Kinect-Based Algorithms for Motion Analysis
João Cabral and Pedro Lima

Abstract—Motion detection is an essential block of multiple systems. There are usually some constraints related to the motion of the camera, or the type of scene. This work proposes a framework to detect and classify multiple moving objects in an 3D space using the Microsoft Kinect camera without any motion constraint.

To solve the detection problem, a 3D alignment step is carried out using the Iterative Closest Point (ICP) algorithm followed by multiple 2D frame-difference techniques, ending up with the construction of 3D clusters containing the moving objects. Two types of entities were considered on this work - people and balls.

To classify people, a classification algorithm relying on Histograms of Oriented Gradients (HOG) is built. A Random Sample Consensus (RANSAC) model is used to find balls.

The proposed solution was tested in realistic indoor scenes.

I. INTRODUCTION

The need for systems able to detect, identify and take action based in the evaluation of the situation is emerging. The aging of the world population and the many illnesses we are facing create the need for continuous care, and considerable research is currently actively seeking solutions on this area. Many companies are now offering nursing-care robots able to help and take actions autonomously in a hospital or even at home. In 2006 the RoboCup [1] started a new league called RoboCup@Home [2] that motivates researchers from all over the world to develop new algorithms which can improve domestic robots performance.

One of the key aspects of the relationship between persons and robots is the ability of the robot to recognize a person and be ready to receive a command from her. Other important task is the ability to detect moving objects and, based on their classification to avoid or follow them. This is usually achieved with the robot in a static position.

The presented solution gives a framework to detect spacial differences and classify the entity that suffered motion. RGB and Depth information are obtained from a Microsoft (MS) Kinect. Although the depth information could be used directly from the sensor, filtering is required to make the depth data less noisy.

The two images are merged together to obtain the scene point cloud. Each scene point cloud is processed through the registration algorithm to get aligned with the previous scene point cloud (thus removing the motion of the camera - ego-motion). Once the two point clouds are aligned the differences between them are computed and fed into the classification module, a component which classifies the object that created the motion.

A. Related work

1) Motion detection: Motion detection has been a research topic for many years and it is still a challenging problem in computer vision.

The usual strategy for addressing this problem is to subdivide it into two smaller ones:
• Calculation of the camera motion.
• Detection of object motion.

Monocular and stereo cameras are used in the literature and pose different challenges. When using monocular cameras some researchers (e.g. [3]) solved the motion detection problem by extracting feature points (in this case, by the use of Good Features to Track) from the input images, applying Kanade-Lucas-Tomasi (KLT) [4] feature tracker. They matched the points and got the homography between the images by means of a RANSAC scheme [5]. Using this they can finally compare all pixels between two frames and calculate the differences. After some morphological process the place in the image frame where the movement happened is detected.

Mito et al [6] while using a stereo camera used the same method, but this time in 3D with RANSAC to filter outliers and, added a further step, an optimization of the model given by RANSAC.

Recently Microsoft presented the KinectFusion work [7] and [8]. Although not exclusively related to the detection of motion, it covers the tracking of the camera and the reconstruction of a scene. This work presuppose a static environment. The tracking of the camera is done using the iterative closest point (ICP) algorithm that tries to align the actual point cloud with a model of the scene that is being construct every time they successfully track the camera. Its tracking part will be used, upon some modifications, in this work.

2) Person classification: Detecting people is a research topic with increased interest in the last years. The capability of detecting people is an important feature for any robot that
interacts with humans.
One used technique is the use of Histograms of Oriented Gradients (HOG), created by Dalal et al.[9]. Their algorithm obtained nearly perfect results on the MIT pedestrian database[10].

II. METHODOLOGY
A. Filtering depth image
The first step in the system is to filter the depth image. Without this filter small variations coming from noise on the computed depth, would translate to the point cloud, turning planes into wobbly surfaces and affecting the point cloud alignment step.

The bilateral filter was chosen due to its capability to keep edges while filtering the image. The idea of this kind of filter is to filter the image on the edges while filtering the image.

