Autonomous Navigation for Mobile Service Robots in Urban Pedestrian Environments

• •

E. Trulls, A. Corominas Murtra, J. Pérez-Ibarz, and G. Ferrer
Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Barcelona, Spain
e-mail: etrulls@iri.upc.edu, acoromin@iri.upc.edu, jnperez@iri.upc.edu, gferrer@iri.upc.edu

D. Vasquez
Swiss Federal Institute of Technology, Zürich, Switzerland
e-mail: vasquez@mavt.ethz.ch

Josep M. Mirats-Tur
Cetaqua, Centro Tecnológico del Agua, Barcelona, Spain
e-mail: jmirats@cetaqua.com

A. Sanfeliu
Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Barcelona, Spain
e-mail: sanfeliu@iri.upc.edu

This paper presents a fully autonomous navigation solution for urban, pedestrian environments. The task at hand, undertaken within the context of the European project URUS, was to enable two urban service robots, based on Segway RMP200 platforms and using planar lasers as primary sensors, to navigate around a known, large (10,000 m^2), pedestrian-only environment with poor global positioning system coverage. Special consideration is given to the nature of our robots, highly mobile but two-wheeled, self-balancing, and inherently unstable. Our approach allows us to tackle locations with large variations in height, featuring ramps and staircases, thanks to a three-dimensional, map-based particle filter for localization and to surface traversability inference for low-level navigation. This solution was tested in two different urban settings, the experimental zone devised for the project, a university campus, and a very crowded public avenue, both located in the city of Barcelona, Spain. Our results total more than 6 km of autonomous navigation, with a success rate on *go-to* requests of nearly 99%. The paper presents our system, examines its overall performance, and discusses the lessons learned throughout development. © 2011 Wiley Periodicals, Inc.

1. INTRODUCTION

Large, modern cities are becoming cluttered, difficult places to live in, due to noise, pollution, traffic congestion, security, and other concerns. This is especially true in Europe, where urban planning is severely restricted by old structures already laid out. Ways to alleviate some of these problems include enhancements to public transportation systems and car-free areas, which are becoming common in city centers. In May 2010 New York City closed to motor vehicles two key sections of midtown, Times Square and Herald Square, after a pilot program in 2009 that reduced pollution, cut down on pedestrian and bicyclist accidents, and improved overall traffic by rerouting. A Green Party initiative to close to vehicles 200 streets in the center of Geneva, Switzerland, had been approved in principle in early 2010. Barcelona already features an iconic hub in La Rambla, a prominently pedestrian-only thoroughfare more than 1 km in length running through the historic center of the city.

It is expected that urban service robots will be deployed in such areas in the near future, for tasks such as automated transportation of people or goods, guidance, or surveillance. The study of these applications was a basic requirement of URUS: Ubiquitous networking Robotics in Urban Settings (Sanfeliu & Andrade-Cetto, 2006; URUS,–) (2006–2009), a European IST-STREP project of the Sixth Framework Programme, whose main objective was to develop an adaptable network robot architecture integrating the basic functionalities required to perform tasks in urban areas. This paper is concerned with autonomous navigation for a mobile service robot in pedestrian environments.

In recent years significant advances have been experienced in the area of autonomous navigation, especially thanks to the efforts of the scientific and engineering teams participating in the DARPA Urban Challenge (Montemerlo, Becker, Bhat, Dahlkamp, Dolgov, et al., 2008; Rauskolb, Berger, Lipski, Magnor, Cornelsen, et al., 2008), as well as other contests (Luettel, Himmelsbach,

Multimedia files may be found in the online version of this article.

Hundelshausen, Manz, Mueller, et al., 2009; Morales, Carballo, Takeuchi, Aburadani, & Tsubouchi, 2009). Even if most of 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, especially in terms of navigation architecture and software integration. However, urban pedestrian areas present additional challenges to the robotics community, such as narrow passages, ramps, holes, steps, and staircases, as well as the ubiquitous presence of pedestrians, bicycles, and other unmapped, dynamic obstacles. This leads to new challenges in perception, estimation, and control. For instance, global positioning system (GPS)-based systems remain an unreliable solution for mobile robots operating in urban areas, due to coverage blackouts or accuracy degradation (Levinson, Montemerlo, & Thrun, 2007; Yun & Miura, 2007), so that additional work is necessary for robot localization.

This paper presents a fully autonomous navigation solution for urban service robots operating in pedestrian areas. In this context, the navigation framework will receive *go-to* queries sent by some upper level task allocation process or directly by an operator. The *go-to* query will indicate a goal point on the map coordinate frame. The system is designed as a collection of closely interrelated modules. Some of them have been applied successfully on other robots during the URUS project demonstrations, whereas the lower level modules are geared toward our Segway robots and take into account their special characteristics. The main contribution of this paper is the presentation of a set of techniques and principles that jointly yield a valuable experimental field report: (1) the consideration of real-world urban pedestrian environments, with inherent features such as ramps, steps, holes, pedestrians, and other dynamic obstacles; (2) the use of Segway-based platforms, which provide high mobility but create perception and control issues successfully addressed by our approach; (3) real-time three-dimensional (3D) localization, without relying on GPS, using state-of-the-art techniques for online computation of expected range observations; (4) the successful integration of all navigation software modules for real-time, high-level actions; and (5) extensive field experiments in two real-world urban pedestrian scenarios, accomplishing more than 6 km of autonomous navigation with a high success rate.

The paper is organized as follows. Section 2 describes the locations where the experiments were conducted. Section 3 presents the robots at our disposal and the sensors onboard. Section 4 presents the architecture of the navigation system. Sections 5 and 6 present our path planning and path execution algorithms. Section 7 summarizes the localization algorithm, a 3D map-based particle filter. Section 8 is concerned with our low-level navigation module, an obstacle avoidance (OA) system capable of dealing with terrain features such as ramps. Field results are summarized in

Section 9, and Section 10 presents the main lessons learned by the scientific team and identifies critical aspects to work on in the future.

A previous version of this work was presented in Corominas Murtra, Trulls, Sandoval, Pérez-Ibarz, Vasquez, et al. (2010). A new localization algorithm, using full 3D information, and several improvements on the path execution and OA modules allowed us to increase our success rate on *go-to* requests from 79% to nearly 99%. We also present experiments in two urban areas instead of one. All experimental data presented in this paper are new.

2. SITES AVAILABLE FOR EXPERIMENTATION

Most of the experiments were conducted at the Campus Nord of the Universitat Politècnica de Catalunya (UPC), located in Barcelona, where a large section was outfitted as an experimental area (Barcelona Robot Lab) for mobile robotics research. This installation covers more than 10,000 m^2 and is equipped with wireless coverage and 21 Internet protocol (IP) video cameras. Our robots are currently self-contained, using only onboard sensors for navigation. A more thorough overview of the capabilities of this lab is available in Sanfeliu, Andrade-Cetto, Barbosa, Bowden, Capitán, et al. (2010). Additional experiments were carried out at Passeig de Sant Joan in the district of Gràcia, also in the city of Barcelona, with the cooperation of the city's administration. This area comprised a 1,900-m^2 section of a pedestrian-only boulevard, with bicycle lanes on both sides and a large monument in the middle. Figure 1 shows an aerial view of both locations.

The main experimental site, the campus, is situated in a hilly region, close to a mountain range. It features differences in height of up to 10 m within the experimental area, resulting in ramps—which the robot must be able to navigate—sudden level changes and staircases—which should be avoided—and other obstacles such as bulletin boards, bicycle stands, trashcans, and flower pots. A geographic information system (GIS) map with elevation data was built for the project and later extended to a full 3D model. All tests made during the robots' development stage were conducted in this area. Figure 2 presents a top view of the 3D model and some photographs of the area. Campus buildings are situated over a four-row, six-column grid and labeled according to row (letters from A to D, bottom to top) and column (numbers from 1 to 6, left to right), e.g., A1 or D6. The experimental area covers the eastern part of the campus. The main features found in this area are the terrace at the bottom of the map, the FIB (Computer Faculty) square and cafeteria, and a promenade with another terrace above it, between rows B and C.

The site at Passeig de Sant Joan does not feature ramps or staircases but is again on sloped terrain, rising more than 2 m in height along 70 m of length, with a relatively even slope of nearly 2 deg. This poses a problem for
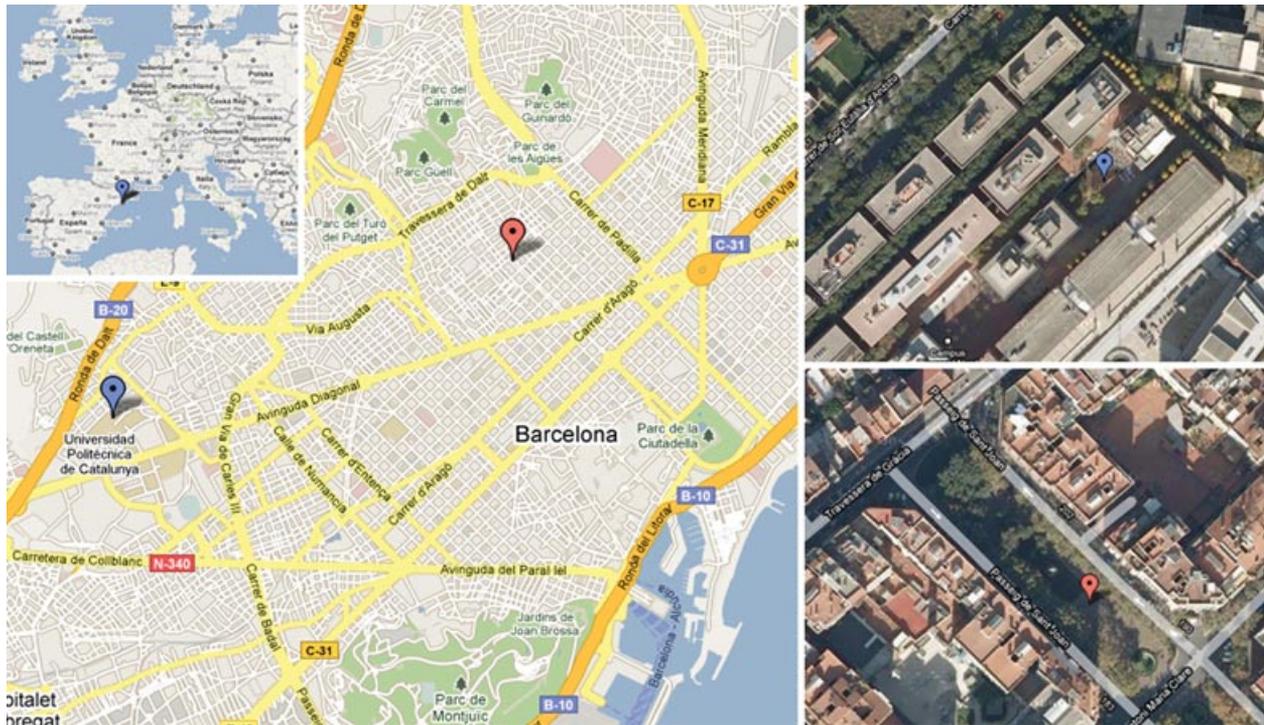
Figure 1. Campus site (blue) and Gràcia site (red).

two-wheeled robots such as ours, as will be further explained in Section 3. It is of particular interest that there are few clear landmarks such as walls, as most of the area is encircled by hedges on either side, and the monument in the middle was at the time (spring) surrounded by rose bushes. Whereas the campus scenario is somewhat controlled and mostly populated by students, the Gràcia environment is a very crowded public street in the middle of a large city, frequented by pedestrians, children, and bicyclists. A 3D model of the area was built from scratch, in much lesser detail. The area is pictured in Figure 3. We placed four fences below the monument for safety reasons, which were included in the 3D map. Later we had to put other fences in place to reroute part of the traffic, but these were not included in the map.

