Lecture Notes for
IROS 2007 Workshop III
Monday, October 29,

Benchmarks in Robotics Research

Organizers

Angel P. del Pobil
Univ. Jaume I
pobil@uji.es

Rad Madhavan
ORNL/NIST
madhavan@nist.gov

Elena Messina
NIST
elena.messina@nist.gov
Program

Synthetic Approach to Cognitive Systems: A Perspective from Cognitive Robotics,
Kaz Kawamura

Benchmarking Urban 6D SLAM,
Oliver Wulf, Andreas Nuchter, Joachim Hertzberg, and Bernardo Wagner

The Jacobs Test Arena for Security, Safety, and Rescue Robotics (SSRR)
Andreas Birk, Kaustubh Pathak, Jann Poppinga, Soren Schwertfeger, Max Pfingsthorn and Heiko Bulow

Towards Quantitative Comparisons of Robot Algorithms: Experiences with SLAM in Simulation and Real World Systems
Benjamin Balaguer, Stefano Carpin, Stephen Balakirsky

Reliability Testing for Embodied Autonomous Systems
L. F. Gunderson and J. P. Gunderson

Advances in the Framework for Automatic Evaluation of Obstacle Avoidance Methods
J.L. Jimenez I. Rañó, J. Mingeuz

Good Experimental Methodologies in Robotics: State of the Art and Perspectives,
Fabio P. Bonsignorio, John Hallam, and Angel P. del Pobil
Cognitive Robot and Internal Rehearsal

It is known that humans are able to have sensory experiences in the absences of external stimuli. It thus seemed reasonable to assume the existence of an ‘inner sense’ where sensory experiences and consequences of different behaviors may be anticipated in cognitive robots. The idea of the existence of such an inner sense does not necessary go against the theory of embedded intelligence advocated by a number of researchers who de-emphasize the role of internal world models and instead emphasize the situated and embodied nature of intelligence. An alternative to internal world models is the ‘simulation hypothesis’ by Hesslow [1] which accounts for the ‘inner world’ in terms of internal simulation of perception and behavior. Our approach may be termed as a “grounded internal simulation” utilizing one type of internal representation of perception and behavior (Figure 2).

Cognitive Robot and Internal Rehearsal

It is known that humans are able to have sensory experiences in the absences of external stimuli. It thus seemed reasonable to assume the existence of an ‘inner sense’ where sensory experiences and consequences of different behaviors may be anticipated in cognitive robots. The idea of the existence of such an inner sense does not necessary go against the theory of embedded intelligence advocated by a number of researchers who de-emphasize the role of internal world models and instead emphasize the situated and embodied nature of intelligence. An alternative to internal world models is the ‘simulation hypothesis’ by Hesslow [1] which accounts for the ‘inner world’ in terms of internal simulation of perception and behavior. Our approach may be termed as a “grounded internal simulation” utilizing one type of internal representation of perception and behavior (Figure 2).

Cognitive Robot and Internal Rehearsal

It is known that humans are able to have sensory experiences in the absences of external stimuli. It thus seemed reasonable to assume the existence of an ‘inner sense’ where sensory experiences and consequences of different behaviors may be anticipated in cognitive robots. The idea of the existence of such an inner sense does not necessary go against the theory of embedded intelligence advocated by a number of researchers who de-emphasize the role of internal world models and instead emphasize the situated and embodied nature of intelligence. An alternative to internal world models is the ‘simulation hypothesis’ by Hesslow [1] which accounts for the ‘inner world’ in terms of internal simulation of perception and behavior. Our approach may be termed as a “grounded internal simulation” utilizing one type of internal representation of perception and behavior (Figure 2).

Reference:
Benchmarking Urban 6D SLAM

Oliver Wulf, Andreas Nüchter, Joachim Hertzberg, and Bernardo Wagner

Abstract—In the past many solutions for simultaneous localization and mapping (SLAM) have been presented. Recently these solutions have been extended to map large environments with six degrees of freedom (DoF) poses. To demonstrate the capabilities of these SLAM algorithms it is common practice to present the generated maps and successful loop closing. Unfortunately there is often no objective performance metric that allows to compare different approaches. This fact is attributed to the lack of ground truth data. For this reason we present a novel method that is able to generate this ground truth data based on reference maps. Further on, the resulting reference path is used to measure the absolute performance of different 6D SLAM algorithms building a large urban outdoor map.

I. INTRODUCTION

Algorithms for solving the robotic simultaneous localization and mapping (SLAM) problem are a key scientific issue in mobile robotics research. Solutions to SLAM are of core importance in providing mobile robots with the ability to operate with real autonomy. SLAM algorithms integrate robot action and sensor readings and exploit the fact that previously mapped areas are recognized. Global optimization methods yield consistent maps. Nevertheless, these consistent maps might be incorrect and therefore ground truth experiments have to be made. This paper presents ground truth experiments using a novel empiricism.

Popular mapping algorithms work with 3DoF pose estimates, i.e., robot poses are represented by three degrees of freedom \( P = (x, y, \theta) \). For indoor environments this choice is appropriate, but a current trend for mapping outdoor environments are mapping algorithms that represent poses in 6DoF, i.e., 6D SLAM \[17\]. These algorithms consider the 6DoF pose \( V = (x, y, z, \theta_x, \theta_y, \theta_z) \) of the mobile robot with 3 position coordinates and roll, pitch and yaw angles. Robot motion and localization on natural surfaces must regard these 6 degrees of freedom. Recently, 3D mapping of large environments received much attention, \[4\], \[18\], \[24\]. A framework for benchmarking these large experiments is still missing.

This paper evaluates algorithms and methods for autonomous mapping. A mobile robot, equipped with a fast 3D scanner gages the environment, while it is steered through a large urban environment. The maps generated by online and offline algorithms are compared to odometry based, gyro based and GPS based pose estimates. Ground truth is provided by a Monte Carlo Localization (MCL) using accurate reference maps.

A. Ground Truth Experiments

In doing experiments with ground truth reference, researchers aim to measure the objective performance of a dedicated algorithm. Based on this benchmark it is possible to give an experimental prove of the effectiveness of a new algorithm. Furthermore measuring the performance of algorithms allows to optimize the algorithm and to compare it to other existing solutions.

Benchmarking is a common scientific instrument. A good example for successful performance measurement in computer science is the computer vision community. There are several projects that aim at providing image data bases to other researchers \[12\] \[23\]. These image databases are supplemented by ground truth images and algorithms that calculate performance metrics. In doing so, the community is able to make progress and to document its progress in fields like image segmentation and object recognition.

Unfortunately this kind of performance measurement is not widely spread in the robotics community. Even though there are several ways of comparing the performance of robotic algorithms and systems, one basic step is to provide experimental data and results to other research groups. Up to now this is only done by small projects \[14\] \[19\] or
individual researchers. Another way of comparing robotic systems are competitions like RoboCup [7], ELROB [8] or the Grand Challenge [5]. With this kind of competitions it is possible to measure the level of system integration and the engineering skills of a certain team, but it is not possible to measure the performance of a subsystem or a single algorithm.

Objective benchmarking of localization and mapping algorithms is only achieved by comparing of experimental results against reference data. The practical problem is the generation of this ground truth data. In computer vision, ground truth data is either available for synthetic images, or needs to be hand labeled. In case of mobile robot navigation one way of gathering ground truth data is the use of precise global positioning systems (RTK-GPS) [11]. Unfortunately, this data is only available in open outdoor environments and not for urban outdoor environments or indoor environments. Another possibility is to use complex external measurement setups.

Another benchmarking method for robotic algorithms comprises simulation. Realistic simulation enables researchers to perform experiments with defined conditions and to repeat these experiments. However, real life differs from simulation. Experiments, involving sophisticated sensors such as cameras or laser scanners can only be simulated up to a certain level of accuracy, e.g., capturing environments must regard surface properties such as material, local structures and reflexions. Therefore, using real robotic data sets is favored for benchmarking.

With this paper, we present a novel method of gathering ground truth data in indoor and urban outdoor environments. The procedure is making use of a highly accurate environment map (provided by the land registry office), a Monte Carlo Localization that matches sensor data against the reference map and a manual supervision step.

B. State of the Art in Metric Robotic Mapping

1) Planar Mapping: State of the art for metric maps are probabilistic methods, where the robot has probabilistic motion models and uncertain perception models. Through integration of these two distributions with a Bayes filter, e.g., Kalman or particle filter, it is possible to localize the robot. Mapping is often an extension to this estimation problem. Beside the robot pose, positions of landmarks are estimated. Closed loops, i.e., a second encounter of a previously visited area of the environment, play a special role here: Once detected, they enable the algorithms to bound the error by deforming the mapped area to yield a topologically consistent model. However, there is no guarantee for a correct model. Several strategies exist for solving SLAM. Thrun [21] surveys existing techniques, i.e., maximum likelihood estimation, expectation maximization, extended Kalman filter or (sparsely extended) information filter SLAM. FastSLAM [22] approximates the posterior probabilities, i.e., robot poses, by particles.

SLAM in well-defined, planar indoor environments is considered solved. In principle probabilistic methods are extendable to 6DoF. However, to our knowledge no reliable feature extraction mechanisms nor methods for reducing the computational cost of multihypothesis tracking procedures like FastSLAM (which grows exponentially with the degrees of freedom) have been published.

2) Mapping Environments in 3D: An emerging research topic is 6D SLAM, i.e., while mapping the robot pose is represented with six degree of freedom. In previous work, we used a 3D laser range finder in a stop-scan-match-go-process to create a 3D map of the environment by merging several 3D scan into one coordinate system [17], [20]. Similar experiments have been made by Newman et al. [16]. A current trend in laser based 6D SLAM is to overcome stop-and go fashion of scan acquisition by rotating or pitching the scanner while moving [4], [24], [25]. In the most recent work Pfaff et al. [18] employ two rotating SICK scanners for data acquisition, odometry, IMU and DGPS positioning, a variant of the iterative closest point (ICP) algorithm and a loop closing procedure to map large urban environments in 3D.

Feature-based 6D SLAM methods are investigated by Udo Frese, who adapted his fast treemap algorithm to six degrees of freedom [10]. Among the category of feature based 6D SLAM are the visual SLAM methods, i.e., the MonoSLAM system of Davison et al. [6].

The remainder of the paper is structured as follows: Next, we describe the sensor system for generating large 3D maps and the two pairs of evaluated mapping algorithms. In section III we present the MCL based benchmarking technique. Then we present results from an experiment consisting of 924 3D scans. Section VI concludes.

II. GENERATION OF LARGE URBAN 3D MAPS

A. 3D Range Sensor

The sensor that has been employed for the experiments is a fast 3D laser range scanner, developed at the Leibniz Universität Hannover (see Fig. 2). As there is no commercial 3D laser scanner available that meets the requirements of mobile robots, it is common practice to assemble 3D sensors out of standard 2D laser range sensors and additional servo drives.

The specialties of our RTS/ScanDrive are a number of optimizations that are made to allow fast scanning. One mechanical optimization is the slip ring connection for power and data. This connection allows continuous 360° scanning without the accelerations and high power consumption that are typical for panning systems. Even more important than the mechanical and electrical improvements is the precise synchronization between the 2D laser data, servo drive data and the wheel odometry. Having this good synchronization, it is possible to compensate systematic measurement errors and to measure accurate 3D point clouds even with a moving robot. Detailed descriptions of these 3D scanning methods and optimizations are published in [27].

Having these optimizations described above the limiting factor in bulding faster 3D laser scanner is the maximal
number of 13575 (75 × 181) points that can be measured with a SICK LMS 2xx sensor in one second. The only way of building faster SICK LMS 2xx based 3D scanners is the use of multiple 2D measurement devices [24]. For this reason we first present the RTS/ScanDriveDuo with this paper. This 3D scanner makes use of two SICK LMS 291 2D laser scanners. Thus the measurement time for 3D scans with 32580 points in 1.2 sec. Right: The mobile robot Erika.

![Image](https://example.com/image.png)

**Fig. 2.** Left: 3D laser range sensor RTS/ScanDriveDuo. Measuring full 3D scans with 32580 points in 1.2 sec. Right: The mobile robot Erika.

### B. The Mobile Robot Erika

The mobile service robot Erika is build out of the Modular Robotics Toolkit (MoRob-Kit). The over all size (LxWxH) of Erika is 95x60x120cm. With its differential drive motors it is possible to drive up to 1.6m/s in indoor and urban outdoor environments. The battery capacity is designed to supply the electric wheelchair motors, sensors and a 700MHz Embedded PC for at least 2 hours or 5km.

In addition to the 3D laser scanner the mobile robot is equipped with wheel odometry, a 3 axis gyroscope and a low-cost SiRF III GPS receiver. The measured data of the wheel odometry and the gyroscope are fused to result in the OdometryGyro that is used as the internal sensor for both MCL and SLAM. In contrast to the odometry sensor the GPS receiver that has got no influence on neither the MCL nor the SLAM results. It is only logged to have another laser independent reference.

### C. 6D SLAM with ICP based Scan Matching

We use the well-known Iterative Closest Points (ICP) algorithm [1] to calculate the transformation while the robot is acquiring a sequence of 3D scans. The ICP algorithm calculates iteratively the point correspondence. In each iteration step, the algorithm selects the closest points as correspondences and calculates the transformation \((R, t)\) for minimizing the equation

\[
E(R, t) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} ||m_i - (Rd_j + t)||^2, \quad (1)
\]

where \(N_m\) and \(N_d\) are the number of points in the model set \(M\) or data set \(D\), respectively and \(w_{i,j}\) are the weights for a point match. The weights are assigned as follows: \(w_{i,j} = 1\), if \(m_i\) is the closest point to \(d_j\) within a close limit, \(w_{i,j} = 0\) otherwise. The assumption is that in the last iteration the point correspondences are correct. In each iteration, the transformation is calculated by the quaternion based method of Horn [13].

To digitalize environments without occlusions, multiple 3D scans have to be registered. Consider a robot travelling along a path, and traversing \(n + 1\) 3D scan poses \(V_0, \ldots, V_n\).

A first straightforward method for aligning several 3D scans is to use a pairwise ICP, i.e., matching the scan taken from pose \(V_i\) against the scan from pose \(V_0\), matching the scan taken from \(V_2\) against the scan from pose \(V_1\), and so on. Here the model set \(M\) is formed the the 3D data from pose \(V_{i-1}\) and the data set \(D\) that of the pose \(V_i\) for all \(i\) in \(1, n\). A second plausible method is to form of all previously acquired 3D scans a so called metascan and match the last acquired one against this metascan. This method is called metascan ICP. Here, the model set \(M\) consists of the union of the 3D scans from the poses \(V_0, \ldots, V_{i-1}\) and the data set \(D\) that of pose \(V_i\), for all \(i\) in \(1, n\).

### D. 6D SLAM with Global Relaxation

Both, pairwise ICP and metascan ICP correct the robot pose estimates, but registration errors sum up. SLAM algorithms use loop closing to bound these errors. If two estimated robot poses \(V_i\) and \(V_j\) are close enough, i.e., their distance falls below a threshold (here: 5 meter) then we assume these scans overlap and are matchable. To a graph, initially containing the sequence of all poses \((V_0, V_1), (V_1, V_2), \ldots, (V_{n-1}, V_n)\), the edge \((V_i, V_j)\) is added. While processing the scans with pairwise ICP or metascan matching, we detect closed loops using this simple distance criterion. Once detected, a 6DoF graph optimization algorithm for global relaxation based on the method of Lu and Milios [15] is employed, namely Lu and Milios style SLAM (LUM). This is a variant of GraphSLAM. Details of the 6DoF optimization, i.e., how the matrices have to be filled, can be found in [2], thus we give only a brief overview here.