A bilateral filter applied on an image $f(x)$ produces an output image defined as

$$h(x) = k^{-1}(x) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi)c(\xi, x)s(f(\xi), f(x)) \, d\xi$$

where $c(\xi, x)$ is a measure of the geometric closeness between the neighbourhood center $x = [v_c, \nu_c]^T$ and a nearby 2D point $\xi = [v_p, \nu_p]^T$, $s(f(\xi), f(x))$ measures the photometric similarity between the pixel $x$ and the neighbourhood pixel $\xi$ intensity. Finally $k^{-1}(x)$ is given by

$$k(x) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, x)s(f(\xi), f(x)) \, d\xi$$

Gaussian functions can be used as the functions $c$ and $s$

$$c(\xi, x) = \exp\left(-\frac{d(\xi, x)^2}{2\sigma_d^2}\right)$$

$$d(\xi, x) = d(\xi - x) = ||\xi - x||$$

$$s(\xi, x) = \exp\left(-\frac{\delta(\phi, f)^2}{2\sigma_r^2}\right)$$

$$\delta(\phi, f) = \delta(\phi - f) = ||\phi - f||$$

where $\sigma_d$ is the geometric spread in the domain and is based on the amount of filtering desired and $\sigma_r$ is the photometric spread in the image range. Large $\sigma_d$ combines pixels from farther image locations and so it blurs the image more. While large values of $\sigma_r$ means that the filter output is making less distinction between pixel intensities.

Fig.2 shows the difference between a point cloud image without using the bilateral filter in the depth image, and the same image with a bilateral filter. The difference is notable, making the noise to disappear.

B. Point cloud creation
After filtering the depth information, it is necessary to calculate 3D information, that is, to convert a pixel (a 2D point) into a voxel (a 3D point). The conversion is done with the depth image only, because this is where the 3D information is, the RGB image is just used to add color information to the points.

The perspective transformation (Equation 4) projects the image captured from the world into the camera image plane.

$$s \begin{bmatrix} u' \\ v' \\ w' \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P = \begin{bmatrix} f_x & 0 & c_x & T_x \\ 0 & f_y & c_y & T_y \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} u' \\ v' \\ w' \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}$$

where $X,Y,Z$ are the 3D coordinates of a point in the world, $u,v$ are the image plane 2D coordinates, $w$ is a multiplication scalar, $f_x, f_y$ are the focal point, $c_x, c_y$ are the central point in the image plane and $T_x, T_y$ are the translation between individual cameras inside a stereo camera, but as the Kinect drivers already align the two cameras the translation between the cameras is 0.

Inverting Equation 4 we obtain:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} u - c_x \\ v - c_y \\ w \end{bmatrix}$$

Equation 5 defines a line and not a single point. But using the information from the depth image the $Z$ is known and it is possible to arrive to a single 3D point from a pixel in the depth image.

C. Motion analysis
1) Set up: After receiving the point clouds, the first step is to calculate the point normals and make a point cloud pyramid (Fig.3) with three levels, each level having half the resolution of the previous. This is done to accelerate the next step.
2) **Point cloud registration:** To detect ego-motion we will make use of the assumption that the most present part of the scene is the background. Finding the translation and rotation that the majority of the scene suffered gives us the ego-motion and allows to eliminate the motion of the camera. As such, we arrive to the motion suffered by objects on the scene. This block starts by defining the key point cloud. This is the point cloud that will be used in the alignment step, and is defined at the start of a new cycle. The motion of the camera is tracked on every frame by finding the transformation between the actual point cloud and the key point cloud. At every three frames, this transformation is used to align the two point clouds (removing ego-motion) and allowing the detection of spacial differences (non-ego-motion). This point cloud is then defined as the new key point cloud.

This scene registration between a previous scene and an actual scene was made by using Iterative Closest Point (ICP) algorithm[12][13]. The ICP algorithm can be stated as:

- The source point set \( P \) with \( N_p \) points \( p_i \) and the target point set \( X \) with points \( x_i \) are given.
- The iteration is initialized by applying a initial transformation matrix to the source point set (if given).
- Steps 1, 2, 3, and 4 are applied until convergence or until a number of iterations is reach.
  1) Compute the closest points between source point set \( P \) and target point set \( X \).
  2) Compute the registration \( T = [R|T] \).
  3) Apply the registration.
  4) Compute the error in the registration.