3. ROBOTS

Two mobile service robots, designed to operate in urban, pedestrian areas, were developed for the URUS project. These are Tibi and Dabo, pictured in Figure 4. They are based on two-wheeled, self-balancing Segway RMP200 platforms and as such are highly mobile, with a small footprint, a nominal speed up to 4.4 m/s, and the ability to rotate on the spot (while stationary).

They are equipped with the following sensors:

- Two Leuze RS4 two-dimensional (2D) laser range finders, scanning over the local XY plane, pointing forward

and backward, respectively, at a height of 40 cm from the ground. These scanners provide 133 points over 190 deg at the fastest setting, running at approximately 6 Hz. This device has a range of 64 m, but in practice we use a 15-m cap. Front and back laser observations are notated as $o^t_{L_F}$ and $o^t_{L_B}$, respectively.
- A third 2D laser scanner, a Hokuyo UTM-30LX, mounted at a height of 90 cm, pointing forward and rotated 90 deg over its side, scanning over the local XZ plane. This scanner provides 1,081 points over 270 deg at 40 Hz and has a range of 30 m, again capped to 15 m. Aperture is limited to 60 deg to ignore points interfering with the robot's frame or aiming too high for our needs. This observation is notated as $o^t_{L_V}$.
- Wheel encoders, providing odometry readings o^t_U from the Segway platform.
- Inclinometers from the Segway platform, providing pitch and roll data o^t_I.

The robot also features two stereo camera pairs and a GPS receiver, which are not used in this work. The user can interact with the robot through a touchscreen, entering *go-to* requests manually. Two off-the-shelf laptop computers running Ubuntu Linux are onboard the robot, one for navigation and the other for communications and human–robot interaction. Experiments were performed using only one robot at time, Tibi or Dabo.
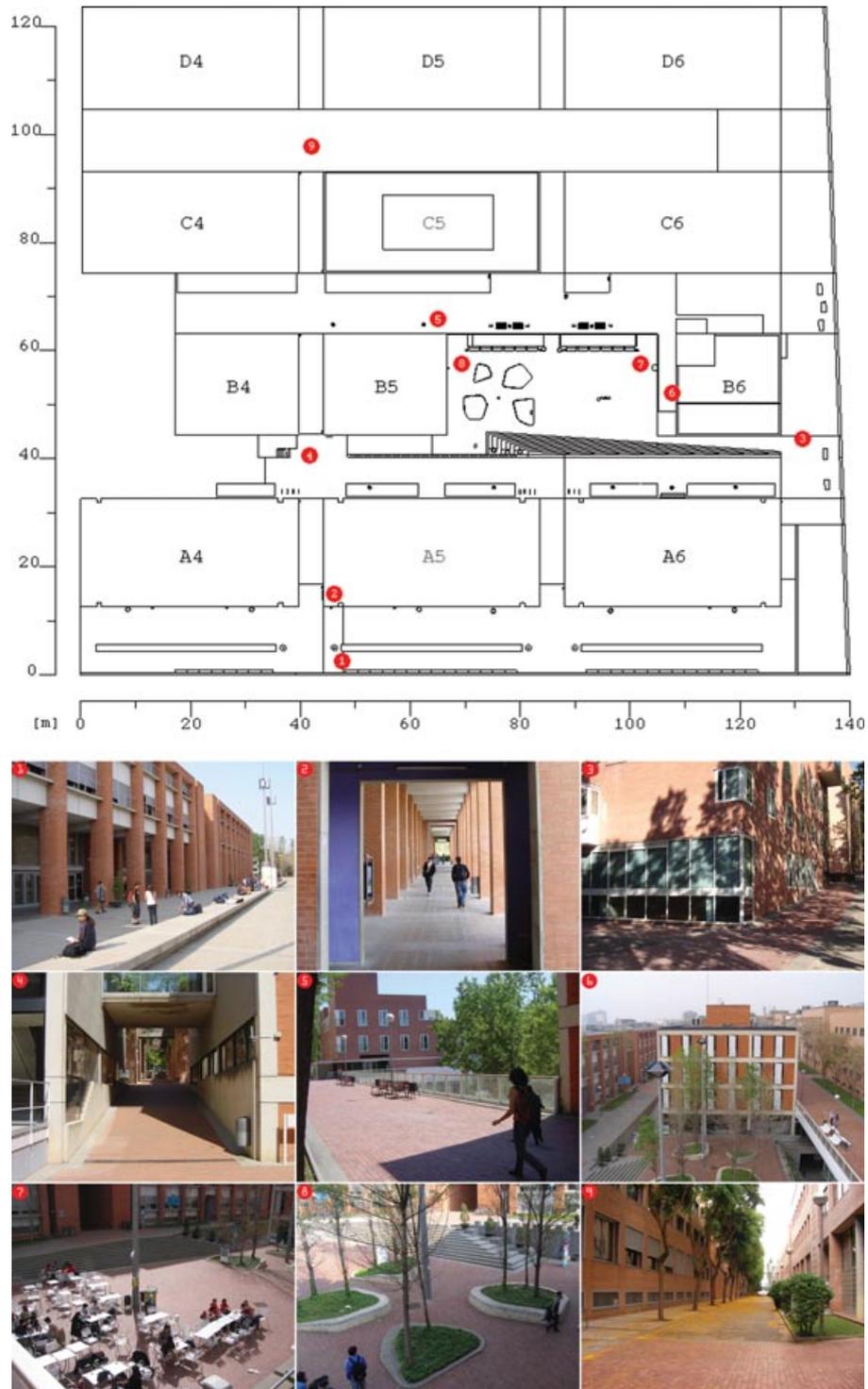
Figure 2. Top view of the 3D model for the campus site, with a set of pictures highlighting certain features of the environment. Note the many ramps (3, 4), the steps around the square (7, 8), changing obstacles such as the cafeteria terrace (7), narrow passages such as around the trees at the square (8), the prevalence of glass windows at waist and foot level (3, 4), and the transparent plastic balcony on (5). Note also the ubiquitous presence of pedestrians.
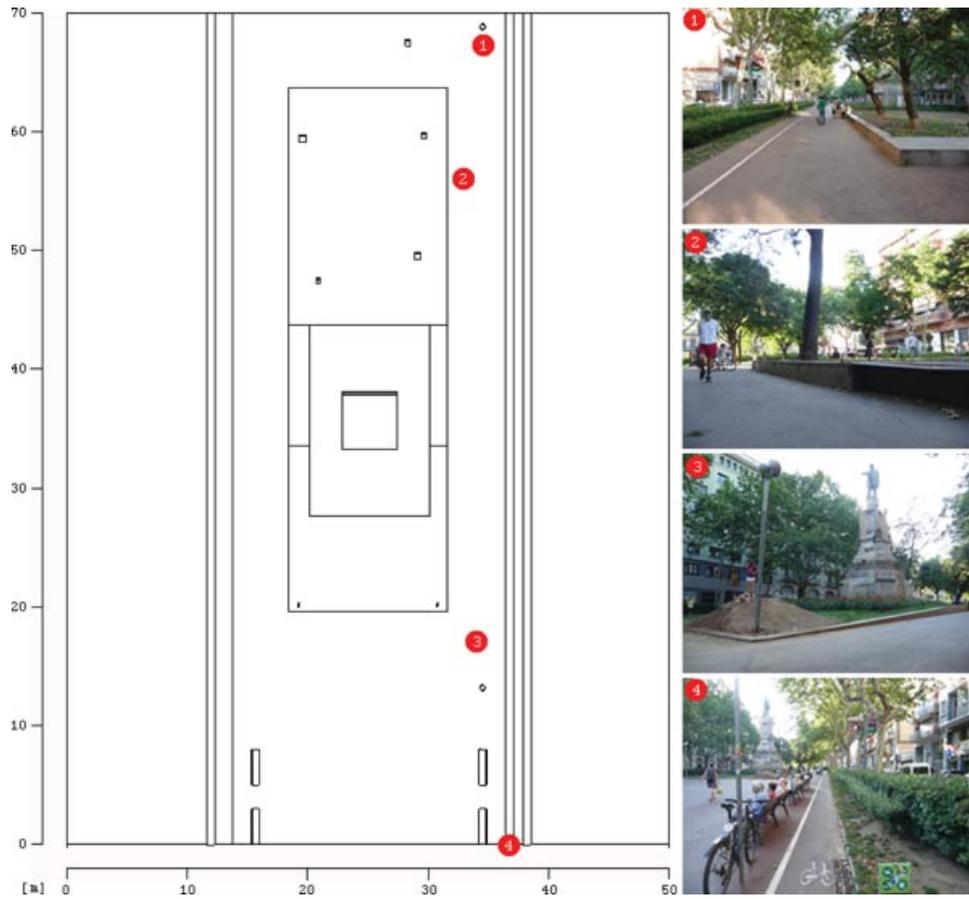
Figure 3. Top view of the 3D model for the Gràcia site, with some pictures of the area. The wide lanes on either side of the map are the road, with buildings beyond. Note the lack of clear landmarks. The site is encircled by vegetation (1, 4), and the elevation around the monument (1, 2, 3) rises to either below or just at laser height (cf. Section 3).
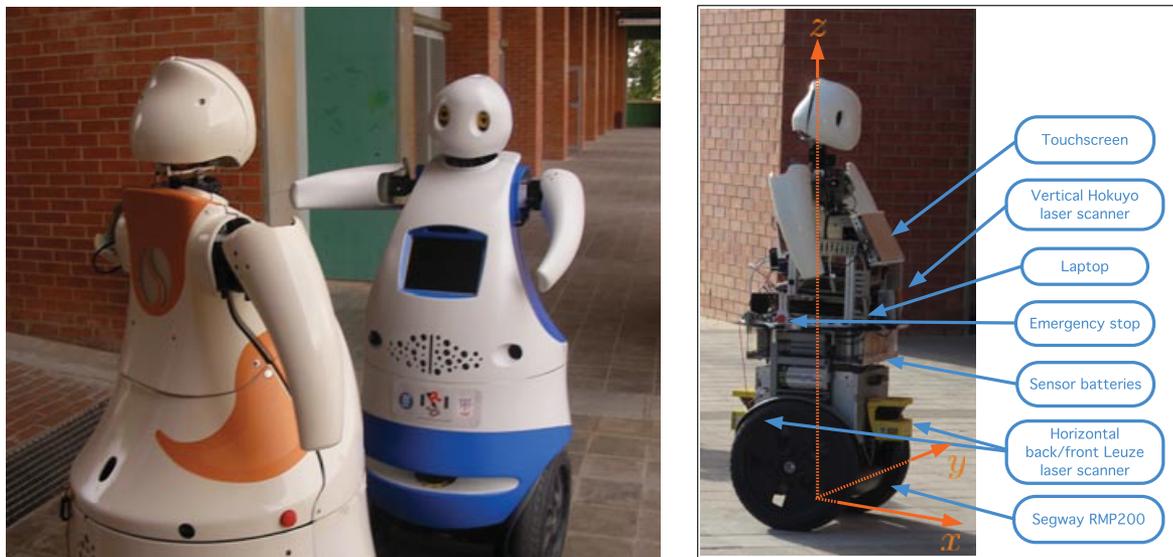
Figure 4. Left, Tibi (left) facing Dabo. Right, Onboard devices used in this work and the robot coordinate frame.
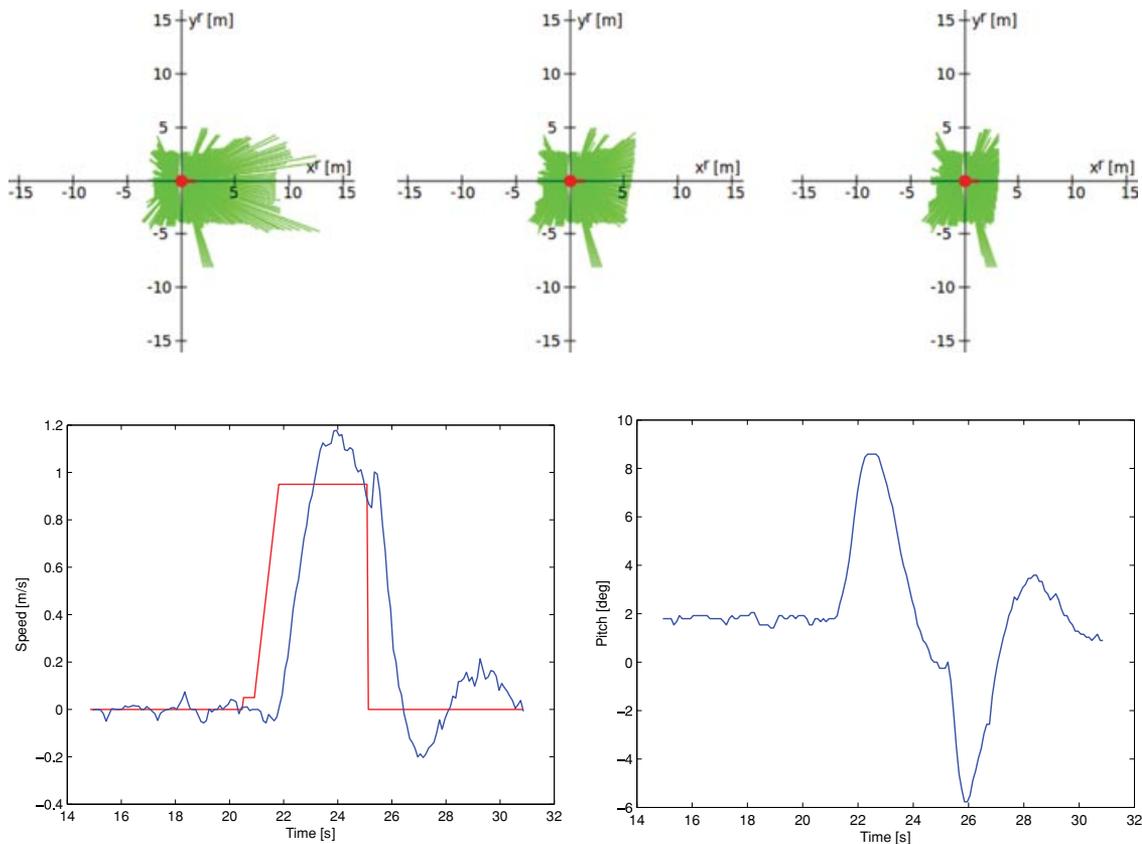
Figure 5. Top, left to right, sequence of horizontal laser scans with the robot accelerating forward on a flat surface. Time between scans is about 0.6 s. Bottom left, Commands for translational velocity (*v*) in red and its estimation from the Segway platform in blue. Bottom right, Pitch estimation from the Segway platform.

