6DoF Monte Carlo Localization in a 3D world with Laser Range Finders

A. L. Ballardini, A. Furlan, A. Galbiati, M. Matteucci, F. Sacchi, D.G. Sorrenti

research report n. 2012-01
may 2012

Title: 6DoF Monte Carlo Localization in a 3D world with Laser Range Finders

QD Quaderni - Dipartimento di Informatica, Sistemistica e Comunicazione,
Università degli Studi di Milano - Bicocca

Research Report n.1, May 2012

ISSN 1828-3357-2012-01

Printed by Dipartimento di Informatica, Sistemistica e Comunicazione,
Università degli Studi di Milano - Bicocca
Submitted to QD Quaderni on May 31th, 2012

6DoF Monte Carlo Localization in a 3D world with Laser Range Finders

A. L. Ballardini[1], A. Fúrlan[1], A. Galbiati[1]
, M. Matteucci[2], F. Sacchi[1], and D. G. Sorrenti[1]

[1] Dept. Informatica, Sistemistica e Comunicazione,
Università degli Studi di Milano - Bicocca,
Building U14 v.le Sarca 336, 20126, Milano, Italy
{ballardini,galbiati,sacchi}@ira.disco.unimib.it,
{furlan,sorrenti}@disco.unimib.it,
[2] Politecnico di Milano,
Dept. Elettronica e Informazione,
P.zza Leonardo da Vinci 32, 20132, Milano, Italy
matteucci@elet.polimi.it

Abstract. In mobile robotics, localization plays a key role in every task assigned to a robot. This report describes the probabilistic module we developed to solve the localization problem in our autonomous driving vehicle. Our method uses a complete 6DoF approach within a 3D motion model, in order to correctly integrate the error caused by dead reckoning. Furthermore, the description of the environment is a 3D voxel representation. We base on particle filtering, in the localization process. An ad-hoc simulation environment has been developed, for testing the effectiveness of the motion model. Finally, we performed some field experiments, to demonstrate that our approach consistently locates the 6DoF position of the vehicle along the driven path on the test site.

1 Introduction

Mobile robot localization is a challenging task that has been intensively analyzed over the last few years, see e.g. [12] [18] [20] being itself at the base of any task that could be assigned to a mobile robot.

The accuracy a robot can achieve in localization assumes a key role in order to avoid collisions against the structural elements belonging to the surrounding environment, e.g. against static and dynamic objects placed within the scene. This becomes even more critical in the field of autonomous driving, where a proper and accurate localization is of primary relevance because on it depends the individuals safety.

The risk of collisions against the environmental static components such as sidewalks, walls, guardrails, or against the environmental dynamic components such as other vehicles, pedestrians and animals, must be reduced to a minimum, foreseeing a series of emergency procedures to be activated in case of unexpected situations.

The robot localization cannot be managed using purely odometric data (dead reckoning) [19] [20] [14]. Wheel sliding due to particular ground surfaces, special weather conditions, variations on the wheels diameters e.g. due to differential vehicle loads, or subsidences, bumps or uneven road surfaces, makes essential the use of algorithms to determine the vehicle position when using other kind of sensors [20]. It must be noticed that in urban environments the GPS system, apparently an immediately available solution, has an absolutely not adequate reliability, with respect to the localization and navigation requirements, due to frequent lack of signal [12] [18].

While the state of the art provides different solutions for the two dimensional localization problem, those solutions are primarily designed for indoor robotic environments, where the analysis of the 3D motion space can be simplified, favoring an estimation of the robot displacement, limited to the 2D motion plane. The main three dimensional approaches within the technical literature use methodologies which adapt two dimensional movements to a three dimensional space. This procedures adopt a 3DoF probabilistic motion model in two dimensions that do not allow accurate modeling of the movement in a 6DoF space [3].

In [18] is used an approach similar to ours for the representations of the three dimensional space by means of voxel maps. However the robot positions are modeled only on a two dimensional basis and the vector state is composed only by the components x, y e δ (yaw angle). The other three components z, roll angle ϕ and pitch angle ψ are calculated from the two dimensional estimation and from the structure of the voxelized ground surface. Furthermore, the motion prediction does not consider possible interactions over the single components source errors, introducing uncertainty on the movement single components accordingly to a velocity model. The independency between the single pose components is also assumed in other works [2].

In [12] it is used a multilevel environment representation called multi-level surface maps. This technique is proposed as an extension of the elevation maps used in [13] and introduced in [6] and allows modeling vertical structures within a grid map used in localization with laser range finders. However these structures do not allow the representation of a typical urban outdoor complex situation like a bridge or a multilevel parking lot. Furthermore in [12] the motion model, even more sophisticated respect to the model proposed in [18], uses as a starting basis an evolution of the model introduced in [16] and similar to that illustrated in [20], again a purely two dimensional motion model.

The inadequacy of these simplifications in an urban outdoor situation has driven us to develop a probabilistic motion model, based on the modeling of a spatial generic movement considering all the six components of the 6DoF state vector, with an innovative proposal concerning how the uncertainty is introduced. The out coming system, adaptable to different robot types by means of adequate parameters, has been design in order to allow 6DoF movements predictions even without a complete transition vector (lack of some vector components). To achieve this goal, several thresholds values were introduced. These thresholds represent the minimum and maximum errors acceptable by the model in case of having or not inertial sensors.

In this report we illustrate the outdoor environment localization system used on our autonomous driven vehicle [3] [1]. The problem has been tackled introducing a completely three dimensional probabilistic motion model and the creation of three dimensional maps, extending the two dimensional occupancy grid in a three dimensional voxel map. The uncertainty of the localization within the three dimensional voxel map is dealt with using a particles filter and searching the correspondences by means of laser range finders.

2 Overview

The system illustrated in this report allows estimate the vehicle 6DoF localizations within a three dimensional existing map using a particles probabilistic approach. The system needs a voxel map and robot odometric measurements (dead reckoning). We have create the *extended odometer* concept which enhanced odometric vector measures including the three missing components of the two dimensional model, the coordinate z and the angles rool and pitch. These three components have been thought to be introduced with an Inertial Measurement Unit (IMU). If the IMU is not available, the extended odometer produces null measures of these three parameters. In the section 3 we'll show how the system interprets the lack of these values.

The system relies on ROS [15] environment for the integration and intercommunication between all hardware components. The system, consisting in an adaptive Monte Carlo Localization method, has been designed based on [9] which was a development of [7] [8] [20]. In our work several significant changes were introduced in order to adapt the different subparts of the system to a three dimensional environment:

– The two dimensional grid map structure has been converted in the three dimensional voxel map representation.
– The robot pose has been enhanced to 6 components. Three Cartesian coordinates and three attitude coordinates, identified by the angles Roll, Pitch e Yaw.
– The geometric orientation measures have been represented using quaternions, allowing more simple computation.
– Particles clustering operation has required the computation of quaternion average orientation. This operation has been performed according to [11] [10].
– For the weighting operation of the particles within the average orientation process, the SLERP [17] was used.

– The Bresenham algorithm [4], used in the scan-matching process, has been extended to a three dimensional environment.

3 Proposed motion model

The model we propose is based on the 3DoF formulation illustrated on [20, Sect. 5.4, p. 132]. In such work the evolution of the two dimensional displacement is divided in a sequence of three steps: an onsite rotation, a translation and a new rotation. This decomposition allows the introduction of an error on each displacement action which is composed through a Gaussian error having average value equal to zero and variance dimensioned ad-hoc, on the base of single elementary movements that can disturb each single parameter.