Given a network with \(n + 1\) nodes \(X_0, \ldots, X_n\) representing the poses \(V_0, \ldots, V_n\), and the directed edges \(D_{i,j}\), we aim at estimating all poses optimally to build a consistent map of the environment. For simplicity, the approximation that the measurement equation is linear is made, i.e.,

\[
D_{i,j} = X_i - X_j \quad (2)
\]

An error function is formed such that minimization results in improved pose estimations:

\[
W = \sum_{(i,j)} (D_{i,j} - D_{i,j})^T C^{-1}_{i,j} (D_{i,j} - D_{i,j}). \quad (3)
\]
where $\tilde{D}_{i,j} = D_{i,j} + \Delta D_{i,j}$ models random Gaussian noise added to the unknown exact pose $D_{i,j}$. This representation involves to resolve the non-linearities resulting from the additional roll and pitch angles by Taylor expansion. The covariance matrices $C_{i,j}$ describing the pose relations in the network are computed, based on the paired closest points. The error function eq. (3) has a quadratic form and is therefore solved in closed form by sparse Cholesky decomposition. The algorithm optimizes eq. (3) gradually by iterating the following three steps: First, for every network link the corresponding covariance is computed based on the point correspondencies of the scan matching. Then the error function (3) is minimized by solving a linear system of equations. In the third step, the local transformations are applied to the poses, resulting in improved pose estimates.

Using the global optimization, two more strategies have been implemented: In pairwise LUM, we use pairwise matching of scans for initially estimating the robot poses. After a loop has been closed, the global relaxation to all previously acquired scans is applied. In metascan LUM, every new scan is initially matched against all previously acquired scans. In both algorithms, global relaxation is started after a closed loop is detected. The relaxation considers all previously acquired scans.

E. Mapping Strategies

Fig. 3 depicts how the mapping strategies are interleaved. 6D SLAM is the result of a 6 DoF ICP algorithm combined with the extension of Lu/Milios Scan Matching to 6 DoF as global relaxation. The SLAM backend uses fast matrix computations exploiting the sparse structure of the corresponding SLAM graphs [3]. Using different path in Fig. 3 the different mapping strategies are created.

Animations of the four mapping strategies, pairwise ICP, metascan ICP, pairwise LUM, metascan LUM are given in the accompanying video and on the following web page: http://kos.informatik.uni-osnabrueck.de/download/6DSLAMbenchmarking. Note the maps presented in the video are rotated about 190°.

III. Benchmarking Technique

This paper introduces a new benchmarking technique for SLAM algorithms. The benchmark is based on the final SLAM results and a reference position that is obtained independently of the SLAM algorithm under test.

As highly accurate RTK-GPS receivers can not be used in urban outdoor environments, we present a technique that is based on surveyed maps as they can be obtained from the German land registry offices. The process of generating this ground truth reference positions can be divided into a Monte Carlo Localization step that matches the sensor data to the highly accurate map and a manual supervision step to validate the MCL results.

As the SLAM algorithm under test and the MCL algorithm use the same sensor data, the SLAM results and the reference positions are not completely independent. But on the other hand, global localization algorithms and incremental localization and mapping algorithms work differently. Incremental mapping algorithms like odometry and SLAM can suffer from accumulating errors and drift effects. However pure localization algorithms eliminate these errors by continuously matching to an accurate given map. For this reason the remaining error of the manually supervised reference position is at least an order of magnitude smaller than the discussed SLAM errors.

A. Reference Map

As part of their geo information system (GIS) the German land registration offices features surveyed data of all buildings within Germany. The information about these building is stored in vector format in the so called "Automatisierte
Liegenschaftskarte (ALK). The vector format contains lines that represent the outer walls of solid buildings. Each line is represented by two points with northing and easting coordinates in a Gauss-Krüger coordinate system. The upper error bound of all points stored in the ALK is specified to be 4 cm. Up to now there are no further details about doors, windows or balconies available.

B. Monte Carlo Localization

The Monte Carlo Localization (MCL) is a commonly used localization algorithm that is based on particle filtering [9]. As the theory of MCL is well understood we focus on the sensor model that is used to match the 3D sensor data to the 2D reference map with this paper.

The key problem of matching a 3D laser scan to a 2D map is solved by using a method called Virtual 2D Scans [25]. The method splits up into two steps. The first step reduces the number of points in the 3D point cloud. The reduction step is based on the assumption that the reference map presents plain vertical walls. For this reason all 3D measurement points that do not belong to plain vertical surfaces need to be removed (Fig. 4). A sequence of 3D segmentation and classification algorithms that is used to do this reduction in urban outdoor environments is described in [26]. By this means the ground floor, vegetation and small objects are removed from the 3D data. Measurement points on the outer walls of buildings and on other unmapped vertical obstacles remain.

Having this reduced 3D point cloud, the second step of the Virtual 2D Scan method is a parallel projection of the remaining 3D points onto the horizontal plane. After this projection the z coordinate contains no information and can be removed. By this means, the Virtual 2D Scan has got the same data format as a regular 2D scan. Thus it can be used as input data of a regular 2D MCL algorithm. To reduce the computational complexity of the successive MCL algorithm the remaining measurement points are randomly down sampled. Experimental results show that less than 100 measurement points are needed for sufficient localization. Thus the average 3D point cloud with about 30000 measurement points is reduced to a Virtual 2D Scan with only 100 point without losing information that is needed for localization in urban outdoor environments.

Due to the 2D nature of the reference map and the used 2D MCL algorithm it is only possible to estimate the 3DoF pose $P_{\text{REF}} = (x, y, \theta_z)$ of the robot. There is no reference information on the robots height $z$. Further more the roll and pitch components $\theta_x, \theta_y$ of the 6DoF robot pose can not be estimated with this 2D method. These angles need to be measured and compensated with a gyro unit before the generation of the Virtual 2D Scans.

C. Manual Supervision

Unlike MCL algorithms used in fully autonomous navigation the generation of reference positions needs manual supervision. Even though the human supervisor is not able to identify the absolute accuracy of the estimated MCL position, it is possible to check the conditions that are needed for proper operation. If all of these conditions are fulfilled the MCL algorithm is able to find the true position of the robot in global coordinates.

There are several conditions that need to be checked to attest proper operation:

- At first the sensor data needs to be checked for a sufficient number of landmarks. Namely, walls as they are given in the reference map. In case of an open area without landmarks in the surrounding of the robot, occluded landmarks or insufficient Virtual 2D Scans the MCL results only depend on odometry and are therefore not accurate.
- The second step is to supervise the numerical condition of the particle filter. As a particle filter only presents a sampled belief an efficient distribution of the finite number of particles is essential for correct operation. For this reason the human supervisor needs to make sure that enough particles are located around the true position. The estimated position can be corrupt if particles are located around more than one maximum or around wrong local maxima.
- Finally, the human supervisor can valuate the overall soundness of the localization and mapping results. For this reason it is necessary to display the reference map with overlaid sensor data. As the sensor data is transformed with the MCL results, fatal matching errors can be detected by the supervisor.

D. Benchmark Criteria

Up to this point the MCL positions and SLAM positions are given in different coordinate systems. The MCL positions are given in the global Gauss-Krüger coordinate system of the reference map and the SLAM positions are given in a local coordinate system that is centered in the robots start position. To be able to compare the positioning results it is necessary to transform the SLAM positions into the global coordinate system based on the known start position.

Having the trusted MCL reference $P_{\text{REF}}$ and the SLAM results $V_{\text{SLAM}}$ in the same coordinate system, it is possible
to calculate objective performance metrics based on position differences. The first metric based on the 2D Euclidean distance between the SLAM and MCL position

$$e_i = \sqrt{(x_i^{SLAM} - x_i^{REF})^2 + (y_i^{SLAM} - y_i^{REF})^2}. \quad (4)$$

The second metric is based on the difference between the SLAM and MCL orientation

$$e_{\theta,i} = |\theta_i^{SLAM} - \theta_i^{REF}|. \quad (5)$$

As the MCL position has got only 3DoF, the robots elevation, roll and pitch angle can not be tested.

To compare the performance of different SLAM algorithms on the same data set, it is possible to calculate scores like the standard deviation

$$\sigma = \sqrt{\frac{1}{n+1} \sum_{k=0}^{n} e_i^2}, \quad (6)$$

and the error maximum

$$e_{\text{max}} = \max e_i. \quad (7)$$

Of course these statistic tests can be done analogously on the orientation errors $e_{\theta,i}$ resulting in the scores ($\sigma_{\theta}$ and $e_{\theta,\text{max}}$).

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

The presented experiment has been carried out at the campus of the Leibniz Universität Hannover. The experimental robot platform that was used to collect the data was manually driven on the 1.242 km path closing a total of 5 small and large loops. On this path 924 full 3D scans have been collected at an average robot speed of 4 km/h and a maximum speed of 6 km/h. In addition to the 3D laser data wheel odometry and fused wheel/gyro odometry have been stored with a data rate of 10 Hz. And the position fixes of a low-cost GPS have been logged with 1 Hz.

B. Ground Truth Data

The section of the ALK that is used as the reference map contains 28 buildings represented by 413 line segments. To avoid huge coordinate numbers a constant offset of 5806400 m northing and 3548500 m easting is subtracted from all Gauss-Krüger coordinates. This offset corresponds to the position 52°23'58'' north, 9°42'41'' east in WGS84 coordinates.

The MCL reference positions are calculated online on the Pentium III 700 MHz processor included in the 3D sensor. The particle filter runs with 200 samples and a generous estimate of the sensor variance of 30cm. This estimate includes the sensor range error, errors from scanning while moving and map uncertainties. The localization results are plotted as a solid gray line in Fig. 1.

The result of the offline manual observation is that the MCL positions can be used as reference positions for 3D scan indexes 1 to 197 and 242 to 924. On the other hand positions corresponding to 3D scan indexes 198 to 241 can not be used as there are not enough landmarks visible to the 3D sensor (MCL error box in Fig. 1). Due to that particles diverge and the calculated position follows the drifting odometry. Starting with 3D scan 138 the Virtual 2D Scan contains new landmarks and thus the MCL converges quickly to the true position.

For that reason results from 3D scan indexes 198 to 241 are not considered in the following analysis.

C. Mapping Results

1) Mapping with Internal Sensors and GPS: Since all sensors are inaccurate the maps generated using internal sensors for pose estimation are of limited quality as has been demonstrated many times before. For odometry and the gyro based localization the error for orientation and postion are potentially unbounded. However, since paths usually contain left and right turns, these errors partially balance. The GPS shows problems close to buildings, where the orientation is poorly estimated and the position error reaches its maximal value. Fig. 5 shows the orientation errors of the internal sensors in comparison to ICP scan matching.
2) Mapping with ICP: Mapping with ICP was done using two different methods, namely pairwise ICP and metascan ICP. The latter method outperforms pairwise ICP since it considers all previously acquired 3D scans leading to slower error accumulation. Fig. 6 shows the scan matching errors in comparison to methods using explicit loop closure that are described next.

3) Mapping with ICP and Global Relaxation: The performance of the methods pairwise LUM, metascan LUM have also been evaluated. As expected, loop closing reduces the position error at the positions, where the loop is closed to approximately zero, e.g., Fig. 6 at scan index 100, where the first loop was closed and at the indices 300–400 and 600–700. At these locations, the Lu/Milios style SLAM methods outperform the pairwise ICP and metascan ICP methods. However, pairwise LUM, and metascan LUM may also fail, if the loop cannot be closed. This case occurs in our experiment in the final part of the trajectory, i.e., when the scan index is greater than 700 (cf. Fig. 6 and Fig. 9). This last loop was not detected by the threshold method described in section II-D.

Finally, Tab. I and II compare all localization/mapping methods. Fig. 7 shows the final map generated with metascan LUM. The left part contains the first 720 3D scans that have been matched correctly, whereas the right part contains all scans including the errors, due to the undetected loop. Fig. 8 shows a 3D view of the scene including two close-up views.

D. Computational Requirements

Of the compared mapping methods only the internal sensor based and the pairwise ICP are online capable. Pairwise ICP using an octree based point reduction and kd-tree search are performed in less than 1.2 sec. using standard computing hardware. In metascan ICP, mapping the computing time for closest point calculations increases with the number of scans; therefore, the scan matching time increases to 11.2 sec. for

![Fig. 9. 3D view of the problematic loop closure in Fig. 7 (right, yellow rectangle). Loop closing is not possible due to accumulated elevation errors.](image-url)
matching scan No. 920 with all previous ones, i.e., matching 32580 against 29 Mio. points.

Pairwise LUM and metascan LUM spend additional time on computing the point correspondences for scans represented by the nodes in the graph. Due to the iteration required by our GraphSLAM algorithm, both methods are not online capable [2]. The total map processing time was 207 min and 371 min, respectively. The largest portion of the computing time was spent by calculating closest points.

V. JUSTIFICATION OF THE RESULTS
To validate our experimental methodology, i.e. generating ground truth reference positions using MCL as described, we match the acquired 3D scans with a 3D map generated from the 2D reference map. For this the 2D map is extrapolated as 3D map, by cuboids representing the boundaries of the buildings. Fig. 10 shows the final map with the point clouds representing the buildings.

This 3D map which is a 2D map extended by some height
is used for comparison using the following three strategies:

1) The ICP algorithm is used to match every single 3D scan with the point cloud based on the 2D map. As it turns out, this method can only be applied to the first 200 scans, since the map does not cover the whole robot path. However, in comparison MCL successsfully deals with this problem by applying the motion model to the particles until scan matching is possible again. This method is referred to as \textit{map ICP (first part)}.

2) The ICP algorithm is used to match a 3D scan with the metascan consisting of the 3D points from the map and all previous acquired and registered 3D scans. The method will be called \textit{metascan map ICP}.

3) The previous method is used with the extension, that points classified as ground are not included, i.e., only the blue points are used for computing point correspondences and the transformation. Thus it is called \textit{metascan map ICP wo ground}. It is expected that this restriction results in better convergence of the ICP algorithm.

Fig. 11 and 12 compare the additional localization/mapping methods. Tab. III and IV gives quantitative results. It turns out that these justification methods are behave similar to MCL and produce comparable results, i.e., the MCL trajectory differs only by statistical nois from the trajectories produced by ICP scan matching using a 3D point cloud derived from the map.

VI. CONCLUSION AND FUTURE WORK

Benchmarking of algorithms and research in experimental methodology are topics that get more and more important in robotics. Thus this paper presents a novel evaluation method
for SLAM in urban outdoor environments. The evaluation is based on a comparison of the final SLAM results and ground truth reference positions. In our case these reference positions are generated with a manually supervised Monte Carlo Localization working on surveyed reference maps. Having this reference positions it is possible to calculate objective benchmark scores that can be used to improve and compare algorithms. This evaluation technique is demonstrated with experimental data and four different 6D SLAM strategies. The experiment that contains 924 full 3D scans on a 1.2 km path was carried out on the campus of the Leibniz Universität Hannover.

Needless to say that much work remains to be done. Future work will be done on two aspects: First, research in robotic benchmarking techniques needs to be emphasized. And second this ideas need to be spread out in the robotics community. To this end, we plan to cooperate with the Radish: The Robotics Data Set Repository and the OpenSLAM [19] project.

