Autonomous Navigation for Urban Service Mobile Robots

A. Corominas Murtra[1], E. Trulls[1], O. Sandoval[1], J. Pérez-Ibarz[1], D. Vasquez[2], Josep M. Mirats-Tur[3], M. Ferrer[1] and A. Sanfeliu[1]

***Abstract*— We present to the robotic community a fully autonomous navigation solution for mobile robots operating in urban pedestrian areas. We introduce our robots and the experimental zone, overview the architecture of the navigation framework, and present the results after 3.5km of autonomous navigation. We expose the main lessons learnt by the scientific team and identify the issues to improve future works.**

I. INTRODUCTION

Pedestrian areas are becoming common in modern cities, due to environmental and social concerns, and it is expected that urban service robots will be deployed in such areas in the near future, to aid people in tasks such as transportation of goods, guidance or taxi service. The study of these issues was one of the goals of the URUS project (Ubiquitous networking Robotics in Urban Settings), which has presented scientific and technological advances on these topics [1].

This paper is concerned with autonomous navigation for a urban, service, mobile robot. In this context, the navigation framework will be requested with *go to* queries sent by some upper-level task allocation process, or directly by an operator. These queries will indicate a goal point in the map coordinate frame, and thus the robot's navigation framework also works in the map coordinate frame. GPS-based navigation remain an unreliable solution for mobile robots operating in urban areas, due to coverage blackouts or accuracy degradation [2], so that additional work is necessary for robot localization.

In recent years significant advances have been experienced in the area of autonomous navigation, specially thanks to the efforts of the scientific and engineering teams participating in the DARPA Urban Challenge as [3], as well as other contests [4]. Even if this body of work is designed for car-like vehicles running on roads, some important ideas translate to robots of different configurations operating in pedestrian areas, specially in terms of software and navigation architecture. However, urban pedestrian areas present additional challenges to the robotic community, such as narrow passages, ramps, holes, steps and staircases, as well as the ubiquitous presence of pedestrians, bicycles and other unmapped, dynamic obstacles. All this leads to new challenges in terms of perception, estimation and control.

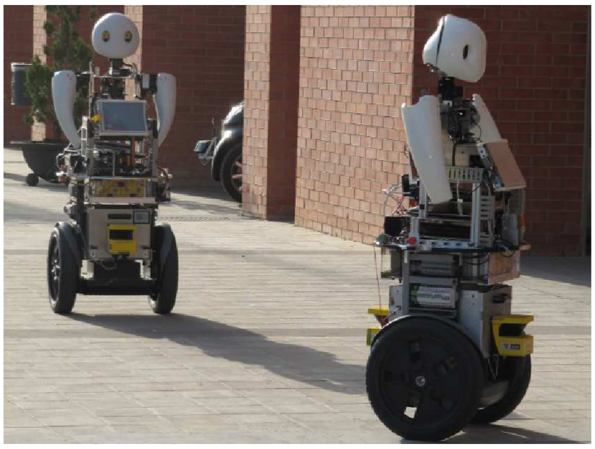
Fig. 1. Tibi and Dabo, urban service mobile robots.

In this context, the paper presents to the robotic community a fully autonomous navigation system designed for urban service robots operating in pedestrian areas. The experimental area is a 10000m^2 section of a car-free university campus. The experiments were carried out with our robots Tibi and Dabo, pictured in figure 1. After reviewing the navigation architecture, we present the results of autonomous navigation experiments totalling over 3.5km, highlighting the issues we encountered and their cause, and exposing the main lessons learnt by the scientific team in order to identify the critical aspects to improve future works.

This paper is organized as follows. Section II briefly describes the robots at our disposal and the experimental area. Section III presents the navigation architecture. Section IV summarizes the localization algorithm. Sections V and VI present our path planning and path execution algorithms. Section VII is concerned to the obstacle avoidance system, dealing with terrain features such as ramps. Results are exposed in section VIII, and section IX presents the main lessons learnt by the scientific team.

