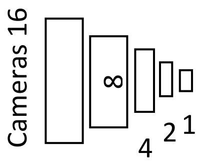


Figure 4.16. 846 points on 18 people and 16 cameras (EXP. 2E).

Reconstructed points:

[0] [538] [27]



Remaining points:

[846] [308] [281]

The unreconstructed points are concentrated in the centre of the room. This is because the cameras are 8 m far from each other and so there will be some areas not covered by three cameras.

In all these tests, at least 2/3 of the total markers are reconstructed in the first half of the tree and most of the reconstructed 3D points are always in the second. This is very important to guarantee that the algorithm makes most of the calculations in the first levels, when the number of nodes is high and the elaboration is spread between many different CPU.

4.7 Full tree analysis

The tree obtained for a set-up of 16 cameras is in Table 4.1:

Level 1	**Level 2**	**Level 3**
[1 12]		
[2 9]		
[3 10]	[1 12 4 11]	
[4 11]	[2 9 3 10]	[1 12 4 11 2 9 3 10]
[5 8]	[5 8 6 7]	[5 8 6 7 13 16 14 15]
[6 7]	[13 16 14 15]	
[13 16]		
[14 15]		

Table 4.1. Tree representation.

The 16 cameras in the *5x5x5* m space are installed as in Figure 4.17:

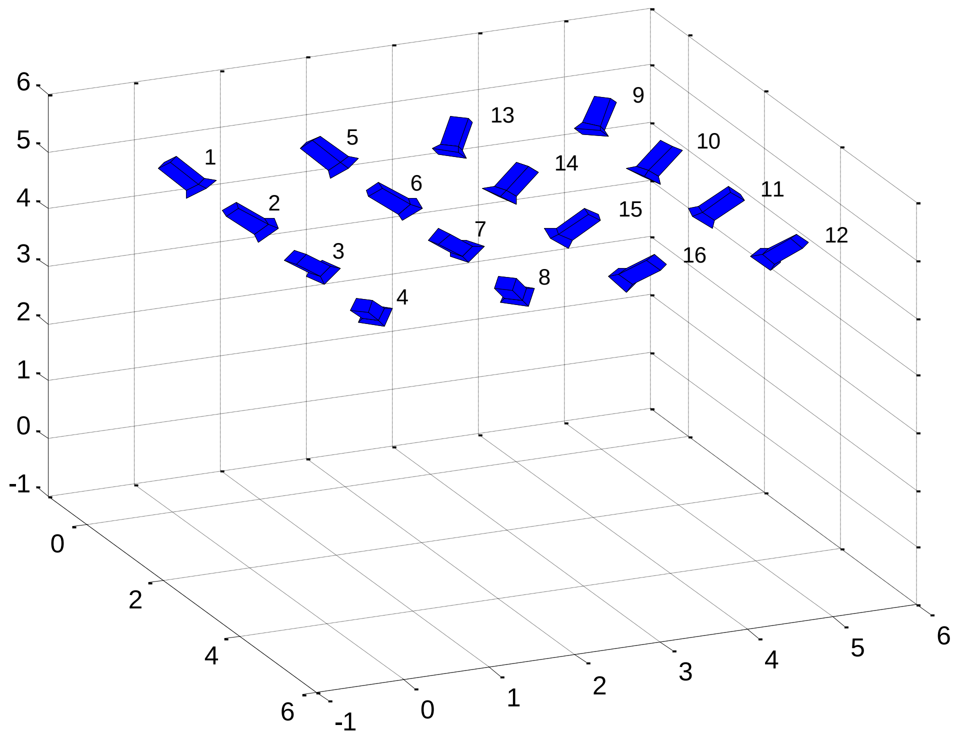


Figure 4.17. 12 cameras set-up.

In Figure 4.18 is possible to see the volume that cameras 1 and 12 share.

The normal angle of the two cameras is *48.21°* and it is bigger than all the others.

The cameras also share the biggest number of voxels. Considering a voxel size of 0.25 m, they are 4868.

These two cameras are associated at the first level of the tree. In the second level of the tree, cameras 4 and 11 are grouped together with 1 and 12. As it is possible to see in Figure 4.17, camera 4 has the best angle in respect to both cameras 1 and 12 and shares 4836 voxels with camera 1. Camera 11 has a good angle with camera 1 and shares a good volume with both cameras 1 and 12.

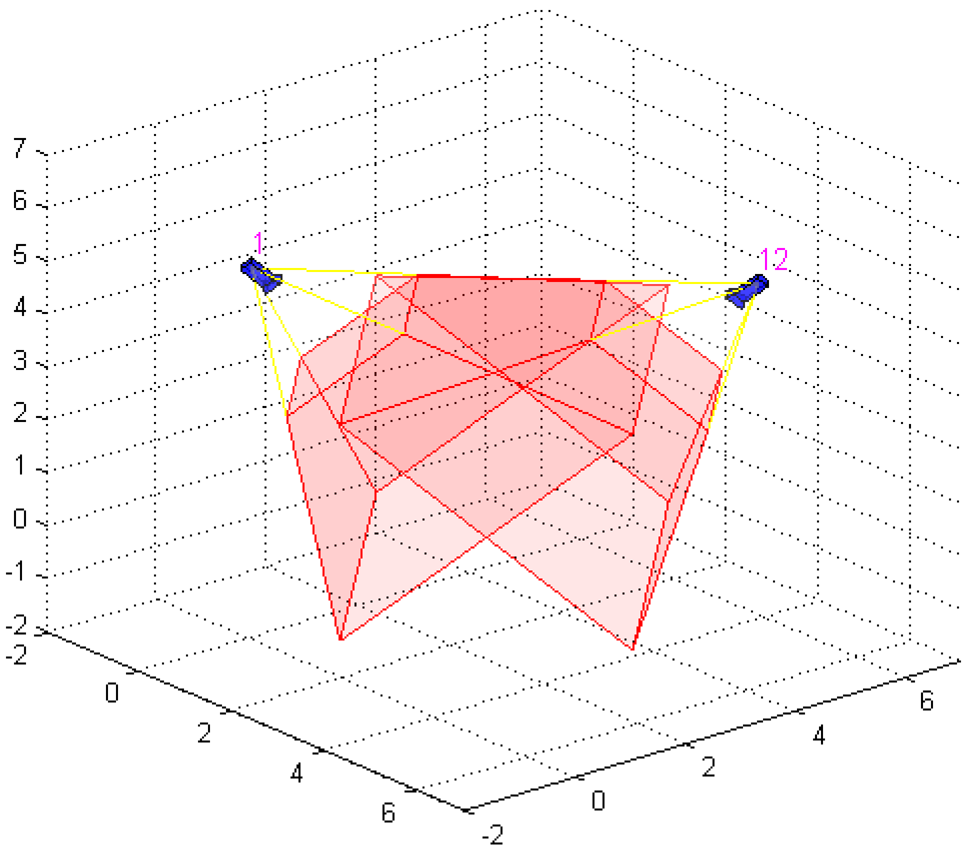
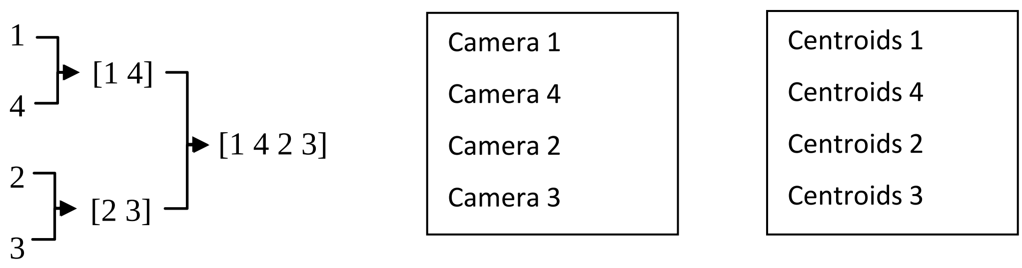


Figure 4.18. Cameras 1 and 12 visibility volume.

5 Results

In this chapter, all results obtained during the internship are shown.
The first tests presented are about the time required by the distributed algorithm to do the reconstruction using a random or an ordered system to create the association tree. Ordered system is referred to the use of the functional cost for creating the tree.
A set-up is synthetically created with 256 cameras and 846 markers using the *Matlab* functions that store that information in two txt files. The position of the 846 markers is taken from a real file with three groups of 6 people. This information is read by the *Reconstructor class*. The cameras and centroids information are saved in the same order of the association tree.

The txt file created contains the information in the same order of the tree



5.1 Random Data

The random tests are done saving all cameras information and centroids randomly in the files. In Table 5.1 there is the mean result of 10 different tests.
In the tables below, there are in general 4 columns: the **Total time** is the sum of the times taken by all nodes in each level of the tree. Their sum is the total time required by the distributed algorithm to be executed on a single core. The **Max time** column contains the time required by the slowest node in each level of the tree. Their sum is the total time required by the distributed algorithm to be executed in a real

distributed environment (excluding the times of the communication). The **2D** and **3D data** in each level is the sum of the 2D and 3D point in each node.

Level	Total time (s):	MAX time (s):	2D Data:	3D Data:
1	0.0130	0.0030	7588	123
2	0.0298	0.0045	7530	409
3	0.0378	0.0065	7348	810
4	0.0583	0.0148	6767	1454
5	0.0830	0.0218	5542	2050
6	0.0780	0.0280	3046	2205
7	0.0395	0.0208	750	1618
8	0.0193	0.0183	148	906
		Sum of total time= **0.1175**		

Table 5.1. Random data results with 256 cameras.

5.2 Ordered Data

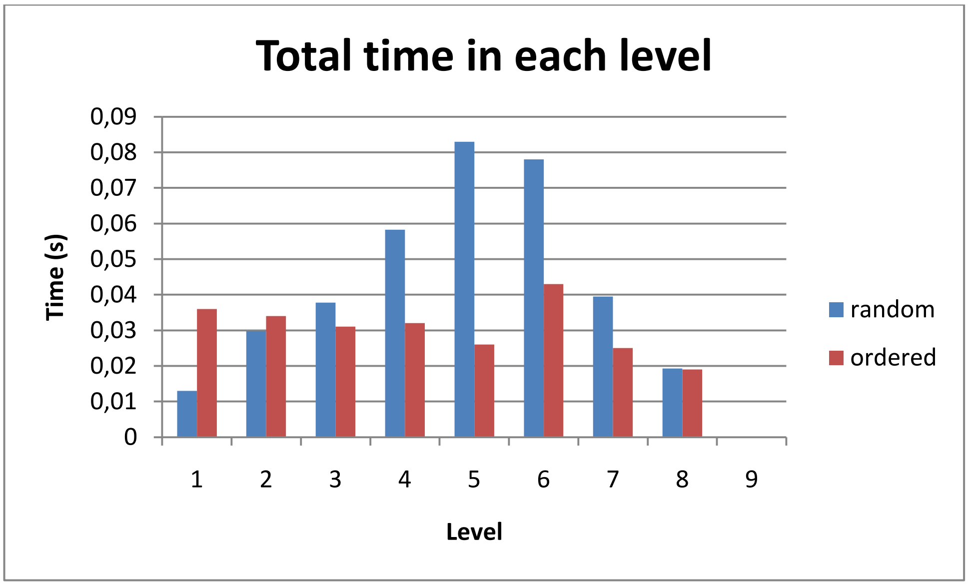
Using the functional costs for creating the tree, the results are Table 5.2:

Level	Total time (s):	MAX time (s):	2D Data:	3D Data:
1	0.036	0.005	7588	1892
2	0.034	0.004	5349	2256
3	0.031	0.006	3436	2169
4	0.032	0.01	2856	2072
5	0.026	0.011	2157	1923
6	0.043	0.02	1496	1966
7	0.025	0.017	492	1313
8	0.019	0.019	102	853
		Sum of total time = **0.092**		

Table 5.2. Ordered data results with 256 cameras.