### REFERENCES


### TABLE III

<table>
<thead>
<tr>
<th>method</th>
<th>σ</th>
<th>θ_{max}</th>
</tr>
</thead>
<tbody>
<tr>
<td>map ICP (first part)</td>
<td>0.3</td>
<td>1.77</td>
</tr>
<tr>
<td>metascan map ICP</td>
<td>0.38</td>
<td>1.79</td>
</tr>
<tr>
<td>metascan map ICP wo ground</td>
<td>0.38</td>
<td>2.98</td>
</tr>
</tbody>
</table>

### TABLE IV

<table>
<thead>
<tr>
<th>method</th>
<th>σ</th>
<th>θ_{max}</th>
</tr>
</thead>
<tbody>
<tr>
<td>map ICP (first part)</td>
<td>1.24</td>
<td>4.00</td>
</tr>
<tr>
<td>metascan map ICP</td>
<td>1.40</td>
<td>6.37</td>
</tr>
<tr>
<td>metascan map ICP wo ground</td>
<td>1.38</td>
<td>6.32</td>
</tr>
</tbody>
</table>
The Jacobs Test Arena for Security, Safety, and Rescue Robotics (SSRR)

Andreas Birk, Kaustubh Pathak, Jann Poppinga, Sören Schwertfeger, Max Pfingsthorn and Heiko Bülow

Abstract—The Jacobs University Bremen features a special test site for mobile robots. It consists of two large arenas with test elements, which are particularly suited to evaluate the performance of systems operating in challenging domains like Security, Safety, and Rescue Robotics (SSRR). The site is one of only six worldwide, which has been established in close cooperation with the US National Institute of Standards and Technology (NIST). This paper presents the Jacobs arenas and a set of metrics for evaluating the mobility, sensing capability, and onboard intelligence of robots. The tests are illustrated by using a Jacobs rescue robot, which is equipped with state of the art sensors. The relative strengths and weaknesses of these sensors are evaluated in a variety of situations; many of them are typically encountered in SSRR applications.

I. INTRODUCTION

In addition to their well-established role as programmable tools for industrial automation, robots are increasingly used as autonomous intelligent systems in challenging environments that are not adapted for automation. These robots have to function without continuous supervision by a human operator. They have to be able to adapt to unforeseen circumstances. They must deal with various working environments where they encounter an enormous variety of different objects with various types of surface materials, shapes, sizes, and so on. One such application domain is the field of search and rescue robotics. As discussed in detail in [1], rescue robotics is an important domain that can serve as an important milestone on the road to truly autonomous systems.

Existing systems have demonstrated their usefulness in the field and there is a sound, growing market for them. Any bit of autonomous intelligent functionality added can improve their overall usefulness leading up to full autonomy in the long term. But though this line of reasoning is valid on the general level, it is much more difficult to assess the real benefit of a particular implementation of a concrete function. Any additional intelligent capability increases the complexity, requires space and payload capacities, consumes energy, and increases the price. Customers, end users and developers have hence an interest in being able to properly measure the benefits of a particular function and to compare the benefits to the costs.

Task based performance testing, just like observing a mouse in maze, has been proposed for this purpose [2]. In addition to the temporary test arenas used at field tests and competitions like RoboCup events, there are six permanent test sites based on the work by the NIST. The following list gives an overview of their locations and the years of opening:

- National Institute of Standards and Technology: Gaithersburg, USA (2000)
- Jacobs University Bremen1: Bremen, Germany (2004)

The performance evaluation aspects described in this paper are based on work at the test site at the Jacobs University Bremen. This test site consists of two parts as shown in figures 1 and 2. The first arena is available since spring 2004 at Jacobs [4]. It is based on a high-bay-racking system. This allows a large floor-space on a relatively small area. The first arena has a footprint of 5.60m by 4.70m and it is approximately 6m high. It has 3 main floors and several intermediate floors, which are interconnected. This testbed consists of three different levels, the so-called yellow, red, and the orange zone as motivated in detail in [5][3]. Yellow models an intact building structure with a normal office or home interior that has been mildly affected by a disaster. In the orange zone, the effects to the interior as well as to the building itself are much more severe. Last but not least, the red zone is a model of a pancake collapse with large amount of rubble and highly instable structures.

The second arena provides an additional collection of locomotion and sensing challenges that supplement the original arena. The second test arena is mainly organized in a flat, 2D fashion with a large floor area. It features test elements like maze-like structures, but also some local 3D obstacles like ramps, stairs and elevated floors. It also offers a large collection of random step fields, i.e., test elements that realistically simulate various forms of rubble. Both arenas are furthermore equipped with components to simulate various features of humans like according form, motion, sound or respiration.

II. LOCOMOTION

Mobility is the most fundamental aspect for SSRR. The environment presented in the test scenarios has a typical selection of challenges that are to be expected in real scenarios. The simplest scenario element is built as a U-shaped track

1The International University Bremen was re-named Jacobs University Bremen in spring 2007.
(see Fig. 3) in which the robot has to start in the area denoted by S and proceed into the end area E. These scenes are designed based on a square Random Step Field of size 1.35 meters. Random Step Fields are standardized test elements developed by the NIST, which very realistically model the challenges of rubble and uneven terrain. The robot is always placed such that it faces the correct direction ± 45 degrees. The points A, B, C and D depict where obstacles/difficulties might be placed. Usually the robot has to reach the end area E as fast as possible. The robot might touch the walls and other obstacles softly but not ram them hard, which would lead to an abortion of this run with a score of zero.

Fig. 3(b) shows the dimensions of the standard U-shaped track. The width was chosen such that two random step fields fit in the U-shape. The timing line is two meters from the upper border of the U-shape. The lower border is at least two meters from the timing line to allow some variations in the start positions of the robots. A rugged robot or rugbot [6], [7], the search and rescue robot of the Jacobs Robotics Group, is used throughout this paper to demonstrate some of the challenges the test element cover. Fig. 4(a) shows the Rugbot driving toward the end area of an empty U-shape.

The list of locomotion challenges that can be combined to generate a complete test course for a robot is as follows:

- **Empty U-Shape** An empty U-Shape is used to test the most basic teleop and autonomy functions. A system that fails here cannot be expected to continue elsewhere in the performance evaluation.

- **Littered Ground** The U-Shape is filled with loose paper and small sticks. The paper can easily obscure the field of view (FOV) of important sensors while wheeled robots could have problems moving over small sticks.

- **A Short Ramp** A two sided ramp is placed in the U-Shape such that the top on the line between the points A and D, filling the whole passage. The angle of the ramp is approx. 15 degrees while the height is 20 cm. The challenge for e.g. an autonomous robot is to stay at the outer borders of the scene in order to be able to enter the ramp, to climb the ramp, to go over the top, and find its way to the end area.
The task for the robot is to get from S to E U-shape without bumping into the walls or the specific obstacle which might be at A, B, C or D.

The dimensions of the U-shape.

Three random step fields in the U-shape.

Fig. 3. The standard test scene - U-shape.

- **Orange Random Step Fields**: Three orange random step fields are placed in the U-shaped track as shown in Fig. 3(c).

- **Red Random Step Fields**: Now three red random step fields are used. The reason for using this configuration is, that it is even more difficult to make an aimed turn on this difficult terrain than just driving straight. Autonomous robots are not expected to cope with this scenario in the near future.

- **A Steep Ramp**: A long and steep ramp as shown in the left image of Fig. 2 has to be crossed by driving the ramp up, turning and driving down again. Autonomous robots most likely will have to be guided into the right direction by placing walls around the entrance and turning corners. The difficulties in this scenario are the entering of the ramp as well as having enough friction to prevent sliding down.

- **Stairs**: Climbing up stairs is an important topic in mobile robot locomotion. See Fig. 4(b) for a Rugbot climbing the stairs in the rescue arena.

**III. MAPPING**

Maps are an important mission delivery in SSRR applications, e.g., to guide first responders to a victim that is found. Also, it is often desirable to have some information whether a certain area is likely to be void of persons. For this purpose, proper map based exploration is a must. Furthermore, mapping is a central issue from the research perspective. As mentioned in the introduction, there is a tremendous potential for fruitful interaction between application oriented work and basic research in the field of SSRR. Especially, this field and therefore also the related performance evaluations based on the NIST arenas can play a major role in the development of autonomous intelligent robots in general.

Map quality as performance evaluation criterion is composed of two main elements: **coverage** and **precision**. The coverage component simply refers to the size of the map, i.e., the amount of area of the environment that is represented in the map. Precision refers to the related accuracy of this representation. With respect to precision there are two fundamentally different approaches. First, there is the option to solely take the topology of the representation into account. Second, a metric error measure between the representation and ground truth can be used. Fig. 6 shows two maps generated by the same robot in parallel. The first one shown on top is based on the classic occupancy grid algorithm [8][9] while the second one shown below uses a state of the art algorithm for simultaneous mapping and localization (SLAM) [10]. The second SLAM algorithm gives a precise metric representation of the environment at the cost of high computational requirements. The first occupancy grid algorithm is less computationally demanding, but it does not give a proper metric representation. Nevertheless, it provides at least a good topological representation of the environment.
As a reproducible metric for a test scenario a Slalom course can be chosen. The main criteria are the requirements regarding the capabilities of the robot and the comparability of the designed test arena. Metrics for a slalom course are straightforward and a corresponding test arena can easily be rebuild. The demands on the robots odometry are high due to the many rotations during the course. These reasons finally lead to a comparability between different robot teams. Figure 5 shows the slalom test element at Jacobs. In figure 6 two different maps generated by the Rugbot are shown.

![Fig. 5. Slalom test with the robot at the start(left) and the robot traversing the corridor(right). Small sticks are scattered on the way to irritate the odometry of the robot.](image)

**IV. PERCEPTION**

Rescue scenarios and therefore also our rescue arena pose different challenges to the kind of sensors usually found on a mobile robot. A set of tests, most of them placed in the U-shaped track, have been developed to measure how well the robot supports a human operator coping with these difficulties. Even more challenging is to pass these tests in autonomous mode. To illustrate the problems these challenges have been explored with a Rugbot (Fig. 7). As can be seen from the figure, it has a wide array of sensors, whose output can be seen in figure 8:

- a 3D time-of-flight (TOF) camera (SR: SwissRanger SR-3000 from CSEM),
- a stereo camera (STOC from Videre),
- a webcam,
- a pan-tilt-zoom-camera (from Panasonic), and
- two laser range finders (LRF) (both URG from Hokuyo).

**A. Dark Scene**

In the dark scene the U-shape is covered such that normal cameras provide no information. Points are being awarded if the robot traverses from the start to the end without bumping the arena, which indicates that the environment had been perceived in one form or the other. To cope with this challenge the robot should be equipped with a light or use active light sensors such as laser range finders. In the example given in Fig. 9 an additional heat source (victim) and a box have been placed in the cave. The IR image identifies the heat source. The LRF and the SR determine the spatial structure whereas the optical sensors webcam and Stereo receive only poor images.

**B. Glass**

In this scenario, a part of the middle-wall near the point A and the back wall between B and C are replaced by a transparent material (plexi-glass). Many sensors have difficulties with this material, as can be seen in Fig. 10. The webcam is not able to detect the obstacle except for the case when light or objects are reflected. A benefit is the recognition of movement within the image by what a recognition of victims giving signs is possible. The LRF returns error beams (which signify free space). The TOF camera, which uses near IR light, does not detect the glass plate, only the reflection of itsillumination unit. The far IR light as detected with the IR camera is absorbed by the glass such that victims behind glass can not easily be spotted this way. Echo-sound sensors, which have not been tested with this robot, are likely to provide the most reliable data with this scene.

**C. Mirror**

The mirror scenario is quite similar to the one above, except that the glass is replaced by a mirror. As could be expected, the result (Fig. 11) are similar to the ones with the plexiglass. Visible and near IR light is reflected, far IR light is not. For sensors, whose output is processed, this can be an advantage as well as a disadvantage: if a mirror is sensed unknowingly, it will produce faulty data. If, however, a mirror is used on purpose, it can be used to look around the corner or to widen the FOV (like in an omni-cam). The images in Fig. 12 show that this is also possible for the 3D sensors. Again echo-sounders might be able to sense mirrors without problems.

**D. Absorptions**

In this scene black material is used to absorb the beams of typical laser range finders. The back wall from B to C
Fig. 6. The results of two different mapping algorithms running on the same robot, namely once an evidence grid (left) and once a state of the art SLAM algorithm (right). Being able to use either of the two approaches allows to trade processing speed for precision.

![Webcam image](image1)
(a) Webcam image

![Pan-tilt-zoom camera](image2)
(b) Pan-tilt-zoom camera

![Pan-tilt-zoom camera, zoomed in](image3)
(c) Pan-tilt-zoom camera, zoomed in

![Pan-tilt-zoom camera, zoomed in more](image4)
(d) Pan-tilt-zoom camera, zoomed in more

![Pan-tilt-zoom camera, maximum zoom](image5)
(e) Pan-tilt-zoom camera, maximum zoom

![Pan-tilt-zoom camera, panned and tilted to show something else](image6)
(f) Pan-tilt-zoom camera, panned and tilted to show something else

![TOF camera, distance image](image7)
(g) TOF camera, distance image

![TOF camera, intensity image](image8)
(h) TOF camera, intensity image

![Stereo camera, disparity image](image9)
(i) Stereo camera, disparity image

![Stereo camera, greyscale image](image10)
(j) Stereo camera, greyscale image

![Laser range finder](image11)
(k) Laser range finder

![Infrared camera](image12)
(l) Infrared camera

Fig. 8. The output of all sensors in a scene which does not produce any errors as well as parts of the wall near A are covered with this material. Fig. 13 shows a different scene, where the robot is standing in front of a ramp covered with said material. A unique strength of the TOF camera can be seen there. Both the LRF and the stereo camera fail to deliver correct measurement. For the stereo camera, this is because it has difficulties identifying features its two images due to the repetitive pattern of the cover. For the LRF on the other hand, it is due to the absorptions of the material. This weakness of the 2D case would, of course, also carry over to a possible 3D extension of the LRF.

E. Featureless Scene

Robots relying on stereo cameras will have difficulties with featureless scenes. Those are tested by covering the walls with a plane, white, featureless material. Fig. 14 shows this specific weakness of the stereo camera in another scene: it hardly detects any distances. The TOF camera in contrast has no such weakness. It has just an error at the very edge of the ramp caused by the tape applied there.
Fig. 9. Sensor Data from the cave. The darkness is a challenge to all the sensors which depend on the presence of visible light.

F. Range

Two additional tests are used to score the range of the sensors used. These scores are given for the closest distance to an obstacle where the robot still receives correct data and the farthest distance.

In Fig. 15, output is reproduced from sensors being very close to an obstacle, with two different distances. While the LRF can handle both situations well, the stereo camera already fails in the first set-up and the TOF cam fails in the second set-up. For the stereo cam, the inability to determine distance is due to the large difference in the images of the two lenses. The TOF camera, on the other hand, suffers from its active sensing. The error seen in Fig. IV-F is caused by excessive brightness of the ambient light. This phenomenon can also occur shortly in other situations (such as the first scene), where the camera is still able to automatically adjust the brightness after a short period of time.

In order to calculate the score for each sensor the minium
(a) Webcam image

(b) TOF camera – In the right image, the robot is clearly visible. But as this the distance image where color signifies distance, it should not. All parts of it should have roughly the same color. Also, none of the distances in the mirror are correct. The white/black spot is due to the reflection of the camera’s illumination unit.

(c) Stereo camera – Here, the distances in the mirror appear correct. Yet the mirror is not recognized as such, it looks like a square hole in the wall.

Fig. 11. Sensor Data with errors caused by a mirror

(a) Webcam – the obstacle around the corner is visible in the mirror.

(b) LRF – The two long groups of beams roughly between 11 and 12 o’clock are reflected by the mirror. The shorter beams in between detect the pole which is also visible in the mirror (Subfig. IV-C).

(c) TOF camera – distances are correctly measured.

(d) Stereo camera – the pole is visible by its edges.