II. ROBOTS AND EXPERIMENTAL AREA

Tibi and Dabo (figure 1) are two mobile robots designed to operate in urban pedestrian areas. Each robot is based on a 2-wheel, statically unstable Segway RMP200 platform and is equipped with two Leuze RS4 2D laser scanners, one pointing forward and the other backwards, at a height of 40cm from the ground, scanning the plane parallel to the ground on a flat surface. A third 2D laser scanner, a Hokuyo

Authors are with the Institut de Robòtica i Informàtica Industrial[1], IRI (UPC-CSIC), Barcelona, Spain, [acorominas, etrulls, osandoval, jnperez, mferrer, asanfeliu] @iri.upc.edu, the Swiss Federal Institute of Technology[2], Zurich, Switzerland, vasquez@mavt.ethz.ch, and the Cetaqua, Centro tecnológico del agua[3], Barcelona, Spain, jmirats@cetaqua.com.

This research was conducted at the Institut de Robòtica i Informàtica Industrial of the Universitat Politècnica de Catalunya and Consejo Superior de Investigaciones Científicas. It was partially supported by Consolider Ingenio 2010, project CSD2007-00018, CICYT project DPI2007-61452, and IST-045062 of the European Community Union. O. Sandoval is supported by a scholarship from the Mexican Council of Science and Technology.

UTM-30LX, is mounted at a height of 90cm scanning the vertical plane in front of the robot. Segway platform also provides wheel odometry data.

A large section of the Universitat Politècnica de Catalunya's (UPC) Campus Nord in Barcelona was outfitted as an experimental area. This installation covers over 10000m^2, including six buildings and a square. The campus is placed on a hilly region and features differences in height of up to 10m in the experimental area, resulting in a series of ramps, which the robot must be able to navigate, and drops and staircases, which should be avoided, as well as typical obstacles such as bulletin boards, bicycle stands, trashcans or flower pots. Figure 2 shows two pictures of this environment. The map of the experimental zone can be seen in figure 4.

Fig. 2. Two pictures of the experimental zone.

III. NAVIGATION ARCHITECTURE

Our navigation (figure 3) consists of four separate modules: localization, path planning, path execution, and obstacle avoidance. The obstacle avoidance module is in turn made up of three blocks: traversability inference, local planning and motion control. The path planning module is responsible of finding a global path between the platform and its goal on a static map of the environment upon a *go to* request. The other components conform two different control loops. The first loop, the obstacle avoidance module itself, is *reactive* and is in charge of moving the robot to local goals, expressed in the robot coordinate frame, $X^r_{g_i}$. It is important to point out that this loop does not depend on the localization estimate since it only ensures that the robot will arrive to a local goal while avoiding the obstacles perceived by on board sensors. The second loop is *deliberative*, and is tasked with guiding the robot through the different waypoints forming the path $\{X^m_{g_1}, ..X^m_{g_N}\}$, up to the goal. The deliberative loop includes the localization module and the path execution process, which uses the current localization estimate $\hat{X}^m_r$ to transform the waypoints from map coordinates $X^m_{g_i}$, to robot coordinates $X^r_{g_i}$. This local goal is the input of the obstacle avoidance module, thus closing the deliberative loop.

Both loops, platform driver and acquisiton processes run concurrently in the same computer. The reactive loop runs at 10Hz and the deliberative one at 3Hz. Since the platform moves at speeds up to 1m/s these rates are deemed sufficient.

The software framework follows a service-oriented architecture, with the aim to ease software integration between the developers: each block of figure 3 has been implemented as an independent process, accessible through an interface.

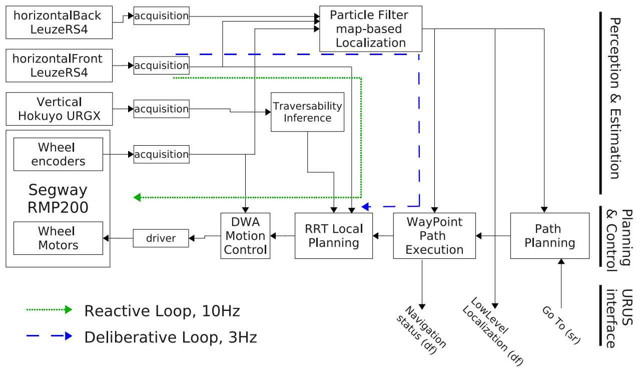

Fig. 3. Process diagram for our navigation architecture. Each block is an independent process. Arrows are TCP connections.