The motion model extended to 6DoF must represents accurately the vehicles possible experimented displacements and contemporary it must divides the generic displacement into a series of elementary displacements which allow the introduction of a parameterized error on each one of the six components of the pose, it mean on the spatial coordinates (x, y, z), but also on the three rotation components called (ϕ, ψ, θ), Roll, Pitch and Yaw respectively. Our proposal includes a series of additional parameters that will be used when the new displacement parameters $(\phi, \psi$ e $z)$ were not available in the robot used. This particularity makes our model adaptable to robot not provided of specific 3D navigation sensors.

To reach this goal, the first action to be performed is considering the expansion of the state vector: from the three components $x_t = (x, y, \theta)$ of the two dimensional model to the six components which allow to individuate a rigid body within a three dimensional world $x_t = (x, y, z, \phi, \psi, \theta)$.

Our system propose, starting from a defined pose $x_{t-1} = (position_{t-1}$,-$orientation_{t-1})$, a decomposition of the generic displacement action in six elementary displacement actions divided in two categories: three displacement acts to define the new $\overline{position_t}$ [3] and three displacement acts to define the new $\overline{orientation_t}$. To each one of the six displacement acts it will be added an uncertainty in order to create the same conditions of the two dimensional model. We will discuss the six elementary displacements that allows us to transform a $pose_{t-1}$ in a new $\overline{pose_t}$, starting from the part $\overline{position_t}$ (see Fig. 1 for the comparison with the three dimensional model). All the rotations and translations must be understood as intrinsic ones.

1. Rotation δ_{rot1}, Yaw type (around Z axis), necessary to align X_{t-1} axis on the $position_t$ directions; corresponding to two dimensional models rotation δ_{rot1}; we identify with X'_{t-1} the rotated axis orientation;
   $X'_{t-1} = X_{t-1} \cdot Yaw$.

[3] With the script $\overline{abc}$ we indicate the prediction

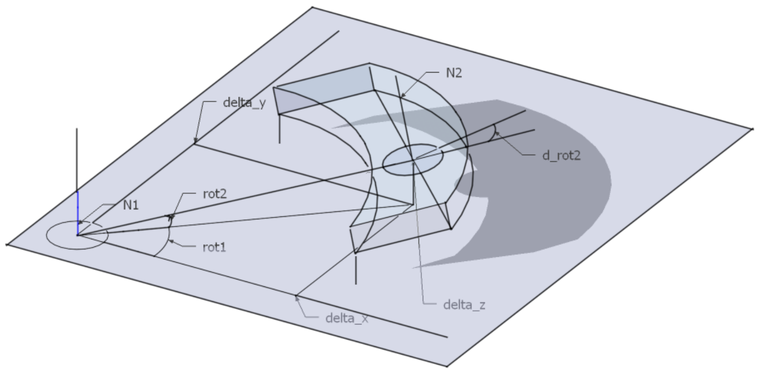

Fig. 1. Uncertainty area of the 3D Model.

2. Rotation δ_{rot2}, Pitch type (around Y axis), necessary to align X'_{t-1} axis on the new $position_t$ directions; this new displacement introduces the possibility of a change on elevation value;
   $X''_{t-1} = X'_{t-1} \cdot Pitch.$
3. Translation δ_{trans}, along X''_{t-1} axis; this translation will take the rotated reference system to the position $position_t$

The three parameters δ_{rot1}, δ_{rot2} e δ_{trans} identify the coordinates of the spherical system illustrated on Fig. 2. In order to move from the Cartesian reference system to the spherical reference system the equations 1 to 3 are used, while for the contrary the equations 4 to 6 are used:

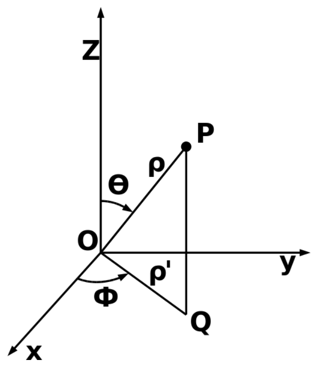

Fig. 2. Spherical coordinate system.

$$\rho = \sqrt{x^2 + y^2 + z^2} \tag{1}$$

$$\phi = arctan(\frac{y}{x}) = arcsin(\frac{y}{\sqrt{x^2 + y^2}}) \tag{2}$$

$$\theta = arccos(\frac{z}{\sqrt{x^2 + y^2 + z^2}}) = arccot(\frac{z}{\sqrt{x^2 + y^2}}) \tag{3}$$

$$x = \rho\ sin\theta\ cos\ \phi \tag{4}$$

$$y = \rho\ sin\theta\ sin\ \phi \tag{5}$$

$$z = \rho\ cos\theta \tag{6}$$

Once calculated the $\overline{position_t}$ to complete the definition of the $\overline{pose_t}$ we need the calculations of the $\overline{orientation_t}$; our solution propose to move a step behind to recover the information of the pose orientation $pose_{t-1}$, from which we can extract the orientation part $orientation_{t-1}$.

To such orientation we include a generic rotation composed of three rotation components $\Delta_{rot} = (\Delta_{Roll}, \Delta_{Pitch}, \Delta_{Yaw})$ individuated by the extended odometer.

The general rotation to apply to the $pose_{t-1}$ requires detailed attention because it has to be expressed as a composition of simple rotations. Rotations in a three dimensional space differs from two dimensional due to different reasons and multiple maths methods allow to handle these operations (i.e. rotation matrix, euler rotations, quaternions) each one with pros and cons. From the operational standpoint the most effective rotation method is given by quaternions.

Then, we introduce the three components of a simply rotation within a *Quaternion* structure.

- $Q_{t-1} \Leftarrow Roll_{t-1}, Pitch_{t-1}, Yaw_{t-1}$
- $Q_{\Delta} \Leftarrow \Delta Roll, \Delta Pitch, \Delta Yaw$

The composition of the quaternion Q_{Δ} with quaternion Q_{t-1} allows to obtain the new orientation $\overline{Q_t}$ from which is possible the calculation of the new values of $\overline{Roll_t}$, $\overline{Pitch_t}$ and $\overline{Yaw_t}$.

So the calculation of all the components of the pose $\overline{pose_t}$ is over and will be the composition of the part *position* and the part *rotation* calculated individually. It must be noted that this procedure goes along the same lines used by the two dimensional model proposed in [20].

In order the displacement model can generate believable hypothesis, it is necessary to introduce errors on the components of the state vector x_t; the errors, applied on each single component, will be generated by means of a extraction from a normal distribution with mean value equal to zero and standard deviation calculated according to the following specific considerations for each single displacement action. We start describing the standard deviations on the errors of the displacements actions that we identify with the *position* position.