The Segway RMP200 is in many ways an ideal platform on which to build an urban robot. Humanoid robots are not yet ready for outdoor environments, and four-wheeled vehicles have a much larger footprint and are more restricted in their mobility. Moreover, Segway robots can carry heavy payloads, up to 45 kg for this model. On the downside, Segway platforms are statically unstable, keeping their balance using embedded gyroscopic sensors to track and correct their tilt. The robot will pitch forward or backward to accelerate or decelerate, or simply to keep its balance while stationary. This behavior presents two issues for their use in robotics.

On one hand, it creates a perception issue for onboard 2D laser scanners. A 2D laser range finder scanning over the local *XY* plane, a very common solution in robotics for navigation or simultaneous localization and mapping (SLAM), may point higher toward the sky/roof or, more critically, lower toward the ground. Using this configuration may result in spurious features or obstacles unless some kind of filtering is used. Figure 5 displays a sequence of 2D range data over time, starting with the robot in a sta-

tionary, upright position, which is then instructed to move forward and later to stop. The front laser visibility is reduced significantly due to the platform's tilt, up to 2 m on a flat surface and less on a ramp. The figure also shows velocity commands and the estimation for velocity and pitch from the Segway platform, for the same sequence. These data were taken under laboratory conditions, on a flat, regular surface. In outdoor environments this behavior is much more pronounced, especially on slopes and changes in slope.

The second issue in using Segway platforms is control: the platform's own control algorithm takes precedence over the user's instructions, as its first priority is to stay upright. This problem, present in all Segway platforms, is compounded by the fact that our robots weigh about 120 kg, which slows them down. In practice, the platform typically takes 1–2 s to react to the user's commands, or even more in extreme situations such as when moving from a flat surface to a slope or vice versa. This ultimately means that it is not possible to execute previously planned trajectories with a high degree of accuracy. An example can be seen in Figure 5.
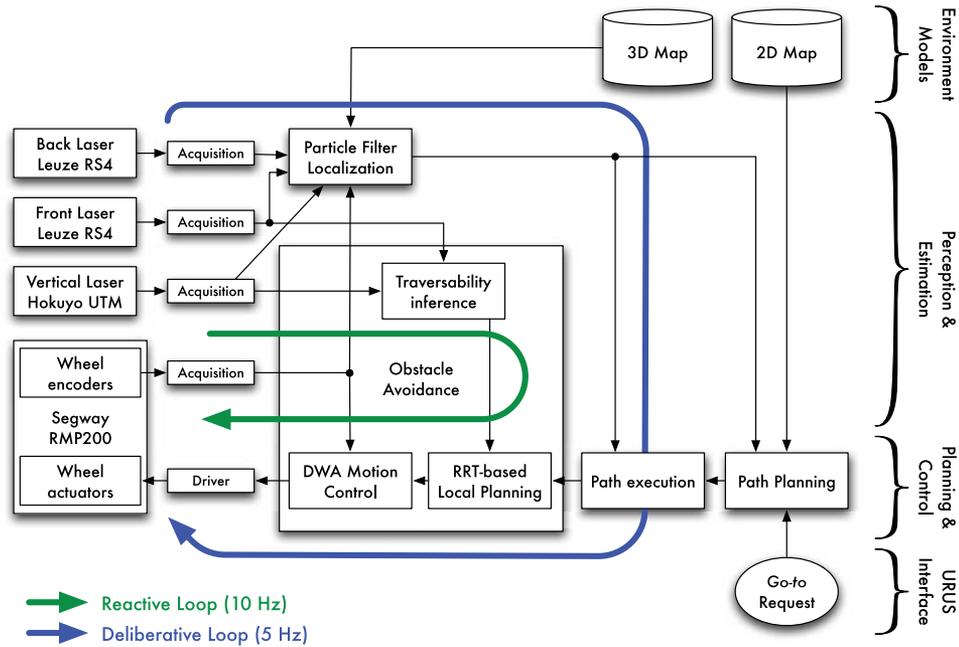
Figure 6. Process diagram for our navigation architecture. Each block is an independent process. Arrows are transmission control protocol (TCP) connections.

Our navigation solution deals effectively with these issues. To solve the perception problem caused by the platform tilt, we use full 3D information for localization and the vertical scanner to determine surface traversability for navigation. The control problem is tackled with a loose, low-level navigation scheme. These procedures are explained in further detail in Sections 7 and 8.

4. NAVIGATION ARCHITECTURE

Our complete navigation framework for Tibi and Dabo is diagrammed in Figure 6. This solution is divided into four different blocks, in decreasing level of abstraction: path planning, path execution, localization, and obstacle avoidance. The obstacle avoidance module consists of three blocks: traversability inference, local planning, and motion control.

The path planning module is tasked with finding a global path between the platform's current position and a goal upon a *go-to* request, in the form of a list of waypoints $\mathcal{W}^M = \{X_{g_1}^M \dots X_{g_{N_w}}^M\}$ in global (map) coordinates. The localization, path execution, and obstacle avoidance modules constitute two different control loops. The obstacle avoidance module is the only component in direct control of the robot's motion and constitutes by itself the *reactive* loop. Its mission is to move the robot to a local goal, expressed in the robot coordinate frame $X_{g_i}^R$. It is important to point out that this loop does not depend on the localization estimate because it ensures only that the robot will arrive

at a local goal while avoiding the obstacles perceived by onboard sensors. The second loop is *deliberative* and is tasked with guiding the robot through the waypoints computed by the path planning module. The deliberative loop includes the localization module, a map-based particle filter, and the path execution process, which uses the current localization estimate $\hat{X}_r^M$ to transform the waypoints from map coordinates $X_{g_i}^M$ to robot coordinates $X_{g_i}^R$. This local goal is the input to the obstacle avoidance module, thus closing the deliberative loop. This is a particularly suitable solution for our Segway robots, for which it is possible to execute planned trajectories only loosely.

We use two different environment models, a 2D map and a 3D map. The 3D map is a model containing the environment's static geometry. The 2D map is inherited from previous work (Corominas Murtra, Trulls, Sandoval, et al., 2010) and is required for path planning. The reactive loop runs at 10 Hz, and the deliberative loop runs at 5 Hz. Because the platform moves at speeds up to 1 m/s, these rates are deemed sufficient.