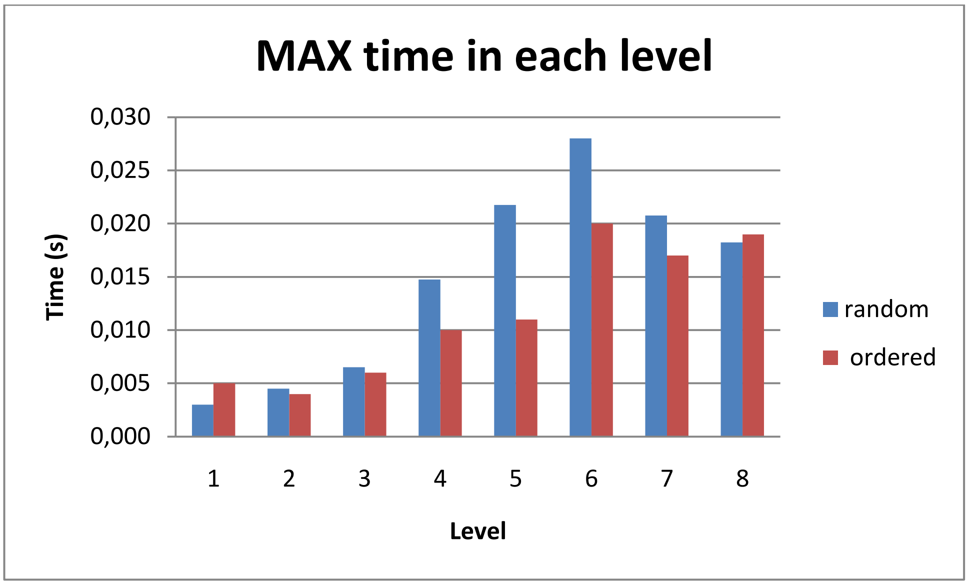
5.3 Detailed analysis of the difference between the two versions

From Graph 5.2 to Graph 5.7 it is possible to see a comparison between the two versions.



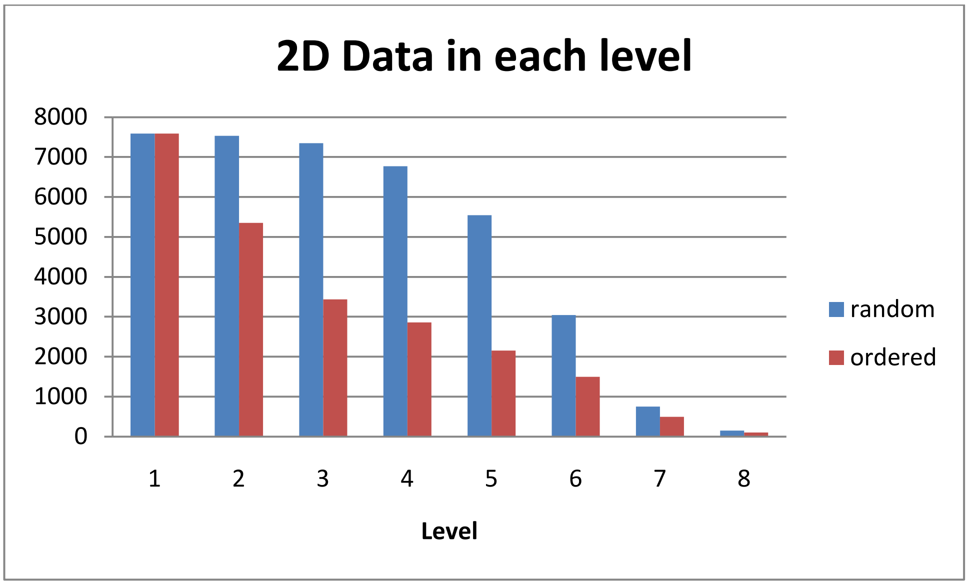
Graph 5.1. Total time in each level (ordered version).

From Graph 5.1is possible to notice that the random version is slowly in all levels apart from the first two in which the distributed version do most of the reconstruction. Comparing each column of Graph 5.1 with Graph 5.2 it is possible to have an idea of the distribution of the calculation among the node. For example in the ordered version, the first level, takes 0.035 s. In that level there are 128 nodes so each node may constribute only with 0,27 ms. Instead the slowest node constributes for 5 ms.



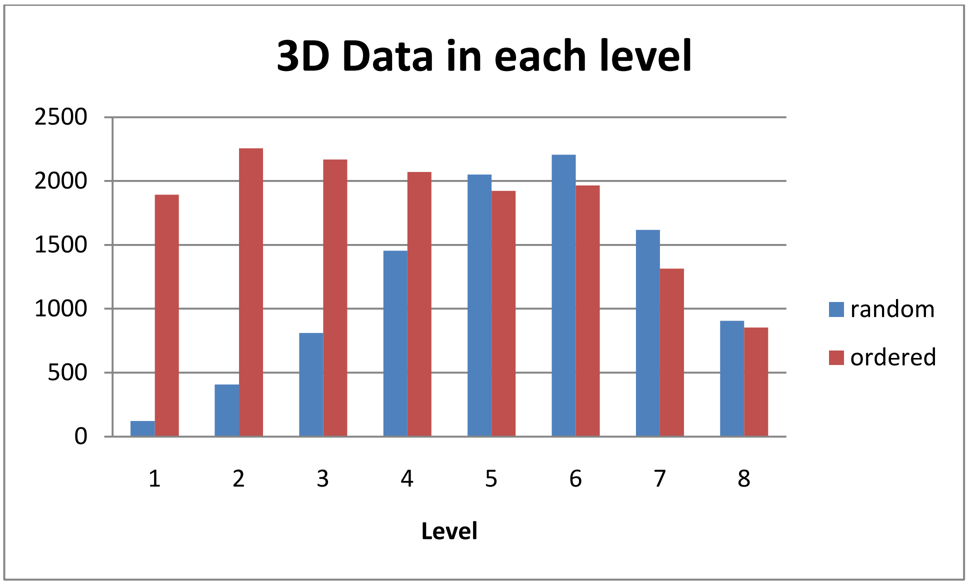
Graph 5.2. MAX time in each level (both versions).

In Graph 5.2 it is possible to see the differences in time spent by the slowest node in each level of the tree. In the first level, the algorithm using the ordered tree is slower than the random one and this is normal considering that the ordered algorithm in this phase should be able to do more reconstruction. The main differences are in levels 4, 5, 6 and 7. The random algorithm in these phases is slower because the number of 2D points to analyse is very high. The ordered algorithm, instead, has already cut most of the 2D points, so the time spent in each level is shorter. In the last level, the time spent is almost the same for the two versions because the number of 2D points is small for both and most of the computation time is spent for merging the 3D points.



Graph 5.3. 2D Data in each level (both versions).

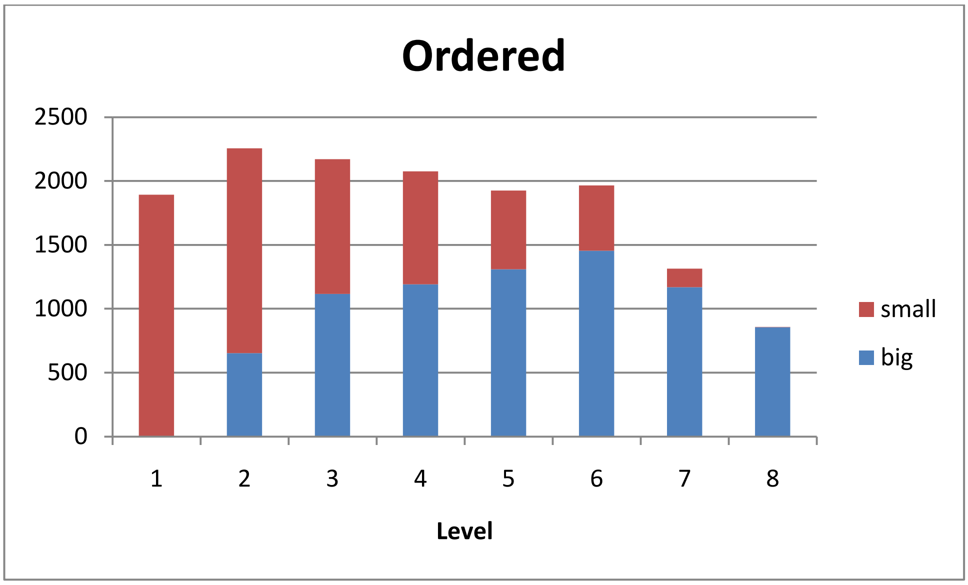
In Graph 5.3 it is possible to see the importance of having a good affinity method.
In the first level of the tree no 2D points are deleted. In the second level, it is possible to notice how the use of the ordered system permits to delete the 30% of the 2D points and in the third level another 35% of the points are cut. The random reconstruction cuts a mere 4% between level 1 and level 3. These results perfectly demonstrate the importance of using a good association function. Between level 4 and 6, each node analyzes from 16 to 64 cameras, so it is fundamental to keep low the number of 2D points.



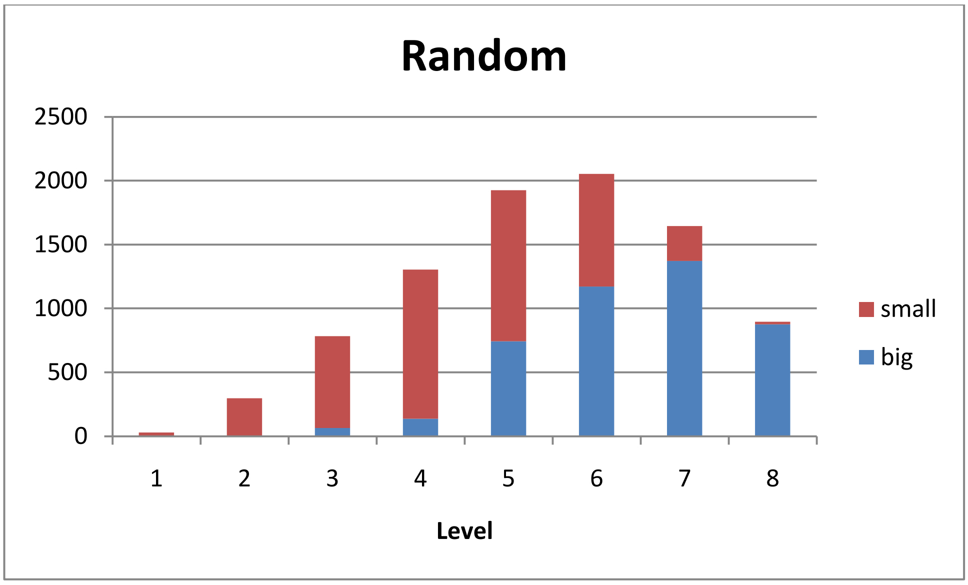
Graph 5.4. 3D Data in each level (both versions).

In the 3D data analysis in Graph 5.4 it is possible to see how the ordered reconstruction is able to create lots of 3D points already in the first level of the tree. The total number of points in the graph is bigger than the real number of points (846), because it is the sum of the 3D points reconstructed in all nodes of the level. Part of them will be deleted, because they are ghost points and other will be merged in other levels of the tree. As it is possible to see in Graph 5.5 and Graph 5.6, the number of BIG 3D points is higher in the ordered tree. This can be correlated to the faster reduction of the 2D points, because when a BIG 3D point is created all the 2D points associated with it are deleted.

The bigger number of 3D points in the first levels of the ordered version doesn't slow the algorithm very much because they are distributed in different nodes.



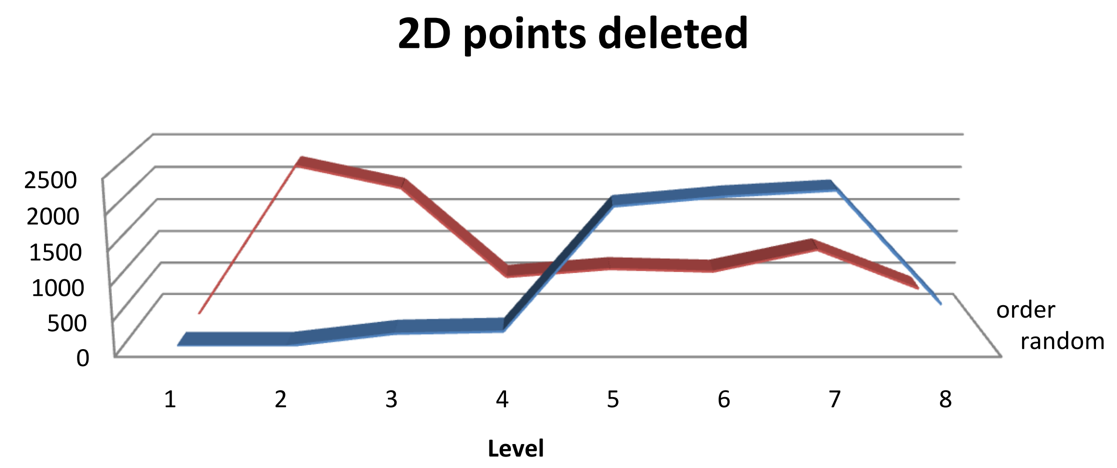
Graph 5.5. 3D BIG and SMALL points in each level (ordered version).



Graph 5.6. 3D BIG and SMALL points in each level (random version).