Fig. 12. A mirror can be used to look around a corner, but only if it is recognized as such. For a computer, however, this is very difficult. It is more likely to mistake a mirror for a kind of window.

(a) Webcam – A ramp covered with plastic.

(b) LRF – The material absorbs the laser, causing the LRF to report free space in front.

(c) Stereo camera – The repetitive pattern of the material confuses the stereo camera such that only the edges are correctly measured.

(d) TOF camera – Only the SwissRanger outputs a gradient.

Fig. 13. Sensor Data from scenes involving absorptions
distance is determined. This average distance over all sensors is then used in the following formula:

\[
\text{Score} = (20\text{cm} - \min(\text{dist}, 20\text{cm})) \times 20. 
\]

This leads to a maximum score of 40 and a minimum score of zero at an average distance of 20 or more centimeters.

We also score the maximum usable range of a sensor. Some sensors have considerable difficulties with large distances. While the LRF reliably reports “error beams” for distances beyond its maximum of 4m, the TOF and the stereo camera produce errors, most notably the TOF camera. As it measures distances indirectly by measuring the phase shift of the emitted modulated light, the distance measurement wraps around after 7.5 m. For examples, refer to Fig. 16.

The score is calculated using the average maximum distance of all sensors as follows:

\[
\text{Score} = 40 \times \frac{\min(\text{dist}, 20m)}{20m} 
\]

Again the maximum possible score at 20m distance is 40 and the minimum score is zero.

**G. Perception performance of the Rugbot**

To illustrate the performance of the Rugbot a typical rescue scene is shown in Fig. 17: a human. This is, of course, the raison d'être for the IR camera being onboard. (Fig. IV-G). The other sensors show different performances. The LRF does generally well, but again (see Sec. IV-D) has problems with the black material. The TOF camera delivers a dense image with errors at the tape. As this scene has too few features for the stereo camera, it can only give distance values for the edges. Additionally, it is confused by the repetitive pattern on the right edge of the image.

In Fig. 8 (page 5), an average scene for rescue robotics is shown: A hallway with a ramp and victim at a distance. This set-up poses no difficulties for any of the sensors. The geometry is correctly captured by the distance sensors, just the far end of the LRF is out of range and produces error beams between 12 and 1 o’clock. In the swiss ranger distance image even the shape of the far away box is captured. The out-of-range measurements in the top right corner are dropped. The stereo camera provides a relatively dense image with correct distances. The cameras deliver good images, especially the zoomed in picture is impressive, and in the image of the IR camera, even the relatively distant heat source is clearly visible.

To round off this summary, a collection of images taken by the Rugbot is shown in Fig. 18, where one of the sensors reported wrong information with respect to the ground truth.

Table I shows a summary of the sensor’s performance in the different scenes. A minus (-) means that the sensor performed very poorly, a zero (0) indicates that the sensors returned mostly correct readings while a plus (+) states that the sensor was not affected by the challenge at all. Regarding perception, one can say that on using sensor fusion, the Rugbot has enough sensor data to cope with most of the challenges quite well.

**V. TESTING ROBOT AUTONOMY**

Due to the high difficulty level of the test arenas, autonomous operation of the robot is usually carried out in a sub-arena which is much simpler for locomotion. In particular, stairs and random step fields are currently not used, although ramps (pitch as well as roll) are used. The main criteria for evaluating autonomy are two-fold: (i) decision making and perception using sensors, and (ii) local and global motion planning. These are discussed in more detail next.
1) **Decision making based on sensor perception:** An autonomously moving robot has to make informed decisions such as in the following situations:

1) **Human detection:** The two common ways a human like a victim or an intruder can be detected are by using IR camera images to capture body heat, and through motion detection using camera images for a static robot. For the latter case, if the robot is equipped with more sophisticated algorithms to detect motion while the robot itself is moving, it gets assigned extra points. Human voice or whistle recognition or any other non-traditional techniques also accrue extra points.

2) **Obstacle detection using sensor-fusion:** In view of the pitfalls encountered by various sensors in different scenarios as described in Sec. IV, the robot should be able to fuse different sensor data, analyze them, and come up with the right identification of the obstacle type. The U-Shaped track along with obstacles of type described in Sec. IV can be used. Another ability is to distinguish a climbable ramp from an obstacle which is insurmountable and hence is to be avoided. Extra points are also awarded if the robot is able to avoid falling down ditches or table tops.

2) **Local and Global Planning:** Local planning involves a reactive approach to obstacle avoidance. The reactive
VI. MAPPING AS GENERAL PERFORMANCE CRITERION

One disadvantage of purely scene based performance evaluation is, that it only covers a part of the problems robots have to cope with, namely locomotion, mapping, perception and autonomy. For successfully completing real SSRR missions the following aspects are of importance, too.

The mobile robots fielded have to be robust enough to succeed in the difficult environment. Their energy source, which are most often batteries, has to last for the typical mission time which is often more than two hours. Another aspect is the communication range which can be quite limited if wireless systems are used. Multi-robot systems and how they coordinate their actions is another increasingly important topic as well as active rescue systems which not only observe but also help by, for example, enabling communication between the victim and the rescue personell or by providing water or maybe pain relief. Also the interface between the robot and the operator (Graphical User Interface, GUI; Joysticks; etc.) as well as between the operator and the rescue personell (victim reports; maps; PDA; etc.) is of great importance for the success of the mission.

Some of these aspects (locomotion, perception, mapping, autonomy, robustness, energy, multi-robots systems) can very well be measured if a complete test arena is available. This arena could be temporally one, for example during RoboCup, or a permanent one like the one at Jacobs University. A number of competitors will try to map as much of the arena as possible in the given time. The only criterion needed to measure the performance can then be the coverage and
Whenever possible, an exact metric measure of precision like the mean squared error between the generated map and ground truth is desirable. But when the ground truth is not readily available or the generated map is too imprecise for a meaningful evaluation, a topology match can be used for precision. In doing so, the number of nodes and vertices in the largest common sub-graph of the topological map and an accurate environment representation can be used. In this case, the measure includes coverage at the same time. Mapping as an evaluation criterion automatically reflects robot capabilities in other aspects like locomotion and sensors. Good locomotion implies better coverage and hence a more extensive map of the area. The sensing and perception capabilities of the system are obviously reflected in any map. Some concrete state of the art sensors and related challenges are discussed in more detail later on in this paper. Directly related to the obstacle sensing is, of course, the task of mapping itself, which is an important research issue as well as mission delivery in search and rescue missions. In addition to the sensor related perception challenges, state of the art mapping algorithms differ significantly in their precision but also computational requirements. An overview of current approaches is given in [11]. Several techniques have been developed to solve the so-called simultaneous localization and mapping problem (SLAM), e.g., by using Kalman filter based approaches [12] or expectation maximization techniques [13].

Note that the majority of work on mapping is focused on 2D maps. There is an increasing amount of research on 3D mapping [14], [15], [16], [17] and, as shown in detail later on in this paper, mapping in the arenas requires 3D perception. From the practical viewpoint of an end user perspective as well as with respect to performance evaluation, 2D maps are desirable as final delivery, either by using 3D perception in 2D mapping or by projecting 3D structures onto 2D planes.

Map quality can also be used to assess the autonomous capabilities of a robot. The map coverage is directly linked to the exploration capacities [18], [19]. This includes navigation skills [20] and intelligent decision making to for example find safe terrain for driving [21], [22], [23], [24], [25], [26]. In general, maps play a core role for almost any algorithm for goal-oriented autonomous behaviors [27], [28], [29]. This holds also with respect to cooperation between intelligent robots and humans [30].

Finally, multi-robot cooperation is an important aspect for the evaluation of autonomous intelligent systems. Map quality is also here a reliable indicator. The better the robots cooperate, the larger the amount of coverage by the joined map [15], [31], [32], [33], [34], [35]. The maps again form the main basis for further algorithms like exploration, which in turn are improving the skills of the robots as reflected in even higher amounts of coverage. The multi-robot cooperation can also be used to increase robustness, which is then reflected in the precision of the maps [36], [37].

VII. Conclusion

The paper presented the Jacobs Test Site for Security, Safety, and Rescue Robotics (SSRR). The site consists of two arenas with test elements that can be used to evaluate mobile robot performance in a repeatable and comparable manner. The performance assessment covers aspects from locomotion, perception, mapping and autonomy. A Jacobs rescue robot was used to demonstrate the usage of the test elements.

Acknowledgments

The authors gratefully acknowledge the financial support of Deutsche Forschungsgemeinschaft (DFG) for their research.

Please note the name-change of our institution. The Swiss Jacobs Foundation invests 200 Million Euro in International University Bremen (IUB) over a five-year period starting from 2007. To date this is the largest donation ever given in Europe by a private foundation to a science institution. In appreciation of the benefactors and to further promote the university’s unique profile in higher education and research, the boards of IUB have decided to change the university’s name to Jacobs University Bremen (Jacobs). Hence the two different names and abbreviations for the same institution may be found in this paper, especially in the references to previously published material.

References


Towards Quantitative Comparisons of Robot Algorithms: Experiences with SLAM in Simulation and Real World Systems

Benjamin Balaguer, Stefano Carpin
School of Engineering
University of California, Merced, USA
E-mail: \{scarpin,bbalaguer\}@ucmerced.edu

Stephen Balakirsky
National Institute of Standards and Technology
E-mail: stephen@nist.gov

Abstract— Autonomous robotics has been plagued by the lack of quantitative comparisons between different solutions for the same problem. The situation arose due to a lack of theoretical background, recognized benchmarks, and the existence of a culture that is not oriented towards the free sharing of ready-to-use code for scientific research. In this paper we leverage a recent paradigm shift, and contrast different algorithms for Simultaneous Localization And Mapping (SLAM) readily available to the scientific community. In particular, we have run the same algorithms in two different settings. The first one is based on a P3AT robot operating inside a large building hosting office space and research labs. The second scenario is a virtual replication of the identical floor plan, implemented inside the USARSim simulation environment. In other words, the simulated scenario features the exact models of the environment and robot. The experimental setup offers a matrix where weaknesses and strengths of different SLAM algorithms can be contrasted in real and virtual environments, also outlining the degree to which the simulated results can be extrapolated to measure or predict real world systems performance. We conclude that the availability of open source algorithm implementations, data sets, and simulation environments is the key to promote accelerated research in autonomous robotics. In particular, it appears that available SLAM implementations are robust and easy to use for environments like those used in our experiments, and therefore research efforts should be accordingly re-modulated.

I. INTRODUCTION

One of the cornerstones of the scientific method is repeatability. Experimental tests confirming or disproving a certain theory should be carried out by different researchers and lead to the same conclusions, within the defined error bounds, provided that the operative conditions are the same. Robotics has not yet enjoyed such a rigorous approach, a fact that can be explained by a multitude of reasons. Robots are complicated systems composed of many interacting units, each of them characterized by its own behaviors and errors. Robots’ observed behaviors do not only depend on the software and hardware, but also on the surrounding environment. Evidently, even if two researchers buy the same robot and perform the same experiment with the same software, they are likely to observe very different findings. Another problem that has not helped in making the situation better, but that finds roots in the same issues, is the lack of a widely shared base of reusable code that is maintained and exploited by different research groups. While it is true that certain middle-ware softwares are enjoying significant popularity [1] [2], they are basically interfaces to gain portable access methods to different sensors and actuators. Code exchange for algorithms solving general purpose tasks is still a fairly rare occurrence. All of the above has contributed to a situation where quantitative comparisons between different approaches is still missing. As a consequence, it is still too often observed that when a new project is started, certain tasks are coded again from scratch, rather than relying on existing libraries. The situation described clearly is a detriment to the development of more capable robots. Fortunately, in recent years a shift has been observed, and the direction seem to be changing. Among these events we list the following:

- the commercial success of certain robotic platforms has created a situation where many research labs use the same robots
- the establishment of on-line repositories of sensor data, like the Robotics Data Set Repository [3], allow different groups to execute different algorithms on exactly the same data set, thus outlining strengths and weaknesses
- the open-source approach is starting to become more accepted in the robotics community, with the creation of repositories of code for similar purpose [4]

In this paper we propose to start a systematic investigation and comparison of different algorithms for the well known simultaneous localization and mapping problem (SLAM). In particular, we present a juxtaposition between real world validation and experimental runs within the high fidelity USARSim simulator [5]. The use of a simulator is particularly appealing for quantitative comparisons of SLAM algorithms because it allows the generation of huge data sets without investing too many resources. Moreover, the simulator is highly configurable and allows one to specify different noise levels for the various sensors, thus permitting a careful evaluation of robustness with respect to noise sources. The paper is organized as follows. In section II we provide an overview of the USARSim software, while in section III we illustrate and contrast three SLAM algorithms. Section IV reports on the experimental setup that was implemented and the results that were produced. Finally, conclusions are offered in section V.
II. THE USARSim FRAMEWORK

The current version of Urban Search and Rescue Simulation (USARSim) [6] is based on the UnrealEngine21 game engine that was released by Epic Games as part of Unreal Tournament 2004. The engine may be inexpensively obtained by purchasing the Unreal Tournament 2004 game. The USARSim extensions may then be freely downloaded from [7]. The engine handles most of the basic mechanics of simulation and includes modules for handling input, output (3D rendering, 2D drawing, and sound), networking, physics, and dynamics. USARSim uses these features to provide controllable camera views and the ability to operate multiple robots. In addition to the simulation, a sophisticated graphical development environment and a variety of specialized tools are provided with the purchase of Unreal Tournament.

The USARSim framework builds on the Unreal game engine and consists of:

- standards that dictate how agent/game engine interaction is to occur,
- modifications to the game engine that permit this interaction
- an Application Programmer’s Interface (API) that def
- 3-D immersive test environments
- models of several commercial and laboratory robots and effectors
- models of commonly used robotic sensors

While there exists quite a few robotic simulators, USARSim was chosen for many different reasons, the most important of which being its accuracy. Indeed, a lot of time and research is spent every year improving the robotic platforms, authenticating the sensors, building additional robots, sensors, and environments, and validating the physics engine. More specifically, [8] [9] [10] [11] [12] provide details about USARSim validation both quantitatively and qualitatively. Additionally, USARSim provides the same robotic interface as the real P3AT, allowing researchers to run two robots, one in USARSim and one in the real world, with a single input (e.g. a joystick).

A simple but effective command-and-message interface is used to interact with the USARSim robots: string commands are sent to the robot and string messages are sent by the robot. The USARSim interaction standards consist of items such as robot coordinate frame definitions and unit declarations while the API specifies the command vocabulary for robot/sensor control and feedback. Both of these items have become the de facto standard interfaces for use in the RoboCup Rescue Virtual Competition which utilizes USARSim to provide an annual Urban Search and Rescue competition. In 2007 this competition had participation from teams representing 5 countries.

Both laboratory and commercial vehicles with different mobile platforms (skid-steered, Ackerman-steered, omni drive, legged, humanoid, nautical, and aerial) are incorporated within USARSim. Additionally, a set of robotic arms and plan-tilt mechanisms can effortlessly be mounted and utilized on any robot. Figure 1 shows a small collection of some of the robots that USARSim has to offer. The list of available sensors and effectors is also quite extensive and includes range scanners, sonars, cameras, grippers, RFID tags, and INS sensors.

Fig. 1. A subset of available robots in the current version of USARSim.

Highly realistic environments are also provided with the USARSim release, ranging from simple planar mazes to multi-level collapsed structures. The environments encompass challenging robotic problems that cover many areas of research including mapping, planning, mobility, cooperation, communication, image-processing, and victim detection. Furthermore, the environments accommodate all the USARSim robotic platforms by providing indoor buildings, urban roads and highways, lakes and rivers, and large flying spaces. Example indoor and outdoor environments may be seen in Figure 2. In addition, an editor delivered for free with the game engine and the ability to import models simplifies the creation of worlds.