Each sensor has an associated data acquisition process. All navigation and data acquisition processes run concurrently on the same computer. The software framework follows a publish/subscribe architecture, with the aim to ease software integration between developers: each block of Figure 6 has been implemented as an independent process, accessible through an interface. The resulting specification runs over YARP as middleware, a free, open-source, platform-independent set of libraries, protocols, and tools
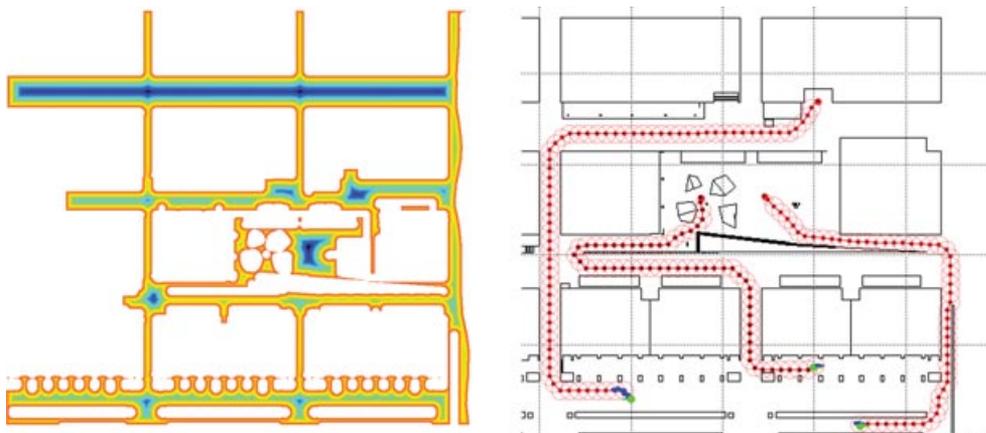
Figure 7. Left, The cost map for the UPC campus. Warmer tones indicate high costs, and white indicates unreachable places. Right, Three sample paths displayed using our simulation environment. Red dots indicate the path computed by the path planning module. Red circles correspond to the circle path, which will be introduced in Section 6. Green and blue dots correspond to the localization estimate and mark the starting position for each iteration. Further examples are available in the videos introduced in Section 9.

aimed at decoupling the transmission of information from the particulars of devices and processes in robotic systems (Metta, Fitzpatrick, & Natale, 2006). For a further description of our software architecture, please refer to Corominas Murtra, Mirats-Tur, Sandoval, and Sanfeliu (2008).

5. PATH PLANNING

Our planning algorithm has been developed in the context of the URUS project, having as a key requirement the ability to effectively deal with the diversity of platforms involved in the project. Thus we have privileged reliability and flexibility over other concerns such as online replanning. That said, it is worth noting that limited online planning capabilities are actually fulfilled by the local planning component of our architecture (cf. Section 8.1).

The global planner takes as input a global cost 2D grid map (Figure 7), as well as the physical properties of the robot such as its size and kinematic constraints. The cost we have used in our experiments is the distance transform (i.e., distance to the nearest obstacle), computed from a binary map of the static obstacles in the environment. By using such a cost, we maximize the distance between the path and the obstacles in the same way as using a Voronoi graph, with the advantage that the cost is defined also for points that are not part of the graph. Another advantage of the distance transform is that a single map can be used for coarse collision testing on all the different platforms by simply comparing the cell's value against the radius of the platform's bounding sphere.

The planner computes a search graph in which nodes represent robot poses and graph edges represent collision-free motion arcs that are deemed to be feasible according to the robot's kinematics. To limit the size of 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, in which 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 implies, in most cases, that the plan is not able to reach the exact goal pose, making it necessary to use an acceptance threshold. However, in practice this has not been a problem. We have used a threshold of 30 cm, which is precise enough for our particular application.

As stated above, this module was common to all robotic platforms in the project. Our navigation system defers online replanning to the obstacle avoidance module and has simpler requirements for global path planning: the distance between waypoints is set to 2 m for Tibi and Dabo, and we disregard heading angle data for the waypoint. Examples of its application are displayed in Figure 7.

6. PATH EXECUTION

The task of the path execution algorithm is to provide local goal points to the robot so that the trajectory computed by the global planner is followed in a smooth manner, even in the presence of unmapped obstacles that force the robot to stray out of the path. Our approach consists of defining circle-based search zones centered on the plan's waypoints. The localization estimate is then used to determine which circle the robot lies on, if any, and the next waypoint to target, which is then transformed into robot coordinates and sent to the obstacle avoidance module as a local goal.
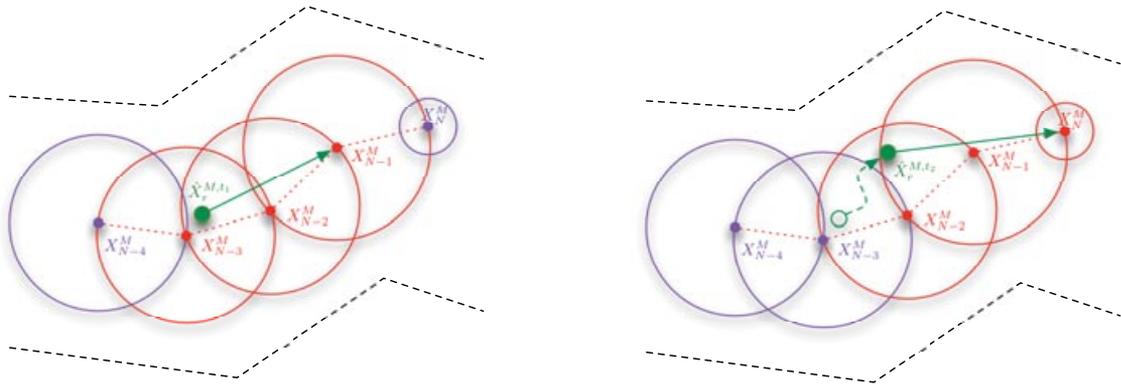
Figure 8. Illustration demonstrating the behavior of the path execution algorithm under normal operating conditions. Waypoints and the circle path are plotted in red when considered by the path execution algorithm, in purple when not. The localization estimate and an arrow signaling the current target are plotted in green. The environment (unbeknown to the path execution module) is plotted in black. On the left, the circle currently being followed is C_{N-2}, centered on waypoint X_{N-2}^M, and so the current target is X_{N-1}^M. The algorithm considers this circle and its neighbors in this order: first C_{N-1}, then C_{N-2}, and finally C_{N-3}. As the first circle that contains the localization estimate is C_{N-2}, the target does not change. On the right, the robot moves forward and enters circle C_{N-1}, so that the new circle being followed is C_{N-1} and X_N^M becomes the new target.

The circle path is created upon receiving a new path from the global planner, once per *go-to* request, and is defined as a list of circles $\{C_1 \ldots C_{N_w}\}$ with center each waypoint and radius the distance to the following waypoint. The radius for the last circle, with center the goal, is defined as the goal tolerance d_g, a global parameter set to 0.5 m. The algorithm stores an index to the circle currently being followed, k, which is initialized to 2, as $X_{g_1}^M$ is the robot's starting position. During run time, the algorithm determines whether the circle currently being followed and its adjacent circles C_{k-1}, C_{k+1} contain the localization estimate $\hat{X}_r^M$, starting from the higher index and moving down. Whenever this is true, the process stops and k is set to the index of the compliant circle. The waypoint to target is in every case the center of the next circle, C_{k+1}, which by definition lies on the circumference of C_k. That is, when a robot nears a waypoint (enters its associated circle), the goal will switch to the next waypoint (the center of the next circle). We check only the next circle to enforce smoothness and the previous circle as a safeguard against small variations on the localization estimate. This procedure is illustrated in Figure 8.

If no circle contains the localization estimate, we compute its distance to the path, defined as the shortest distance to a waypoint. If this distance is smaller than the recovery distance d_r, set to 3 m, the path execution algorithm will enter recovery mode, sending the robot to the closest waypoint. When the robot is farther away than the recovery distance, we presume that recovery is not possible, stop the robot, and request the path planning module for a new path to the same global goal. These situations are illustrated in Figure 9. Figure 7 displays examples of circle paths in the campus area.

This approach is valid for obstacle-free environments but may fail if an unmapped object rests over a waypoint that thus cannot be reached. We solve this by computing not one but an array of goal candidates $\mathcal{G}^R$, which are offered to the obstacle avoidance module as possible targets. X_j^M being the current target, we consider all waypoints $\{X_i^M | i \in [j, N_w]\}$. We take a maximum of N_{OA} points, and only candidates closer to the robot than d_{OA} are considered valid. The first candidate that violates this rule is truncated to d_{OA}, and the rest are ignored. For our implementation we use $d_{OA} = 5.5$ m and $N_{OA} = 8$. This guarantees at least three goal candidates within range. The obstacle avoidance module considers one candidate at a time, starting from the lower index, and selects the first candidate that may be reached, as explained in Section 8.

7. MAP-BASED LOCALIZATION

Localization is the process in charge of closing the deliberative loop (Figure 6), thus allowing the path execution module to convert goal points from the global planner, in map coordinates, to local goal points, in robot coordinates. Localization plays a key role in autonomous navigation for mobile robots, and a vast amount of work can be found in the literature. It is accepted by mobile robot researchers that GPS-based solutions are not robust enough in urban environments due to insufficient accuracy and partial coverage. This fact has forced the mobile robotics community to design alternative or complementary methods for localization (Georgiev & Allen, 2004; Levinson, et al., 2007; Nuske, Roberts, & Wyeth, 2009; Thrun, Fox, Burgard, & Dellaert, 2001; Yun & Miura, 2007).
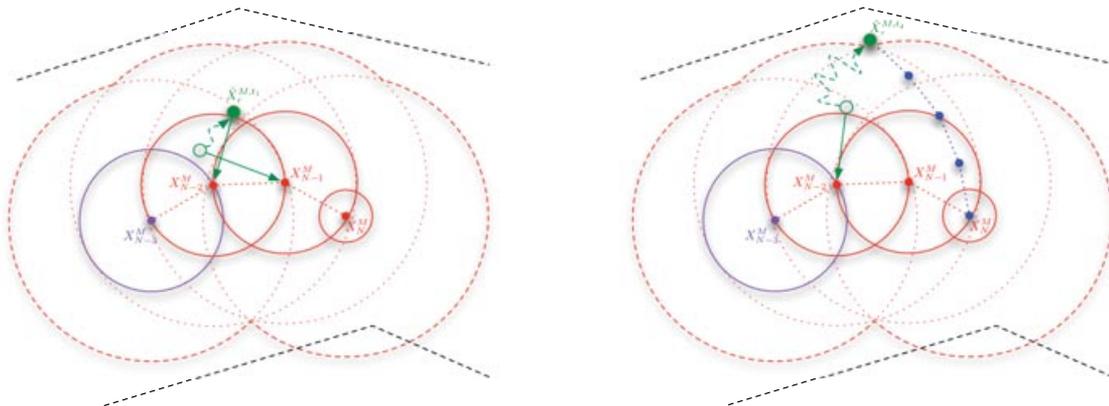
Figure 9. Illustration demonstrating the behavior of the path execution module when the robot strays off the path. The dashed circles determine the recovery zone. On the left, the robot is following circle C_{N-2}, targeting waypoint X_{N-1}^M, but moves off the circle path and is instructed to return to X_{N-2}. On the right, the robot strays farther off the path and moves out of the recovery zone, prompting the algorithm to stop the robot and request a new path from the global planner, plotted in blue. The old path is discarded.