The ICP tries to minimize a error function. In this work a point-to-plane error function was chosen

\[
f(R, T) = \sum_{i=1}^{N} ||n_i \cdot ([R|T] \cdot p_i - x_i)||^2
\]

where \( n_i \) is the surface normal of the point \( p_i \). The minimization of Equation 6 is a nonlinear minimization problem. It can be solved using the Levenberg Marquardt Algorithm[14], or it can be linearised for small rotations and translations, and solved by any linear minimization algorithm.

The advantages of using a point-to-plane method is that it converges in less iterations and it is less susceptible to the local minima problem than the point-to-point version. In Fig.4 it is possible to see the alignment step in a real scene point cloud.

3) **Finding spacial differences:** The differences between the two point clouds will be calculated from the RGB and depth images.

First step is to decompose the 3D point cloud back to a RGB and depth image. This is done by creating two blank images, one for color information and the other for depth information. Using Equation 4 we obtained the pixel coordinates and with the color and depth information, from the 3D point cloud, we can build the RGB and depth images. In this step a mask for the valid points is also created. Fig.5 shows the RGB and depth images\( \) obtained from the point cloud decomposition.

The two masks arriving from the decomposition of the two point clouds\( \) of the previous and the actual,\( \) are added together by means of a bitwise OR operation, resulting in a so called “final mask”.

Finding differences on the RGB image is a matter of finding pixels with enough difference in color. The International Commission on Illumination (CIE)[15] calls this color difference \( \Delta E \). In 1994 CIE recommended a new formula to calculate the color difference called \( \Delta E_{94} \)[16][17]. Before finding the color differences the RGB images are passed by a small Gaussian filter, to remove any noise arriving from the camera. A binary matrix stores the detected differences that are above a threshold.

The depth image is treated differently. Here the absolute difference between the two depth images is computed. If the difference is greater than a defined threshold that pixel is signalized as different. The result of this process is stored on another binary matrix.

Both binary matrices, containing the differences found in the comparison of the RGB and the Depth images, are multiplied element-wise by the “final mask” to remove any points showing invalid differences. Differences are invalid if at least in one of the compared images, one of the points was already invalid. Invalid points are created in the point cloud decomposition process to represent space without 3D information.
After, a blob filter removes any blob with an area below 20 pixels. The resulting matrices are joined by another bitwise OR operation.

This last mask is applied on both previous and actual 2D images resulting from the point clouds (RGB and Depth), and the point clouds are reconstructed using that result. This process creates two point clouds having only points representing differences between previous and actual scenes. Finally these point clouds are merged into one point cloud that contains all the differences founded.

4) Filtering: First it is possible that some of the differences found are due to the fact that while the object is moving some areas of the scene from one point cloud to another are occluded and some are unobstructed/unblocked. Those differences appear on the background and are not desirable. To filter it a ray trace algorithm[18][19] is used, removing any point that is occluded by another.

There is some random noise in the detected differences that needs to be clean. Part of the noise is due to a small error on the alignment, other is due to the intrinsic error on the depth calculation from the Kinect.

To remove noise a chain of filters was created, each one using a different filtering technique.

The first on this chain is a simple radius filter. It works by searching, for each point, all its neighbors in a given radius. The point remains only if the number of neighbors within that radius, is superior to a given quantity. In this work the radius was set to 0.1 meters and the minimum number of neighbors to 10.

The second filter is a Statistical outlier removal. It runs in two iterations. In the first, it scans all points searching for the nearest k neighbors. Having the k neighbors of a point the average of the distances is calculated. After analyzing all points the mean and standard deviation of all the distances are computed to determine a distance threshold that is equal to the \( \text{mean} + \text{std.dev} \). On the second iteration it scans all points and the ones that remain are the ones that have their average distance below the threshold distance for that point cloud. The value chosen for k is 50 neighbors, based on experimental results.

The last step on the filter chain is the creation of clusters on the point cloud containing the differences between previous and actual point cloud scene. This acts as a double function, besides filtering the differences point cloud, it also clusters the motion detected allowing for multiple motions, more precise classification of the motion and, in the case of the ball detector, detection of multiple balls in the same scene.

Clustering is constructed through the use of Euclidean distance, it clusters all the points inside a given radius, saying they all belong to the same object. This allows filtering, as the number of points inside each cluster is counted and if they are bellow a certain threshold, the cluster is rejected and those points are removed from the point cloud. The value of the radius is 0.3 meters, as it is close to, but less then, the shoulder-to-shoulder distance in a person, and the threshold for the number of points is 1000 points.