Process communications are based on YARP open-source middleware [5]. For a further description of the software architecture, please refer to [6].

IV. MAP-BASED LOCALIZATION

The localization process estimates the 2D robot position referenced to the map coordinate $\hat{X}^m_r = (\hat{x}^m_r, \hat{y}^m_r, \hat{\theta}^m_r)$, and the associated uncertainties. This has been implemented with a basic particle filter (PF), a recursive algorithm that estimates a probability density function of the state of a system given a set of observations and a prediction model. The density function is represented by a set of samples of the state space, each one having a weight related to the probability of current observations given the system state is in a sample point. The pair formed by a sample vector and a weight is called a *particle*. Further details on PF's in [7].

The proposed filter integrates the wheel odometry and the front and back laser data. The **propagation** step is made by advancing each particle following a simple differential kinematic model with the wheel odometry data. At the **correction** step of iteration t, each particle X^t_i updates its weight computing the likelihood between the real laser observation available, o^t_L, and the *expected* one, $o^s_L(X^t_i)$:

$$p(o^t_L|X^t_i) \propto \mathcal{L}_L(o^t_L, o^s_L(X^t_i)) \quad \in [0,1] \tag{1}$$

Our approach computes on-line the expected observations from particle positions. To compute such expected observations, the robot needs a model of its environment, implemented as a 2D geometric map of the experimental area. This map, coded following a vector format compatible with the standard Geographical Information Systems (GIS), describes a set of obstacles, while each obstacle is represented with a set of segments. However, the map is augmented with height information, thus each segment has a height component [8], necessary to compute expected laser observations considering the laser device mounting height on board the robot.

Figure 4 shows the localization output in blue and the wheel odometry position in green for a single execution on the campus. Our approach has been proved to deal robustly with the presence of unmodelled obstacles such as pedestrians, thanks to comparing expected and real laser observations, without any feature extraction step. The proposed PF has achieved an execution rate of 3Hz. Some issues are encountered dealing with ramps and other three dimensional elements, which are discussed in section IX.

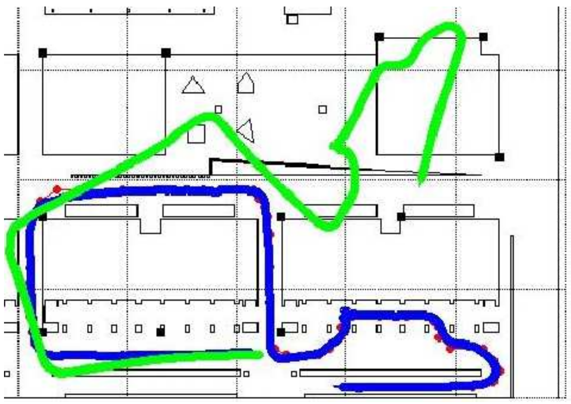

Fig. 4. Localization estimate (blue) and odometry (green) during an autonomous navigation test in the Campus experimental area. Dotted lines form a 20m×20m grid.

V. PATH PLANNING

The path planning module computes global paths between the platform and its goal on a static map of the environment, so that the output of this module is a set of waypoints in the map coordinate frame. It is executed when a new global goal is requested. The module takes as input a global cost gridmap, as well the physical properties of the robot such as its size and kinematic constraints. The cost map we have used in our experiments is the distance transform, computed from a binary map of the static obstacles. The planner computes a search graph in which nodes represent robot poses and graph edges represent collision-free motion arcs fulfilling robot's kinematics. In order to limit the search space, graph expansion is performed using a fixed arc-length and a discrete number of arc curvatures. The graph is explored using the A^* algorithm, where the heuristic is the naïve grid-distance to the goal, computed on the cost map using Dijkstra's algorithm. It is worth noting that using a fixed arc length and angle discretization usually implies that the plan is not able to reach the exact goal pose, making it necessary to use an acceptance threshold. Tibi and Dabo have used a threshold of 30cm, precise enough for our application.