General consideration

The time taken by the ordered version is 0.092 seconds, while the mean time of the random version is 0.1175 seconds that corresponds to 27% more.



	1	2	3	4	5	6	7	8
random	0	0	184	216	2011	2149	2235	608
order	0	2248	1916	573	695	655	987	391

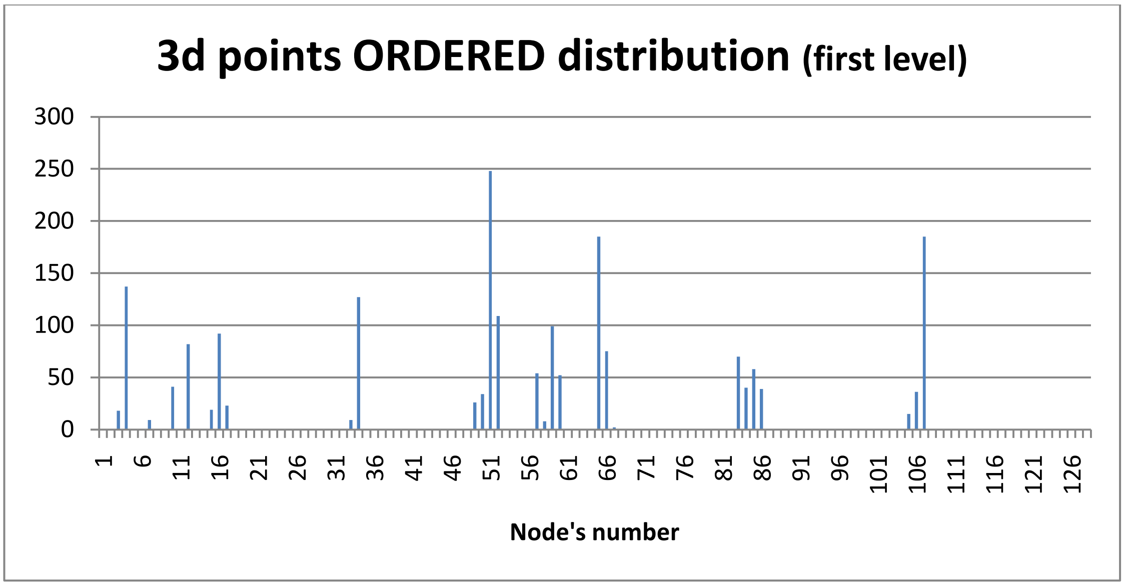
Graph 5.7. 2D points deleted in each level (both versions).

In this Graph 5.7 is shown in more detail the number of 2D points that are deleted in each level of the tree. The ordered version reduces them in the first levels of the tree, so the total time required to analyze them is lower.

During the internship the algorithm has been improved a lot so the time required may vary respect to the first results. Every time that a comparison between two versions has been done, to make consistent comparison, the same data has been used and the algorithm was at the same development version.

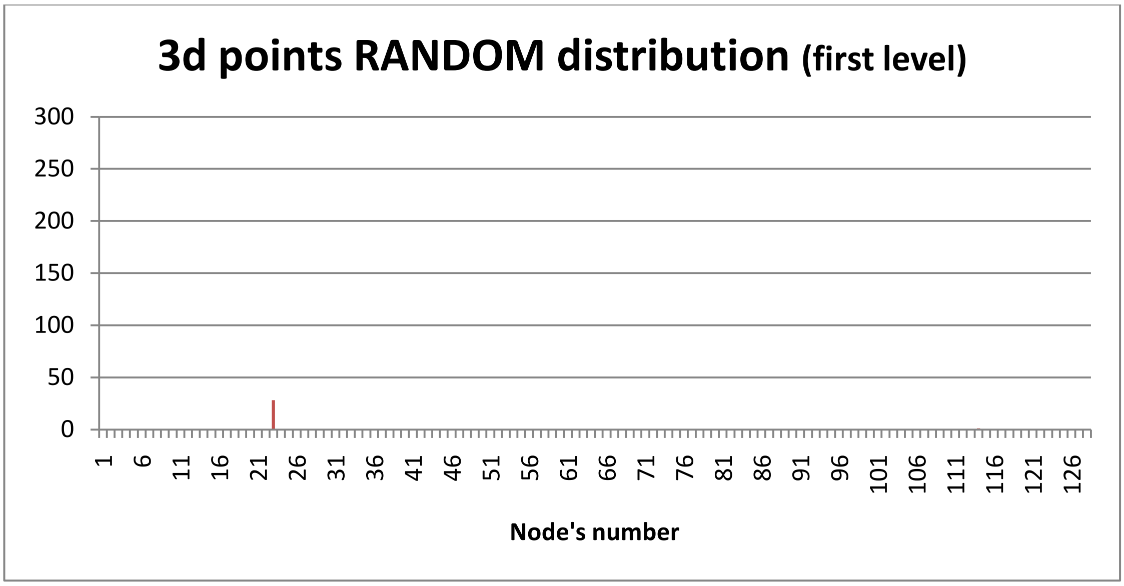
5.4 Detailed Analysis of the distribution of the 3D points in the nodes of the tree

The total time required by the algorithm is calculated summing for each level of the tree the time required by the slowest node using equation (3.1). That time is influenced by the distribution of the markers in the volume. In case the markers are uniformly distributed, each camera sees only a fraction of them and the reconstruction will be faster, because the calculation are uniformly distributed on all the nodes. Instead, in a real case it is more likely to have zones with a big amount of markers and zones with no markers. To recreate a real situation, in the volume there were 3 groups with 6 people each one. Between Graph 5.8 and Graph 5.13 is possible to see the number of 3D points reconstructed in each node of the ordered and random version of the tree.



Graph 5.8. 3D points distribution in the first level using the ordered tree.

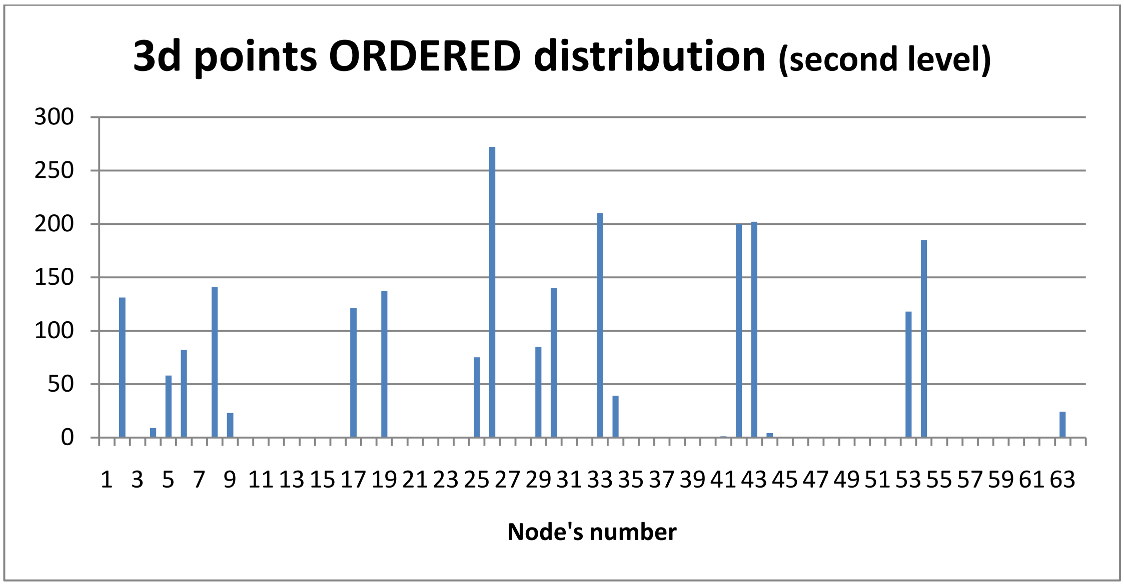
Graph 5.8 represents the number of 3D markers reconstructed by each node of the first level of the tree. It is possible to notice how there are some nodes that reconstructs a big number of points and this should guarantee that the set-up represents a real situation and it is similar to a worst case scenario. The second thing is that there are “clusters”[2] of nodes that reconstruct a big number of 3D markers, for example node 51 and 52 participate respectively to the reconstruction of 248 and 109 3D points. In the next level of the tree all those points will be merged together. It is very important to highlight the presence of “cluster”, because this guarantees that at the next level this information will be matched and it is very likely that a lot of SMALL 3D points will be matched with other SMALL 3D points to create BIG 3D points.



Graph 5.9. 3D points distribution in the first level using the random tree.

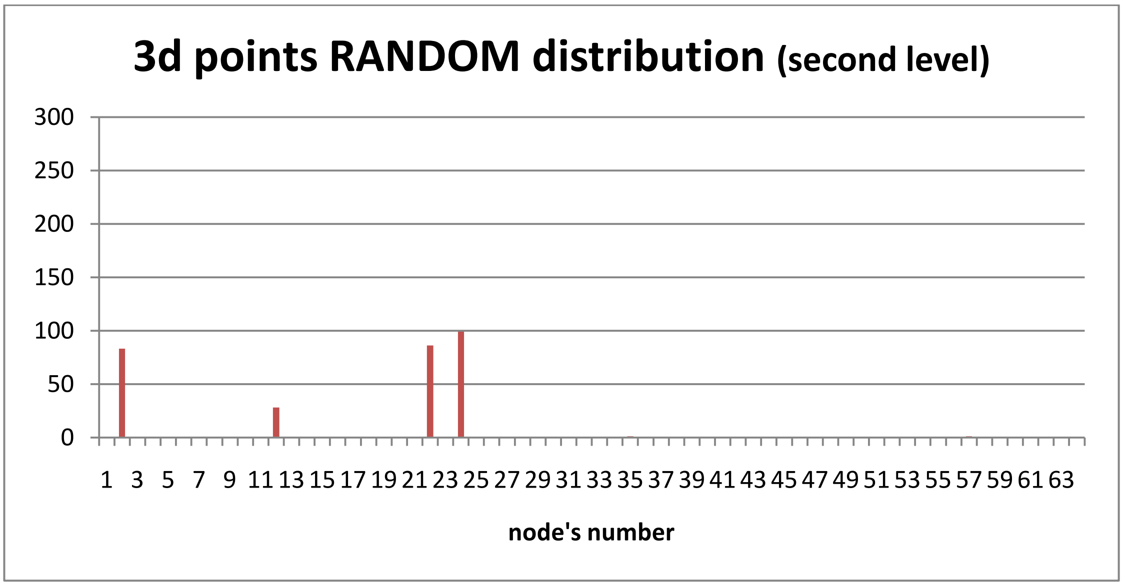
Graph 5.9 represents the number of 3D points reconstructed in each node of the first level of the random tree. As it is possible to see, only node 23 is able to reconstruct something.

[2] A cluster is a group of nodes that will be analyzed in the next level of the tree; every odd node is merged with the following even node. If the odd node number x and the node x+1 reconstruct lots of 3D points, in the next level, all this points will be merged



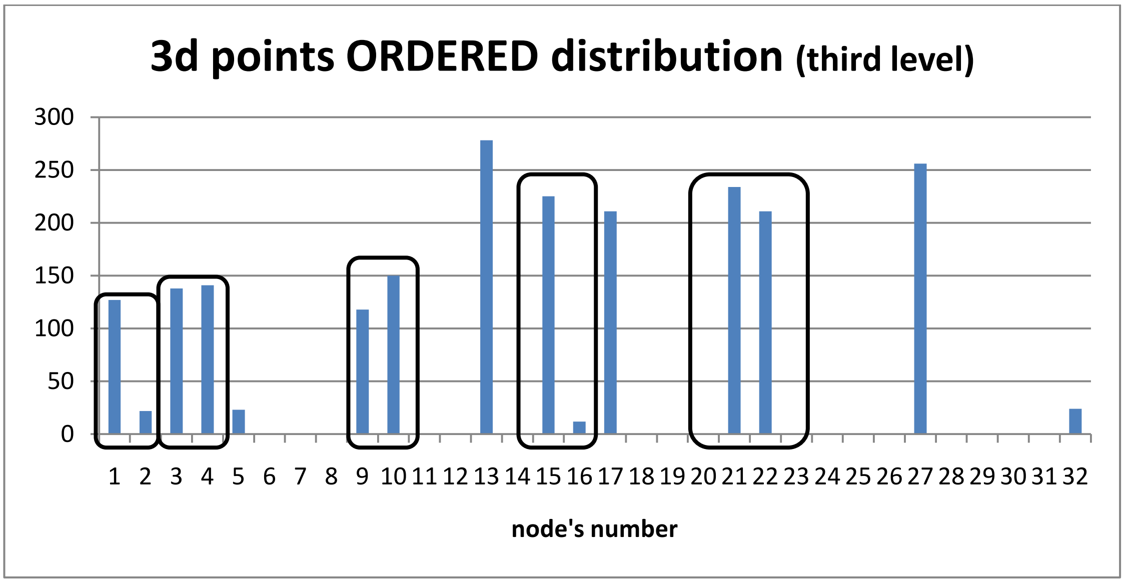
Graph 5.10. 3D points distribution in the second level using the ordered tree.