1. The first displacement action, it means the rotation over the Z δ_{yaw1} axis is influenced by:
   - how much the vehicle has rotated.
   - how much linear space the vehicle has covered: the longer the displacements the higher error probability accumulated on the odometric measures
2. The second displacement action, it means the rotation around the Y δ_{pitch1} axis is influenced by:
   - how much the measure Δ_z changes, measured by the extended odometer
3. The third displacement action, it means the translation along the X axis of the reference system rotated with the first two movements δ_{trans} is influenced by:
   - how much linear space the vehicle has covered: the longer the displacements the higher error probability accumulated on the odometric measures
   - how much the vehicle has rotated around the Y axis, it means the variation Δ_{Pitch} measured by the odometer

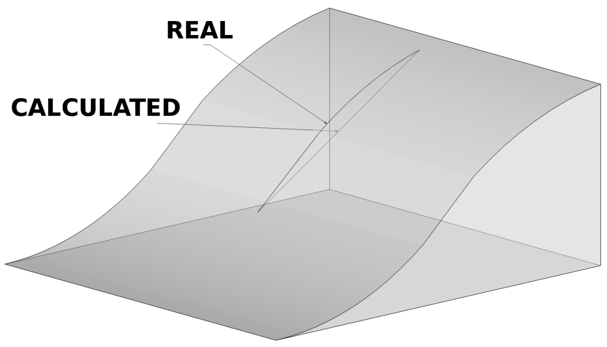

Fig. 3. Uncertainty from Pitch component.

Since the change of *Pitch* in a translation identifies a displacement over a not plane surface, we add a quantity depending on *Pitch* on the trial of explaining the error. In the figure Fig. 3 are illustrated the calculated translation and the real translation. In Fig. 4 it is illustrated a further explanation of the reason why it is introduced an error based on the variations of *Pitch*

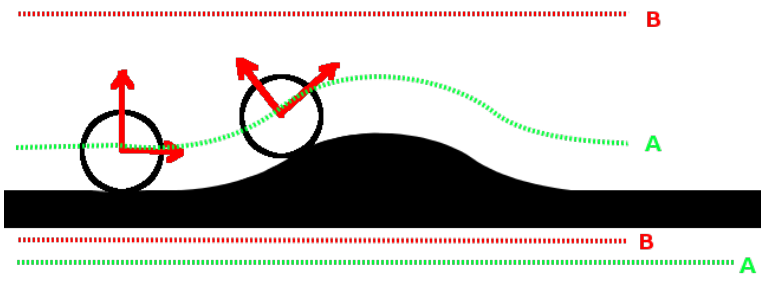

Fig. 4. Uncertainty from Pitch component.

– how much the vehicle has rotated around the X axis, it means the variation Δ_{Roll} measured by the extended odometer

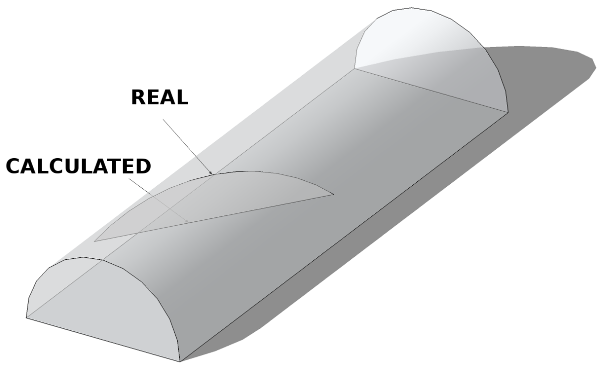

Fig. 5. Uncertainty from Roll component.

Since the change of *Roll* in a translation identifies a displacement over a not plane surface, we add a quantity depending on *Roll* on the trial of explaining the error. In the figure Fig. 5 are illustrated the calculated translation and the real translation.

  - how much the vehicle has rotated around the Z axis, it means the variation Δ_{Yaw} measured by the extended odometer (remember that the displacement Yaw is calculated by the encoder placed on the wheels).

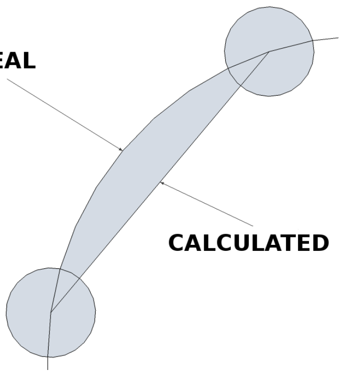

Fig. 6. Uncertainty from Yaw component.

Since the change of *Yaw* in a translation identifies a displacement over a not plane surface, we add a quantity depending on *Yaw* on the trial of explaining the error. In the figure Fig. 6 it can be seen the translation calculated and the real translation.

Let us see now the part *orientation*.

4. the final rotation around the X $\delta_{Final-Roll}$ axis is influenced by:
   - how much the vehicle has rotated around the X axis, it means the variation Δ_{Roll} measured by the extended odometer (remember that the displacement Roll is calculated by the inertial sensor)
5. the final rotation over the Y $\delta_{Final-Pitch}$ axis is influenced by:
   - how much the vehicle has rotated around the Y axis, it means the variation Δ_{Pitch} measured by the extended odometer (remember that the displacement Roll is calculated by the inertial sensor)
6. the final rotation over the Z $\delta_{Final-Yaw}$ axis is influenced by:
   - how much the vehicle has rotated around the Z axis, it means the variation Δ_{Yaw} measured by the extended odometer (remember that the displacement Yaw is calculated by the encoder placed on the wheels).
   - how much linear space the vehicle has covered: the longer the displacements the higher error probability accumulated on the odometric measures

So are defined the standard deviations (equations (7)...(12)) that will be applied to the six elementary displacement actions formerly individuated. Finally, in order to better control the model behavior, the components α (one for each δ) has been applied in order to increase or decrease the weight of each component of the system.

$$\sigma_{yaw1} = \alpha_1 \cdot \delta_{yaw1} + \alpha_2 \cdot \delta_{trans} \tag{7}$$

$$\sigma_{pitch1} = \alpha_3 \cdot \Delta_z \tag{8}$$

$$\sigma_{trans} = \alpha_4 \cdot \delta_{trans} + \alpha_5 \cdot \Delta_{Yaw} + \alpha_6 \cdot (\Delta_{Roll} + \Delta_{Pitch}) \tag{9}$$

$$\sigma_{Final-Roll} = \alpha_7 \cdot \Delta_{Roll} \tag{10}$$

$$\sigma_{Final-Pitch} = \alpha_8 \cdot \Delta_{Pitch} \tag{11}$$

$$\sigma_{Final-Yaw} = \alpha_9 \cdot \Delta_{Yaw} + \alpha_{10} \cdot \delta_{trans} \tag{12}$$

It should be noted how *sigma* valutes are differently composed according to the uncertainty origin (ie which part of the extended odometer system, encoder or IMU). We expect the IMU uncertainty won't be correlated with the encoder uncertainty. While $\sigma_{Final-Roll}$, $\sigma_{Final-Pitch}$ and σ_{pitch1} are calculated and influenced only from IMU part of the extended odometer the σ_{trans} is influenced both from encoder readings and IMU values (see Fig. 5 Fig. 3 and Fig. 6).