VI. PATH EXECUTION

The mission of the path execution algorithm is to have the robot follow a series of waypoints, as provided by the path planning algorithm, into a smooth trajectory, even with the presence of unmapped obstacles. Most path planners [9] had the peculiarity that, when a path is clear of obstacles, they generate a fairly straight path with rather separated waypoints. With obstacles in the way, however, there are several twists in the resulting path and the waypoints are very close to one another. Inspired by that fact, te proposed procedure uses circle-based search regions to smoothly correct deviations from the path. This algorithm carries a low computational load. Figure 5 shows a sample situation and the following pseudo code overviews the algorithm.

Algorithm 1 Path execution

```
Require: $path$ and $positionEstimate$
  $circleList = createCircleList(path)$
  $index = 1$
  repeat
    for ($i = index + 1$; $i >= index - 1$; $i - -$) do
      if $positionEstimate$ is inside $circleList[i]$ then
        $globalGoal = circleList[i + 1].center()$
        $index = i + 1$
        $localGoal = globalGoal.Tr(positionEstimate)$
        send $localGoal$ to $obstacleAvoidance$
        break
      end if
    end for
  until $positionEstimate$ is not inside $circleList[last]$
```

When a new path is provided by the planning module, the path execution algorithm generates a list of circles with center at each waypoint and radius the distance to the following point. The algorithm then searches the circles around the last known robot position estimate. If the robot is inside one of those circles, the center of the next circle is sent as the new local goal to the obstacle avoidance. Otherwise, we check if the robot is inside a recovery zone, defined as the mean distance between points around the path. If this is the case the robot is sent to the last known goal; otherwise, a request is sent to the path planner for a new path. Figure 5 shows the procedure.

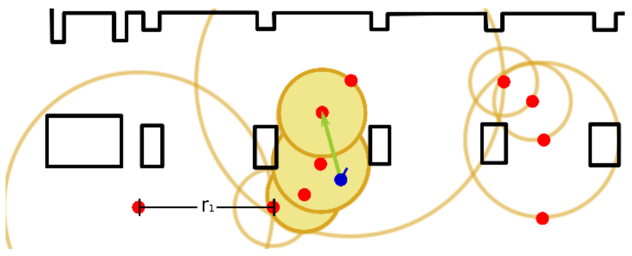

Fig. 5. Waypoints provided by the path planner in red, path execution circle list in yellow with search circles filled, robot position estimate in blue, and green arrow pointing the new local goal.

VII. OBSTACLE AVOIDANCE

The motion planning problem is well studied when using a priori information [9]. However, many techniques are not applicable when the environment is unknown or highly dynamic. This problem is compounded by the fact that both the environment and the robot carry uncertainties due to sensing and actuation, so that it is not feasible to treat motion planning separately from its execution. To solve these problems, sensory information is required in the planning and control loop, enabling for reactive navigation. A widely-used, real-time approach, based on the artificial potential field, was presented in [10] and extended by [11]. Other methods extract higher-level information from the sensor data, such as [12], based on inferring regions from geometrical properties. While these methods don't take into account the physical properties of the platform, two common approaches doing so are the curvature velocity method [13] and the dynamic window (DW) approach [14].

Our proposal combines a local planner with a slightly modified DW, producing motion control commands suitable for the platform. Decoupling planning and execution is a common practice in obstacle avidance (OA), as the full path planning problem is typically too complex for real-time processing. Inputs to the local planner are the local goal, X_g^r, provided by the path execution module, and sensor data: front laser $o_{L_H}^t$ and odometry o_U^t. The output of the local planner is an obstacle-free goal, denoted by $X_g^{r'}$. This goal is the input for the motion controller, which computes commands for translational and rotational speeds.