It is worthwhile to note that USARSim does not supply a robot controller. In other words, USARSim simply provides a well-defined interface to communicate with the robots and it is the researcher’s responsibility to appropriately use the interface to achieve desired results. Researchers do not have to write a controller from scratch, however, since several open source controllers may be freely downloaded. These include the community developed Mobility Open Architecture Simulation and Tools (MOAST) controller [13], the player middle-ware [1], and any of the winning controllers from previous RoboCup competitions. Winning controllers, from RoboCup 2006, may be found on the robocuprescue wiki [14]. A description of the winning algorithms may be found in [15].

1Certain commercial software and tools are identified in this paper in order to explain our research. Such identification does not imply recommendation or endorsement by the authors, nor does it imply that the software tools identified are necessarily the best available for the purpose.
III. SLAM ALGORITHMS

In this section we briefly describe the three algorithms that were compared. We selected three of the seven packages currently available on the OpenSlam website. The selection was mainly driven by the desire to compare algorithms with similar characteristics in terms of requested input data, rather than by the desire to perform a comprehensive comparison.

A. GMapping

The GMapping algorithm produces a grid map and takes a particle filter approach [16] [17]. In particular, each particle is associated with a possible map. The main challenge for the algorithm, therefore, is to reduce the number of particles, because of the significant overhead associated with each particle. The algorithm uses a so-called Rao-Blackwellized filter and its major contribution is in the definition proposal distributions and resampling techniques that allow one to decrease the number of particles without incurring in problems related to undersampling.

The GMapping implementation available on OpenSlam is coded in C++ and processes data logs encoded using the Carmen log format [18]. Basically, the algorithm requires time-stamped odometry (pose, translational and rotational velocities) and time-stamped readings from the SICK laser.

B. GridSlam

GridSlam also uses a Rao-Blackwellized filter, and is particularly aimed to mapping environments with loops, a problem known to be challenging [19]. GridSlam also tries to decrease the number of particles used in the filter, but takes a different approach from GMapping. In GridSlam, a model of the residual error from scan registration is learned on the fly and used to contract the number of particles.

The GridSlam implementation is also coded in C++ and, similarly to GMapping, processes data provided in the Carmen log format.

C. DPSlam

DPSlam also uses a particle filter to estimate the robot’s pose and maps [20]. However, unlike the former approaches that aim to carry along a restricted set of candidate maps, DPSlam exploits a peculiar map representation that allows one to track a huge set of candidates (thousands, according to the authors).

DPSlam is also implemented in C++ and requires the same data as the former algorithms (i.e. odometry and laser scans) although not encoded in the Carmen log format.

IV. EXPERIMENTAL SETUP AND RESULTS

The experimental setup aims to not only compare different SLAM algorithms, but also assess the fidelity of the simulation engine. In fact, if we are able to show that results extracted in the simulation environment can be safely extrapolated to real world scenarios, we have then installed a very powerful tool to generate a massive amount of test data with minimal effort. The methodology developed to conduct this twofold evaluation will be described shortly.

For real world validation, we use a P3AT platform. The robot is equipped with odometry sensors and a SICK PLS range finder. A wireless-capable laptop is mounted on the robot, and the robot is controlled using the Player middleware [1]. The robot used in simulation is the corresponding P3AT model available in USARSim. The simulated robot is also controlled using Player. Data collection for the real robot took place in the hallway of the School of Engineering of UC Merced. For the simulated experiments, we developed a model of the same building, using the original blue prints provided by the architects. Figure 3 shows matching screenshots, in simulation and the real world, of the robots collecting data.
labs, especially for experiments aiming at assessing the simulation accuracy. The reason for this constraint stems from the lack of detailed footprints of lab and office furniture. Data collection was performed in parallel. A control application gets input from a user via a joystick and then sends the same commands, i.e., rotation and translation speeds, to the two robots. The user does not directly see any of the robots, but rather controls them by observing the output coming from the SICK sensor equipped on the real robot. The described approach requires careful tuning of the robot model in simulation in order to match the performance of the real platform.

We preliminary compared the performance of the various algorithms while processing data coming from the real robot and from the simulation. Figures 4 and 5 show the output of the GMapping algorithm for data collected by the real robot and the simulator, respectively. Figures 6 and 7 show the same results for the GridSlam algorithm. Finally, figures 8 and 9 show the same results produced by the DPSlam algorithm.

It is important to stress that, in this set of tests, we did not strive to find the best fine tuning for the algorithms, but rather to assess the similarity between results produced in simulation and with the real robot. A few observations can be made from the figures:

- There is a good correspondence between maps produced in simulation and in reality by the GMapping algorithm. In both cases the produced map and the tracked path basically agree with ground truth.
- For the considered dataset, the DPSlam algorithm seems to fail both for real and simulated data. This does not mean that DPSlam is not working properly in general, but rather that both specific set of simulated and real
Fig. 8. A map produced by the DPSlam algorithm on a data set produced by the real P3AT robot.

Fig. 9. A map produced by the DPSlam algorithm on a data set produced by a simulated P3AT robot.

Fig. 10. A map produced by the GMapping algorithm on a data set produced by a simulated P3AT robot.

Fig. 11. A map produced by the GridSlam algorithm on a data set produced by a simulated P3AT robot.

Fig. 12. A map produced by the GridSlam algorithm on a data set produced by a simulated P3AT robot.

It is understood that Gaussian noise is highly suboptimal when it comes to reproduce data coming from odometry, a fact that needs to be better addressed within the USARSim framework. We then executed the three SLAM algorithms on a data set produced by a simulated robot whose SICK laser was affected by an additive noise of intensity 5%. This means that every value $d_i$ returned by the sensor is altered accordingly to the following formula

$$d'_i = d_i (1 + 0.05x)$$

where $x$ is a random variable with uniform distribution over the interval $[-1, 1]$. Produced maps are illustrated in figures 10, 11 and 12.

The above results illustrate that there is a reasonable correspondence between results produced with simulated and real world data. Such correspondence is fairly strong for GMapping and DPSlam (in terms of success or failure) but less evident for GridSlam.

The next set of tests aims to measure the robustness of the three algorithms with respect to signal noise. Previously illustrated runs were obtained under the following simulated conditions. Readings from the SICK laser were affected by an additive noise with intensity 0.1% while the odometry was affected by Gaussian noise with 0 mean and 0.1 covariance.

The final set of tests was produced in a similar setting, with an additive noise of 10%. Resulting maps are illustrated in figures 13, 14 and 15.
V. CONCLUSIONS

Some general conclusions can be drawn. First, the use of the USARSim framework in order to compare different SLAM algorithms appears appropriate. Indeed, results obtained in simulation nicely translates to real robots. The performance of the simulated SICK laser is comparable to the real one, and an additive noise of 0.1% seems a reasonable choice. It appears necessary to develop a better model for odometry noise in order to accommodate the incremental nature of this disturbance. The three algorithms seem to be equally and reasonably robust to noise in the SICK laser. A level of 5% noise that visually appears much larger than anything observed in real world systems can still be dealt with by the algorithms. Both GMapping and DPSlam suffer from a lack of accuracy when noise is increased to 10%, but this is a level that is hardly ever observed in reality.

Comparing the different algorithms, GMapping emerged to be the more stable one in terms of performance, seldom incurring in severe problems of map consistency. Along a different line for comparisons, DPSlam required the most time when processing batch data logs offline and, as acknowledged by the authors, is very demanding in terms of memory. It is important to note that we have not fine-tuned the algorithms, resulting in a possibly unsatisfactory set of parameters. Different sets of parameters could have easily produced different results. However, if the robotics community aims to a wide sharing of open source algorithms, the availability of easy to use and tune algorithms is a must.

An aspect that still seems to be underconsidered is the availability of well defined metrics for mapping algorithms to, for example, quantitatively measure the correlation between a produced map and the corresponding ground truth. Visual inspection still seems to be the most widely used means of performing such evaluation, but a more rigorous approach is needed. Tools coming from the field of computer vision, related to image similarity, could be useful but have
yet to enjoy significant popularity.

REFERENCES

Reliability Testing for Embodied Autonomous Systems

L. F. Gunderson and J. P. Gunderson

Abstract — Autonomous systems are intended to function effectively in dynamic and uncertain environments. Unfortunately, traditional testing methodologies are grounded in the assumptions of static environments and deterministic analysis. As a result, these methodologies fail to test the very characteristics that are critical for the deployment of autonomous systems. At the same time, there is more and more demand to provide quantitative benchmarks for autonomous systems.

New agile development methodologies, such as eXtreme programming (XP), are becoming standard. XP, in particular, relies on constant automated testing to provide both the developers and the customers with some level of confidence that the systems are performing correctly and reliably. However, XP is not designed to handle the additional challenges posed by embedded systems. We provide an augmented XP methodology that is specifically designed to address the issues associated with the automated testing of embodied (i.e., robotic) autonomous systems in dynamic environments.

In this paper, we present the results of over four years of designing and developing tests for embodied autonomous systems. We use a brief case study to provide examples of several key pitfalls, and provide a high-level overview of the requirements that must be met to test these systems.

INTRODUCTION

Let me set the stage. It is late on Sunday and your robot is scheduled for a demo tomorrow. Your team has been tweaking the code to get a new behavior to work properly. You call a halt to a process and start a final test of the robot, and it fails. What do you do now? If you are like most of us, you have a large set of possible error sources, a hardware component could have failed, the new software components could have failed, or the new software could be causing a communications error between the hardware components. If you start on the hardware and it turns out to be the software, you have wasted valuable time, and vice versa. It looks like it is going to be a long night. This work was inspired by that frustration.

In this paper we discuss ways to implement eXtreme Programming (XP) methodologies to embedded autonomous systems. We will start with a brief discussion of XP, and the advantages that it brings to software development. Then, we will discuss the unique challenges posed by embedded/robotic systems and propose some solutions. Finally, we will discuss an XP methodology that we have implemented in the development of robotic code.

I. eXtreme PROGRAMMING

In the last decade many methodologies have been proposed to improve the quality and reduce the cost of developing software. Among these tools, eXtreme Programming is becoming widely accepted as a methodology that supports the development of highly reliable software. For an overview see[2]. This is due, in part, to four specific practices: continuous testing, continuous code review (pair programming), continuous integration, and pragmatic coding standards. Most of the practices required by XP are the same whether they are applied to a pure software development project, or to an embodied system. However, the introduction of co-development of hardware, firmware, and software requires significant adjustment of some traditional XP practices. In specific, the application of continuous integration, and continuous testing require some re-evaluation in the hardware/firmware domain.

According to the National Institutes of Standards and technology (NIST), inadequate testing infrastructures result in tens of billions of dollars in additional costs, and significant over-runs in development time[12]. As mentioned above, one of the core practices of XP is continuous integration and testing. XP relies on the use of automated test tools such as the xUnit family (JUnit for Java, CppUnit for C++, etc.) of programs. These tools allow the developer to produce detailed unit tests (tests at the function or method level) which can be automatically run, and the results evaluated, summarized, and displayed to the developer. It is not uncommon to have several hundred test cases, covering every method and execution path, run in a few seconds at the press of a button. This allows the developers to make significant changes to the code (re-factoring) while remaining confident that no unanticipated side effects have broken the code base. This capability also supports the nearly continuous integration of the new code into the build process.

While these techniques have been well developed for traditional software projects, their integration into embodied systems lags behind[15] [9][7][4][3]. In the current model of hardware/software co-development, this has frequently resulted in software being developed well in advance of the hardware that it is designed to control. As a result, the high-level software is often tested in isolation, and integration testing becomes a significant factor in total development cost and time to deployment. It would be extremely powerful to have the benefits of continuous testing when
II. METHODOLOGY FOR TESTING EMBODIED SYSTEMS

In this section we discuss a methodology that we have developed for testing embedded systems. In the interests of clarity, the following definitions are used in this paper:

- **Software** is the high level code, for example the planning and reification system.
- **Firmware** is the lower level code that controls the hardware.
- The interface allows the passing of information and control between firmware and software modules.
- The system is the combination of the software, the firmware, the interfaces, and the hardware.
- A scenario is a preplanned sequence of inputs, for which the preferred output is known.
- A simulation is an unplanned or random sequence of inputs for which the preferred output is only known in a probabilistic way.

In traditional software deployment, every effort is made to abstract away from the specific hardware underlying the virtual machine. This is possible because, for the most part, the hardware is completely substitutable. In fact, the Apple/Windows/Linux wars can be considered an indirect result of a lack of substitutability. In embodied systems, the hardware is an integral part of the system, and the substitutability assumption can not be made. This lack of a (more or less) uniform platform causes the developers to be responsible for the production and testing of not only high level code but also lower level interface and firmware control code as shown in Figure 1.

The fact that the developers may be responsible all of these components leads to a complex process of co-development. The high level software must interact with the firmware, which must be written to interact with the high level software. This tight coupling significantly increases the chances of software failure. The potential of software failure can be mitigated by automated testing, but this requires testing both the high level software and the firmware[1]. We have developed a different way of thinking about testing in embodied systems.

The first test that happens is bench testing, when the individual components are tested for input and output parameters, often on the robotic chassis. Once the behavior of the component is known, then the component can be “mocked up” in software (see Rainsberger[14] for more details on Mock objects). This allows the programmer to create and run regression tests on that component in a software environment such as jUnit.

The next set of static testing will be done on the high level software (but as a good XP programmer, you already have a set of regression tests on your high level software). Once the firmware and software have been tested, then the interfaces between them can be tested. So far so good, but thus far only the behavior in a static environment has been tested, now, we need to test the system in the presence of...
chaos, dropped messages, moving targets, moving obstacles, and other hazards.

Dynamic testing comes in four different flavors. For the purpose of this paper, we assume that the high level planning software has been run through simulation testing. However, in order to make the firmware and interface testing fit the XP framework, the tests take the form of prescribed scenarios, for which the appropriate behavior is known. This allows the programmer to have the same confidence in the system’s dynamic behavior that the regression tests give in the static behavior. It also would be a good idea to write scenario tests for the high level planner in addition to the probabilistic simulation tests. Last, but not least, on to system tests, where the robot is taken out and allowed perform in the real world. This is a much less frustrating experience, since at this point you have some confidence that the software is working correctly.

So, back to the example in the introduction, what went wrong? The bench test was probably done correctly as was a static test of the high level software. The system failed at the system test. Without the static and dynamic test of the firmware and interface it will take a significant amount of time to discover the mismatch between the output of a GPS board and the input of the high level software. By partitioning the tests, this error can be detected before the expensive field testing.

A. Benefits of Partitioning the Tests

So, what lift do we get from this partitioning? Ever since the introduction of functional decomposition, it has been clear that neatly decomposed software is easier to develop. One key reason for this is that software that has been decomposed into (relatively) independent modules has lower levels of coupling.

In this paper we are presenting a methodology for decomposing tests into roughly independent classes, and addressing cross product terms explicitly. This is driven by the need to effectively cover a large, complex space of test cases, in a domain that is not easy to exhaustively cover. From the test decomposition shown in Table 1, we can extract six classes of “software only” tests. High level software has been extensively researched with respect to testing, and it is not uncommon to have two to three test cases for every function or method in the code. While the automated testing of firmware is not as well researched, it is expected that approximately the same test-to-function ratio will hold. Without any type of decomposition, the number of tests needed to cover the combination of software, firmware, and interfaces would be:

\[ \text{TestCount} = \text{Test}_{f} \times \text{Test}_{i} \times \text{Test}_{u} \]

Where: \(\text{Test}_{f}\) is the number of firmware tests, \(\text{Test}_{i}\) is the number of interface tests, and \(\text{Test}_{u}\) is the number of high-level software tests.