In the second level of the tree (Graph 5.10 and Graph 5.11), it is possible to notice how some nodes do most of the work while some other don't reconstruct any points. The distribution of the markers again is in "clusters" and it means that also the function used in the high levels of tree well associate cameras.



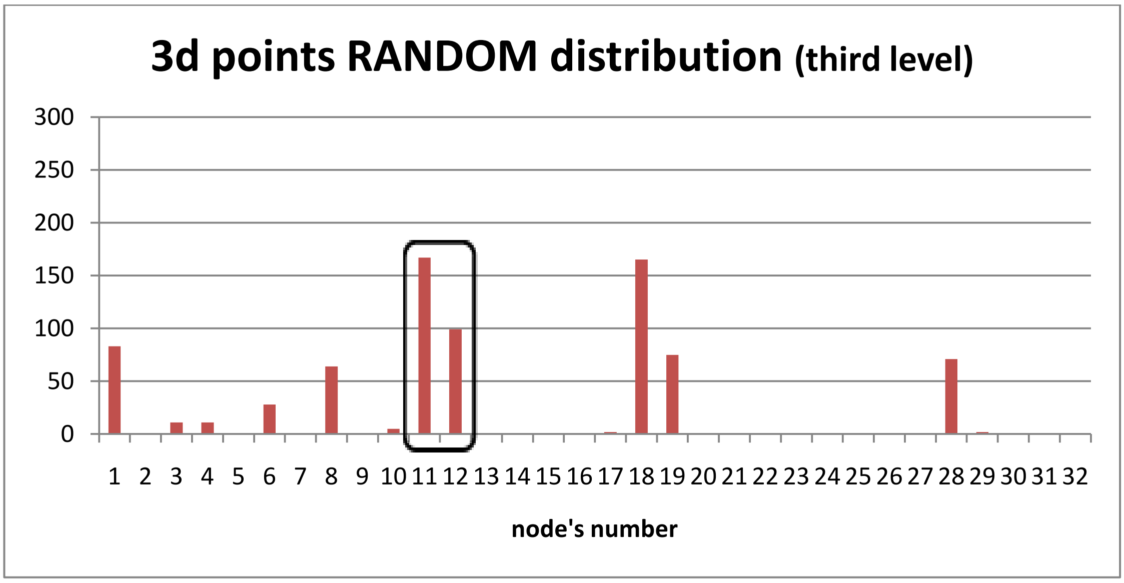
Graph 5.11. 3D points distribution in the second level using the random tree.

In the random version, on the second level of the tree the quantity of the 3D points reconstructed in the nodes is still very low.



Graph 5.12. 3D points distribution in the third level using the ordered tree.

Even at the third level of the tree (Graph 5.5 and Graph 5.6), the nodes that merge lot of points are in cluster, in this example there are 5 of them (as is possible to see). In Graph 5.13 with the random version there is only one cluster (node's number 11-12).



Graph 5.13. 3D points distribution in the third level using the random tree

5.5 Estimating the performance using a different number of cameras

An important thing to analyze is the speed of the algorithm changing the number of cameras.

The tests are done with 128, 256, 512 and 1024 cameras and using an ordered tree. The volume is *32x32x5* m and all cameras are placed on the ceiling. The actual algorithm works only if the number of cameras is a power of 2 but it could be easily modified adding dummy cameras. For every test, the minimum visibility distance for each camera is always 2 m and the maximum visibility distance is specified.

Test with 128 cameras (846 markers):

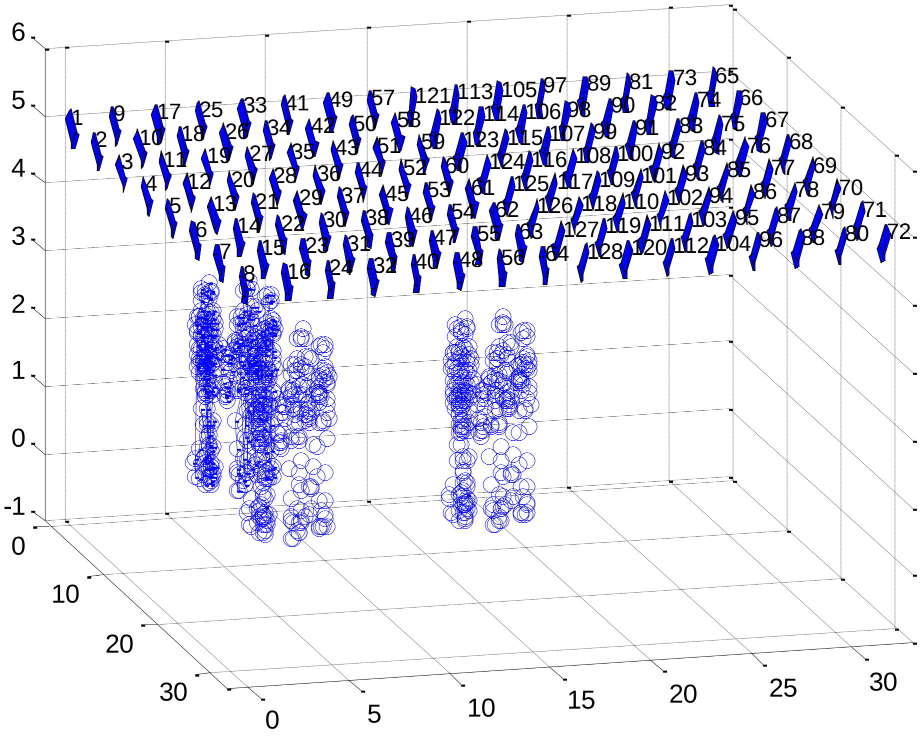


Figure 5.1. Set-up with 128 cameras and 846 markers.

D_{max} = 6 m

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.003	3456	0	0	978
2	0.004	2905	551	185	1048
3	0.004	2089	816	418	1148
4	0.009	1618	471	450	908
5	0.013	1552	66	472	880
6	0.018	737	815	592	460
7	0.027	267	470	700	140
	Total time = 0.078				

Table 5.3 846 markers and 128 cameras (Dmax = 6 m)

Markers found = 700 out of 846 Ghost = 0.

Not all markers of Figure 5.1 are reconstructed. As it is possible to see from Figure 5.2, most of the points not found are in the lower part of the volume near the borders.

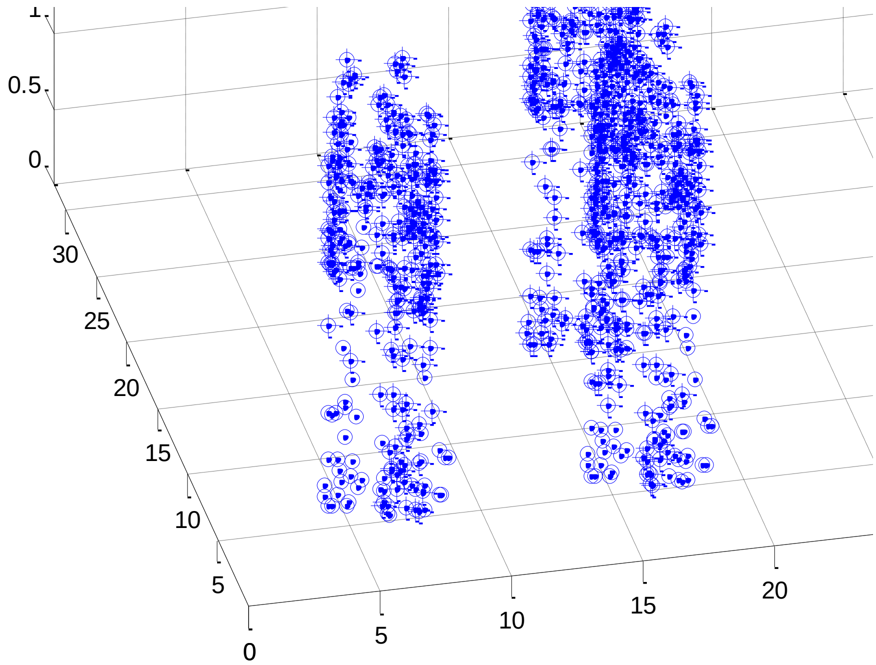


Figure 5.2. Detailed view of the markers not reconstructed (the ones without the "+") by the 128 cameras setup.

To avoid the problem of the markers that are not reconstructed, the visibility distance of the cameras is increased to D_{max} = 8 m.

After some trials with different parameters the results are in Table 5.4:

```
const double distance3D = 0.01;
const double distance2d3d = 0.02;
const double SamponImageError = 0.000005;
```

Level	MAX time:	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.008	7428	0	0	5367
2	0.016	3411	4017	1190	1982
3	0.011	1257	2154	1457	880
4	0.004	528	729	1260	344
5	0.013	528	0	1260	344
6	0.008	232	296	1003	183
7	0.015	11	221	849	2
	Total time = 0.075				

Table 5.4. 846 markers and 128 cameras (Dmax = 8 m).

Increasing the visibility distance of the cameras, all markers are found and the time required is lower because the algorithm will delete faster the 2D points.

Test with 256 cameras (846 markers):

After several optimisations of the code, using the same parameters as the other experiment, the results with 256 cameras are in table:

D_{max} = 6m

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.006	7588	0	0	1892
2	0.003	5369	2219	635	1626
3	0.006	3517	1852	1079	1090
4	0.01	2957	560	1156	914
5	0.009	2230	727	1285	638
6	0.013	1558	672	1433	533
7	0.01	556	1002	1162	145
8	0.018	167	389	846	0
	Total time = 0.075				

Table 5.5. 846 markers and 256 cameras (Dmax = 6 m).

Test with 512 cameras (846 markers):

D_{max} = 6m

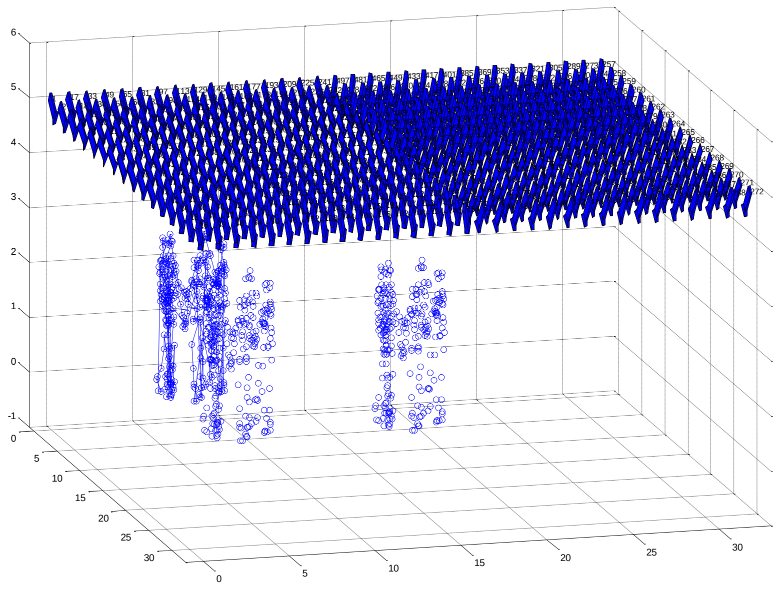


Figure 5.3. Set-up with 512 cameras and 846 markers.

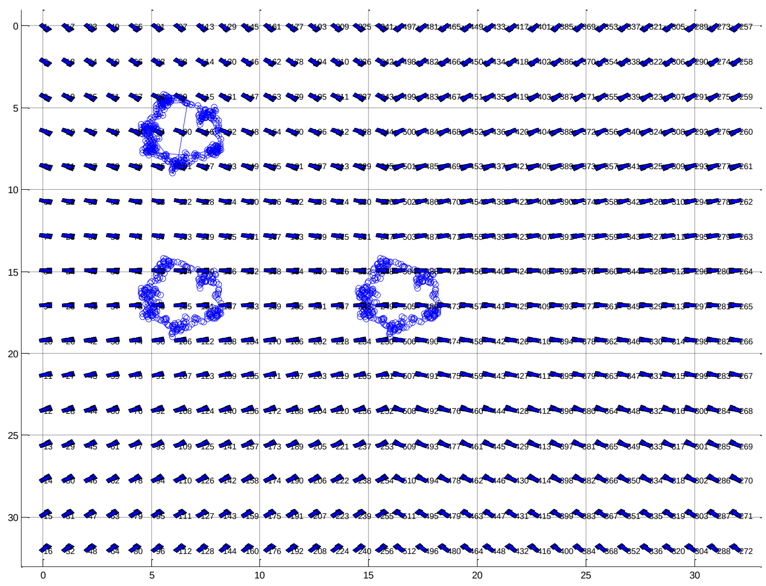


Figure 5.4. Set-up with 512 cameras and 846 markers (ceiling view).