While this approach is sufficient for traversing flat areas, some modification must be made before it can handle an environment containing ramps (which must be overcome), and staircases or drops (which must be avoided). This problem cannot be solved with a single laser scanner, since the entry to a ramp is seen from its base as a wall at a distance determined by the laser's mounting height. This is compounded by the fact that our robots are statically unstable and pitch forward or backward for self-balancing, most notably when entering or traversing a ramp, reducing laser visibility up to 1m. Therefore, we propose an additional component to the OA module, in charge of performing traversability inference over the data provided by the vertical laser $o_{L_V}^t$. We introduce the local planner and motion controller first, and later present the traversability inference method. Figure 6 shows the OA block diagram.

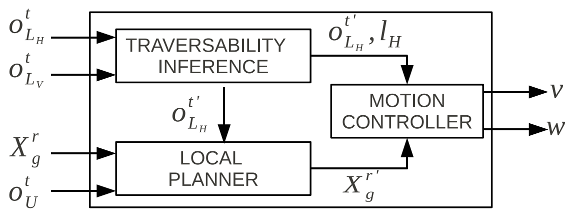

Fig. 6. Obstacle Avoidance block diagram.

A. Local RRT-based planner

The local planner has been implemented using a Rapidly-exploring Random Tree (RRT) [16]. RRT's explore a workspace by incrementally building a tree, creating new branches by generating points randomly in the workspace and linking them to the closest point already in the tree for which an obstacle-free line of sight exists. The front laser is used to determine the presence of obstacles, and the search space is restricted to sectors of a circle centered on the robot with radius the distance to the goal. The sectors are initially restricted in angle, increasing the search space each iteration until reaching the maximum scanner's aperture (190^o). Radius limit is necessary to avoid faulty solutions caused by the sensors' inherent weakness to occlusions, and the path execution module must provide a goal within range. If the local planner is unable to find a solution, the robot stops and upper-level modules are notified. Otherwise, the path is smoothed and the tree discarded, and the first point in the path is provided to the motion controller as the new local goal $X_g^{r'}$. This process is depicted in figure 7. Please note that we do not perform any kind of feature extraction.

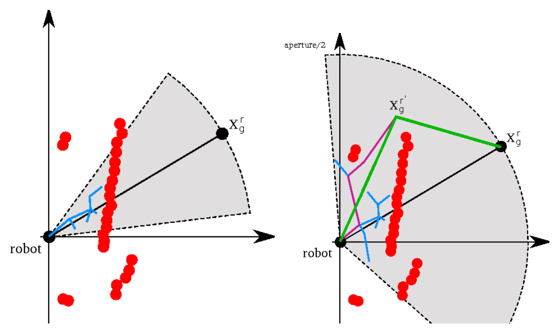

Fig. 7. Local RRT planner with incremental search space in gray. Obstacles, as the front horizontal laser scan points with a clearance, in red. On the left the tree, in blue, after a few iterations. On the right the solution in purple and the final, smoothed path in green.

We do not allow the robot to move backwards. This is because (1) our Segway robots can rotate 180^o with ease, and (2) as will be described later in this section, the third laser scanner, pointing forwards, is required to safely navigate certain features of the experimental zone.

B. Motion controller

Our motion controller is based on the DW approach [14]. This method circumvents the complexity of the full path planning problem by considering small time increments periodically, at a high rate. The approach considers only those configurations reachable within the allotted time frame for the current state (the *dynamic window*), implicitly complying with the robot's dynamic constraints. This space is then discretized into a number of cells, for which an objective function is maximized. This function considers a trade-off between velocity, target heading and clearance to obstacles:

$$G(v,\omega) = \alpha_v f_v(v,\omega) + \alpha_\theta f_\theta(v,\omega) + \alpha_c f_c(v,\omega) \qquad (2)$$

The clearance cost function measures the time until collision for the cell's configuration, relative to the platform's breaking time, as described in [15], while the cost functions for velocity and heading measure closeness to the configurations that maximize translational velocity and minimize the angle to the goal, respectively. The platform's dynamics must be incorporated into the velocity cost function so as not to overshoot the goal, and into the target heading cost function to avoid an oscillating behavior. We achieve this by defining:

$$\begin{aligned} T_{stop}^V &= \frac{V_T}{a_T max};\ T_{goal}^V = \frac{d_{goal}}{V_T};\ K_T = \frac{T_{goal}^V}{T_{stop}^V} \\ T_{stop}^R &= \frac{V_\theta}{a_\theta max};\ T_{goal}^R = \frac{\theta_{goal}}{V_\theta};\ K_R = \frac{T_{goal}^R}{T_{stop}^R} \end{aligned} \qquad (3)$$