This results from the need to test each function with each combination of tests for the other classes. This is significant since in not uncommon to encounter test libraries with thousands of individual automated tests. This can easily result in millions of test cases to assure coverage.

It is, of course, possible to reduce the rather large number of test cases by effectively partitioning the space. If partitioned perfectly, the total number of test cases reduces to the sum of the test cases in each class, a significant reduction. It is rarely possible to partition the test space perfectly, since there are interactions between the various software components.

As a result, we test the firmware and hardware classes as though they were partitioned perfectly, and then test the interface between the layers as a separate class of tests. The key condition for the interface testing is the ability to rely on the successful testing of the components on either side of the interface. In effect, given that the firmware is correct, and that the software is correct, it becomes possible to test the interface as an independent class of tests. Given the validity of this assumption, the total number of tests becomes:

\[ \text{TestCount} = \text{Test}_{f} + \text{Test}_{i} + \text{Test}_{u} \]

Where: \(\text{Test}_{f}\) is the number of firmware tests, \(\text{Test}_{i}\) is the number of interface tests, and \(\text{Test}_{u}\) is the number of high-level software tests.

For the case where we have approximately 700 firmware tests, 1100 high-level software tests, and 200 interface tests; we go from 770,000 un-partitioned tests to 2000 partitioned tests.

III. GENERAL TESTING GUIDELINES

As mentioned above, the structural decomposition is into three layers: high-level software, firmware, and hardware. Each of these has characteristics that affect the way in which testing is done. In order to reduce the costs and delays associated with systems integration, we have developed several guidelines for the design of automated tests for the firmware layers in embodied systems.

This code is written in a requirements driven development environment. Practitioners of test driven development will notice that the following code guidelines will violate some of their rules. We have found that for embedded systems the constraints imposed by the combination of the requirements and the physical environment are limiting enough that the addition of test driven development may make the coding task impossible. So, the following are our “in house” testing rules.

- Some things cannot be tested
- Do NOT change the code to make testing easier
- Use public accessors for setting and testing variables
- Test physical constants
- Mock hardware components and interfaces. Note: this often requires duplicating hardware performance in testable software modules
- The static tests are unit tests, test unit level functions.

A. General Partitioning Guidelines

Another possible source of confusion is the line between static and dynamic testing, so here are our rules for that
partitioning. Static unit tests are traditional automated tests, in which the required behavior of a method or function is verified. These tests typically involve setting up a test harness for a method, invoking the methods with a specific input set, and confirming that the return data are the expected value(s). These tests are the staple of all unit testing frameworks. Dynamic unit tests are used to test the performance of a module under changing conditions. These tests are critical for embodied systems, such as autonomous robots, since they will almost always be deployed into dynamic and uncertain domains. Dynamic testing is done using a scenario generator. This software is built out of traditional unit testing components, which have been extended to allow the specification of a sequence of tests, which correspond to a requirement of dynamic behavior. For example, consider an autonomous ground vehicle that is expected to detect a dead-end, and avoid getting ‘trapped.’ This required behavior can best be tested by creating a scenario which consists of mocking the sensory data that would correspond to the robot entering the dead-end and detecting the blockage. This scenario would have an expected behavior that corresponds to reversing course, extricating the robot from the dead-end, and proceeding in a manner that would avoid the dead-end.

IV. CASE STUDY

A. Hardware

As a case study, we will use an autonomous ground vehicle that is currently under development. In Figure 2, the autonomous ground vehicle (Kitty) is shown undergoing unconstrained environment testing at a recent robotics exposition. (Please note that the robot was moving towards the people when the photo was taken) While there is an emergency stop control system, during these tests the E-Stop was unnecessary.

Kitty has an architecture that is biologically inspired. Based on research into neurophysiology, the core structure uses a model based on the brainstem and functional units of simple terrestrial vertebrate nervous system. In Figure 3, the UML deployment diagram for the brainstem of the vehicle is shown. Each of the major nodes (the cubes in the figure) is an independent processor module. These are real-time modules which are programmed in a subset of the Java Programming language, and which emulate the parallel processing capability of living systems. For the entire system for to perform reliably, each of these modules and their components must function correctly and the interactions between each module must perform correctly. The XP methodology relies on continuous testing to assure the developers that any new code that has been integrated is performing correctly, and to assure that the most recent changes have not caused any previous software to fail.

Figure 2 The autonomous ground vehicle ‘Kitty’ during unconstrained field tests. Note: Kitty is driving towards the people at about 4 m/s, and has just adjusted the steering for a hard right turn. Kitty will clear the woman (and her child) with about 15 cm to spare.
B. Static Tests

1) Hardware Bench tests: The first stage of testing is making sure that the hardware is doing what it is supposed to be doing. Without this foundation, it is extremely difficult to debug software failures. In traditional software-only development, this step is typically skipped, since the underlying hardware is assumed to be tested and functional. If the development is being embodied into a commercial, off-the-shelf robotic platform, this step is often provided by the platform manufacturer. However, if the embodied system is a custom, or in-house developed, platform some level of assurance must be provided that the hardware is functioning correctly. The second benefit of this bench testing is that it provides the ‘gold-standard’ data needed to successfully build software mock objects to use in the firmware testing.

2) Static Firmware Tests: The firmware acts as an abstraction layer between the hardware and the high-level software. In the past, firmware was frequently developed in hardware specific application languages, which generated obstacles to automated testing. Recently, there has been growing acceptance of higher level languages as the primary development tools for embedded micro-controllers, and low-level hardware interfaces. The use of a high-level language such as Java or C++ allows significant flexibility in testing firmware.

For example, Kitty’s brainstem is running on five independent Parallax Javelin[13] chips, microcontrollers which are programmed in a subset of the Java programming language. These chips provide Java wrappers for hardware components such as Universal Asynchronous Receiver/Transmitters (UARTs), timers, and motor control systems. When the embedded code is running on the actual chip, these hardware components are instantiated, and connected to the Java code.

During static testing, it is necessary to mock these components, enabling the embedded code to compile and execute. By building the mock components, we have the ability to verify the function of the firmware, without having to run the actual hardware[6]. This means that automated static test libraries can be developed and archived. They can then provide full regression tests at the firmware level.

As an example we have mocked the UART wrapper used by the Javelin chip. A UART is used to provide serial communications between independent devices. The firmware can instantiate a UART and assign it to any input/output (I/O) pin on the chip. The mocked UART has the ability to accept data from the firmware and store it into a transmission buffer, which is visible to the verification tests. This allows the static test to automatically verify that the order in which the data are transmitted is correct, and that each datum has the correct value. If, during development, a change is made that results in a software error affecting this information packet (perhaps the order of data is altered, or a data item is skipped) the automated tests will detect and flag the error.

It is important to note that the mocked hardware components should only have enough fidelity to enable testing the firmware. The focus is not on testing the hardware itself, that step was completed by the bench testing in the previous step.

The question under consideration is: “If the hardware is working correctly, will the current firmware code meet the requirements?”

3) Static Software Tests: The static software tests are traditional unit tests. Many of the resources previously cited provide coverage of the process of testing traditional software. However, it is important to note that by previously testing the firmware, the testing of the high-level software can be separated from the testing of the underlying firmware. As mentioned above, this partitioning significantly reduces both the number of tests needed to cover the high-level software, and reduces the complexity of those tests.

4) Static Interface Tests: The final leg of the static testing is to test the interfaces between the different components. It is entirely possible for two software components to each pass all their individual tests, yet fail to function as a complete system. In embodied systems this can become a major problem, since there are typically numerous independent microcontrollers, processors, and discrete components.
hardware components that must all interact to meet the system’s requirements.

As an example, consider the navigation and communications modules as shown in Figure 3. The individual tests on the firmware code running on each of these modules shows that the navigation board is correctly reading and packaging the GPS and compass data, and loading that data into the UART for transmission. In the same manner, the static tests on the communications board show that it is correctly receiving the data packet, and unpacking it into local storage for dissemination. However, even though all the tests are passing, the system fails in the field tests. This comes about because the navigation board and the communications board (developed by different teams) have an incompatibility. This was the cause of the failure of the Mars Climate Orbiter[11]. The use of automated interface testing can detect and flag these types of errors. In the case of the interface between the communication board and the navigation board, the tests would instantiate a copy of the interface between the communication board and the navigation firmware, and load known values, mock the transmission of the data to the communications board, and then verify that the correct data is stored on the communications board. This could be caught during the test writing process as a developer writes:

```
AssertEquals(COMM.HeightInFeet(),
             NAV.HeightInMeters());
```

In addition, during dynamic testing, the behavioral requirements would detect and flag the error.

C. Dynamic tests

The static tests on the firmware focused on the fixed behavior of the methods and functions. The dynamic tests address issues that occur during continued operations. These tests include verifying correct behavior in cases such as buffer over-runs, loss of communication, updating rapidly changing data, and run-time performance. These types of tests require extending the model of unit tests. In keeping with the notion of building firm foundations, dynamic tests of the firmware are the logical starting point.

1) Dynamic Firmware Tests: In the static testing of the firmware, it was necessary to create mock objects that corresponded to the structure of the hardware components that the firmware utilized. These objects must be extended to provide some of the dynamic behavior that the hardware objects display. As an example, in the case of the UART provided by the Javelin chip, there is a fixed size transmit/receive buffer. During static testing it was sufficient to mock the capacity of the buffer, however to support testing of buffer over-run behavior, it is necessary to mock a certain amount of the run-time behavior. The description of this run-time behavior may be described in the technical documentation of the hardware, or it may be necessary to discover the behavior by running additional bench tests. However, not knowing how the hardware performs under dynamic load means that the developers have a large class of errors that will only occur at run-time.

A case in point was the behavior of the Pathfinder Mars Rover, which underwent system level resets (unfortunately, on Mars[5]) when a hardware priority inversion fault occurred. The Dynamic firmware tests for the Navigation board include building a scenario which will continuously generate and export compass heading data, to confirm that the firmware does not get overwhelmed. Additional tests include updating the mocked data registers on the GPS module during the read by the firmware to verify that the data are consistent. In addition, general performance tests are run on the firmware loops, to verify that requirements of update rates are being met.

2) Dynamic Software Tests: The Dynamic software tests are a natural extension of the firmware tests. Since the hardware has been mocked to provide realistic dynamic behavior, and the firmware has been tested to verify that its dynamic behavior meets the requirements of the system, it is possible to develop similar tests of the dynamic behavior of the high-level software. In many embodied systems there are additional requirements for the high level software. These may include adaptive behaviors, learning, and autonomous behavior. These are aspects of the high-level software that require dynamic testing of a slightly different nature than the lower level firmware. For example, if the system is supposed to function autonomously, it may not do the same thing, in the same way, time after time. This may require dynamic tests which setup the same situation time after time, and record the behavior of the embodied system. No single test can necessarily answer the question “Is the system performing correctly?” Rather, it may require statistical analysis of the aggregated result before the automated test can be verified. If the system is supposed to learn from its actions in the environment, then we compound this verification problem. It is necessary to run a series of aggregate tests to establish the current behavior, allow the system to experience failures and learn from them, and then rerun the aggregate test to establish that the learning has occurred. Now the test harnesses must provide a complex and dynamic set of test cases, and measure the change in aggregate response distribution.

While it is certain that designing, developing, and, ironically, testing these test scenarios is a major undertaking; it is clear that attempting to do this in the field is far more time consuming, and physically challenging. (Consider setting up a physical test to introduce reliable, repeatable noise into a sonar sensor, versus pumping ‘noise’ into the software mock of the sonar sensor. See the recent report by Tse et. al. [16] for an example in the manufacturing domain.)
3) Dynamic Interface Tests: The dynamic interface tests are extensions of the previous tests. They are more complex since they are testing the interactions between multiple components in the system. But the same types of tests that were run to verify the dynamic behavior of the individual components can be extended with the static interface tests to establish correct dynamic behavior.

D. System Level testing

Systems level testing is actually composed of two phases. The first is a general systems test which is run every time the robot turned on. The second is more interactive formal systems test protocol.

1) The Robot dance (systems self-test): The first phase is the something we call “the robot dance,” which is run every time that the system is started. In the “robot dance” the hardware is self-tested by the machine and the success or failure of that hardware is announced in a way that the human can understand.

So Kitty, on startup, uses a voice output to say her name and software revision numbers, turns the steering to the left and right, activates the sonar sensors and announces the results, runs backwards and forwards, activates the compass and announces the result, and activates the GPS and announces the results. The steering and drive train tests explain the “robot dance” label. This sequence becomes a hardware regression test, although the complexity of the test is limited by the need for a human to observe and interpret the results. As we add new hardware, new tests are added to the dance. This test perhaps the most tedious of the set, because it happens just as you are ready to test all the new cool behaviors that have just been added and tested since you had the robot out last. However, this test is also essential. If you have completed all of the tests, you have an assurance that any anomalous results you may see in the final test sequence are the result of unexpected interactions and not testable software or hardware bugs.

2) Formal Systems tests: Finally, we come to the last tests. At this point, you are probably saying “Enough already,” which is run every time that the system is started. In the “robot dance” the hardware is self-tested by the machine and the success or failure of that hardware is announced in a way that the human can understand.

So Kitty, on startup, uses a voice output to say her name and software revision numbers, turns the steering to the left and right, activates the sonar sensors and announces the results, runs backwards and forwards, activates the compass and announces the result, and activates the GPS and announces the results. The steering and drive train tests explain the “robot dance” label. This sequence becomes a hardware regression test, although the complexity of the test is limited by the need for a human to observe and interpret the results. As we add new hardware, new tests are added to the dance. This test perhaps the most tedious of the set, because it happens just as you are ready to test all the new cool behaviors that have just been added and tested since you had the robot out last. However, this test is also essential. If you have completed all of the tests, you have an assurance that any anomalous results you may see in the final test sequence are the result of unexpected interactions and not testable software or hardware bugs.

References

9. H. Koehnemann and T. Lindquist, “Towards target-level testing and debugging tools for embedded software”. In Proceedings of the


Advances in the Framework for Automatic Evaluation of Obstacle Avoidance Methods

J.L. Jimenez I. Rañó
Dpto. de Informática e Ing. de Sistemas
Universidad de Zaragoza, Spain
irano@unizar.es

J. Minguez
I3A, Dpto. de Informática e Ing. de Sistemas
Universidad de Zaragoza, Spain
jminguez@unizar.es

Abstract—This paper will describes the advance in our project for benchmarking obstacle avoidance techniques for mobile robots. The core of the project is to create a methodology/software to evaluate the performance of the methods given a wide range of work conditions. These work conditions usually include scenarios with very different nature (dense, complex, cluttered, etc). The performance is measured in terms of robotic parameters (robustness, optimality, safety, etc). In the paper we will give an overview of the project and we will focus on the project analysis from a software engineering point of view. At this state the software design decisions are critical and could impede a proper later development, therefore we have developed a great effort in the analysis and design of the project.

I. INTRODUCTION

There is currently a great effort in the robotic community to find standards as a way to measure the quality of a wide range of technologies. Good examples are the RosTa EU project, the special group of interest raised by EURON and some other launched by the NIST. Many of these efforts are focussed in creating benchmarks for a given area. Our project is concerned with the standardization but from the automatic evaluation point of view. Instead of working on benchmarks, we address an automatic evaluation system is being constructed for obstacle avoidance algorithms. On the one hand, benchmarking just will try to match the results of an algorithm with some expected output, a desired algorithm result. On the other hand, our automatic evaluation framework will provide the needed results to perform the benchmarking, but also will generate a composition performance of the obstacle avoidance algorithms in very different situations.