In recent years, researchers worldwide have opted for particle filter–based solutions for localization (Levinson et al., 2007; Nuske et al., 2009; Thrun et al., 2001), which offer design advantages and greater flexibility than approaches based on the Kalman filter (Arras, Castellanos, Schilt, & Siegwart, 2003; Georgiev & Allen, 2004; Yun & Miura, 2007). However, when particle filter localization is developed for autonomous navigation, it has to deal with real-time requirements. Particle filters need to compute expected observations from particle positions. Computations can be performed offline and then stored in large lookup tables discretizing the space of positions, so that during online executions these lookup tables will be queried from particle positions (Levinson et al., 2007; Thrun et al., 2001). However, when the robot operates in large environments and the position space has a dimensionality greater than three, precomputing expected observations becomes a critical database issue.

In this section we describe a 3D map–based localization method, consisting of a particle filter that computes the expected observations online by means of fast manipulation of a 3D geometric model of the environment, implemented using the OpenGL library (OpenGL, –). Using OpenGL for online observation model computations has already been proposed by some researchers (Nuske et al., 2009). However, in that paper the authors use an edge map of the environment and compute only expected edge observations. Our approach does not perform any feature extraction step and deals with online computation of the full sensor model, so that real sensor data are directly compared with expected sensor data to score the particles in the filter loop. This approach overcomes the issue of feature occlusion due to the ubiquitous presence of pedestrians and other unmodeled obstacles around the robot, achieving

robust tracking of the robot's position. Our solution runs at 5 Hz, enough for our platform's speed.

7.1. State Space

The state space considered in our approach, **X**, is that of 3D positions, parameterized as a (x, y, z) location referenced to the map frame and the three Euler angles, heading, pitch, and roll, (θ, ϕ, ψ), defined starting with the heading angle with respect to the x map axis. In this section, all positions will be referenced to the map frame if no specific mark or comment indicates otherwise.

At each iteration t, the filter produces a set of particles, P^t, in which each particle is a pair formed by a position in the state space and a weight:

$$P^t = \{s_1^t \ldots s_{N_P}^t\}; \; s_i^t = (X_i^t, w_i^t); \; X_i^t = (x_i^t, y_i^t, z_i^t, \theta_i^t, \phi_i^t, \psi_i^t),$$

(1)

where s_i^t is the ith particle produced by the tth iteration, $X_i^t \in \mathbf{X}$, and $w_i^t \in [0, 1]$.

7.2. 3D Environment Model

The environment model used by the localization module, also referred to as the map and notated as $\mathcal{M}$, is a geometric 3D representation of the static part of the area where the robot operates. In both the campus and Gràcia areas, the static part considered includes buildings, stairs, ramps, borders, curbs, some important vegetation elements, and urban furniture such as benches and streetlamps. Our implementation uses the *.obj* geometry definition file format (*OBJ*, –), originally developed for 3D computer animation and scene description, which has become an open format and a de facto exchange standard.

Both maps were built by hand, taking measurements with laser distance meters and measuring tape, which were used to build a coherent 3D model. Even if the maps incorporate the most important geometrical elements of each experimental area, they are always considered incomplete models: for instance, trees were modeled only partially, due to the difficulty of the modeling, and minor urban furniture was not always mapped. Thus the localization approach should be robust enough to address this issue. Further details are available in Corominas Murtra, Trulls, Mirats-Tur, & Sanfeliu (2010).

7.3. Kinematic Model

In particle filtering, having a motion model allows us to propagate the particle set, thus limiting the search space to positions satisfying the motion model constrained to given sensor inputs. Probabilistic kinematic models (Thrun, Burgard, & Fox, 2005) compute a new sample set, called the *prior*, P^{t^-}, based on the previous set, P^{t-1}, constrained to the platform's motion. We define the platform wheel odometry readings as

$$o_U^t = (\Delta_\rho^t, \Delta_\theta^t), \tag{2}$$

where Δ_ρ^t is the translational 2D increment in the local XY plane and Δ_θ^t is the rotational increment around the local Z axis of the platform. Both increments are the accumulated odometry from iteration $t-1$ up to iteration t. The Segway RMP200 platform also features embedded inclinometers that provide a pitch increment measure from $t-1$ to t:

$$o_I^t = \Delta_\phi^t. \tag{3}$$

With these two input observations, at the beginning of each iteration the state of the ith particle is moved according to the probabilistic kinematic model described by

$$
\begin{aligned}
\tilde{\Delta}_{\rho,i}^t &= \mathcal{N}(\Delta_\rho^t, \sigma_\rho^t); \ \sigma_\rho^t = \epsilon_\rho \Delta_\rho^t, \\
\tilde{\Delta}_{\theta,i}^t &= \mathcal{N}(\Delta_\theta^t, \sigma_\theta^t); \ \sigma_\theta^t = \epsilon_\theta \Delta_\theta^t, \\
\tilde{\Delta}_{\phi,i}^t &= \mathcal{N}(\Delta_\phi^t, \sigma_\phi^t); \ \sigma_\phi^t = \epsilon_\phi \Delta_\phi^t, \\
x_i^t &= x_i^{t-1} + \tilde{\Delta}_{\rho,i}^t \cos\left(\theta_i^{t-1} + \frac{\tilde{\Delta}_{\theta,i}^t}{2}\right), \\
y_i^t &= y_i^{t-1} + \tilde{\Delta}_{\rho,i}^t \sin\left(\theta_i^{t-1} + \frac{\tilde{\Delta}_{\theta,i}^t}{2}\right), \\
\theta_i^t &= \theta_i^{t-1} + \tilde{\Delta}_{\theta,i}^t, \\
\phi_i^t &= \phi_i^{t-1} + \tilde{\Delta}_{\phi,i}^t,
\end{aligned}
\tag{4}
$$

where the first three lines draw, for each particle, random data with normal distribution centered at the platform data $(\Delta_\rho^t, \Delta_\theta^t, \Delta_\phi^t)$ with standard deviation depending linearly with each respective increment by parameters $\epsilon_\rho, \epsilon_\theta, \epsilon_\phi$, so

that large increments imply a more sparse propagation. Epsilon values were set to $\epsilon_{\{\rho,\theta,\phi\}} = 0.2$ during the experimental sessions.

Please note that the orientation angles of a Segway robot do not necessarily indicate a displacement direction because the platform is unconstrained in pitch. Thus this kinematic model approximates displacements in the local plane provided by the platform odometry as displacements in the global (map) XY plane. This approximation leads to an error that is negligible in practice because the slopes in our test environments have at most an inclination of 10%. Note also that the kinematic model does not modify z_i^t and ψ_i^t because these two variables are constrained by gravity, as will be explained in the next subsection.

7.4. Gravity Constraints

A wheeled robot will always lie on the floor, due to gravity. For relatively slow platforms, such as those presented in Section 3, it can be assumed as well that the whole platform is a rigid body, so that a suspension system, if present, does not modify the attitude of the vehicle. With these assumptions, and for two-wheeled self-balancing platforms, there are constraints on the height z and roll ψ dimensions of the position space, given a (x, y, θ) triplet and the environment model $\mathcal{M}$. Both constraints will be computed using OpenGL for fast manipulation of 3D models.

The height constraint sets a height z for a given coordinate pair (x, y). To compute it, the floor part of the map is rendered in a small window (5×5 pixels) from an overhead viewpoint at (x, y, z_{oh}), limiting the projection to a narrow aperture (1 deg). After rendering, we obtain the depth component of the central pixel, d_c, and compute the constrained z value as $z = z_{oh} - d_c$.

The roll constraint fixes the roll component for a coordinate triplet (x, y, θ). Its computation is based on finding z_r and z_l, the height constraints at two points to the left and to the right of (x, y, θ). These points are separated a known distance L (i.e., size of the platform), so that the roll constraint can be computed as $\psi = \text{atan2}(z_l - z_r, L)$.

The computation of the roll constraint is based on computing height constraints, which can be viewed as a simple 2D height map, relating (x, y) pairs with a height value z. This leads us to precompute, offline, a height grid of the environment, so that during online executions gravity constraints will be resolved with simple and fast queries to a table. Figure 10 shows the height grid for the UPC Campus site. The cell size of this grid is 0.2×0.2 m^2.

Note that this approach is valid for maps with a single traversable z level such as ours and although our algorithms can be directly applied to multilevel maps, further work would be required in determining the appropriate map section to compute. To avoid discretization problems, especially when computing the roll constraint using the height grid, we use lineal interpolation on the grid.
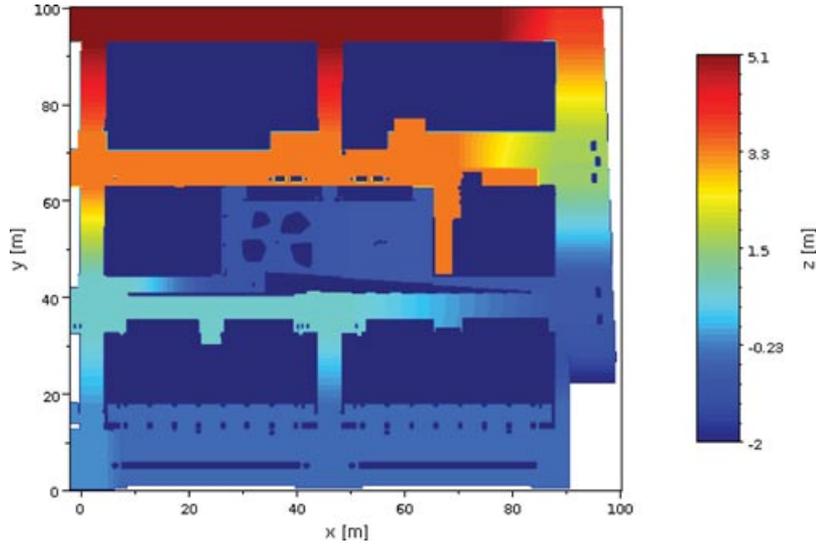
Figure 10. Height grid for the UPC campus site.

7.5. Range Observation Model and Similarity Metrics

Fast and accurate computation of observation models is a key issue for successful particle filtering. The result of computing an observation model is an expected observation computed from a particle position, denoted as $o_L^s(X_i^t)$ for a laser scanner. Given this expected observation, the conditional probability of an actual laser observation given that the robot is in particle position X_i^t can be approximated as

$$p(o_L^t|X_i^t) \sim \mathcal{L}[o_L^t, o_L^s(X_i^t)], \in [0, 1], \qquad (5)$$

where $\mathcal{L}$ is a similarity function measuring the closeness of two laser observations. This subsection first details how the expected observations o_L^s are computed and then presents the similarity function used to compare real and expected laser scanner data.