So that K_T and K_R give us a measure of the difficulty of stopping the robot in either case. We determine adequate acceleration rates for different values of K_T and K_R and use them to devise a control law experimentally, as we find it too complex to fully model the behavior of the Segway robots. The control law and current state provide us with target values for the velocity and heading cost functions, which are a measure of closeness to these values. The functions are weighted as follows: $\alpha_v = 1$, $\alpha_\theta = 2$, $\alpha_c = 5$.

The reactive loop runs at a higher frequency than the deliberative loop, and so while the path execution module does not provide a new, updated goal, the OA algorithm updates its current goal by odometry data only.

C. Dealing with ramps

False obstacles due to ramps may be filtered by incorporating the localization estimate and using the GIS map to identify the situation, but we feel this solution would greatly compromise the robustness of our navigation system. Thus, our approach is based on an additional laser scanner placed at the robot's waist, rotated 90^o on its side so that it scans the vertical plane in front of the robot. This laser scanner is used to infer whether the robot can traverse this region of space, and enables us to detect some obstacles outside the field of view of the horizontal laser as well.

The campus features three different kinds of traversable surfaces: flat, sloped with a relatively even incline, and transitions from one to the other. The sensor's observations can thus be modeled with one or two line segments. Linear regressions are extracted from the sensor data by least squares fitting, using the average regression error to determine its quality. Prior to this computation, the vertical laser scan is pre-processed by removing points beyond the OA's algorithm range (8m) or due to interference with the robot chassis. The process is divided into three steps, executed in order, and it is terminated whenever one of these steps produces a satisfactory solution. We consider: (1) a single regression using all valid data, (2) two regressions, using all data sorted over the horizontal axis and divided into two sets by a threshold, for a set of thresholds over the horizontal axis, and (3) a single regression, iteratively removing the points furthest away from the robot over the horizontal axis until conditions are met. In all cases a maximum regression error and a minimum regression length must be satisfied. In case 2, two additional conditions are enforced in order to ensure the compatibility between segments: the vertical gap and the angular difference between regressions must be sufficiently small. Thresholds are determined empirically for the available sensors at the campus environment.

This inference process enables the robots to enter and traverse ramps by removing points from the front laser's scan incorrectly indicating the presence of obstacles prior to local planning (see figure 8). Staircases are easy to discriminate when seen from the bottom. From above, the laser's accuracy presents a problem, resulting in observations similar to those of a ramp, and the slope's steep incline is then used to identify the surface as not traversable. The length of the traversable surface is used as an additional parameter for the motion controller, limiting the translational speed or directly stopping the robot in close proximity of an obstacle.

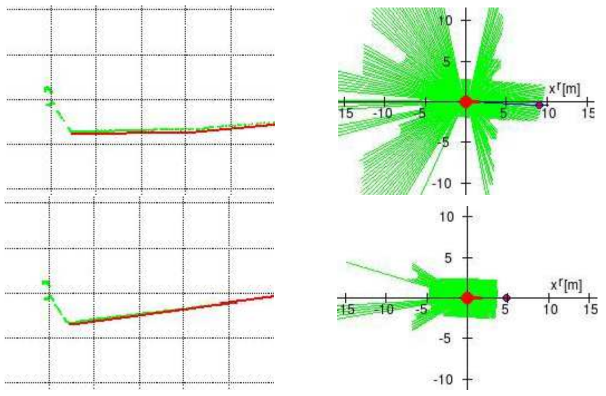

Fig. 8. Vertical (left) and frontal (right) laser scans for a Segway robot in front of (top), and in the middle of (bottom), a ramp. Regressions are shown in red, where every square side equals $1m$. In either case, the local goal indicated in the picture cannot be reached without traversability inference.