In this direction, we have been working in the development of a software tool evaluate the obstacle avoidance algorithms in the context of service robotics. The objective of this paper is to present the first steps: the analysis and design of the problem from a software engineering perspective. The project is expected to grow up, hence this software engineering perspective is very important at this stage. Since this is currently an open research field, the application is expected to change as new ways of evaluation appear. In order to design a useful application in the future, including those desirable characteristics, it is important to follow good software engineering practices. An early implementation has been done and some tests have been performed as the final steps of the analysis of requirements and design of modules and system. The rest of the paper is organised as follows: Section II introduces the evaluation system global analysis and its requirements. In Sections III, IV and V the analysis and design of the identified independent applications is presented. Some details of our early implementation and preliminary results are depicted in Section VI, while Section VII depicts some conclusions and next steps in our project on automatic evaluation system for obstacle avoidance.

II. OVERVIEW OF THE EVALUATION SYSTEM FRAMEWORK

In this section we will present an overview of the whole evaluation system framework to analyse and identify the possible modules and its requirements. Some aspects of the evaluation system has been introduced in [2]. The idea behind the application is to evaluate obstacle avoidance methods (measuring quantitative parameters of the solutions) given a wide range of working conditions (different scenarios). As can be seen from Figure 1 three independent modules can be clearly identified. The colored blocks denoted Module 1, Module 2 and Module 3 correspond to the scenario generator and characteriser part, the robot simulator and trajectory descriptor and the final analysis of the results module. They jointly from the complete automatic evaluation system and can be implemented as separate applications even some interaction exists between them.

Figure 2 draws the interaction between the three modules of the automatic evaluation system as a data flow diagram representation. Instead of keeping direct data flows between the modules, the applications can use two data repositories, namely the scenario files (obtained from the scenario generator) and the execution result files (provided by the robot simulator module). The user will introduce to the system some parameters like the amount of scenarios to generate, the number of test to run, and so on. In view of Figures 1 and 2 the automatic evaluation has been decided to be implemented as three independent applications.

Following [5] the requirement analysis of a software system can be viewed as a set of functional requirements, what the system should do, and a set of non-functional requirements, like efficiency, portability, extensibility and so on. As a ongoing research project some points are not currently defined or are prone to change, therefore software modularity, flexibility and extensibility are mandatory as
non-functional desirable features. On the other hand the functional requirements of the system have to be defined before the analysis stage. The rest of this section is devoted to present the modules and its functional requirements.

a) Module 1: represented in Figure 1 is the scenario generator and characteriser. Its general purpose is to generate simulated environments (randomly or following some criteria) and compute descriptors measuring interesting properties from an obstacle avoidance point of view. The descriptors will be used to classify the different sets of environments according to its nature (dense, complex, cluttered, etc). This is a key module on the system since the obstacle avoidance algorithms must be tested on all the possible situation or environments a robot will face up. A high level analysis shows that its functionality must include:

- It must be able to characterise the scenarios using the defined descriptors and compute their values.
- It must generate scenarios, randomly (for a given number of scenarios) or such that they descriptors cover a given range with a minimum number of them. While in the first case the user must only introduce the number of scenarios to be generated, the former must ask for a number of bins and minimum scenario number in each bin.
- It must handle scenario files and files with scenario lists.

b) Module 2: will compute the trajectories of an obstacle avoidance algorithm for a set of scenarios. It also should provide some performance measurements of the trajectories refereed to an, in some sense, optimal path. The functional requirements found for this module are:

- It should be able to dynamically load any obstacle avoidance algorithm that matches a given function prototype.
- It must simulate a generic robot over a set of scenarios and characterise the robot motion according to some trajectory descriptors.

c) Module 3: builds the evaluation results based on the outputs provided by the two other modules. Given an obstacle avoidance algorithm, for each scenario descriptor and trajectory descriptor this module will generate a result table with the algorithm performance behavior. This module can bee seen as the most general part of the system, since it can be used for other purposes where scalar results from two different features need to be drawn. The minimal functional requirements of this application are:

- It must build tables by crossing scenario descriptors and trajectory descriptors.
- It must be able to open and to create result files for an obstacle avoidance algorithm benchmark in a predefined format.

III. SCENARIO GENERATION AND CHARACTERISATION

In this section we present the application for scenario generation and characterisation (Module 2) analysis and design, while implementation and testing will be treated in a separated section. We choose to follow the OMT (Object Modelling Technique) methodology [3] to build the analysis diagrams of each application in the system and some UML (Universal Modelling Language) tools has been also used to build those diagrams. However, since UML [4] has been derived from OMT and they have many common elements.

The analysis section focuses only on the static relations of data, not treating the dynamic and functional relations, because its smaller significance in our case. Then the design section includes the application architecture grouping the analysed data models into modules.
A. Analysis of the Scenario Generator and Characteriser Application

The key concept in this application is the scenario for a full technical description of scenarios see [2]. They will be characterised according to its qualitative features and will be used to carry out the robot motion simulations. A concept with such an importance must be included as a class to collect the scenario functionality on the application, the Scenario class. The scenarios will have associated sets of characteristics, the descriptors, and encoding models, the way scenarios are implemented in the application, that need to be included in the static class view. Therefore a Descriptor virtual class needs to be created to reflect the scenario characterisation. The general class represents the functionality of any descriptor prone to be implemented in the system, and collects the scenario characterisation functional requirement. Moreover, as the Descriptor class is virtual the concrete defined descriptors will be derived from it. This also fulfills the extensibility non functional requirement above mentioned. Since the calculation of some descriptors can be computationally expensive in terms of time and memory we decided to use the Proxy software pattern [1]. In this way the computation of the descriptor is delayed until it is really necessary, and once the numerical value is get it will be stored for later use. Figure 3 shows part of the UML class diagram generated on the analysis stage of the application. As can be seen from figure, currently only three descriptors have been implemented as Density, Clearness and Confinement derived classes.

Fig. 3. Part of Scenario Class Diagram

The right class box on Figure 3 is the scenario implementation ScenarioImp virtual class. Since the computation of some descriptors can be quite difficult for some scenario representations the ScenarioImp class has been designed to allow different scenario representation on the application. Even not drawn in the figure two classes are derived from this virtual class, the discrete DiscreteImp and continuous ContinuousImp scenario implementations. While the later stores the environment as a set of obstacles defined by its geometrical primitives, the former consists on a binary occupancy grid with configurable size. This allows to compute the descriptors with different resolutions and algorithms for a given scenario. The capability of having different implementations introduces the need for an implementation converter in order to pass from one implementation to another. A Converter virtual class has also been designed using the Strategy software pattern. This pattern defines and encapsulates a family of algorithms, in our case for implementation conversion. From the Converter class any implementation converter can be derived. Currently only a continuous to discrete implementation algorithm is available.

Another functional requirement of this application stated in Section II is the ability to generate scenarios in different ways. A fixed number of random scenarios, for instance, could be desired to perform test on some algorithms. In other cases a set of scenarios need to be generated to cover all the possible value ranges of a given descriptor. A virtual class Generator, not reflected on the Figure 3, has been defined to fulfil the corresponding functional requirement of the application. Since the generation of scenarios can be performed in different ways, the Strategy software pattern has also been selected to design the generator class. Any instance of a generation algorithm must be derived from the Generator class. Moreover, as there are several implementations of the scenario, the virtual class has a method called FixImplementation() to select the implementation kind to be generated.

Up to this point the analysis has only included individual scenarios, however the evaluation system has to perform test on all possible scenario conditions. Therefore a scenario list ScenarioList abstract class needs also to be used to jointly store sets of scenarios. Any other scenario list can be derived from it. On the other side, the lists usually need to be accessed in a sequential way and an iterator software pattern is necessary. Besides this access method for the scenarios on the lists, another functional requirement is the capability of scenario indexing according to the value of some descriptor. A class for a scenario list index has been modelled to serve as an interface for scenario lists allowing to access groups of scenarios with some given ranges of descriptor values.

An important aspect of the scenario generator is data persistence, because all the computed descriptors and its scenarios must persist when the application finishes. To generate and characterise scenarios can be a resource consuming task. To allow this data persistence it was necessary to create a scenario storage file system. Since the internal format in which data is stored is not a part of the analysis stage, but just to take into account the persistence needs, two abstract classes have a reader and a writer been implemented, with some appropriate derived classes to store the scenario lists in a preliminary format. This structure allows for a change in the internal storage way, fulfilling also the flexibility non functional requirement.

The final requirement is to allow the user generate and handle the scenarios and sets of them through a Graphical User Interface. It is common practice to change an application GUI, and therefore it is important to have a small number of classes involved, to have a uncoupled application and GUI implementation. Any visual element on
the application is modelled as a GUIElement class, and are grouped hierarchically. The composite software pattern has been used since it allows to treat visual elements in the same way either being compound or not. It is a good practice to separate the user interaction in two parts: the visual part and the functional part, such that if any needs to be changed the other can be kept. The functional part are implemented through the application commands that represent the simplest actions a user can perform. The Command software pattern proses the creation of command objects to encapsulate and parametrise actions.

B. Design of the Scenario Generator and Characteriser Application

The previous section has depicted the five main concepts related to the scenario generation and characterisation application with its corresponding classes. A set of classes (Scenario, Scenario Implementation, Generator...) and its derived subclasses are grouped around the scenario concept. There are other classes related to the scenario list; the Scenario List itself, the Iterator, Reader, Writer and Indexes. Some others providing Graphical User Interface functionality and finally the command related classes. All of the above can be joined into a new class representing the application itself. Figure 4 groups and relates the main modules to build application, an arrow starting in one module means that it depends on the target arrow module. As can be seen all the modules, except the application, use services of the Scenario module while this one does not use any service of the rest of the layers. This makes the scenario module quite critical in case any change should be done, because all the layers could need an adaptation to the new module interface. The Scenario I/O layer is used by the Commands and GUI modules making it the second most critical module to changes.

Fig. 4. Scenario Generator and Characteriser Application Architecture

The Commands and GUI layers interact with each other, since the GUI uses Commands functionalities and introduces information parameters from the end user to the commands. Finally, the higher layer, the Application is available to the user for the lower level Commands and GUI layers access.

IV. ROBOT SIMULATOR AND TRAJECTORY EVALUATION

This section presents the analysis and design performed on the simulation application, the so called module 1 in Section II. This module must interact with the scenario generator presented in Section III by reading the different scenarios where obstacle avoidance algorithms have to be tested. As stated in Section II the functional requirements of this application are the simulation of a generic robot motion for any dynamically loaded obstacle avoidance technique and trajectory descriptor calculation.

A. Analysis of the Robot Simulator and Trajectory Evaluation

The simulated robot model has a sensorial and motor part. The sensor of the robot is assumed to be a proximity range sensor providing distance measures in a 180° range around the robot front, one for each degree. On the other hand, the robot always move forward with limited speeds and accelerations. As presented in Figure 5 a robot class models all those robot characteristics. We reflect in this class diagram the fact that the motion control is provided by an external library. Through the MovementCalculator class we provide an interface for the implementation of the dynamic library loading process and function call. As can be seen in the figure two classes are derived from this basic one that must take into account the way each operating system loads the dynamic libraries. Using this class heritage to load obstacle avoidance algorithms the non functional requirement of a multi-platform application is accomplished.

![Class Diagram for the Robot Simulator and Trajectory Evaluation](image)

The motion simulator obtains as output the robot trajectory that must be compared to some defined optimal paths to evaluate the trajectory generated by the loaded obstacle avoidance algorithm. The robot class includes a trajectory calculation.

The trajectory evaluation is performed by the computation of some trajectory describers also represented in Figure 5. The motion simulator obtains as output the robot trajectory that must be compared to some defined optimal paths to evaluate the trajectory generated by the loaded obstacle avoidance algorithm. The robot class includes a trajectory calculation. Currently three such derived describers have been defined and implemented, success, optimality and safety. The success describer has a boolean value indicating if the target position has been reached following the motion commands. Optimality is a comparison with the optimal path obtained using the visibility graph from the start to the end position, while safety is computed through comparison with the safest
path, obtained from the Voronoi diagram. Finally, as for the scenario characterisation application the results need to be stored, therefore the robot class also includes a result writer abstract class that allows to store descriptors in a format that can be changed in the future without affecting the rest of the classes.

**B. Design of the Robot Simulator and Trajectory Evaluation**

The robot simulator and trajectory evaluation application design is presented in Figure 6. As can be seen some of the application layers are Scenario and Scenario I/O, defined for the scenario generator but also used here. The design reflects that the only way to access the hard disk storage is through the I/O modules. An operating system layer has been added since the obstacle avoidance library, the Motion Calculator module, is OS dependant. The Robot layer is related with the Scenario, from which simulation data is extracted, with the Trajectories and Characterisation layer, the one that computes trajectories descriptors.

Since two of the designed modules on this trajectory evaluation application are directly dependent on the Scenario layer, as for the scenario generator and characteriser the classes related with the scenario class may change if the scenario is changed. Therefore the scenario layer must have a stable interface with the rest of classes.

**V. RESULT ANALYSIS APPLICATION**

This is the simplest application on the current implementation of the automatic evaluation system. Actually this application could be used to perform data analysis of any system with similar features, since it only builds result tables by crossing descriptor ranges. Besides the capability of building result tables the only functional requirement is to store them into a given format.

**A. Analysis of the Application for Data Analysis**

The key concept in this application is the result interpreter, an element that for a given result sequence and a scenario file is able to statistically summarise results in an automatic way. Since there are multiple table formats and programs to handle them we decided to use in out preliminary implementation a plain text format. However, it can be interesting to build binding with some office applications for data analysis. The main classes include the ResultReader abstract class that forms a base for the result file reader, the reading functionality for results not included on Figure 5. When the reader finished the process of data loading it generates an element of the ResultInterpreter hierarchy which is responsible of statistically cross the data. Once again the Result interpreter class is an abstract one allowing different data interpretation classes to be defined in later researching steps, that is providing extensibility to the application.

**VI. IMPLEMENTATION ISSUES AND APPLICATION TESTING**

Besides all the functional and non-functional requirement for our applications we should choose the proper development tools. Since the obstacle avoidance algorithms will usually be provided as compiled dynamic libraries and the evaluation problem is run on a intensive test basis, an efficient programming language must be selected. Another interesting requirement is to make a multi-platform project, that, as stated before, must be highly modular. On the other hand, both application analysis and design have been done Object Oriented and therefore the implementation language should support objects. We choose C++ as our implementation language since it fulfils all out requirements, but mainly it provides modularity and produces efficient programs. We also used a set of highly standard tools like the Standard Template library (STL) and GTK+ for the application parts that need a Graphical User Interface.

An early implementation of the whole framework has been performed contain more than 11,000 lines of C++ code. The performed tests over the system has been done in two steps; a component test and integration test. The component test where performed over all the classes and modules, and the joint application was also tested once the parts were integrated. Most of the test were black-box testing, where different inputs were provided to the elements and its outputs were checked with the expected output. Once a class or layer...
was fully tested the integration was performed and again tested in a bottom-up way.

A. Scenario Generation and Characterisation

Figure 8 shows a final view of the application Graphical User Interface. The left part of the main window shows the working scenario list, while the right part is split in the current scenario display at the bottom and the descriptors sub-window, where descriptor values are displayed. The scenarios can be generated in different ways, the non-parametric way just generates a random number of scenarios on a scenario list. The parametric generator allows the user to define bins over one descriptor and select a minimum number of scenarios in each bin. Of course all the generated scenarios or lists can be saved to the hard disk with an appropriate format using the application.