At this point we have multiple point clusters that indicate the motion of an object, they represent only the coordinates in 3D space and most probably represent just the borders of the object.

In order to have a representation of the object all the clusters are analysed, searching for the biggest box (bounding box) that includes that cluster. All boxes are expanded by 0.1 meters to allow for small discrepancies on the detection. The boxes are again searched by occlusions, if a bounding box of one cluster occludes the majority of other cluster, then the later is removed.

The remaining bounding boxes are copied to the actual point cloud and the region of interest (ROI) is extracted from inside each box. This leaves us with multiple clusters (multiple point clouds) each one having all the color and position information that describes an object.

Fig.6 shows the various filters output and the progression from the raw differences to the final clusters.

D. Motion classification

After detecting the motion of the scene, it is now necessary to classify what has moved. In this work two types of entities of interest were considered - persons and balls.

To do that, two classifiers were built and trained. The classifiers receive the point cloud clusters and return information regarding the type of object and its coordinates.

1) Person detector: The classifier used in this work to detect persons is the Histogram of oriented gradients(HOG)[9][20].

The idea behind the HOG method is that the local object appearance and the object shape, can be defined by the distribution of local gradients of the image intensity, as such it evaluates the normalized local histograms of the image gradient orientation in a dense grid.
To create HOG features, the image is divided in cells, each one having a local 1D histogram of the gradient directions over the pixels of that cell.

After having the histograms of each cell, it is essential to normalize them. To do it, each cell is grouped into blocks of $\varsigma$ by $\varsigma$ cells, and the gradients are normalized inside each block. Blocks can overlay other blocks, in that case, the normalization is done as before, but the same cell contributes to different blocks. In this paper the size of the blocks were 2x2, so $\varsigma = 2$, and they overlap each other by 8 pixels.

Fig. 7. HOG organization. The blocks are the two big rectangles in blue and red, note how the overlap each other, the smaller rectangles in green are the cells within each block. Using the same parameters as the original paper([20][9]), each block would have 2x2 cells and each cell 8x8 pixels with a block overlapping of 8 pixels.

Having done the normalization in all the blocks that divide the image, all the histogram vectors are concatenated into a single large vector, the HOG vector.

This ends the HOG descriptor phase. To train the detector, the HOG vector, from positive and negative images (ie, images that contain the object to be detected, and images without the object) are fed into a (Support Vector Machine) SVM classifier, in this case a soft ($C = 0.01$) linear SVM.

The negative images are tested for false positives, and a second training phase is done using the negative images and the false positives.

The detection of the trained object in a image is done by a sliding window approach. The input image is scanned by this window, and the HOG descriptors are fed to the trained SVM to classify them as the entity or not. To address for different size possibilities, the image is scaled down in steps until the minimal size (the size used for the training) is reached, each scaled image is tested using the same sliding window technique.

To train the detector a set of images was taken from the motion analysis part of this work. This means that images were always from detected motion, some were from a person and others from errors imposed on the detection part to give what would be the non-moving objects, such as walls and tables. In total 500 images with people and 1354 without were used. The images were treated with an image software like GIMP[21], to create patches with 64x128 pixels. The positive patches were aligned in the same software in regards of the person body, as seen in Fig.8.

The positive and negative patches (and their vertical reflections) were fed to the SVM classifier, in this case SVMLight[22] was used, with a soft linear model with $C = 0.01$ like in [20][9].

2) Ball detector: The idea behind the ball classifier is to try to fit a sphere model on each of the point cloud clusters.

If the inliers of this model is above a given threshold then a ball is detect in the scene and its coordinates with the radius information are passed to the user.

Multiple objects such as a person head or even the part of an arm can be classified as balls if we allow a low number of inliers. Having a fixed high threshold for the minimal number of points that allow positive classification, means that this is only optimized for a fixed ball-camera distance, because as the distance increases the density of the points decreases. The need to use a threshold function is clear.

In order to find that function the relationship between the number of inliers and the distance to the ball was evaluated and a third order polynomial was found. The results are on Fig.9.

Fig. 8. Image patch alignment. a), b), c) are individual patches, in d) it's possible to see that a), b), c) are aligned to put the body parts roughly on the same position. Each patch is 64x128 pixels.