After different trials with 512 cameras the results are in Table 5.6:

```
const double distance3D = 0.01;
const double distance2d3d = 0.005;
const double SamponImageError = 0.000005;
```

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.007	15787	0	0	5172
2	0.004	7951	7836	2240	3490
3	0.003	4275	3676	2501	1867
4	0.004	1644	2631	2453	767
5	0.007	1475	169	2265	694
6	0.004	842	633	1995	356
7	0.004	230	612	1470	79
8	0.01	83	147	1228	29
9	0.012	20	63	861	0
	Total time = 0.055				

Table 5.6. 846 markers and 512 cameras (Dmax = 6 m).

Test with 1024 cameras (846 markers):

D_{max} = 6m

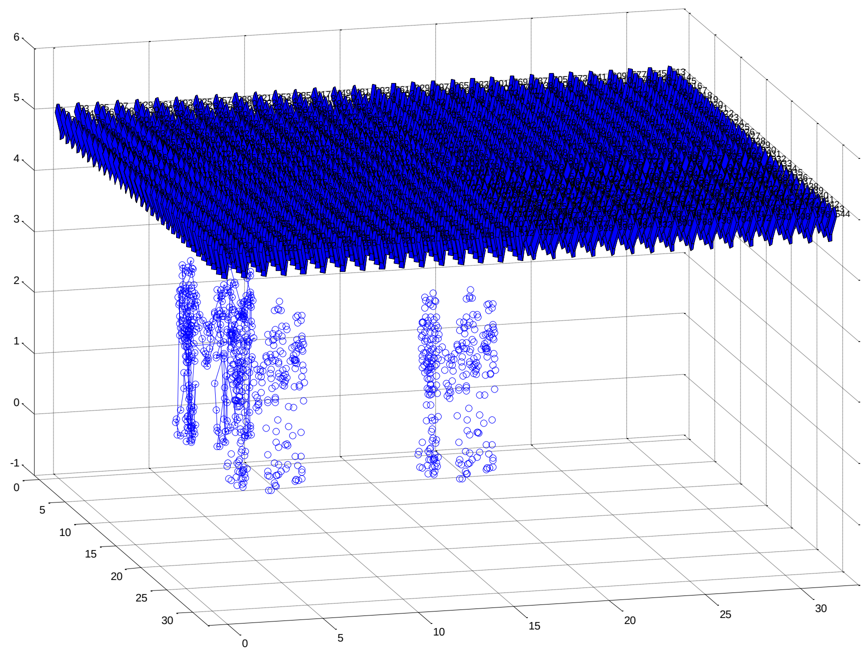


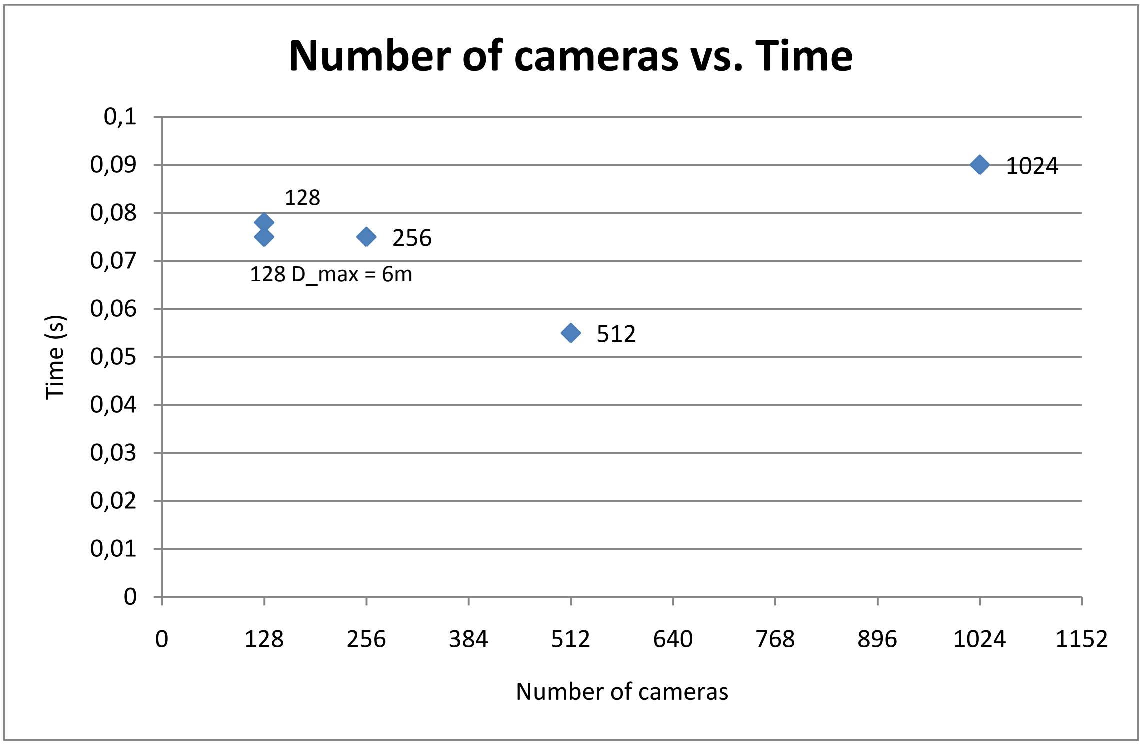
Figure 5.5. Set-up with 1024 cameras and 846 markers.

After several trials with different parameters, the best results are Table 5.7:

```
const double distance3D = 0.01;
const double distance2d3d = 0.005;
const double SamponImageError = 0.000005;
```

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.007	32631	0	0	14212
2	0.004	11042	21589	5898	3789
3	0.003	6378	4664	5269	2137
4	0.006	4696	1682	4863	1701
5	0.008	3820	876	4525	1354
6	0.01	2720	1100	4437	1080
7	0.011	1149	1571	3459	432
8	0.008	348	801	2333	88
9	0.009	256	92	1278	61
10	0.024	120	136	848	9
	Total time = 0.09				

Table 5.7. 846 markers and 128 cameras (Dmax = 6 m).



Graph 5.14. Comparison of the time required by a different number of cameras.

As it is possible to see from Graph 5.14, increasing the number of cameras does not correspond to a proportional increase in time required by the reconstruction. There are two points with 128 cameras (corresponding to different visibility distance).

The fastest reconstruction is done with 512 cameras. It may seem unusual that the fastest reconstruction is the one with 512 cameras because the initial amount of 2D data is bigger than the 128 and 256 cameras tests .

In Table 5.8, Table 5.9 and Table 5.10 is possible to see the amount of 2D and 3D points in each level of the tree for the different numbers of cameras.

2D points			
1024			
32631	**512**		
11042	15787	**256**	
6378	7951	7588	**128**
4696	4275	5369	3456
3820	1644	3517	2905
2720	1475	2957	2089
1149	842	2230	1618
348	230	1558	1552
256	83	556	737
120	20	167	267

Table 5.9. 2D points comparison.

3D total points			
1024			
14212	**512**		
9687	5172	**256**	
7406	5730	1892	**128**
6564	4368	2261	978
5879	3220	2169	1233
5517	2959	2070	1566
3891	2351	1923	1358
2421	1549	1966	1352
1339	1257	1307	1052
857	861	846	840

Table 5.8. 3D points comparison.

Time (s)			
1024:			
0.007	**512:**		
0.004	0.007	**256:**	
0.003	0.004	0.006	**128:**
0.006	0.003	0.003	0.008
0.008	0.004	0.006	0.016
0.01	0.007	0.01	0.011
0.011	0.004	0.009	0.004
0.008	0.004	0.013	0.013
0.009	0.01	0.01	0.008
0.024	0.012	0.018	0.015

Table 5.10. Time comparison.

The green boxes are those with the minimum amount of points or time. It is possible to see the high correlation between the number of 2D points and the time required by the algorithm. The 512 cameras set-up from the sixth-last level has less 2D points. Instead, there is not a strict correlation between the time and the number of 3D points.

The distance between the cameras in the 512 cameras set-up is about 1 m. This is probably the best condition for deleting quickly the 2D points associated with the BIG 3D points.

5.6 Markers placed in different positions

To verify the results obtained, the markers are dislocated to a more central position. All 256 cameras look partially at the centre of the volume so more cameras should see the 846 markers.

The position used in all previous tests was in Figure 5.6:

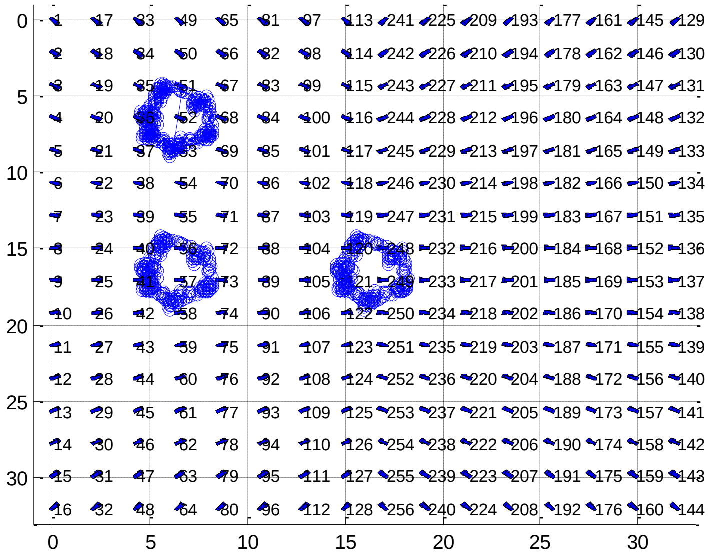


Figure 5.6. Ceiling view of the standard 846 markers set-up.

The new tested position is in Figure 5.7:

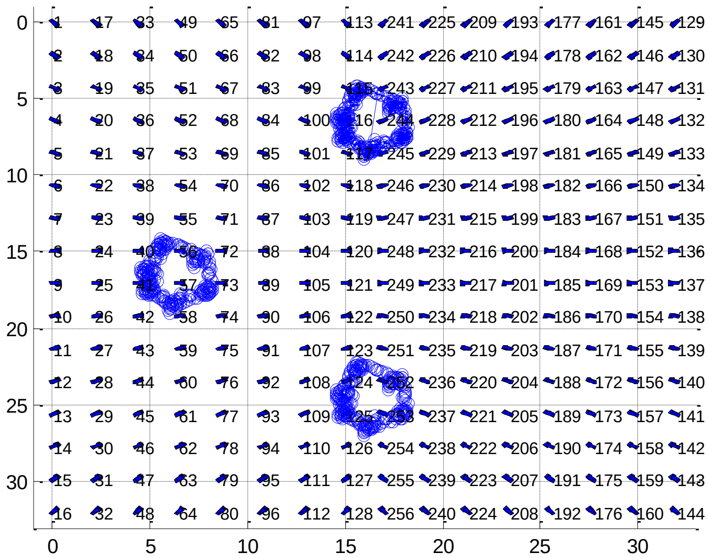


Figure 5.7. Ceiling view of a different 846 markers set-up.

In this case the results are in Table 5.11:

Level	MAX time:	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.008	18592	0	0	6955
2	0.004	6495	12097	3451	2126
3	0.004	2805	3690	3283	809
4	0.008	2138	667	2970	631
5	0.007	1736	402	2638	545
6	0.008	736	1000	2148	226
7	0.008	192	544	1411	33
8	0.016	66	126	847	0
	Total time = 0.063				

Table 5.11. Results with a different position of the markers.

The time required is less than before (0.075 s) and analysing the data, it is possible to notice that with the new set-up the number of 2D data is bigger in the first two levels but then it decreases faster. The reason is that now, the markers are seen by more cameras and the reconstruction is faster. It also confirms that the set-up used for the previous analysis is not the best-case, for the time of the reconstruction.

5.7 Analysis diminishing the amount of data

To evaluate the performance of the algorithm changing the number of markers, different set-ups are done.

For all tests, the number of cameras is kept fix to 256.

The number of points in the first test is half of the original.