We propose a method using the OpenGL library for fast online computation of expected laser scanner observations. The method is based on rendering the 3D model from the viewpoint of the sensor's position given the particle position X_i^t and reading the depth buffer of the computer graphics card. The rendering window size has been minimized to reduce the computation time while keeping the sensor's accuracy given by the scanner aperture, Δ_α, and the number of scan points N_L. Algorithm 1 outlines the procedure to compute an expected laser scan from a particle position X_i^t, given a 3D environment model $\mathcal{M}$ and a set of sensor parameters $(\Delta_\alpha, N_L, r_{\min}, r_{\max})$, being respectively, the angular scan aperture, the number of scan points, and range limits.

Because the Leuze scanner has an aperture greater than 180 deg, we divide the computation into two sectors. For the Hokuyo scanner we use only an aperture of 60 deg, so

---

Algorithm 1 Laser scanner observation model

INPUT: $X_i^t, \mathcal{M}, (\Delta_\alpha, N_L, r_{max}, r_{min})$
OUTPUT: $o_L^s(X_i^t)$

$w = 2r_{min}tan(\frac{\Delta_\alpha}{2}); \quad h = 2r_{min}tan(\frac{\Delta_\beta}{2}); \quad \rho = \frac{w}{h}; \quad //$metric dimensions [m] and aspect ratio of projection
setProjection(1 deg, ρ, r_{min}, r_{max}); //rendering volume: 1 deg of vertical aperture, aspect ratio, depth limits
$\delta_\alpha = \frac{\Delta_\alpha}{N_L}; \quad //$sensor's angular resolution [deg]
$p_\alpha = (int)2\frac{tan(\Delta_\alpha/2)}{tan(\delta_\alpha)}; \quad //$ compute window width [pixels]
setWindowSize(p_α, 5); //set window size [pixels]
$X_s = Transform(X_i^t, X_s^R); \quad //$transform sensor's position from robot frame to map frame
renderUpdate($\mathcal{M}, X_s$); //render the model from the sensor's position
$\mathbf{b}_z = readZbuffer(CENTRALROW); \quad //$read normalized depth values of the central image row
for $j = 1 \dots N_L$ **do**
  $\alpha_j = \Delta_\alpha(0.5 - \frac{j}{N_L}); \quad //$ray angle [deg]
  $k = (int)\left(0.5 - \frac{tan(\alpha_j)}{2tan(\Delta_\alpha/2)}\right)p_\alpha; \quad //$pixel index corresponding to j^{th} ray
  $d_j = \frac{r_{min}r_{max}}{(r_{max}-\mathbf{b}_z(k))(r_{max}-r_{min})}; \quad //$screen depth [m]
  $r_j = \frac{d_j}{cos(\alpha_j)}; \quad //$range of the jth ray [m]
end for
return $\{r_1, \dots, r_{N_L}\};$

---

that a single sector is enough. According to the device parameters detailed in Section 3, the resulting window sizes are 88×5 pixels for the Leuze device (for each sector) and 265×5 pixels for the Hokuyo scanner. This optimized

implementation allows the filter to run at 5 Hz while computing at each iteration $N_P \times (133 + 133 + 241)$ ranges. For $N_P = 50$ particles, this implies 126,750 ranges per second. Further details on computing such expected range observations can be found in Corominas Murtra, Trulls, Mirats-Tur, et al. (2010).

Once both the expected and real observations are available, a similarity function $\mathcal{L}$ computes their similarity. We use the mean over all scan points of the complementary error function (`erfc`) of the difference between actual and expected range data, so that given real and expected observations denoted, respectively, by $o_L^t = \{r_j^t\}$ and $o_L^s = \{r_j^s\}$, the similarity function is

$$\mathcal{L}\left[o_L^t, o_L^s\left(X_i^t\right)\right] = \frac{1}{N_L} \sum_{j=1}^{N_L} \text{erfc}\left[\frac{|r_j^t - r_j^s(X_i^t)|}{\sigma_L \sqrt{2}}\right], \quad (6)$$

where σ_L is the standard deviation of the range observation, set to 5 cm. This similarity function has the desired property that its value is limited to the $[0, 1]$ interval, evaluating to 1 for two identical scans and approaching 0 when the scans diverge, whereas the function depends only on the standard deviation associated with laser range measurements.

7.6. Particle Filter

The preceding subsections have introduced the kinematic model, the gravity constraints, a method for online computation of expected laser observations, and a similarity function to compare them with actual laser data. Algorithm 2 overviews how to combine these elements to build a particle filter to estimate the 3D position of the robot within the map coordinate frame.

The filter is initialized with a 2D position provided by the user, (x_0, y_0, θ_0). The first particle set is initialized within a square of $4\,\text{m}^2$ around the (x_0, y_0) location, within heading range $\theta_0 \pm 3$ deg and with pitch equal to zero (we initialize the localization algorithm with the robot stopped and on flat terrain). After the propagation and correction steps, in order to output a close estimation of the filter, a gaussian parametrization of the particle set is performed. The robot position estimate $\hat{X}_r^t$ is computed as the weighted mean of the particle positions, and the covariance parameters $\hat{C}^t$ are computed as the weighted sample variance. The following equations detail the computations for x and θ. The y and z coordinates are computed as for x, and the ϕ and ψ coordi-

---

Algorithm 2 Particle filter localization algorithm

INPUT: (x_0, y_0, θ_0), $\mathcal{M}$, o_U^t, o_I^t, $o_{L_F}^t$, $o_{L_B}^t$, $o_{L_V}^t$
OUTPUT: $\hat{X}_r^t$, $\hat{C}_r^t$, τ^t //robot position, associated uncertainty and time stamp

$t = 0$; //iteration counter
$P^0 = initialization(x_0, y_0, \theta_0)$; //initialization with prior knowledge
while running **do**
    $t \leftarrow t + 1$
    for $i = 1 \dots N_P$ **do**
        $(x_i^t, y_i^t, \theta_i^t, \phi_i^t) = f(X_i^{t-1}, o_U^t, o_I^t)$; //kinematic model
        $z_i^t = heightConstraint(X_i^t)$; //gravity height constraint
        $\psi_i^t = rollConstraint(X_i^t)$; //gravity roll constraint
    end for
    $\tau^t = timeStamp(NOW)$;
    for $i = 1 \dots N_P$ **do**
        $w_i^t = \mathcal{L}(o_{L_F}^t, o_{L_F}^s(X_i^t)) \cdot \mathcal{L}(o_{L_B}^t, o_{L_B}^s(X_i^t)) \cdot$
        $\mathcal{L}(o_{L_V}^t, o_{L_V}^s(X_i^t))$; //correction
    end for
    for $i = 1 \dots N_P$ **do**
        $w_i^t \leftarrow \frac{w_i^t}{\sum_{j=1}^{N_P} w_j^t}$; //normalization
    end for
    $(\hat{X}_r^t, \hat{C}_r^t) = gaussianParameters(P^t)$;
    $publish(\hat{X}_r^t, \hat{C}_r^t, \tau^t)$; //publish produced data
    $P^t \leftarrow resampling(P^t)$; //draw a new particle set resampling the current one
end while

---

nates are computed as for θ:

$$\hat{x}_r^t = \sum_{i=1}^{N_P} x_i^t \cdot w_i^t; \; (\hat{\sigma}_x^t)^2 = \sum_{i=1}^{N_P} (x_i^t - \hat{x}_r^t)^2 \cdot w_i^t;$$

$$\hat{\theta}_r^t = \text{atan2}\left(\sum_{i=1}^{N_P} \sin\theta_i^t \cdot w_i^t, \sum_{i=1}^{N_P} \cos\theta_i^t \cdot w_i^t\right);$$

$$(\hat{\sigma}_\theta^t)^2 = \sum_{i=1}^{N_P} \left(\text{acos}\left[\cos\left(\hat{\theta}_r^t\right)\cos\left(\theta_i^t\right) + \sin\left(\hat{\theta}_r^t\right)\sin\left(\theta_i^t\right)\right]\right)^2 \cdot w_i^t.$$

$$(7)$$

We consider all cross covariances to be zero, except for $\hat{\sigma}_{xy}^t$ that results from

$$\hat{\sigma}_{xy}^t = \sum_{i=1}^{N_P} (x_i^t - \hat{x}_r^t)(y_i^t - \hat{y}_r^t) \cdot w_i^t. \quad (8)$$

In the last step of the filter, a resampling function draws a new particle set keeping the mean of the current one. Resampling is necessary to avoid particle depletion (Arulampalam, Maskell, Gordon, Clapp, Sci, et al., 2002;

Doucet, de Freitas, & Gordon, 2001), an undesired phenomenon of particle filters in which the particle set collapses to a single state point, rendering the filter no longer capable of exploring new solutions for the estimation and therefore compromising its robustness.

As an aside, the vertical laser is integrated into the correction stage only when appropriate. Most unmodeled obstacles, such as pedestrians or bicyclists, have a relatively small footprint on the XY plane, so that the horizontal lasers remain usable despite numerous occlusions (as our experiments demonstrate). The vertical scanner on the other hand can be nearly fully occluded by a single pedestrian a few meters in front of the robot. In that scenario the filter attempts to match actual and expected observations by pitching the robot forward, lifting the floor surface toward the part of the scan corresponding to the pedestrian, and thus increasing the similarity between scans. This is clearly inadequate and compromises the filter's performance, so we use the vertical laser only when the difference between actual and expected observations, as computed by the similarity function, is smaller than a threshold, determined experimentally. We do not want to perform feature extraction or segmentation over the raw scan, but there exist more elaborate solutions, such as iteratively considering sets of data removing the points farther away from the robot until the threshold is met. These shall be explored in the future.

Section 9 summarizes the field work and discusses in depth the two failures we experienced during the experiments, both due to localization issues.

8. OBSTACLE AVOIDANCE

The motion planning problem is well known and studied when using a priori information (Latombe, 1991). However, many techniques are not applicable when the environment is not known or is highly dynamic. This problem is compounded by the fact that both the environment (i.e., the real world) and the robot carry uncertainties due to sensing and actuation, respectively, so that it is not feasible to treat motion planning separately from its execution. To solve these problems, it is necessary to incorporate sensory information in the planning and control loop, making reactive navigation possible. A real-time approach based on the artificial potential field concept was presented in Khatib (1986), was later extended in Khatib and Chatila (1995), and became widely used, as for instance in Haddad, Khatib, Lacroix, and Chatila (1998). Other methods extract higher level information from the sensor data, such as, for instance, in Minguez and Montano (2004), a reactive obstacle avoidance system for complex, cluttered environments based on inferring regions from geometrical properties. None of these methods takes into account the physical properties of the robot platform itself: two common approaches that do so are the curvature velocity method (Simmons, 1996) and the dynamic window approach (Fox, Burgard, & Thrun, 1997).