Traversability inference is computed in around $1 - 20$ms depending on the number of steps. The local planner typically takes $1 - 10$ms. The computational cost for the DW depends on its granularity, taking for instance $20 - 50$ms for 15×15 cells. The OA loop is conditioned by the sensors update rate and runs at 10Hz.

VIII. ASSESSMENT OF RESULTS

The navigation system was validated in a series of experiments conducted over two days. All the processes shown in figure 3 were running on a DELL XPS 1313 laptop on board the robot. An external laptop connected through wifi to the on board computer was dedicated to send the *go to* requests and for online monitoring. Requests were pairs of XY coordinates selected manually on the map, typically covering large distances over the campus. 34 requests were handled, 27 of which were successfully accomplished by the robot. Total runtime of both experiments was about 155 minutes, with the robot actually navigating for about 87 minutes and covering an estimated distance of over 3.5km. These experiments were conducted on a real, populated setting on a working day, so the robot encountered many obstacles (mostly people) in the way and 13.7% of navigation time was spent in obstacle avoidance mode. Failing requests were due to incorrect localization (5), a software bug (1) and obstacle avoidance issue (1). During each experimental day, the processes were only initialized at the start of the experimental session, and only the localization module was reinitialized after a localization failure. All but one of the instances where the robot got lost correspond to navigation on ramps, where perception is severely impaired since the balancing behavior of the platform forces the horizontal laser to point directly to the ramp surface. The obstacle avoidance failure happened when the robot was dangerously close to a glass door, unseen by the lasers due to illumination issues, and the red emergency button was prudently pushed.

Table I lists the experiments, detailing the total navigation time until reaching the goal, the time spent on active obstacle avoidance, and the final status of the request. Table II summarizes the results. A short video is attached to this paper showing the robot following a path while avoiding obstacles.

TABLE I
DETAILED RESULTS OF THE 34 *Go To* REQUESTS

Nav. time (s)	OA time (s)	Status	Nav. time (s)	OA time (s)	Status
51.8.	0.5	Loc. Fail	95	12	Sucess
260.1	51.3	Success	342	51.8	SW bug
137.2	10.3	Success	125	18.8	Success
159.8	11.3	Success	64	52.7	Loc. Fail
89.6	60.5	OA Fail	97	5.9	Success
190.0	8.5	Success	175	7.8	Success
205.2	4.0	Success	246	19.2	Success
84.1	2.0	Success	212	45.2	Success
128.1	0	Success	119	27.3	Success
164.6	40.5	Loc. Fail	160	34.2	Success
223.2	16.2	Success	154	22.5	Success
176.1	2.7	Success	93	37.7	Success
179.6	6.5	Success	100	4.2	Success
276.2	31.5	Success	116	13.8	Success
127.7	4.2	Success	132	66.8	Loc. Fail
125.2	1.5	Success	104	6.5	Success
136.7	17.3	Loc. Fail	163	16.8	Success

TABLE II
SUMMARY OF THE EXPERIMENTAL SERIES

Number of *go to* requests	34
Success Rate	27/34 (79.4%)
FAILS	5(loc), 1(SW bug), 1(OA)
Total Run Length	3517 m
Total Navigation Time	86.8 min
Time Avoiding Obstacles	720 s (13.8%)
Mean Travel Speed	0.67 m/s

IX. LESSONS LEARNT AND FUTURE WORKS

This paper overviews a full autonomous navigation approach designed for mobile robots operating in urban pedestrian areas. The paper also shows the results after a session of 34 *go to* requests sent to the robot, detailing the cause when errors occurred. The experimental sessions have been useful to validate the proposed framework, accomplishing about 3.5km of autonomous navigation in a pedestrian area during working days. The software architecture, based on independent processes connected through TCP ports, has been proved to be very useful both for navigation purposes and during debugging sessions. The experiments have also highlighted the critical issues to be improved in future works.