The results are:

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.002	7899	0	0	2263
2	0.002	3561	4338	1286	1111
3	0.002	1899	1662	1334	568
4	0.004	1700	199	1296	520
5	0.004	1022	678	1249	302
6	0.004	317	705	1127	73
7	0.003	203	114	610	33
8	0.012	81	122	424	0
	Total time = 0.033				

Table 5.12. 424 markers and 256 cameras.

Then the number of markers is reduced by 1/3:

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.001	5273	0	0	1369
2	0.002	2648	2625	774	792
3	0.001	1408	1240	887	380
4	0.002	1279	129	862	363
5	0.003	838	441	836	211
6	0.003	344	494	773	64
7	0.003	218	126	445	45
8	0.008	81	137	282	1
	Total time = 0.023				

Table 5.13. 282 markers and 256 cameras.

Then by 1/4:

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.001	3978	0	0	1019
2	0.001	2040	1938	573	616
3	0.001	1078	962	670	290
4	0.002	981	97	653	283
5	0.002	637	344	634	171
6	0.002	254	383	588	49
7	0.002	170	84	336	37
8	0.007	61	109	214	1
	Total time = 0.018				

Table 5.14. 212 markers and 256 cameras.

And finally by 1/6:

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.001	2633	0	0	670
2	0.001	1341	1292	382	397
3	0.001	715	626	443	193
4	0.001	647	68	431	185
5	0.001	427	220	414	111
6	0.001	186	241	383	39
7	0.002	114	72	223	26
8	0.006	43	71	141	0
	Total time = 0.014				

Table 5.15. 141 markers and 256 cameras.

Another test is done increasing the number of markers to 1692. These were distributed in six groups of six people like in Figure 5.8:

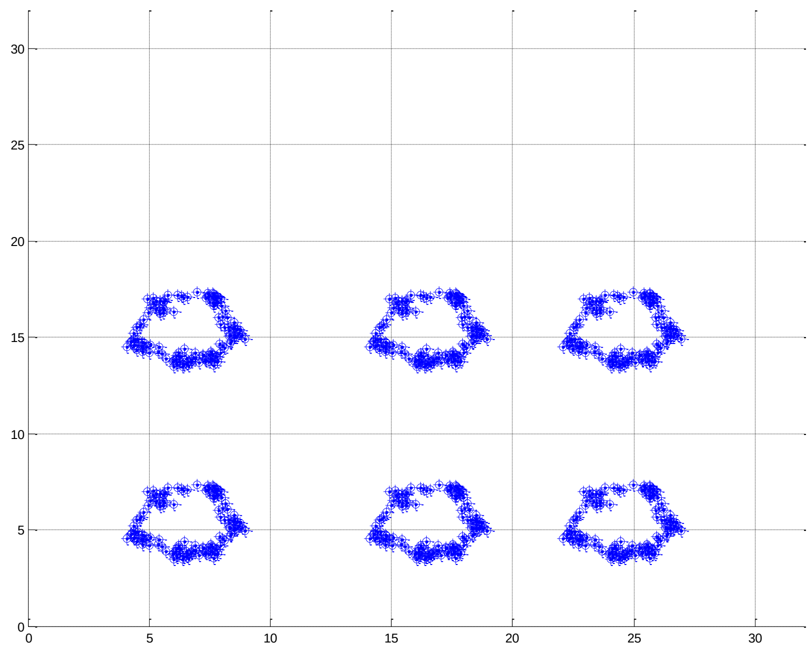
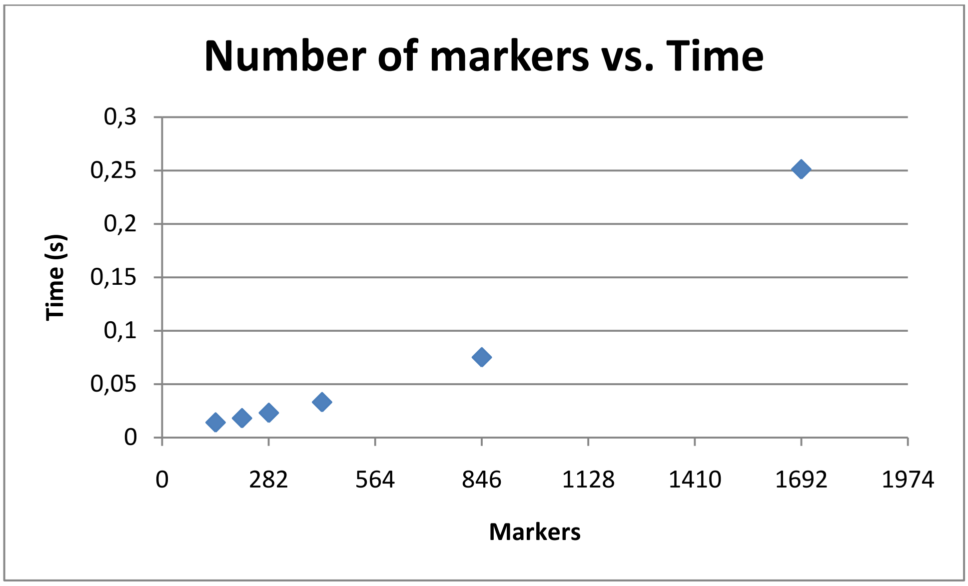


Figure 5.8. Ceiling view of the 1692 markers set-up.

The data obtained are:

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.008	33160	0	0	9996
2	0.008	13956	19204	5556	4461
3	0.021	7205	6751	5556	2315
4	0.033	5706	1499	5379	1772
5	0.024	3846	1860	4750	1149
6	0.034	2165	1681	4102	616
7	0.035	865	1300	2735	118
8	0.088	492	373	1704	16
	Total time = 0.251				

Table 5.16. 1692 markers and 256 cameras.



Graph 5.15. Comparison of the time taken by the algorithm, changing only the number of markers.

As it is possible to see in Graph 5.15 the time required by the algorithm is strictly correlated with the number of markers.

5.8 Adding error to the data

Other tests are done to verify the robustness of the algorithm adding an error to the input data. Before saving the synthetic data, a Gaussian error is added in Matlab to the 2D position of each centroid seen by the cameras. The set-up has 256 fixed cameras and 846 markers. Each camera can see 6 meters far with a focal length of 5 mm; the pixel size is 0.005mm.

The first error added has a variance equal to 0.000625 mm in the image plane. This value corresponds to 1/8 of a pixel or, to a markers moved from its real position of 0.5 mm if it is 4 meters far from the camera.

The *"Missed"* are the real markers that the algorithm cannot reconstruct, while the *"Ghosts"* are markers reconstructed that are not present in the real data set.

The result is:

```
const double distance3D = 0.02;
const double distance2d3d = 0.015;
const double SamponImageError = 0.00000025;
```

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.005	7588	0	0	2485
2	0.004	4739	2849	838	1788
3	0.007	2997	1742	1110	1042
4	0.01	2695	302	1126	963
5	0.01	2129	566	1197	751
6	0.017	1399	730	1368	584
7	0.012	459	940	1148	186
8	0.021	36	423	846	1
	Total time = 0.086				

Table 5.17. Gaussian error = 1/8 of a pixel.

Total Markers= 846, Missed = 0, Ghost = 0.

With this error, all 3D points are found.

The second test is done doubling the error to 0.00125 which consists to a Gaussian error of 1 mm for a marker 4 meters far from the camera:

```
const double distance3D = 0.04;
const double distance2d3d = 0.01;
const double SamponImageError = 0.000007;
```

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.006	7588	0	0	2697
2	0.004	4789	2799	826	1937
3	0.006	3048	1741	1108	1125
4	0.011	2746	302	1126	1043
5	0.01	2194	552	1196	827
6	0.017	1516	678	1355	700
7	0.014	537	979	1162	218
8	0.024	94	443	846	3
	Total time = 0.092				

Table 5.18. Gaussian error = 1/4 of a pixel.

Total Markers= 846, Missed = 0, Ghost = 0.

Also with this error, all markers are found.

Then the error is doubled again to 0.0025, which corresponds to a Gaussian error on each axe of 2 mm for a marker 4 meters far from the camera:

```
const double distance3D = 0.04;
const double distance2d3d = 0.03;
const double SamponImageError = 0.00002;
```

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.006	7588	0	0	3099
2	0.005	4909	2679	796	2339
3	0.008	3199	1710	1087	1406
4	0.013	2906	293	1105	1321
5	0.013	2358	548	1179	1064
6	0.019	1697	661	1345	926
7	0.018	656	1041	1184	268
8	0.028	206	450	844	24
	Total time = 0.11				

Table 5.19. Gaussian error = 1/2 of a pixel.

Total = 844, Missed = 3, Ghost = 1.

With this error, three real markers are not reconstructed and 1 extra marker is reconstructed by the algorithm.

Next test is done doubling again the error to 0.005, which corresponds to a Gaussian error on each axe of 1 pixel or 4 mm for a marker 4 meters far from the camera:

```
const double distance3D = 0.04;
const double distance2d3d = 0.03;
const double SamponImageError = 0.00002;
```

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.006	7588	0	0	2592
2	0.006	5390	2198	703	2220
3	0.007	3668	1722	1085	1320
4	0.015	3389	279	1108	1266
5	0.019	2870	519	1191	1067
6	0.023	2264	606	1347	977
7	0.031	1163	1101	1265	440
8	0.055	606	557	859	159
	Total time = 0.162				

Table 5.20. Gaussian error = 1 pixel.

Total = 859, Missed = 20, Ghost = 33.

An added error of 4mm is quite big considering that the standard marker has a diameter of 1 cm; however only 2.3% of the real markers are not reconstructed.

Last test is done to stress the algorithm: the Gaussian error added is 0.01, which corresponds to a Gaussian error on each axe of 8 mm for a marker 4 meters far from the camera.

```
const double SamponImageError: 0.0002
const double distance2d3d: 0.03
const double distance3D: 0.07
```

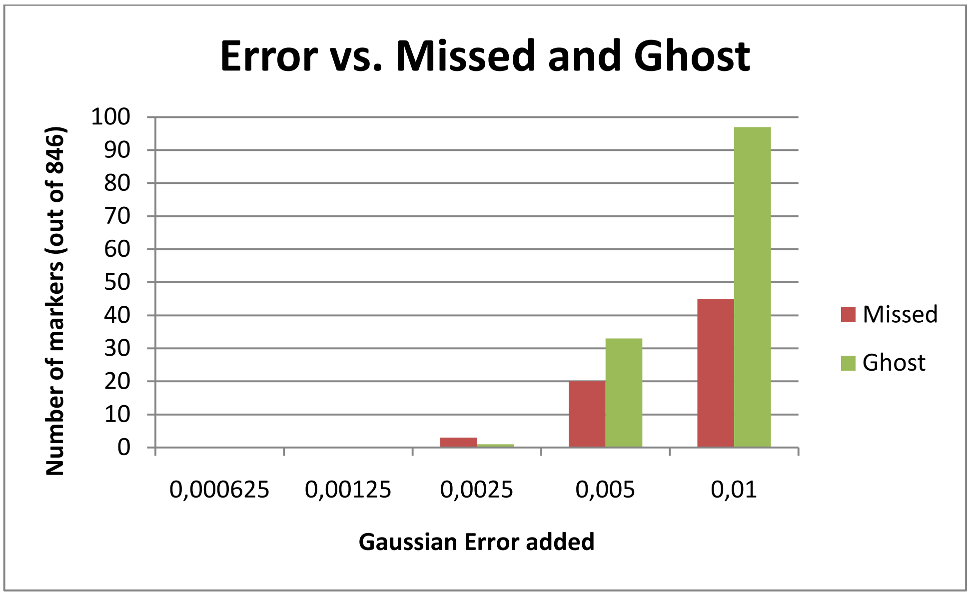
Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.008	7588	0	0	4858
2	0.014	5696	1892	595	4572
3	0.01	3963	1733	1064	3247
4	0.019	3439	524	1146	2730
5	0.026	2875	564	1259	2362
6	0.047	2419	456	1374	2683
7	0.075	1065	1354	1372	851
8	0.066	545	520	898	287
	Total time = 0.265				

Table 5.21. Gaussian error = 2 pixels.

Total=898, Missed = 45, Ghost = 97.