Our proposal consists of an obstacle avoidance method that combines a local planner with a slightly modified dynamic window approach so as to generate motion control commands suitable for the robot platform. Decoupling planning and execution is a common practice in mobile robotics, as the full path planning problem is typically too complex for real-time processing. This is particularly appropriate in our case, as our Segway robots cannot execute trajectories with a high degree of accuracy. Inputs to the local planner are a set of local goal candidates, provided by the path execution module and notated as $\mathcal{G}^R$, and sensor data: the front laser scan $o_{L_F}^t$ and odometry updates o_U^t. The output of the local planner is an obstacle-free goal, denoted by X_f^R. This goal is the input of the motion controller unit that computes suitable commands for translational and rotational velocities.

This approach would be sufficient for traversing flat environments. This is not the case, as urban environments contain features such as ramps, which the robot must be able to navigate, and drops and staircases, which should be avoided. Notably, a configuration of front and back lasers only is not capable of navigating a ramp upward, as a ramp is seen from its base as a wall at a distance determined by the ramp's slope and the laser's mounting height. In addition, our robots suffer from the tilt problem, introduced in Section 3, so that navigation on ramps, or even on flat surfaces when accelerating or decelerating, is impaired as well.

One possible solution lies in using an additional planar laser scanner somewhat tilted toward the ground, as introduced in Wijesoma, Kodagoda, and Balasuriya (2004), where it is used for detection and tracking of road curbs. A similar approach is used in Morales et al. (2009) for navigating cluttered pedestrian walkways. In the latter, the authors use two planar laser scanners tilted toward the ground so that on a flat surface the beams intersect the floor at 1 and 4 m from the robot, respectively. This information is used to perform traversable road extraction and allows the robot to navigate on outdoor paths. We found this technique challenging to implement on our robots for two reasons. First, its application on two-wheeled robots is much more involved than on statically stable robots, due to the additional degree of freedom (pitch). Second, this approach requires the robot to move toward an area of space to determine its traversability. This may negate one of the main advantages of our platform: its ability to rotate on the spot. We should also be able to ensure map consistency in time and deal explicitly with occlusions and dynamic obstacles.

We instead opt for a reactive solution, based on the vertical laser scanner, positioned as explained in Section 3, to perform traversability inference. We introduce the local planner and motion controller first and later present the traversability inference component. The obstacle avoidance module's block diagram is pictured in Figure 11.
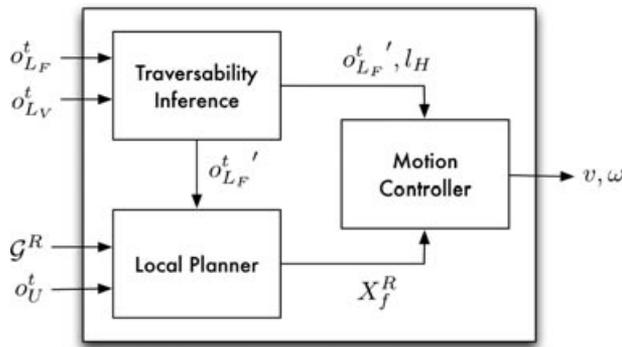
Figure 11. Obstacle avoidance module with local planner, motion controller, and traversability inference. $o_{L_F}^{t}{}'$ is the front laser scan after traversability inference, and l_H is the length of the traversable surface in front of the robot.

8.1. Local RRT-Based Planner

The local planner has been implemented using a rapidly-exploring random tree (RRT) (LaValle, & Ku, 2000). This technique explores 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 if an obstacle-free transition to the point exists. The presence of obstacles is determined by the front laser scanner. Range measurements are transformed into Cartesian coordinates, and each point is made into a circle with radius r_c, the clearance parameter for our robot (half its width plus safety room), set to 1 m. The search space $\mathcal{S}$ is restricted to sectors of a circle centered

on the robot with radius the distance to the goal, in order to better deal with occlusions. The search sectors are initially restricted in angle, to favor solutions going forward, and are expanded on each iteration until reaching the maximum sensor workspace, determined by the laser scanner's aperture Φ_H (190 deg). That is, in polar coordinates (r, φ), for a goal X_g^R, $\mathcal{S} = \{(r, \varphi)|r \in [0, r_g], \varphi \in [\varphi_{\min}, \varphi_{\max}]\}$, where $\varphi_{\min} = \max\{\varphi_g - \varphi_0, -\Phi_H/2\}$ and $\varphi_{\max} = \min\{\varphi_g + \varphi_0, \Phi_H/2\}$. We expand the search space around φ by $\Delta\varphi$ in each direction every time a random point is generated, whether it is possible to add it to the tree or not, while enforcing $\varphi \in [-\Phi_H/2, \Phi_H/2]$. The parameter φ_0 determines the initial sector's width and is set to 15 deg, and $\Delta\varphi$ is set so that the whole search space will be used before we determine that there is no solution and the RRT is not allowed to expand further.

Every time a new branch is added to the tree, we check for an obstacle-free transition to the goal. If it exists, we store the path to the goal and discard the rest of the tree. The path is smoothed, if possible, by determining obstacle-free transitions between pairs of points and eliminating redundancies. The resulting path is denoted $\mathcal{P}$. The first point in the path is provided to the motion controller as its current local goal, X_f^R. Note that while farther points may not be reachable due to occlusions in the laser scan, this point is in sight and thus guaranteed to be obstacle-free. The process is illustrated in Figure 12.

We opt for RRTs for ease of use and efficiency. Unlike potential fields or Voronoi-based approaches, or hybrid techniques (Dolgov, Thrun, Montemerlo, & Diebel, 2010), they do not offer a "best" solution. We defer this step to the motion controller, whose only requirement is an obstacle-free goal. As we do not perform any kind of feature
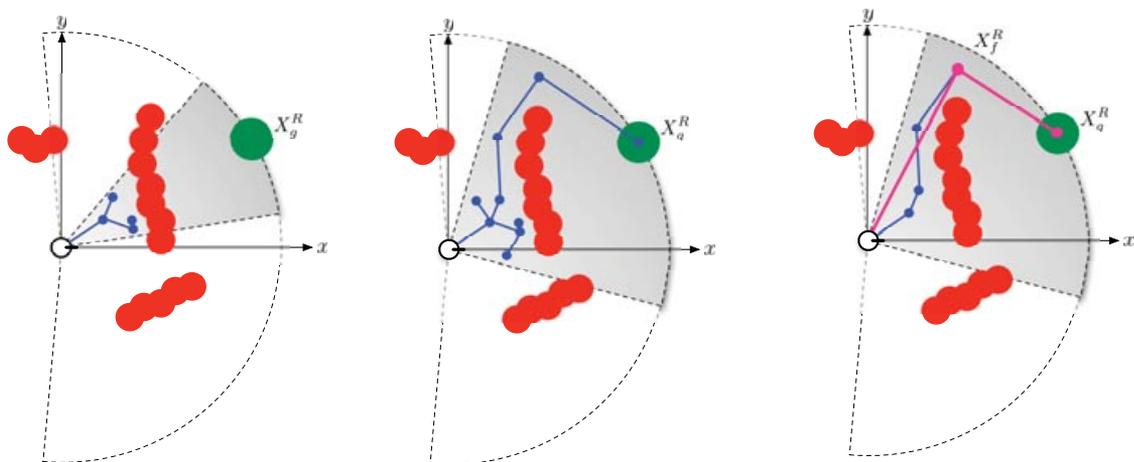

Figure 12. Local RRT planner with incremental search space $\mathcal{S}$, depicted in gray. Obstacles, defined as the front laser scan points with clearance r_c, are depicted in red. The figure on the left shows the tree, in blue, after a few iterations. The figure in the middle shows the tree after finding a solution. The figure on the right shows the RRT path in blue and the final, smoothed path in purple, the first point of which is the obstacle-free goal X_f^R.

extraction or tracking, the local planning step is very fast to compute.

As can be inferred from these constraints, we do not allow the robot to move backward. On one hand, our Segway robots can rotate 180 deg with ease, which is preferable, and second, the vertical laser scanner is a requirement for safe navigation and is facing forward. If the goal is behind the robot, we stop and rotate the robot until the goal lies within sensor range. Likewise, if the local planner is unable to find a path to the goal, we stop the robot and allow it to rotate to attempt to find a solution. This process will be described in Section 8.4.

8.2. Motion Controller

Our motion controller is based on the dynamic window approach (Fox et al., 1997). 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 [in our case, (v, ω) pairs] reachable within a single time increment Δt for the current state (the *dynamic window*), implicitly complying with the robot's dynamic constraints. This workspace is then discretized into a number of cells, for which an objective function G is maximized. This function considers a trade-off between velocity, target heading, and clearance to obstacles:

$$G(v, \omega) = \alpha_v f_v(v, \omega) + \alpha_\varphi f_\varphi(v, \omega) + \alpha_c f_c(v, \omega). \quad (9)$$

The clearance value function f_c is a measure of time until collision for the cell's configuration, relative to the platform's breaking time, as proposed in Philippsen (2004). We define

$$f_c = \begin{cases} 0 & \text{if } t_{\text{col}} \leq t_{\text{stop}} \\ \dfrac{t_{\text{col}} - t_{\text{stop}}}{T_{\text{stop}} - t_{\text{stop}}} & \text{if } t_{\text{stop}} < t_{\text{col}} < T_{\text{stop}}, \\ 1 & \text{otherwise} \end{cases} \quad (10)$$

where t_{col} is the time to collision if the robot continued in a straight line, T_{stop} the time required to stop the robot at

maximum speed, and t_{stop} the time required to stop the robot at the current speed.