Fig. 9. Ball inliers vs distance.
The best third order polynomial function is
\[ y = -1191.9562x^3 + 9272.1513x^2 - 24522.9906x + 22791.1306 \]
to have a safe margin on the detection the function was lowered by 500 inliers, in this configuration the function would reach zero for a distance of 2.99 meters, which is not desirable because it can still find a ball at 3.30 meters, with that in perspective the function was truncated to a minimal of 100 inliners. The final equation is then
\[ y = \begin{cases} 
-1191.9562x^3 + 9272.1513x^2 \\
-24522.9906x + 22791.1306 \\
100 
\end{cases}, \text{ if } x < 2.9 \]
\[ \text{ or } x \geq 3.0 \]

III. RESULTS

The system input is a set of RGB and depth images recorded through the Kinect camera. The output is a set of images containing only the objects that suffered motion, and its classification.

The tests results are compared against a ground-truth define manually by analysing all the images.

A. Test 1 - Camera motion

The first test, verifies the ability of the system to track the motion of the camera, the accuracy of the alignment step and the efficiency of the filters.

The input dataset is a scan through a static room, containing 690 images corresponding to 23 seconds of filming. The camera was hand-held during all the footage.

The results of this test are shown in Table I

<table>
<thead>
<tr>
<th>Total images</th>
<th>Positives</th>
<th>False-Positives</th>
<th>False-Negatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>230</td>
<td>0</td>
<td>2(0.87%)</td>
<td>0</td>
</tr>
</tbody>
</table>

the system shows that it can track well the motion of the camera (ego-motion). It only showed two small regions that were considered as moving (Fig.10). Both detected objects were producing light, and the difference were only found in the RGB image, this indicates that could be a problem with the automatic gain control on the RGB camera.

B. Test 2 - Still camera and moving person

This second test, includes the classification of persons.

The input dataset corresponds to 370 images (125 analysed) in 12.5 seconds of recording. The camera was static, while a person was moving in its field of view.

The expected result is a good segmentation and classification of the person.

Table II shows a good performance of the system, finding motion on 89% of the time, and with only 11% of false negatives. In this test, we didn’t receive any static part appearing as moving (only a small part of a wall appearing from being to close). One justification for the false positives is that sometimes the motion was so small that disappear when passing through the chain of filters.

In this test the system behave like a classic static background-subtraction, hence the almost perfect results.
2) Person classification: The objective of this part of the test was to classify correctly moving persons. It is based in the images where motion were detected. The results are show in the next figure and table. The ground-truth was defined by manually analysing the images searching for persons.

![Image](a) ![Image](b) ![Image](c) ![Image](d)

Fig. 12. Test 3 - Person detection. a) and b) shows a correct classification of a person. c) false positive. d) false negative

<table>
<thead>
<tr>
<th>Total images with persons</th>
<th>positives</th>
<th>false negatives</th>
<th>false positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>56 (72.5%)</td>
<td>21 (26.25%)</td>
<td>1 (1.25%)</td>
</tr>
</tbody>
</table>

Table III shows the performance of the system. The relatively high value for the false negatives can be explained by the fact that in this dataset in multiple frames the person was partially occluded by other objects, and has already seen that can affect the performance of the classifier. Apart from that the system has a good performance on the person classification.

C. Test 3 - Moving camera with moving person

The input dataset corresponds to 450 images (150 analysed) in 15 seconds of recording. The camera was again hand-held, and suffered rotation and translation motions. A person is naturally walking through the scene.

1) Motion segmentation: The ideal result from this test is similar to the one on Test 2. Fig.13 shows some examples of segmentations.

![Image](a) ![Image](b) ![Image](c) ![Image](d)

Fig. 13. Test 3 - Motion Segmentation. a) shows a perfect segmentation. b) as the person is too close to the wall a part of it is included too. c) the image shows a valid segmentation and a invalid one (the wall on the right)

<table>
<thead>
<tr>
<th>Total images misclassified</th>
<th>static objects appearing</th>
<th>no detection of motion</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>8 (47%)</td>
<td>9 (53%)</td>
</tr>
</tbody>
</table>

Table V shows that only a small number of images were misclassified. Comparing with the results in Table IV it is possible to conclude on the good performance and good behaviour of the system.