The six elementary displacement actions composed with their own uncertainties will be the following:

$$\delta_{\hat{yaw1}} = \delta_{yaw_1} + SAMPLE\underbrace{\{\alpha 1 \cdot \delta_{yaw1} + \alpha 2 \cdot \delta_{trans}\ \}}_{\sigma_{yaw1}} \tag{13}$$

$$\delta_{\hat{pitch1}} = \delta_{pitch_1} + SAMPLE\underbrace{\{\alpha 3 \cdot \Delta z\}}_{\sigma_{pitch1}} \tag{14}$$

$$\delta_{\hat{trans}} = \delta_{trans} + SAMPLE\left(\underbrace{\alpha_4 \cdot \delta_{trans} + \alpha_5 \cdot \Delta_{Yaw} + \alpha_6 \cdot (\Delta_{Roll} + \Delta_{Pitch})}_{\sigma_{trans}}\right) \tag{15}$$

$$\hat{\delta_{Final-Roll}} = \delta_{Final-Roll} + SAMPLE\underbrace{(\alpha_7 \cdot \Delta_{Roll})}_{\sigma_{Final-Roll}} \tag{16}$$

$$\hat{\delta_{Final-Pitch}} = \delta_{Final-Pitch} + SAMPLE\underbrace{(\alpha_8 \cdot \Delta_{Pitch})}_{\sigma_{Final-Pitch}} \tag{17}$$

$$\hat{\delta_{Final-Yaw}} = \delta_{Final-Yaw} + SAMPLE\underbrace{(\alpha_9 \cdot \Delta_{YAW} + \alpha_{10} \cdot \delta_{trans})}_{\sigma_{Final-Yaw}} \tag{18}$$

In case an extended odometer as that one described on Paragraph 2 were not available, we use the priori uncertainty values for each parameter, calculated with a specific method described in the following sections.

3.1 Model Thresholds

The following sections will describe some considerations introduced in order to make a more robust model.

Minimum Thresholds These thresholds are used in case the sampling algorithm produces small values. Minimum thresholds introduce and guarantee a minimum dispersion level over sampled data, that is necessary to correct model operation. The introduction of these parameters reflect the attempt to reduce the *particle deprivation.*

Maximum Thresholds Contrary to minimum thresholds, maximum thresholds have been introduced in order to manage situations where the extended odometer does not give 6DoF variations.

Maximum thresholds represent the maximum *a priori* uncertainty (we expect a better estimate of robot movements using a sensor). The σ_{max} associated with every model parameter have to be suitably big to ensure that measurements can be generated with enough dispersion near the mean value, in order to represent possible changes on the given degree of freedom. We have chosen the size of these thresholds considering a maximum vehicle speed of 25 Km/h and a 20 hz acquiring frequency from the odometer. Under these considerations we have determined the σ_{max} values shown on Table 1, using our calculated estimates as 3σ value and the Chebyshev inequality.

parameter	σ_{max}
yaw1_max	0.26
pitch1_max	0.07
trans_max	0.01
roll_max	0.1
pitch_max	0.1
yaw_max	0.1

Table 1. Calculated σ_{max}

4 6DoF Localization

The common feature within all MCL (*Monte Carlo Localization*) algorithms is a representation of the estimated localization with a non parametric posterior probability distribution $bel(x_t)$. The localization is described as a set of $\mathcal{M}$ particles $\mathcal{X}_t = \{x_t^{[1]}, x_t^{[2]}, \cdots, x_t^{[\mathcal{M}]}\}$ and represents the straightforward application of a particle filter over a typical localization problem. Our system is inspired by the Monte Carlo localization algorithm proposed in [20] and follows the structure of the system proposed by Brian Gerkey [9]. The probability density distribution $p(x_t|z_{1:t}, u_{0:t-1})$ is updated as follows:

$$p(x_t|z_{1:t}, u_{0:t-1}) = \alpha \cdot p(z_t|x_t) \cdot \int p(x_t|x_{t-1}, u_{t-1}) \cdot p(x_{t-1})dx_{t-1} \tag{19}$$

On (19) the prediction motion model is represented by $p(x_t|x_{t-1}, u_{t-1})$, while $p(z_t|x_t)$ represents the sensor model. The main contribution given in [3] and described in this report relies on a 6DoF prediction model, based on the considerations shown in section 3. Regarding the sensor model, the Beam Model proposed in [5] has been used. The MCL algorithm uses the KLD-Sampling variation [7, 8] to dynamically adjust the number of necessary particles and the Augmented-MCL [20, Sect. 8.3.5, p. 257] variant in order to recover from global localization failures. To allow the system dealing with three dimensional situations, the following considerations are required.

4.1 3D Raycasting

The Bresenham algorithm [4] has been used to calculate the z_t^{k*} parameters used in the Beam Model. Given two three dimensional points, the algorithm allows to select which voxels on the discretized space have been crossed by the laser beam, allowing to check in an iterative mode the presence of obstacles on the map. This procedure allows the Beam Model to achieve a probabilistic estimate of the measurement. In our system the points used correspond the origin of the laser beam (the 6DoF pose of the LIDAR, considering the angular change of the specific beam) and to the maximum sensor range.

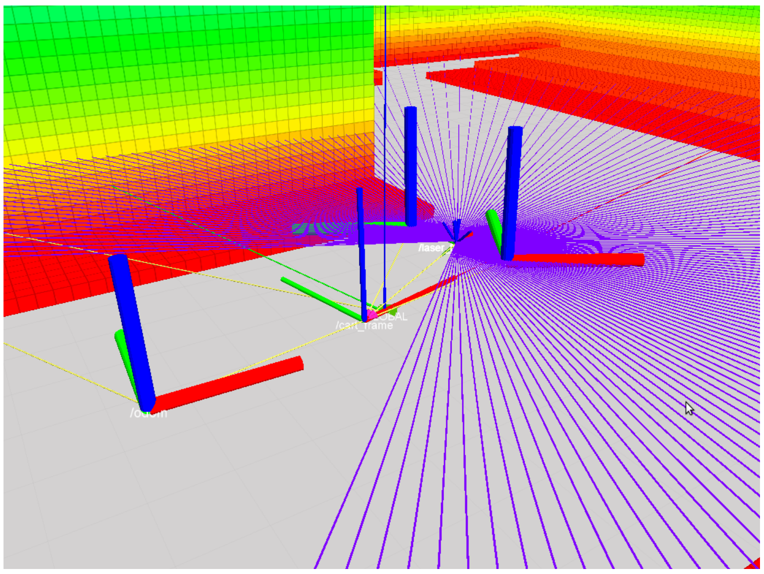

Fig. 7. The raycasting procedure, executed by the simulator (see section 5)

4.2 3D Clustering

Even though the basic idea of the particles algorithms consist in representing the continuous distribution of the robot estimate state with a discrete approximation, the goal of the system is to generate a single estimate for our vehicle. An evaluation of particles is needed in order to give a continuous estimate. This kind of operations are known in literature as Density Extraction algorithms [20, Sect. 4.3.4, p. 104] and differ to each other for the kind of result to be obtained and computational complexity. In our system we use particle clustering method. The clustering is the last logic block of the system and it has been introduced to improve the estimate generated from the particles. The clustering algorithm is performed both in spatial and rotation terms that every single particle can assume. The algorithm follows the pseudocode in Algorithm 1.