At the estimation level, the localization module must be improved, since 5 errors are directly provoked by a localization failure. The proposed approach is based on a particle filter estimating the 2D position of the robot, using a 2D model of the environment to compute the expected observations for particle correction. This approach has met the real-time requeriments dealing with a map of an urban area of 10000m^2. However, in such urban pedestrian areas, where 3D elements are usually found, the computation of the expected observations in a 2D model limits their reliability, specially when the robot navigates on ramps or close to them, since most of the laser scanner rays are colliding with the ramp surface. This issue is even more critical for Segway-based platforms due the balancing behavior. Moreover the balancing movement gives the robot an aditional 3D perspective of the environment poorly modelled with the 2D approach. Future works in terms of localization should be based on a 3D estimation of the robot position and the development of 3D environment models and fast algorithms to compute expected laser scans.

At the motion control level, and due to safety reasons, obstacle avoidance is the most critical module. The proposed approach for OA also meets the computational time requeriments but remains very limited in terms of perception. Using two planar laser scanners, sweeping the vertical and horizontal planes in front of the robot, creates dangerous blind areas, leaving the navigation system susceptible against obstacles such as holes or steps. This limitation could be improved using an hybrid laser-camera approach, sensing the whole forward cone of the robot. Such an approach should meet real-time constraints while facing the challenging illumination situations encountered in outdoor environments.

REFERENCES

[1] http://urus.upc.edu
[2] J. Levinson, M. Montemerlo and S. Thrun, Map-Based Precision Vehicle Localization in Urban Environments, *in Proceedings of the Robotics: Science and Systems Conference*, Atlanta, USA. June, 2007.
[3] M. Montemerlo *et al.*, Junior: The Stanford entry in the Urban Challenge. *Journal of Field Robotics*, Vol. 25, No. 9, 2008.
[4] T. Luettel, M. Himmelsbach, F. V. Hundelshausen, M. Manz, A. Mueller and H.-J. Wuensche. Autonomous Offroad Navigation Under Poor GPS Conditions, *in Proc. of the IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles. St Louis, MO, USA. October 2009.*
[5] G. Metta, P. Fitzpatrick and L. Natale, YARP: yet another robot platform, *Int. Journal on Advanced Robotics Systems*, vol 3(1), 2006.
[6] A. Corominas Murtra, J.M. Mirats-Tur, O. Sandoval and A. Sanfeliu, Real-time Software for Mobile Robot Simulation and Experimentation in Cooperative Environments, *Lecture Notes on Artificial Intelligence 5325. Springer ed.* , SIMPAR08, Venice, Italy. November, 2008.
[7] S. Thrun, D.Fox, W. Burgard and F. Dellaert, Robust Monte Carlo localization for mobile robots, *Artificial Intelligence*, vol. 128, 2001.
[8] J.M. Mirats Tur, C. Zinggerling and A. Corominas Murtra, Geographical information systems for map based navigation in urban environments, *Robotics and Autonomous Systems*, 57(9). 2009.
[9] J.C. Latombe, Robot Motion Planning, Kluwer Academic Pub., 1991.
[10] O. Khatib, Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, *International Journal of Robotics Research*, 5(1), 1986.
[11] M. Khatib and R. Chatila, An extended potential field approach for mobile robot sensor-based motions, *Proc. of Intelligent Autonomous Systems*, 1995.
[12] J. Minguez, L. Montano, Nearness Diagram (ND) Navigation: Collision Avoidance in Troublesome Scenarios, *IEEE Transactions on Robotics and Automation*, Vol. 20, No. 1, February 2004.
[13] R. Simmons, The curvature-velocity method for local obstacle avoidance, *IEEE Int. Conf. on Robotics and Automation*, vol 4, 1996.
[14] D. Fox, W. Burgard, S. Thrun, The dynamic window approach to collision avoidance, *Robotics & Automation Mag.*, Vol. 4(1), 1997.
[15] R. Philippsen, Motion planning and obstacle avoidance for mobile robots in highly cluttered dynamic environments, PhD Thesis, École Polytechnique Fédérale de Lausanne (ref. 3146), 2004.
[16] S. LaValle and J. Ku, Rapidly-exploring random trees: Progress and prospects, *Algorithmic and Computational Robotics: New Directions. Fourth Workshop on the Algorithmic Foundations of Robotics*, 2001, pub. AK Peters, Ltd.