B. Robot Simulator and Trajectory Evaluation

For the simulation application the optimal path between a starting and target positions needs to be obtained. The current implementation state includes a $A^*$ algorithm to compute the optimal path as a sequence of discrete cells, therefore using the discrete implementation of the scenarios. The Voronoi diagram on the discrete scenario implementation has been used to get the safest path. Both, paths are used to compute optimality and safety trajectory descriptors, which are also evaluated using the discrete implementation by performing an integration over the grid cells. Since the amount of simulations to perform is big and done as a batch process, no graphic interface has been designed for this application.

C. Evaluation Results

For the final evaluation results, we implemented a software able to connect the different descriptors of the scenarios with the performance parameters of the methods. This is displayed in the form of tables of performance for visualization and comparison in between methods. Figure ?? shows the performance of one technique selected ?? . In this case, one can see the performance and evolution of the performance descriptors as a function of the different scenarios measured by density, clearness and confinement for example. In fact this type of tables are a great help of researchers, engineers and developers in order to asses their results and to search for possible techniques to work in a given range conditions.

VII. CONCLUSIONS AND FURTHER WORK

This paper present the current state of the automatic evaluation software for obstacle avoidance algorithms. As an ongoing research project some aspects of the software framework development could change. The initial idea is to create an open source software system. A great effort has been performed from a software engineering perspective to start building an extensible and flexible framework in three mostly independent modules. Extensibility and flexibility must be key features on such a system. Some major aspects of the analysis and design have been presented. Our framework allows to include new descriptors to both, scenarios and trajectories, that on the other hand need to be studied and implemented to better evaluate obstacle avoidance mechanisms.

The future steps include the definition and implementation of new scenario descriptors, and a extensive scenario lists generation as a test-bed for the algorithms. Even the system has been developed to be multi-platform this feature has not been tested yet, and maybe some small changes and further development will be needed to actually have such an application. A Developer’s Guide document and online API documentation will be also created in order to help both developers and users.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1998.
Good Experimental Methodologies in Robotics: State of the Art and Perspectives

Fabio P. Bonsignorio, John Hallam, and Angel P. del Pobil, Members, IEEE

Abstract—As the complexity of developed robotic and intelligent systems grows, it is more and more needed to define proper experimental approaches and benchmarking procedures.

Trustable benchmarks are needed in order to allow the comparison of the many research results in service robotics research and enable their industrial application.

On the other end, if robotics aims to be serious science replication of experiments deserves serious attention.

It is necessary to be able to verify if and by which measure new procedures and algorithms proposed in research papers constitute a real advancement and can be used in new applications.

New more successful implementations of concepts already presented in literature, but not implemented with exhaustive experimental methodology, risk to be ignored, if appropriate benchmarking procedures, allowing to compare the actual practical results with reference to standard accepted procedures, are not in place.

Both replication and benchmarking are needed to foster a cumulative advancement of our knowledge of intelligent physical agents and even to correctly appreciate disruptive innovation in the science and technology of robots.

Should we take inspiration from biology and medicine?

In order to address these needs the European Union Network of Excellence on robotics EURON has funded a Special Interest Group on Good Experimental Methodology and benchmarking.

This paper summarize the state of the art in the field, the possible perspective activities identified so far, and the EURON GEM SIG challenges and ambitious plans.

1. INTRODUCTION

All science proceeds from experiment, which motivates the creation of new theory and establishes the limits and validity of the existing theoretical basis. Individual branches of science conduct experiments differently, depending on the topic of investigation, but all have in common a body of knowledge concerning experimental methodology that specifies how to design and conduct 'good' experiments in that discipline.

If robotics aims to be serious science, serious attention must be paid to experimental method.

The research activities in the robotic fields are huge and it is huge the number of published papers.

In order to allow the exploitation of the many results obtained it is at least necessary to able to:

- validate the results by replicating them
- compare the results in term of the choosen performance criteria

Although some work is already carried on, a lot of open issues are still in front of us.

In section II is described the state of the art as regards the replication of experiments in robotics, in section III the situation in benchmarking is reviewed.

II. THE IMPORTANCE OF REPLICATION

Whether you see robotics as the science of intelligent physical agents ('embodied cognition') or as the branch of engineering that, through mechatronic integration, aims to build autonomous or semi-autonomous machines for many diverse tasks, it must be seen as a scientific quantitative discipline.

In any scientific discipline, from physics to engineering and medicine, the models of a system must be able to predict with a certain accuracy the evolution of the variables under study with a given input over time.

A 'good' model must give the same results in the same condition.

Results have to be 'replicable'.

An objection raised sometimes to this kind of considerations is that the robot which are developed are very different in kind, physical morphology, tasks, algorithms, sensors,
As it is known K.Popper [19] defined in a very tight way the requisites for a discipline to be considered 'scientific'. In social science, management and economics exact repetition is often seen as a limit case, experiments that systematically vary one or more input parameters of a system under study to see whether its output parameters remain stable or change according to the expected model in a predictable way. Only when the model fails clearly in a number of varied experimental setup it is considered 'not replicable'. Nethertheless, as already noticed, all disciplines aiming to be considered 'scientific' incorporate a concept of experiment replication and a concept of 'falsification' of theory through experiments. There are different modulation of this concept [20,21,22,23,24], but whether we think we are in a cumulative phase in the development of a scientific field or in presence of a 'disruptive' creative paradigm shift [22], as somebody is claiming in nowadays robotics, a kind of widely accepted experimental methodology is needed in order to be able to ground the advancement of research on a shared quantitative language. On the other end, in different scientific fields like biology and medicine[17,18,19], there are well established experimental procedures to deal with the behaviors of complex systems, at present more complex, and less known, of those under development and study within the robotics community. They suggest that the huge diversity of the developed robots should not prevent us from implementing more reliable experimental procedures. A clinical trial protocol is the detailed written plan of a clinical experiment. It may be inspiring looking at the US NCI guidelines for drafting a clinical trial protocol, [18]: the emphasis on signaling 'adverse events', the definition of 'criteria for response assessment', the necessity of defining clearly principal and secondary hypotheses to be validated. The statistical section of the protocol is asked to define how the data will be analyzed in relation to each of the objectives. In particular it expect that an acceptable trial specify, with reference to the study objectives:

- Method of randomization and stratification
- Total sample size justified for adequate testing of primary and secondary hypotheses
- Error levels (alpha and beta)
- Differences to be detected for comparative studies
- Size of the confidence interval of the estimates.

It seems clear that in robotics the experimental methodology standards are currently in many cases weaker, and the syndrome 'it worked once, in my lab' could be more widespread than we may think. As already noticed, a limit to replication is given by the huge variability of robot machine. Perhaps, following the biomedical analogy, we have to compare behaviors and performances of different 'animals'. Any way a wider capability to replicate research results is probably needed in order to allow a faster development of our field and to foster both cumulative progress and disruptive change.

III. BENCHMARKING IN ROBOTICS

Due to the huge diversity of robotic architectures and approaches and the intrinsic difficulty of measuring the performances of machines which aim to be flexible and autonomous, so far, there are benchmarks defined only for a limited subset of the tasks which can be performed by a robot [5]. There are benchmarks in machine vision and dependability of software systems which could probably be adapted to the robotic field.

There are some interesting conference tracks like NIST, IEEE and ACM Permis and the IARP, IEEE RAS series of workshop on Dependable Robots.

A. Conference tracks

The Permis workshop started in 2000 and in 2007 it reached the seventh edition. This workshops aim to define measures and methodologies for the evaluation of performance of intelligent systems.

The focus is to define quantitative metrics to be able to compare such things as: the level of autonomy, human-robot interaction, collaboration.

To provide methodologies to evaluate components of intelligent systems: sensing and perception, knowledge representation, world models, ontologies, planning and control, learning and adapting, reasoning.

Other topics are infrastructural support for performance evaluation, application specific performance measures.

In this context intelligence is defined as “the ability to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral goals” (J. Albus, “Outline for a Theory of Intelligence”, IEEE Trans. on Systems, Man, an Cybernetics, Vol. 21, No. 3, May/June 1991).

There are some connections between these workshop topics, those addressed in this paper and the scope of the Workshops on Technical Challenges for Dependable Robots in Human Environments co-sponsored by IARP and the IEEE Robotics and Automation Society. The 2005 workshop was co-sponsored by Euron, the EU network of excellence in robotics.

The concept of dependability requires that the capability of...
performing a set of task in a set of changing environments
are performed with replicabilty with given measured
performances.
Finally in both 2006 and 2007 IROS conference was
organized a workshop on performance metrics of robots.

B. Performance Metrics
Performance metrics have been developed in various area of
robotics for specific purposes.
There are a number of initiatives devoted to define adequate
performance metrics in specific subfields.
Here below follows a non exhaustive list, whose main
purpose is to exemplify the community attempts to cope with
the benchmaking problems.

Radish, started in 2003, by Andrew Howard and Nick Roy
is a repository of standard data sets with, currently, a main
focus on localisation and mapping.
The more common format CARMEN, the open source
Carnegie Mellon Robot Navigation Toolkit, for mobile
robots. control that provides basic navigation primitives.
At present it contains mostly logs of odometry, laser, sonar
and other sensor data taken from real and simulated robots
and envrone ate maps created by robots or manually.
The Automatic Control Telelab (ACT) has been developed
to support real-time configuration and observation of
experiments in visual servoing remotely, as well as playback
access to acquired data.
Universitat Jaume I developed a visual servoing simulation
environment called JaVISS. It is written in Java with
graphical rendering. Manipulator kinematics code is based
on the Robotics Toolbox for Matlab by Peter Corke. It is
intended as a tool to simplify the testing and comparison of
different visual servoing approaches.
Another initiative at Universitat Jaume I offers a set of
experiment data on visually-guided grasping of planar
objects with a Barrett hand. This set of standard experiments
allows the definition of benchmarks and associated
performance metrics.
This allows to compare different algorithms and
implementation for visually-guided grasping.
The University of Parma is managing a repository storing
benchmarks designed by different research groups and
related documentation down to file format for motion
planning.
It stores data sets about robots, workspaces and benchmark
problems, related to different kind of robots: mobile robots
on the plane, free-flying robots, manipulators, sequence of
manipulators, union of mobile robot and one or more
manipulator sequences
An other shared repository of motion planning benchmarks,
originated by Movie project, is Movie Models for Motion
Planning maintained by Utrecht University.

It currently contains data sets related to 192 robots and
objects and 75 scenes.
The RAWSEEDS project is an SSA (Specific Support
Action) in the EU 6th Frame Program, providing a
comprehensive, benchmarking toolkit for SLAM
(Simultaneous Localization And Mapping). It will provide a
web accessible repository storing standard data sets, based
on different sensor sets, and related benchmarks, state-of
the-art solutions to SLAM problems in the form of
algorithms and software, and methodologies for the
validation of algorithms.
The NIST USAR (Urban Search And Rescue) 'after disaster'
scenarios, ranked as yellow, orange and red, are used in
RoboCup USAR. They provide an useful conventional
reference scenario for USAR applications together with
USARsim the open source simulation environment based on
the Unreal Tournament gaming engine.

C. Research coordination activities
The RoSta project, [4], started in 2007, is a two-year
coordination action, within the EU 6th Framework Program.
The objective of RoSta, which is linked also with standard
related activities within IEEE, is to identify the action
needed to start formal standards development and the
establishment of de facto standards in service robotics. The
short term aim is to select a few key topics, where
standardization is already possible and whose expected
impact is higher.

ALFUS (Autonomy Levels For Unmanned Systems) [8], is
a similar US federal agencies ad hoc working group
focused on unmanned system autonomy metrics.
Its main objectives are to analyse the needs for autonomy
metrics, the related methodologies and to develop standards.

D. Competitions and Challenges
RoboCup, [9] is probably the most famous competition in
robotics. RoboCup is mostly focused on soccer game as a
primary domain, and organizes the Robot World Cup Soccer
Games and Conferences. Soccer is a very good testbed for
multi (robot) agent technologies.
New competitions in search and rescue, based on NIST
scenarios, and home assistance have been added.
A similar activity, born in 1997, is FIRA robot soccer
league, a korean KAIST initiative.
Eurobot is an international competition with chiefly
educational purpose with rules renewed every year.
The DARPA Grand Challenge is a famous competition for
outdoor robot race on an about 200km circuit in the desert.
The last edition was won by Stanford team, with a modified
version of a VW Tuaregh, and five teams were able to complete the race. DARPA is now organizing the Urban Challenge where the robot has to cope with an urban traffic scenario.

An interesting cleaning robot competition was organized in 2002, in Lausanne, Switzerland jointly with IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS 2002). As an example the task of the floor cleaning section was to clean within 10 minutes 5x5m room covered with sugar.

The European Land-Robot Trial (ELROB), organized by the German Federal Armed Forces (Bundeswehr), is an outdoor robot demonstration with no real competitions or prizes, but otherwise similar to the DARPA Grand Challenge.

It focuses on mobility and RSTA (Reconnaissance, Surveillance, and Target acquisition). It took place in 2006 for the first time, in 2007 a civilian version was organized in Switzerland.

IV. DISCUSSION

Even from the limited survey above it is apparent that the bare replication of experiments and the quantitative comparison of research results in robotics raise many challenging issues. This is due to the variety of applications, tasks, mechanical structures, sensor sets, actuators, control system, software architectures, required levels of flexibility and autonomy, and so on. When we are dealing with Human Robot Interaction in everyday settings also human psychology is involved. On the other end, there are many initiative trying to define proper standards. There are benchmarks in some specific areas like visual servoing, SLAM, motion planning, but there is still a lot of work to do. Possibly we should identify a few limited and simpler tasks and related environments and develop benchmarks for those task that can be accepted and are by the community and then proceed extending the approach to more complex functions. As told we should probably look to biology, medicine and 'soft' sciences for inspiration.

Fig. 1. Sensory motor coordination information metrics examples (Lungarella and Sporns, 2006)

Fig. 2. Sensory motor coordination information metrics measures (Lungarella and Sporns, 2006)

In [13] and [14], see fig. 1 and 2, from [14], and in other experimental works ‘entropy measures’ on the ‘sensory-motor’ coordination of different ‘robotics’ equipment have shown that information metrics can be used to classify, at least, and to get an insight on (semi) autonomous robotics devices, which show an ‘emergent behavior’, while, in [15], entropy measures are used to rank environment complexity, with reference to the navigation task, see fig. 3. In [12] an approach integrating task and environment complexities is proposed. HRI experimental research is sometime conducted by means of protocols deriving from psychology.

Fig. 3. Environment complexity measures (Chatila, 2006)

It might help dividing the robot functionalities into level with an approach similar to the communication OSI level, starting, for instance, from the phisical level, to the control, perception, planning and 'cognitive' levels?

V. EURON GEM SIG

If we want to foster the further development of (service) robotics research and to enable the industrial exploitation of the many results already obtained, it is probably necessary to improve the common experimental practices, looking at both replication of experiments and objective performance evaluation. In order to cope with these needs the EU EURON network...
practice and poor technique. preferably hands-on training as well as lecture material on basic statistical techniques.) will be supported.

It is also envisioned the establishing of the 'Journal of Replicated Robotics Results': a high quality open access web-based journal that encourages the publication of replications of published experimental results, rational reconstructions of systems, and similar.

VI. CONCLUSIONS AND FUTURE WORK

There is a widespread perception of the need of improving experimental practices in robotics, among many others world wide initiatives, the Euron SIG GEM is trying to address these needs.

It is thought that proper and widely accepted replication procedures and performance benchmarks are needed to allow the cumulative progress of robotic science and technologies and even to assess the value of new disruptive ideas.

REFERENCES

[7] [http://www.euron.org/activities/SIGs.html](http://www.euron.org/activities/SIGs.html)
[17] [https://www.ctnbestpractices.org/](https://www.ctnbestpractices.org/)