4.3 Average Weighting of Orientations

Once the clustering process on the particles has been executed the problem of extracting a single *pose* from the cluster set arise. We tackle this as following: using the weights associated to every single particle, the heaviest cluster is chosen. This cluster matches the particles most populated area. Then the value of the final *pose* is calculated averaging the weight of whole *pose* vector, using the weights of particles belonging to the cluster. The need to calculate an average

```
Algorithm Clustering(ParticleSet X);
for $i$=0 to $N$ do
    $X[i].cluster \leftarrow -1$;
end
for $i$=0 to $N$ do
    if $X[i].cluster$ = -1 then
        $X[i].cluster \leftarrow cluster\_count++$;
    end
    for $j$=0 to $N$ do
        if $X[j].cluster \neq -1$ then
            continue;
        end
        if $i==j$ then
            continue;
        end
        $translation = calculate\_euclidean\_distance(X[i], X[j])$;
        $rotation = calculate\_angle(X[i], X[j])$;
        if $traslation <= traslation\_threshold$ then
            if $rotation <= rotation\_threshold$ then
                $X[j].cluster \leftarrow X[i].cluster$;
            end
        end
    end
end
```

Algorithm 1: Pseudo code of the clustering algorithm used in our system

pose both with the spatial part ((x, y, z) components) and the three dimensional rotation part ((ϕ, ψ, θ) components) arise. Furthermore the average pose have to be weighted on both parts (see Fig. 8). If calculating the spatial weighted average can be considered trivial operation, the rotation considering the three components *roll, pitch* and *yaw* requires more complex calculation, since the order of multiplication of the rotation matrix affects the final orientation. In our system quaternion representation for rotations has simplified the weighting average problem. The method which allows to calculate an average three dimensional orientation using quaternions has been described in (20) [11] and [17] and consists in the sum of each component of a quaternion followed by a final normalization.

$$Q_{mean} = \frac{(Q^1 + Q^2 + \cdots + Q^N)}{\|Q^1 + Q^2 + \cdots + Q^N\|} \tag{20}$$

Slerp [17] is the acronym for *Spherical Linear Interpolation* and is a quaternion interpolation method. This operation is needed in order to work with a specific weight for every quaternion, during the rotation averaging process.

Again, using the quaternion representation for rotations makes the operation simpler. The Slerp procedure applied to quaternion objects takes as input a second quaternion and a scalar value t, that matches with the interpolation ratio. When the parameter is equal to zero the resulting Slerp operation correspond to a null rotation, while with a parameter value equal to one the operation results in the second quaternion orientation. The Slerp procedure is given by the BulletPhysics [4] framework.

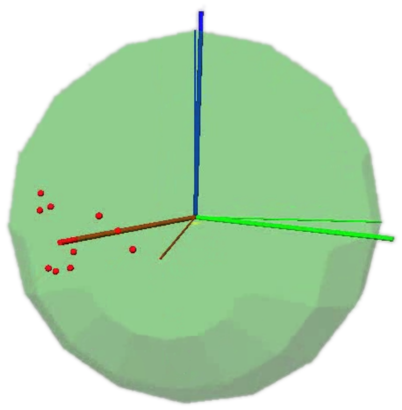

Fig. 8. The result of a non weighted quaternion average. In red the orientations of each quaternion. The rotated reference system matches the average mean of the red quaternions. The thin reference system matches the identity orientation used in the process.

5 Model Validation

Overall system simulations have been required in order to validate our work. Most important simulations have concerned the 6DoF motion model. Two simulation environments has been produced. The first simulator tests only the motion model while the second simulator produces a complete environment for the whole system, including the MCL, 3D raycasting and the clustering algorithms.

The first simulation environment, shown in Fig. 9, is designed to verify the motion model accuracy; the environment allows to analyze the effect of each parameter on the overall behavior, in successive simulation steps. All model parameters can be set. The parameter models are:

[4] Bullet, an open source physics engine. `http://bulletphysics.org`

- 6 parameters relative to the extended odometer (i.e. odometric variations)
- 10 parameters relative to the weighting values of the previous 6 model parameters
- 6 parameters relative to the minimum thresholds described in 3.1
- the simulation steps number
- the number of hypothesis to be evaluated

This simulator has shown useful to understand the effect of each component model over the final displacement generated. It has been essential for an accurate definition of the *min-max thresholds* parameters. In Fig. 9 a screenshot of the simulator is shown.

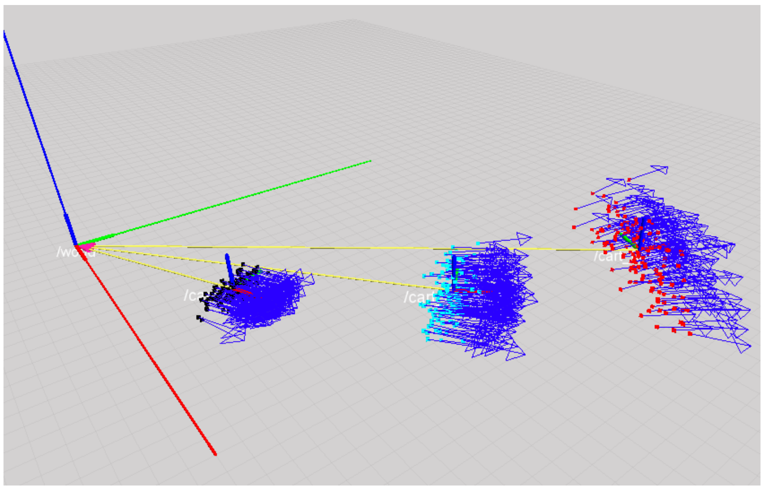

Fig. 9. 6DoF simulator with 3 evolution steps. Each step having the same displacement at each time.

Tests executed inside the second simulation environment (see Fig. 10) have shown excellent behaviors, similar to those using the 3DoF node present in the ROS system. The target of our work was having at least a similar performance as in the previous 3DoF system. Deactivating the four-plane LIDAR we have verified an uncertainty increase over z axis (see Fig. 11).

6 Experimental environment

Our software has been written in C++ under Linux (Ubuntu 10.10, kernel 2.6.35-32-generic-pae, 32 bit arch) and compiled with gcc 4.4.5 (repository

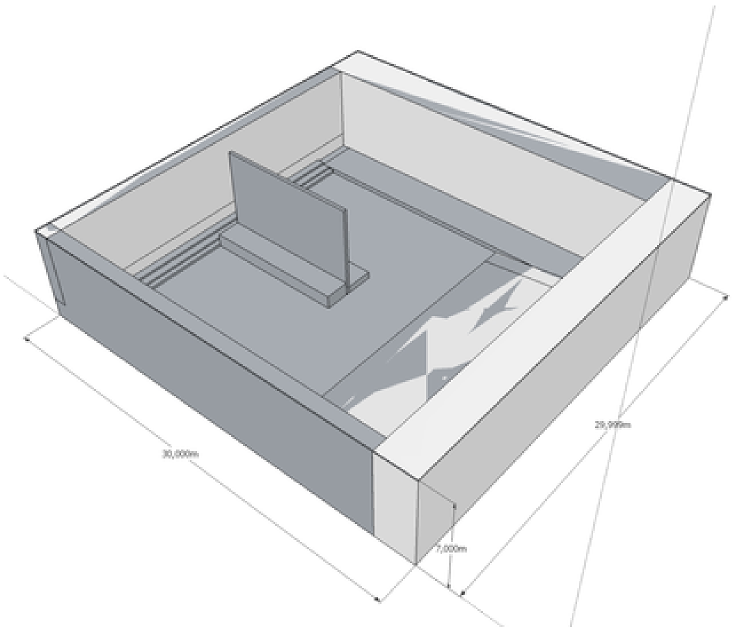

Fig. 10. Simulation environment (same environment is shown voxelized in Fig. 11)

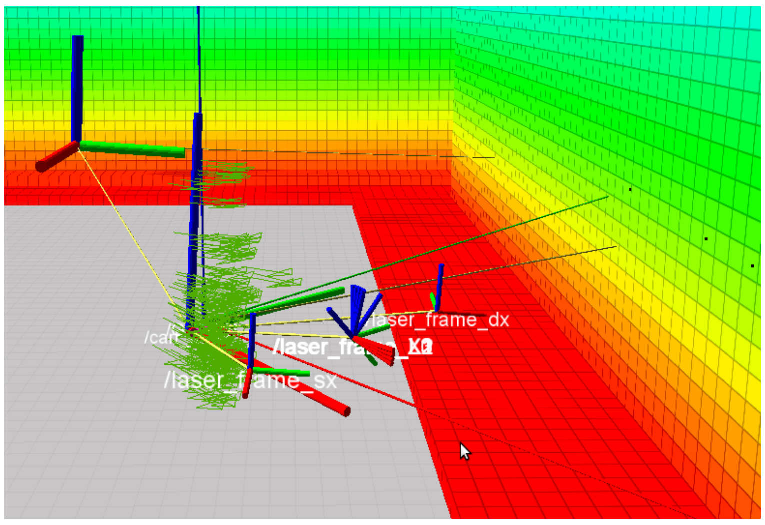

Fig. 11. Result of localization process using only LIDARS parallel to the ground surface. Uncertainty is expressed over the z axis.