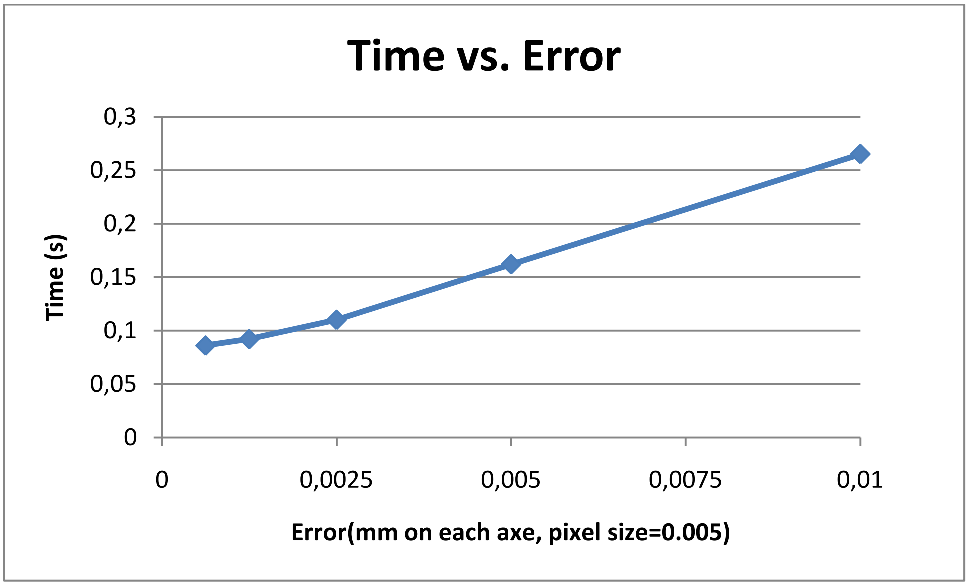
In this situation, the accuracy of the algorithm is low, 5.3% of the original markers are not reconstructed and 11% of the total markers reconstructed are ghosts.

The different performances are in Graph 5.16 and Graph 5.17:



Graph 5.16. Number of missed and ghost point adding a Gaussian error to the data.

The times required by the algorithm vary, the larger the error the longer the time:



Graph 5.17. Time required by the algorithm adding a Gaussian error to the data.

5.9 Comparison with the centralized code

The other main step is to compare between the distributed and the centralised algorithm using real data. The big limitation of these tests is that the tree is not created using the association function but is obtained by random association, so the performance of the distributed algorithm is lower. The second factor is that the distributed algorithm is not as optimised as the centralised version on which Vicon has worked for years.

The results have been obtained using a big file captured for a movie. In this file people has lots of markers on the face and on the hands to capture in details also the information about facials expressions and hands movements.

The number of cameras used is 220 and the number of markers is about 2350. The exact number of markers is not known *a priori* but for evaluating the differences between the two versions this is not very important. With a big amount of cameras, the threshold for the BIG 3D points must be increased to 5 o 6 otherwise it is very likely the presence of ghost markers with 3 or more rays forming it.

The set-up of the cameras is shown in Figure 5.9 (Captured with Vicon Nexus®):

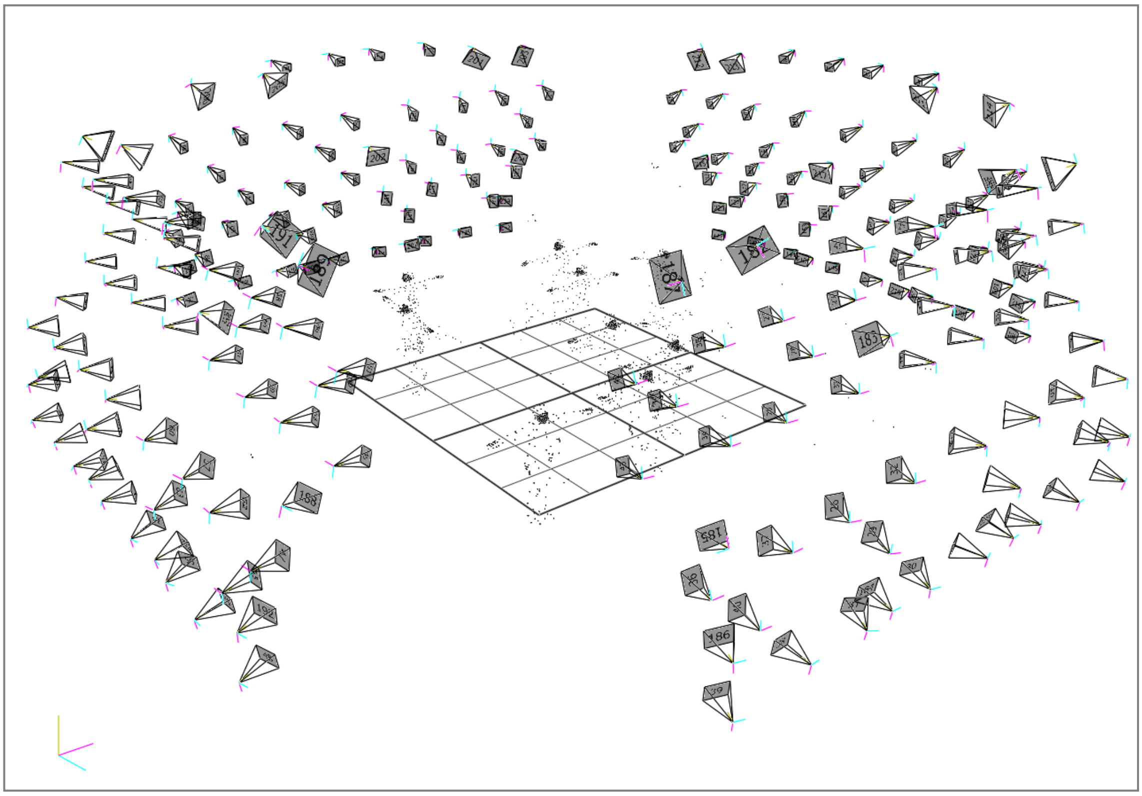


Figure 5.9. Cameras set-up of the real data.

In Figure 5.10 is possible to see more in detail the 9 people with the open arms plus some objects in the scene:

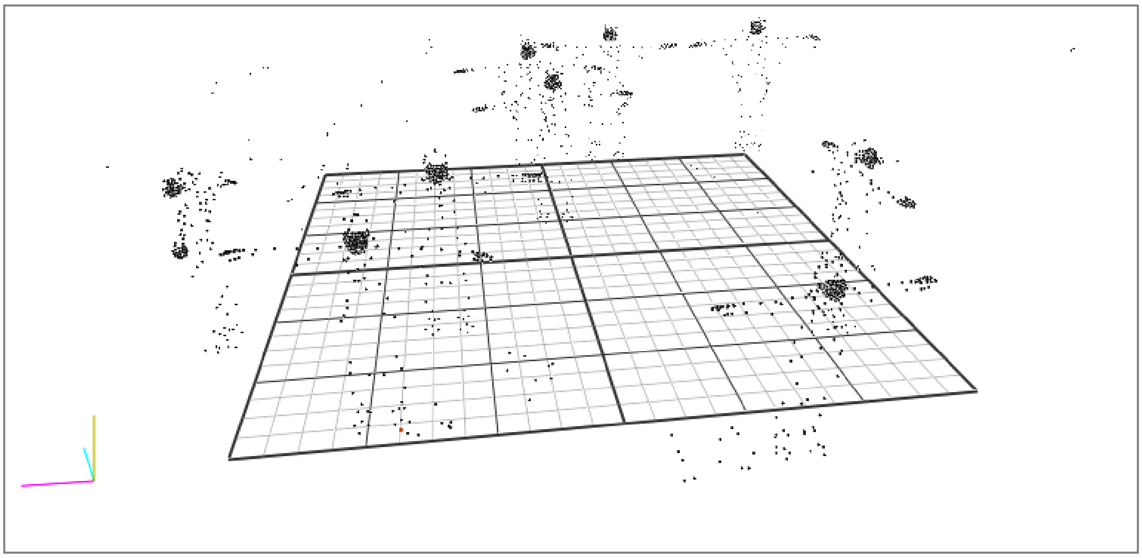


Figure 5.10. Markers set-up of the real data.

The tests are done with a different number of cameras starting from 8 to 128. The results for the distributed version with 8 cameras are:

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.013	2750	0	0	322
2	0.035	2589	502	52	901
3	0.08	1867	2157	260	1039
	Total time = 0.128				

Table 5.22. Distributed algorithm with 8 cameras (real data).

The summarize data for the other set-up is:

Max Time in each level (s):			
128			
0.022	**64**		
0.054	0.022	**32**	
0.152	0.054	0.023	**16**
0.302	0.153	0.043	0.013
0.509	0.346	0.097	0.043
0.814	0.494	0.238	0.097
1.379	0.814	0.474	0.229
Total time = 3.232	Total time = 1.883	Total time = 0.875	Total time = 0.382

Table 5.23 Distributed algorithm with all the cameras (real data)

The results of the centralised and distributed version are:

128 Cameras:

The time taken by the **centralised** algorithm is 3.451
The total number of markers found is 4950
The time taken by the **distributed** algorithm is 3.232
The total number of markers found is 3635

64 Cameras:

The time taken by the **centralised** algorithm is 1.758
The total number of markers found is 3522
The time taken by the **distributed** algorithm is 1.883
The total number of markers found is 2467

32 Cameras:

The time taken by the **centralised** algorithm is 0.811
The total number of markers found is 2225
The time taken by the **distributed** algorithm is 0.875
The total number of markers found is 1371

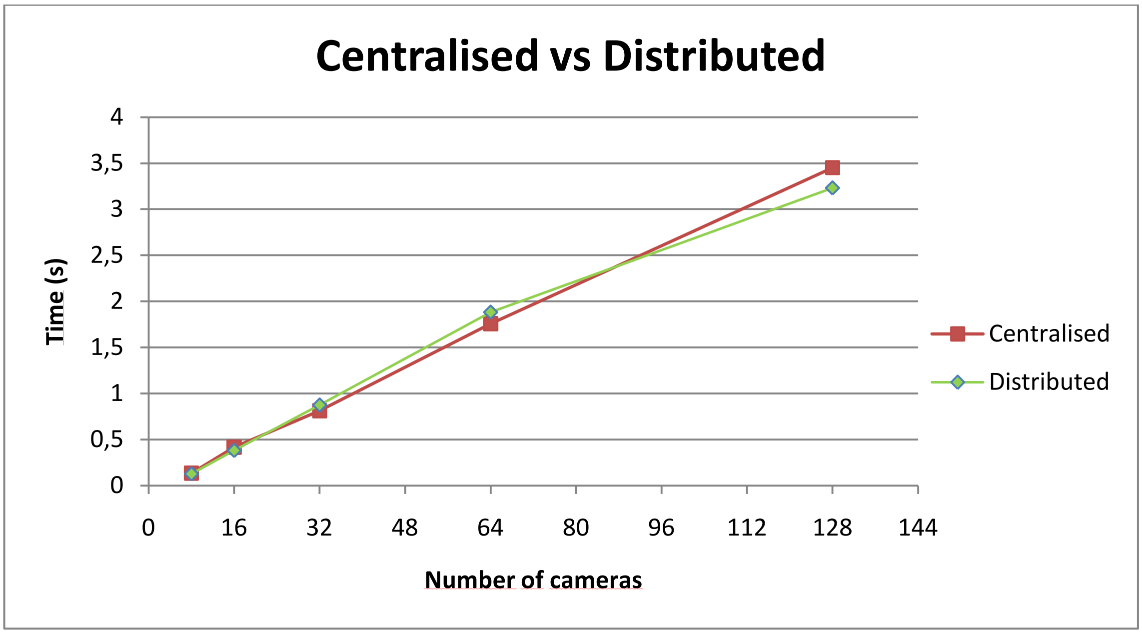
16 Cameras:

The time taken by the **centralised** algorithm is 0.416
The total number of markers found is 1437
The time taken by the **distributed** algorithm is 0.382
The total number of markers found is 700

8 Cameras:

The time taken by the **centralised** algorithm is 0.135
The total number of markers found is 707
The time taken by the **distributed** algorithm is 0.128
The total number of markers found is 206

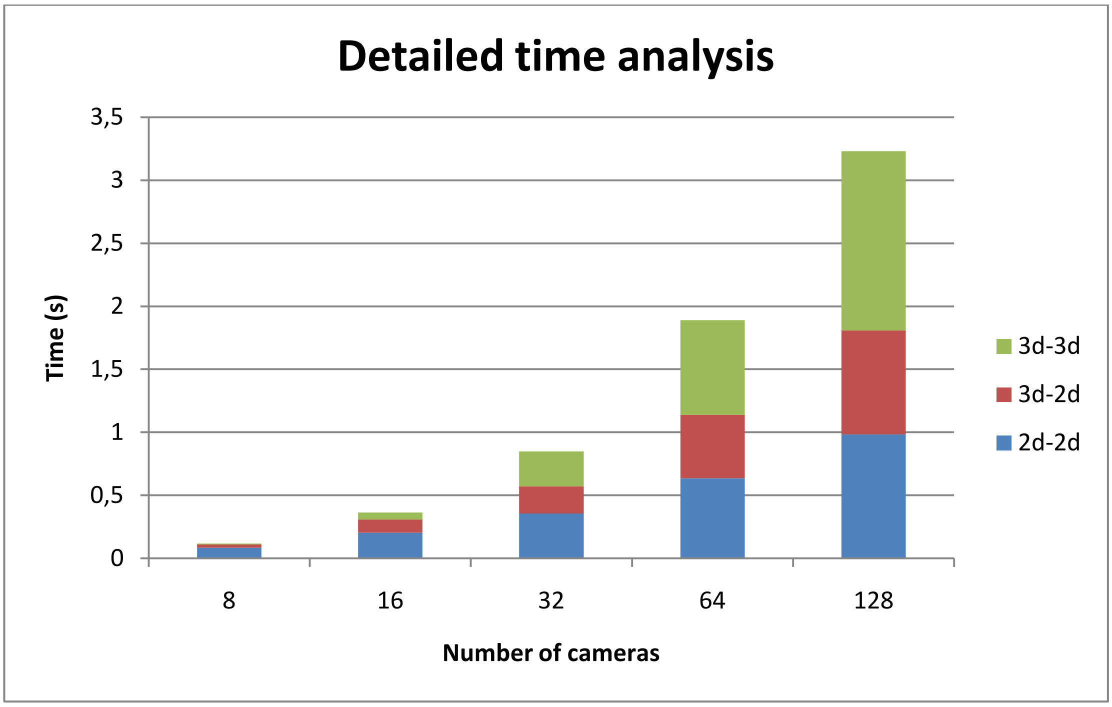
The performance is in Graph 5.18:



Graph 5.18. Comparison between the centralized and the distributed version using real data.