2) Person classification: Fig.14 shows the person classification.

![Image](a) ![Image](b) ![Image](c) ![Image](d)

Fig. 14. Test 3 - Person detection. a) and b) shows a correct classification of a person. c) false positive. d) false negative

<table>
<thead>
<tr>
<th>Total images with persons</th>
<th>positive</th>
<th>false negative</th>
<th>false positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
<td>66 (72%)</td>
<td>19 (21%)</td>
<td>6 (6%)</td>
</tr>
</tbody>
</table>

Table VI shows the performance of the person classifier. As expected the system shows a good performance with a
low number of false negatives and positives and over 72% of correct classifications.

D. Test 4 - Camera motion and ball classification

This test verifies the ability of the system in segmenting and detecting a ball being passed between two persons (not appearing on the scene).

The input dataset consists of 480 images (160 analysed) corresponding to 16 seconds of footage. The camera was handheld suffering rotations and translations, while trying to keep the ball in its field of vision.

1) Ball segmentation: As for the person’s segmentation, an ideal result would be to get the ball only, without any patches of background or any piece of noise. This is certainly achieved in ideal conditions with controlled light and without reflecting materials.

The results of the test are shown in Fig.15 and Table VII.

![Fig. 15. Test 4 - Ball segmentation. a) shows a good segmentation while b) shows a incorrect one.](image)

<table>
<thead>
<tr>
<th>Total images with ball</th>
<th>positive</th>
<th>no segmentation realized</th>
<th>Bad segmented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total images with ball</td>
<td>98</td>
<td>68 (70%)</td>
<td>25 (25%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 (5%)</td>
<td></td>
</tr>
</tbody>
</table>

These results show the difficulty of the system, and the Kinect in dealing with light and reflecting surfaces. The small radius of the ball makes it also more difficult for the system to distinguish valid points and remove noise. The test results are in line with the range of values got in the person’s segmentation test.

2) Ball classification: The objective of this part of the test was to classify correctly moving balls. It is based in the images where motion were detected. The results are shown in Fig.16 and Table VIII.

![Fig. 16. Test 4 - Ball classification. a) and b) shows good classification. c) a good classification and a erroneous one.](image)

<table>
<thead>
<tr>
<th>Total classifications made</th>
<th>positive</th>
<th>false positive</th>
<th>false negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
<td>62 (68%)</td>
<td>21 (23%)</td>
<td>8 (9%)</td>
</tr>
</tbody>
</table>

E. Time analysis

All the test were run in a laptop equipped with an Intel® Core™i7 Q720 CPU running at 1.6 GHz with 4GiB of RAM. The majority of the algorithms used in this work are single threaded, and therefore do not use the CPU to its maximum.

<table>
<thead>
<tr>
<th>Test</th>
<th>Total Time (ms)</th>
<th>Seconds/frame</th>
<th>Frame/seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>9235817</td>
<td>11.84</td>
<td>0.084</td>
</tr>
<tr>
<td>Test 2</td>
<td>4961560</td>
<td>13.09</td>
<td>0.076</td>
</tr>
<tr>
<td>Test 3</td>
<td>6529330</td>
<td>14.32</td>
<td>0.070</td>
</tr>
<tr>
<td>Test 4</td>
<td>6399613</td>
<td>13.01</td>
<td>0.077</td>
</tr>
</tbody>
</table>

IV. Conclusion

The almost overlap of values over all the tests shows coherency in the system performance. The level of values achieved, always close to 70% gives the confidence that its basic blocks are strong and reliable. Having such results and coherency we believe this system can be fine-tuned to achieve better performances and used as a steady base for detection/classification of other types of objects.

The main objective of this work is to develop a system that is able to detect and classify moving objects without imposing any motion constrains to the camera. The system is able to detect the camera motion, and can be used as visual odometry. It can also detect motion from objects in the scene with a high confidence. The false positives are few, and even those, in a real usage situations, represent no problem, because they are most of the time not classified as objects of interest, and can be ignored.

The critical step of this work is the alignment between the two point clouds. If it fails, or finds a pour transformation then more and more static objects will appear as moving, reducing the performance of the system.

Overall and looking at the results, it is safe to say that all objectives were fully achieved. We presented a proof-of-concept that the system works and further investigations and development can be made taking this work as it’s base.
REFERENCES