Ubuntu/Linaro 4.4.4-14ubuntu5). For what concerns the choise of the middleware we studied several solutions including Orocos, Aria, Carmen, Orca. Our choice has been ROS, because of the documentation, and the availability of software in the open-source community (sensor drivers, 2D localization and navigation solutions and 3D graphic simulator environment).

6.1 ROS Middleware

The ROS core offers high level procedures and features including hardware abstraction, both in synchronous and asynchronous message passing between processes (processes are called nodes in ROS). ROS also provides tools to run code on a distributed computing platform. This section will deal with some problems experienced using ROS middleware. The problems we have found during the development of our system were also reported in [9], which has been the system we started from.

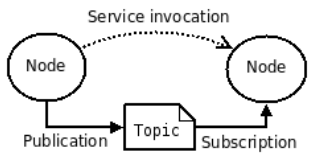

Fig. 12. Communication paradigms over ROS nodes

ROS Queues Message queues attached to every node are a feature of the ROS system. Publishing a message from one node consists in insert some data into one of the node queues. When data is written into the output queue, a special node, called ROS master, delivers the message to the final node, in accordance with internal threads timings. This process allows the user algorithms to avoid being involved with message delivery. Input queues guarantee that the delivery of messages can be achieved even when the user node is oveloaded, e.g. when running a specific algorithm. These functionalities require particular care in the queue sizing. Robots are realtime systems and reading and writing queues brings delays in messages exchange.

To prove how queues could be dangerous we suggest a simple example. Given a ROS node in charge of just laser reading and publishing i.e., a task that creates messages at a 25Hz frequency. Given another ROS node with the task of analyzing the received data with a 100 messages queue length (the queue is FIFO data structure; old elements are discarded when a new message arrives if the queue is full). Suppose that the execution time of the analyzing algorithm is 0.1 seconds. It follows that, after 10 seconds of system execution, the queue of the receiving node will be full and an important problem shows up. If a third node was waiting for the output of the second node and this last node manages an emergency task, e.g, the emergency stop due to the presence of an obstacle in front of the vehicle. Although the detection of the obstacle takes a small time (suppose 0.04 seconds), the emergency signal will be raised as in the following table, due to the saturation of the incoming message queue.

0.04 seconds	needed to get the full LIDAR scan, including message creation and insertion time in the output queue
0.01 second	assumed time needed by ROS master in order to deliver the message
$0.1 \cdot 99 = 9.9$ seconds	needed by the second node to analyze all 99 negative messages preceding the critical one
0.14 seconds	needed by the third node to create the emergency message and put it into the output queue
0.01 seconds	in order to have ROS getting and delivering the message
0.04 seconds	assumed time needed in order to have the third node read the message and start the emergency braking procedure
10.14 seconds	overall time needed before initialization of the braking operation

Real tests, performed at 5 Kmph, demonstrate that the delay computed above is less than the one actually required to activate the braking procedure. Given the quadratic dependency on time, of the space required to stop the vehicle, is obvious that such a delay can't be accepted, even more on an autonomous driving vehicle. This effect was also illustrated by Brian Gerkey with the [9] AMCL [5] node and it is modeled as a spring effect actiong on the correct localization, forcing manual intervention on the brake system.

The solution to this problematic issue is in the reduction of the queue length, up to unit length. In the tests at building U5 we verified that the ROS AMCL version, after reducing the queue length to 1, is no more affected by the delay problem.

6.2 Intraprocess communication

ROS nodes are system processes. During the communication between different nodes, ROS provides serialization and deserialization of C++ data structures,

[5] `http://www.ros.org/wiki/amcl`

in order to compose a message. In case of messages containing large amounts of data, as a complete scan from a LIDAR (which might be not only containing distance data, but also an intensity value corresponding to each distance), the procedure for sending messages can introduce a delay in communication. A solution to this problem is represented by the techniques of *intraprocess publishing* provided since the *Cturtle* ROS version. It consists of the definition of a new type of node: the *nodelet*. Nodelets are designed to allow the execution of several nodes as threads, without incurring in the cost of copying data when messages are exchanged within the same process; optimizations concern handling pointers to data instead of real data. In order to manage nodelets, ROS provides dynamical plugins (library pluginlib[6]); nodelets, used as plugins, are *loaded* on a *nodelet master* node and executed as threads.

6.3 Code Parallelization

Regarding code parallelization and multithreading techniques, Intel® Threading Building Block (TBB[7]) version 3.0 and 4.0 have been used. The TBB library allow to easily implement concurrent code. *Algorithms* represent a set of concurrent patterns that interface a scheduler, which allows the runtime creation of threads; the number of threads is dynamically adapted, based on the available resources, allowing good scalability according to software requests and hardware availability. In order to verify the performances achieved with the TBB parallelization techniques, we performed several tests with an Intel® Core™ i7-740QM, 6mb cache, 8gb ram (quad core) computer. An increase on the performances of the Beam Model algorithm has been noticed, almost linear in the number of available cores. Such increase has not been observed on algorithms with lower computational load.

6.4 Voxeling

With *voxel* (**vo**lumetric pi**xel**) we denote the three dimensional counterpart of the two dimensional *pixel*. If the pixel could be represented as a square, a voxel could be intended as a cube. Voxel maps represent 3D environments in raster format. The voxeling procedure is necessary since the whole system uses discretized 3D maps. These maps have been generated converting vectorial maps drawn with a CAD software (e.g. Autodesk Autocad or Google Sketchup Pro) into binary raster maps. To perform the conversion we used the Binvox[8] [9] tool. In Fig. 10 is represented the testing environment used in the simulation

[6] `http://www.ros.org/wiki/pluginlib`
[7] `http://threadingbuildingblocks.org`
[8] `http://www.cs.princeton.edu/\~min/binvox/`
[9] Additional information about Binvox are available at `www.minecraftwiki.net/wiki/Binvox`

of the 6DoF odometric model; it includes a planar surface where the vehicle can move, with upward edges (in order to simulate sidewalks of a road) and a ramp. In Fig. 13 we can observe the environment obtained with the voxeling process. This procedure will be replaced by 3D mapping algorithms which will use LIDAR sensors to obtain high quality reconstructions of environments e.g. the real slope of a flooring or areas of which a drawing is not available.