The graph shows that the two curves are very similar. With a big number of cameras, in particular 128, the centralised version of the algorithm is slower than the distributed version of the 6.78%. Analysing the trend of the graph, it seems that when the number of cameras increases, the difference between the two versions becomes larger.

An analysis of the time spent in each part of the distributed algorithm shows that the time is almost equally distributed among the three different functions.



Graph 5.19. Time required by each reconstruction function of the distributed algorithm.

A synthetic set-up is created to do an analysis of the algorithm behaviour with a larger number of cameras. It is not a trivial process. First, the 3D position of all markers is calculated and saved using the software developed by the company Vicon Nexus®. Second, to the stage are added 36 cameras in random positions so the total number grows to 256. Third, using a *cpp* code all markers are projected to the image plane of all the cameras and then stored to a file.

The new input file is similar to the previous one with the only difference that there are no occlusions so the total number of markers that the cameras could see is larger. Even in this example the tree is randomly built.

The data obtained with 8 cameras are:

Level	MAX time (s):	2D Data:	2D Data deleted:	3D Data (big):	3D Data (small):
1	0.141	8650	0	0	2194
2	0.134	4563	4087	1334	1861
3	0.044	81	4482	1993	37
	Total time = 0.319				

Table 5.24. Distributed algorithm with 8 cameras (real data, no occlusions).

The other results are:

Max Time in each level (s):				
256				
0.186	**128**			
0.208	0.186	**64**		
0.115	0.209	0.185	**32**	
0.078	0.114	0.209	0.185	**16**
0.048	0.048	0.06	0.14	0.14
0.083	0.038	0.041	0.059	0.139
0.054	0.035	0.031	0.041	0.043
0.053	0.036	0.033	0.03	0.033
Total time = 0.825	Total time = 0.666	Total time = 0.559	Total time = 0.455	Total time = 0.355

Table 5.25. Distributed algorithm with all cameras (real data, no occlusions).

The results of the centralised and distributed version are:

256 Cameras:

The time taken by the **centralised** algorithm is 1.373
The total number of markers found is 2264
The time taken by the **distributed** algorithm is 0.825
The total number of markers found is 2491

128 Cameras:

The time taken by the **centralised** algorithm is 0.882
The total number of markers found is 2264
The time taken by the **distributed** algorithm is 0.666
The total number of markers found is 2264

64 Cameras:

The time taken by the **centralised** algorithm is 0.561
The total number of markers found is 2261
The time taken by the **distributed** algorithm is 0.559
The total number of markers found is 2260

32 Cameras:

The time taken by the **centralised** algorithm is 0.393
The total number of markers found is 2256
The time taken by the **distributed** algorithm is 0.455
The total number of markers found is 2237

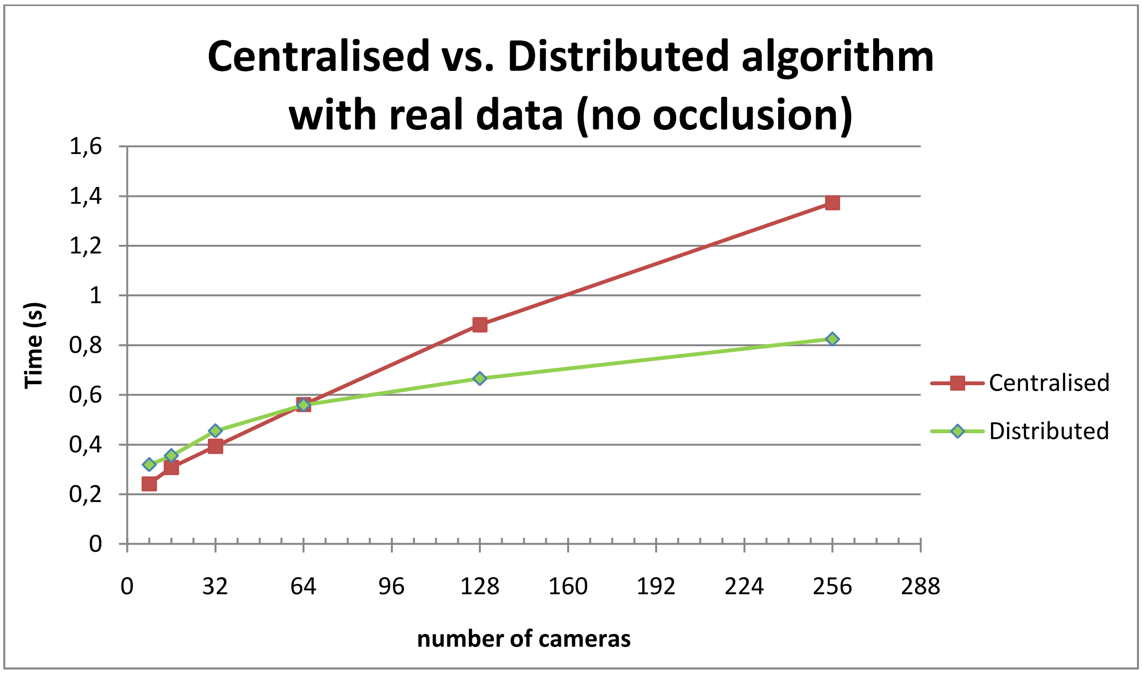
16 Cameras:

The time taken by the **centralised** algorithm is 0.308
The total number of markers found is 2232
The time taken by the **distributed** algorithm is 0.355
The total number of markers found is 2220

8 Cameras:

The time taken by the **centralised** algorithm is 0.242
The total number of markers found is 2030
The time taken by the **distributed** algorithm is 0.355
The total number of markers found is 1993

In Graph 5.20 there are the new results obtained.



Graph 5.20. Comparison between the centralized and the distributed version using real data (no occlusion).

With 128 cameras the centralised version is 24.5% slower and with 256 cameras it is 40% slower than the distributed version.

Using synthetics data, both versions are faster. This fact may seem strange because without occlusions, cameras see a bigger amount of 2D points. A possible explanation of this difference is that now, in the distributed version more 2D points are deleted in the first levels of the three and in the centralised version, for the same reason, a 3D points will mark its rays sooner and the algorithm will not try to use them for other reconstructions.

The trend of the Graph 5.20 suggests that the gap between the two versions could further enhance increasing as the number of cameras.

6 Conclusions and future steps

The main aim of this research work was to find a way to do motion capture faster, optimizing its performance. The aim was reached using a distributed approach instead of the centralized one used before.

- The first argument discussed was to verify if it is possible to find a good way to associate groups of cameras in the nodes of the binary tree used to distribute the elaboration and to design a distributed algorithm. The experiments done demonstrate that using functional costs that evaluate the volume and the angle between the cameras, the time required by the algorithm to do the reconstruction is the 22% lower than a random function.

  Using the data generated in Matlab, are obtained other results. It is important to highlight that the synthetic set-up used, with lot of markers near the edge of the room, is similar to a worst-case scenario, because lot of points are seen only by few cameras. The position of the points on three groups of six people is not the best situation because there are volumes with a high density of markers and volumes with no markers so the reconstruction is not equally distributed between all the nodes. In a distributed system created using a binary tree, the bottleneck is the slowest node in each level so if a node has more works, it will slow the entire algorithm.

  The results show that the algorithm is also robust to error in the camera data. With a reasonable Gaussian error introduced on both axes of one pixel, the number of markers not found is the 2.36%. This result is very good considering that the accuracy of the cameras is under one pixel.

  It is very hard to study the computational complexity of the algorithm because as soon as a 3D point becomes BIG, all 2D points associated with it are not used again by the algorithm, so the total number of points to analyze varies during the execution. The speed of the 3D point's creation is very dependent on the position of them and the cameras. To try to find a trend of the algorithm speed, have been performed some tests.

Keeping the number of markers fixed and increasing the numbers of cameras, the time required does not increase. Surprisingly the fastest reconstruction is with 512 cameras. The reason is that using 512 cameras, in that particular configuration, the number of 2D points to reconstruct decrease quickly respect to the same set-up with 128, 256 or 1024 cameras.

Keeping the number of cameras fixed and increasing the number of markers, the reconstruction time required increases. The curve *time* versus the *number of markers* looks like quadratic but this is normal considering that in the 3D+3D phase the comparison between the two lists of 3D points is quadratic.

- The second main aim of this work was to compare the distributed algorithm with the centralized one. To obtain this data, the distributed algorithm has been modified to work in the same condition as the centralized and using the same data.

  The tests show that with real data and 128 cameras, the distributed version is the 6.34% faster. Increasing the number of cameras to 256 and using the same data (but without occlusions) the distributed version is the 40% faster. The real data consist of a very difficult scenario with almost 2300 marker of different sizes very close each other and 2 types of cameras.

  With a smaller amount of cameras, the performances of the two versions are very similar.

- The results of the distributed algorithm are very impressive considering that it is not optimized. In fact the time required by the 3D+3D function (that is not present in the centralized version) is quadratic respect to the number of 3D points and in the 3D+2D function, while the distributed algorithm have to scan all the centroids in the image plane, the centralized algorithm, using a more complex data structure, does this search almost immediately.

The time required by the distributed algorithm, in any case, does not consider the communication times. The amount of data to transfer is not very high considering the fast network connections existing nowadays.

The biggest problem may be in the times required for initializing the connection. This problem can be partially solved using multicore CPU or motherboard with two or more multicore CPU installed, so that many nodes can be executed on the same machine.

A very interesting prospective is to make the distributed algorithm, to execute on a graphic card that has lot of different cores.

Future steps

Possible optimization or analysis of the solution can be:

- In the volume cost function, give a bigger weight to the voxels closest to the camera, because the accuracy of their capture depends on how far are they from the camera.
- In the 3D+3D function, use a more efficient way to merge the two 3D inputs using a more complex data structure.
- In the 3D+2D function, try to increase the accuracy of the reconstruction of the 3D point when matching them with a 2D point.
- Using real data, create the association tree using the functionals cost.
- Test the accuracy of the reconstruction using partial data in the second-last level of the tree.
- Try the distributed algorithm on a real distributed system.

7 Bibliography

[1] Menache, A. (1999) *Understanding motion capture for computer animation and video games*, Morgan Kaufmann.

[2] Davison A. J., Deutscher J. and Reid I.D. (2001)*Markerless Motion Capture of Complex Full-Body Movement for Character Animation*, Robotics Research Group, Department of Engineering Science, University of Oxford.

[3] http://www.vicon.com/

[4] http://www.c3d.org/

[5] Report of group 1 and group 2 on http://automatica.dei.unipd.it/people/schenato/teaching/PSC/PSC_09.html

[6] Hartley, R. and Zisserman A. (2004) *Multiple View Geometry in Computer Vision*, Cambridge University Press.

[7] Harker M. and O'Leary P. *First Order Geometric Distance (The Myth of Sampsonus)*, Institute for Automation, University of Leoben, Austria.