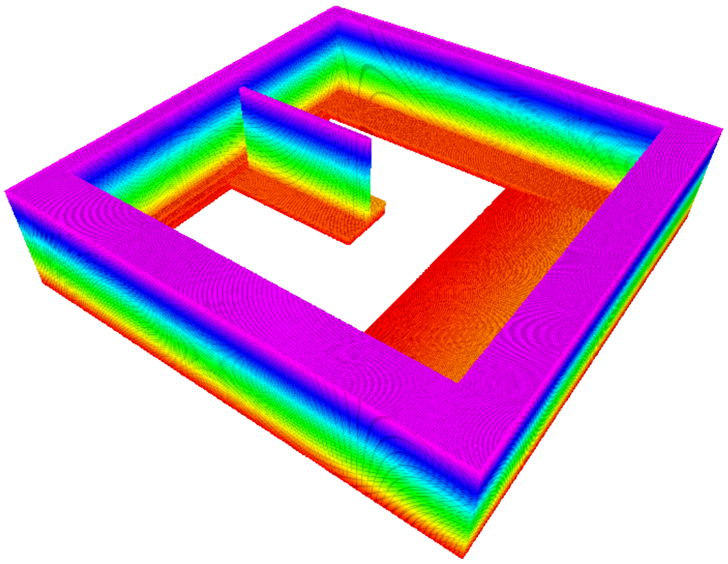

Fig. 13. Voxel version of the simulated scene. Colors change with height

7 Experimental Results

After the simulation tests, we started verifying the software on the real vehicle. The environment where the tests have been performed is the parking area of the U5 building (see Fig. 15). We made the tests in this order: first of all we verified that the localization was correct when moving on the almost planar surface of the underground parking, obtaining results comparable to the results that can be obtained with the state of the art software [9] reference tests. Then we drove both on the ramp of the parking and outside, still localizing successfully. Despite roll and pitch data were available, thanks to the inertial MTi X-sens sensor, we verified that using only the available LIDAR sensors, together with the minimum and maximum thresholds of the odometric model proposed in 3.1, sufficed for a correct localization. We also noticed that using together both kind of LIDAR models (LMS111 and LDMRS4001, see below) is extremely useful, since they measure on different scanning planes.

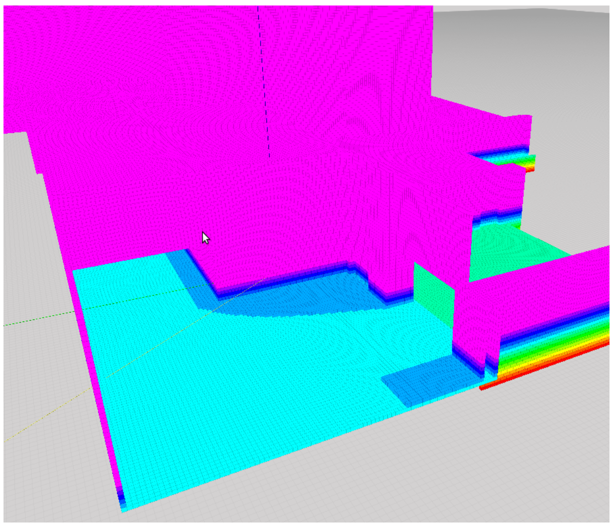

Fig. 14. Voxel representation of the U5 building underground parking. Here the exit that goes to the ramp.

7.1 Experimental Robot Configuration

The autonomous vehicle used during this project is an Alpaca Golf Cart, produced by Ecology Runner [10] (see Fig. 17). The vehicle is an integral part of the research conducted by the Informatics for Robotics and Automation Laboratory (IRALab) on the USAD [11] project, which aims to perform autonomous navigation in an urban outdoor environment. At a technical level, the vehicle's movement is controlled by an ad-hoc control board developed by IRALab which interfaces with a PC by means of an USB connection; the main component of the board consist in a Microchip DSPic micro-controller unit. This MCU, among other activities, controls the readings taken by a couple of infrared emitters and receivers mounted on the shaft of the posterior wheels, paired with an optic wheel. Regarding the positioning of the LIDAR sensors we built a support bar

[10] `http://www.ecologyrunner.it`
[11] `http://irawiki.disco.unimib.it`

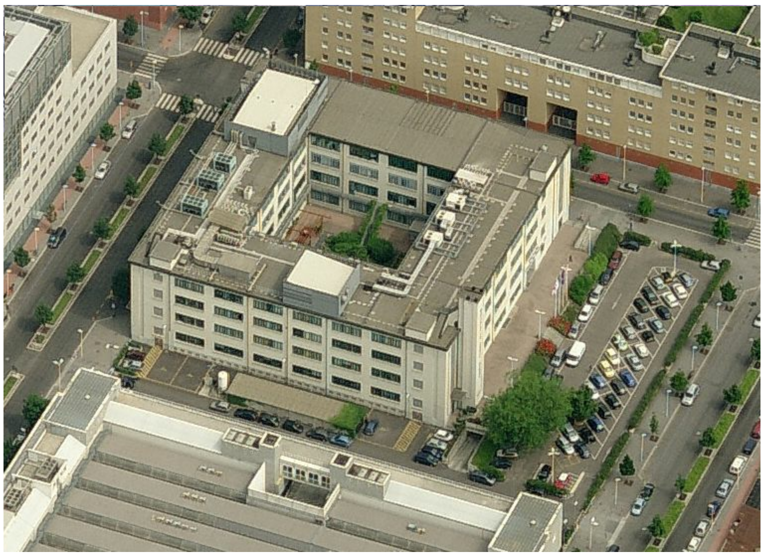

Fig. 15. U5 building - Aerial view

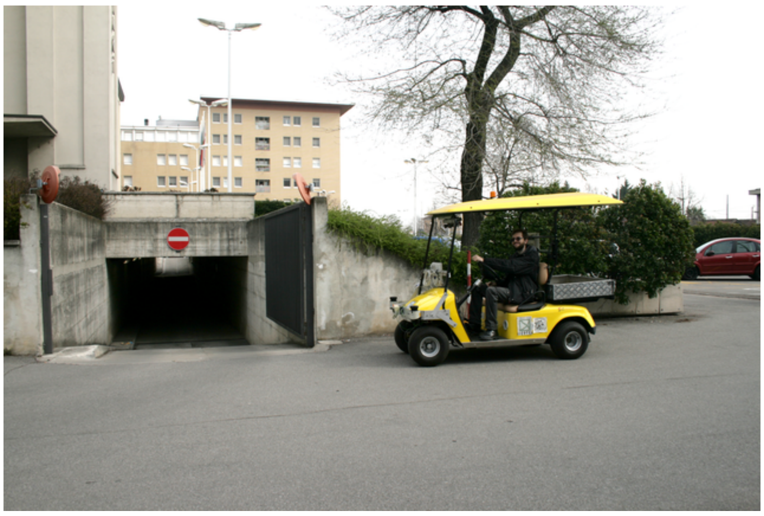

Fig. 16. Garage ramp of the U5 building (picture taken from the external parking lot).

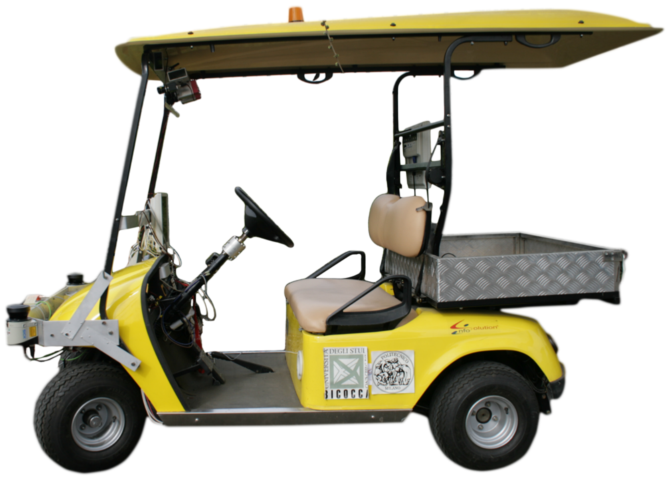

Fig. 17. Our vehicle at the current development stage. On the front the three LIDARs can be noticed.

on which scanners are fixed. During the construction of this support particular attention has been paid to the choice of the anchoring points on the vehicle structure and to the material used for the construction, in order to minimize the spring effect that such structure would have introduced on the laser readings.

7.2 LIDAR Configuration

Our vehicle sports the following LIDAR sensing devices.

Sick LMS111 Laser scanner Sick LMS111, this device produces scans in a single scanning plane, with a field of view of 270 deg (slightly less, so to avoid detecting the vehicle itself as an obstacle), and an angular resolution of 0.5 deg, and a maximum scanning frequency of 50Hz. We used two such devices, at the forward left and right corners of the vehicle, in order to cover the whole area about the vehicle, apart the rear.

The laser scanners have been installed on the vehicle so that their scanning plane is the same, and this to be parallel to the ground plane, in normal vehicle conditions. These devices are very effective, particularly in 2D localization, thanks to their field of view, maximum range ($20m$), and reliability in different lighting conditions. In 3D environments, single scanning plane scanners, mounted parallel to the ground plane, of course do not allow the state space to be fully observable.

For what concerns interfacing the sensors with the computing system onboard the vehicle, we initially used the drivers provided with the ROS LMS1xx [12].

Sick LDMRS-4001 The field of view the Sick LDMRS-4001 lidar is limited to 100 deg, but it can perceive 4 horizontal layers simultaneously (multi layer Sick technology) angularly spaced by 0.8 deg. The maximum range for the Sick LDMRS-4001 is 100 meters with a constant depth resolution of 4cm and an angular resolution, i.e., between the laser beams, of 0.25 deg.

This sensor has turned out to be particularly useful to obtain a proper localization in a 3D map; it has been placed on the frontal support tilted downward so to have the upper scanning plane parallel to the road plane (when the vehicle is still) and at the same height of the two Sick LMS111. The sensor driver has been written from scratch, although recently the ROS community came out.

The use of one Sick LDMRS-4001 and 2 Sick LMS111 provides 6 scanning planes; 3 are pointing toward the street and 3 are parallel to it. Thank to the multilayer sensor, which allows the estimation of vehicle height when the two LMS111 scanning planes are parallel to the road, this set up is enough to obtain a 3D scan of the environment suitable for 6DoF localization.

Conclusions

We have developed a localization system for mobile robots operating in a three dimensional world, i.e., system computes the six parameters which characterize the pose of a rigid body in space. In order to obtain this goal it was necessary the definition of a probabilistic motion model. The develop of such probabilistic model required the development of a simulation system to perform the required testing. We use these developments in several public events as the fair *Electrical Intelligent Vehicles 2010* and an episode of television program Rai3 *Buongiorno Regione*, recorded out the university square between the U1 and U2 buildings named Piazza della Scienza. The work is part of the project *Urban Shuttles Autonomously Driven* (USAD) under development at the IRA laboratory. The goal of such project is to allow an autonomous vehicle to perform navigation within urban outdoor environments as our university campus.

Acknowledgments

This work has been partially supported by project ROAMFREE, a PRIN2009 project, funded by MUR (the Italian Ministry for University and Research).

[12] `http://www.ros.org/wiki/LMS1xx`

References

1. Urban shuttles autonomously driven, `http://irawiki.disco.unimib.it/irawiki/index.php/Outdoor_autonomous_navigation_in_urban_environment`
2. Baldwin, I., Newman, P.: Road vehicle localization with 2d push-broom lidar and 3d priors. In: Proc. IEEE International Conference on Robotics and Automation (ICRA2012?) - ACCEPTED. Minnesota, USA (May 2012)
3. Ballardini, A.L.: 6DoF Monte Carlo Localization in a 3D world with Laser Range Finders. Master's thesis, Università degli Studi Milano - Bicocca (2012)
4. Bresenham, J.: Algorithm for computer control of a digital plotter. IBM Systems Journal 4(1), 25–30 (1965)
5. Dellaert, F., Fox, D., Burgard, W., Thrun, S.: Monte carlo localization for mobile robots. In: Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on. vol. 2, pp. 1322–1328 vol.2 (1999)
6. Eliazar, A.I., Parr, R.: Learning probabilistic motion models for mobile robots. In: Proceedings of the twenty-first international conference on Machine learning. pp. 32–. ICML '04, ACM, New York, NY, USA (2004), `http://doi.acm.org/10.1145/1015330.1015413`
7. Fox, D.: Kld-sampling: Adaptive particle filters. In: Advances in Neural Information Processing Systems 14. MIT Press (2001)
8. Fox, D.: Adapting the sample size in particle filters through kld-sampling. International Journal of Robotics Research 22, 2003 (2003)
9. Gerkey, B.: Amcl, http://ros.org/wiki/amcl
10. Gramkow, C.: Onaveragingrotations. Journal of Mathematical Imaging and Vision 15, 7–16 (2001), `http://dx.doi.org/10.1023/A:1011217513455`, 10.1023/A:1011217513455
11. Humbert, M., Gey, N., Muller, J., Esling, C.: Determination of a Mean Orientation from a Cloud of Orientations. Application to Electron Back-Scattering Pattern Measurements. Journal of Applied Crystallography 29(6), 662–666 (Dec 1996), `http://dx.doi.org/10.1107/S0021889896006693`
12. Kümmerle, R., Triebel, R., Pfaff, P., Burgard, W.: Monte carlo localization in outdoor terrains using multi-level surface maps. In: In Proc. of the International Conference on Field and Service Robotics (FSR (2007)
13. Lacroix, S., Mallet, A., Bonnafous, D., Bauzil, G., Fleury, S., Herrb, M., Chatila, R.: Autonomous rover navigation on unknown terrains: Functions and integration. The International Journal of Robotics Research 21(10-11), 917–942 (2002), `http://ijr.sagepub.com/content/21/10-11/917.abstract`
14. Olson, C.: Probabilistic self-localization for mobile robots. Robotics and Automation, IEEE Transactions on 16(1), 55–66 (feb 2000)
15. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
16. Roy, N., Thrun, S.: Online self-calibration for mobile robots. In: Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on. vol. 3, pp. 2292–2297 vol.3 (1999)
17. Shoemake, K.: Animating rotation with quaternion curves. In: Proceedings of the 12th annual conference on Computer graphics and interactive techniques. pp. 245–254. SIGGRAPH '85, ACM, New York, NY, USA (1985), `http://doi.acm.org/10.1145/325334.325242`

18. Suzuki, T., Kitamura, M., Amano, Y., Hashizume, T.: 6-dof localization for a mobile robot using outdoor 3d voxel maps. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on. pp. 5737–5743 (oct 2010)
19. Thrun, S.: Probabilistic algorithms in robotics. AI Magazine 21(4), 93–109 (2000)
20. Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. Intelligent robotics and autonomous agents, MIT Press (2005)