The value functions for velocity and heading, f_v and f_φ, are a measure of 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 value function so as not to overshoot the goal and into the target heading value function to avoid oscillations around the desired trajectory. We achieve this by defining

$$T_{\text{stop}}^T = \frac{v_t}{a_v^{\max}}, \quad T_{\text{goal}}^T = \frac{d_g}{v_t}, \quad K_T = \frac{T_{\text{goal}}^T}{T_{\text{stop}}^T} = \frac{d_g a_v^{\max}}{v_t^2}, \quad (11)$$

$$T_{\text{stop}}^R = \frac{\omega_t}{a_\omega^{\max}}, \quad T_{\text{goal}}^R = \frac{\varphi_g}{\omega_t}, \quad K_R = \frac{T_{\text{goal}}^R}{T_{\text{stop}}^R} = \frac{\varphi_g a_\omega^{\max}}{\omega_t^2}, \quad (12)$$

where v_t and ω_t are the translational and rotational velocities of the robot and $a_v^{\max}$ and $a_\omega^{\max}$ the maximum accelerations. K_T and K_R give us a measure of the difficulty of stopping the robot in either case. For values of $|K| \gg 1$, the robot may accelerate at the maximum rate, whereas for values greater than and close to 1, time to stop the robot is critical. We determined adequate acceleration rates for different values of K_T and K_R and used them to devise a control law experimentally, as we find the behavior of the Segway robots to be highly dependent on external factors and thus too hard to model. We define the target translational velocity $\tilde{v}_{t+\Delta t}$ as

$$\tilde{v}_{t+\Delta t} = \begin{cases} \min\left(v_t + a_v^{\max}\Delta t, v_{\max}\right) & \text{if } K_T \geq K_T^B \\ \min\left(v_t + a_v^{\max}\Delta t \dfrac{K_T - K_T^A}{K_T^B - K_T^A}, v_{\max}\right) & \text{if } K_T^A < K_T < K_T^B. \\ \max\left(v_t - a_v^{\max}\Delta t, 0\right) & \text{if } K_T \leq K_T^A \end{cases} \quad (13)$$

Analogously, we define the target rotational velocity $\tilde{\omega}_{t+\Delta t}$ as

$$\tilde{\omega}_{t+\Delta t} = \begin{cases} \min\left(\omega_t + a_\omega^{\max}\Delta t, \omega_{\max}\right) & \text{if } K_R \geq K_R^B \\ \min\left(\omega_t + a_\omega^{\max}\Delta t \dfrac{K_R - K_R^A}{K_R^B - K_R^A}, \omega_{\max}\right) & \text{if } K_R^A < K_R < K_R^B \\ 0 & \text{if } -K_R^A \leq K_R \leq K_R^A \\ \max\left(\omega_t + a_\omega^{\max}\Delta t \dfrac{K_R + K_R^A}{K_R^B - K_R^A}, -\omega_{\max}\right) & \text{if } -K_R^B < K_R < -K_R^A \\ \max(\omega_t - a_\omega^{\max}\Delta t, -\omega_{\max}) & \text{if } K_R \leq -K_R^B \end{cases} \quad (14)$$

Table I. Physical and practical limits for our Segway robots.

	v_{max} (m/s)	ω_{max} (rad/s)	a_v^{max} (m/s^2)	a_ω^{max} (rad/s^2)
Platform	4.4	3.15	Undetermined	Undetermined
Our implementation	0.85	0.8	4	2.5

where K_T^A, K_T^B, K_R^A, and K_R^B are derived experimentally for our robot with its particular weight distribution. The velocity and heading value functions are then a measure of closeness to targets $\tilde{v}_{t+\Delta t}$ and $\tilde{\omega}_{t+\Delta t}$, respectively. For this we use simple triangular functions centered on the target values with a given width. The weight distribution for the value functions depends on the robot's capabilities and the formulation of the value functions. Our implementation is weighted as follows: $\alpha_v = 1$, $\alpha_\varphi = 2$, $\alpha_c = 3$. Prioritizing clearance to obstacles is typical, and weighting the heading value function more than the velocity value function allows for tighter bends, which our robot can do easily. We settled on this distribution after some experimentation. The values for maximum velocities and acceleration rates are provided in Table I.

By Eqs. (11) and (13), K_T^A is the parameter that determines when the robot starts decelerating, which may cause the robot to overshoot the goal if it is too conservative. In practice, the robot is commanded to stop whenever it reaches the goal, decelerating at the platform's maximum rate. This happens likewise for Eqs. (12) and (14) when we set $\tilde{\omega}_{t+\Delta t}$ to 0 regardless of its current value. This violates the limits listed in Table I, but not the platform limits, thus still observing the dynamic window principle. We find that this allows for better control of the platform.

8.3. Traversability Inference

False obstacles due to, for instance, ramps may be detected by incorporating the localization estimate and using the 3D map to identify the situation, but this solution dangerously couples the robot's reactive behavior to the robustness of the localization process. This would compromise the safety of our navigation system. Thus our approach is based on the vertical laser scanner, used to infer whether the robot can traverse this region of space. It also enables the robot to detect some obstacles outside the field of view of the horizontal laser scanners.

The campus features three different kinds of traversable surfaces: flat, sloped with a relatively even incline, and transitions from one to the other. The Gràcia environment does not feature noticeable changes in inclination, while being sloped throughout. The vertical laser observations in these environments 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 quality. Prior to this computation, the vertical laser scan is preprocessed by removing points beyond the range of the obstacle avoidance module (8 m) or due to interference with the robot chassis. The inference process is divided into three steps, executed in order, and is terminated whenever one of these steps produces a satisfactory solution. We consider the following:

1. A single regression using all data.
2. Two regressions, using all data sorted over x and divided into two sets by a threshold, for a set of thresholds over x until conditions are met.
3. A single regression, iteratively removing the points farthest away from the robot over x until conditions are met.

In any case, a maximum regression error and a minimum regression length must be satisfied. In the second case 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. These thresholds were determined empirically for our sensors in the campus environment.

This inference process enables the robots to enter and traverse ramps by removing points from the front laser scan incorrectly indicating the presence of obstacles prior to local planning. To do this, we use the linear regressions and the front laser mounting height to determine where the scanner's central point should intersect the floor. We call this distance the sensor's horizon. Range measurements are then transformed into Cartesian coordinates, and points such that $\{X = (x, y) | x > x_{th}, -y_{th} < y < y_{th}\}$ are removed from the scan. x_{th} is shorter than the horizon to account for noise, and y_{th} determines the points to consider and is set to 4 m. Results are shown in Figure 13.

The processed laser scan, $o_{L_F}^{t}{}'$, is required by both the local planner and the motion controller, as illustrated in Figure 11. The length of the traversable surface l_H is used as an additional parameter for the motion controller, limiting the translational speed or directly stopping the robot. The robot is also commanded to stop if the slope is too steep for the platform. Staircases are easy to discriminate when seen from the bottom, but from the top the laser's accuracy presents a problem and some observations are close enough to those of a ramp to fall under the threshold. The staircase's steep incline is then used to disambiguate.
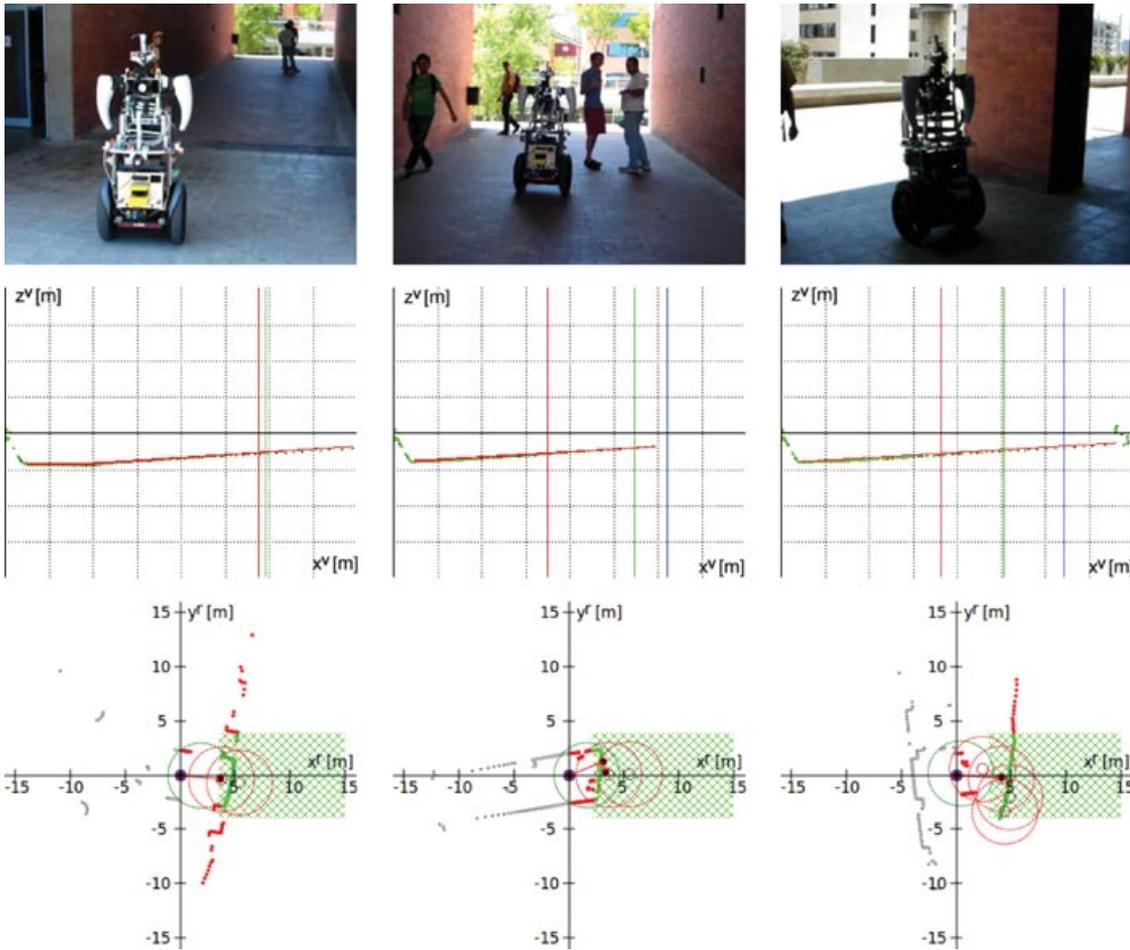
Figure 13. Demonstration of the traversability inference component on the campus site. The first row shows the action captured by a handheld camera, for reference. The second row shows the vertical laser scan as green dots and the linear regressions in red. Note that the coordinate frame in this figure is that of the laser scanner. Vertical lines indicate the central range measurement for the front laser in green, the horizon in blue, and the actual threshold applied over the front laser range data in red. The third row shows the front laser scan after filtering, where points in the green area are to be ignored. The first column shows the robot attempting to enter a ramp. In the second column, the robot is already traversing the ramp, its visibility reduced to approximately 2.5 m. The red dots in front and to the right of the robot correspond to pedestrians. The third column shows the tilt problem of our Segway robots on a flat surface. In any of these situations the robot would be unable to reach the goal without traversability inference.

8.4. Integration

The different components of the obstacle avoidance module are integrated as diagrammed in Figure 14. First, we check for new goal candidates $\mathcal{G}^R$ from the path execution module. If none exists, we update the current set of goal candidates with odometry updates only, closing the reactive loop. We also update the RRT path, if any. If the robot has reached the goal, it stops and waits for further instructions. Otherwise, we perform traversability inference over the vertical laser scan $o^t_{L_V}$, obtaining a filtered laser scan, $o^t_{L_H}{}'$, and the length of the traversable surface in front of

the robot, l_H. If l_H is too small, the robot enters recovery mode: it stops and then turns around in the direction of the goal (φ_g). After a certain number of iterations stuck in this step, we desist and notify upper level modules. This step is required due to the robot's limited perception capabilities and is equivalent to sweeping the space around the robot with the vertical laser scanner. The rotation is controlled by a version of the motion controller that allows only for (v, ω) pairs such that $v = 0$.

If no new front laser scan is available, we use the commands computed in the last iteration. This does happen, as the module runs at a frequency higher than the horizontal
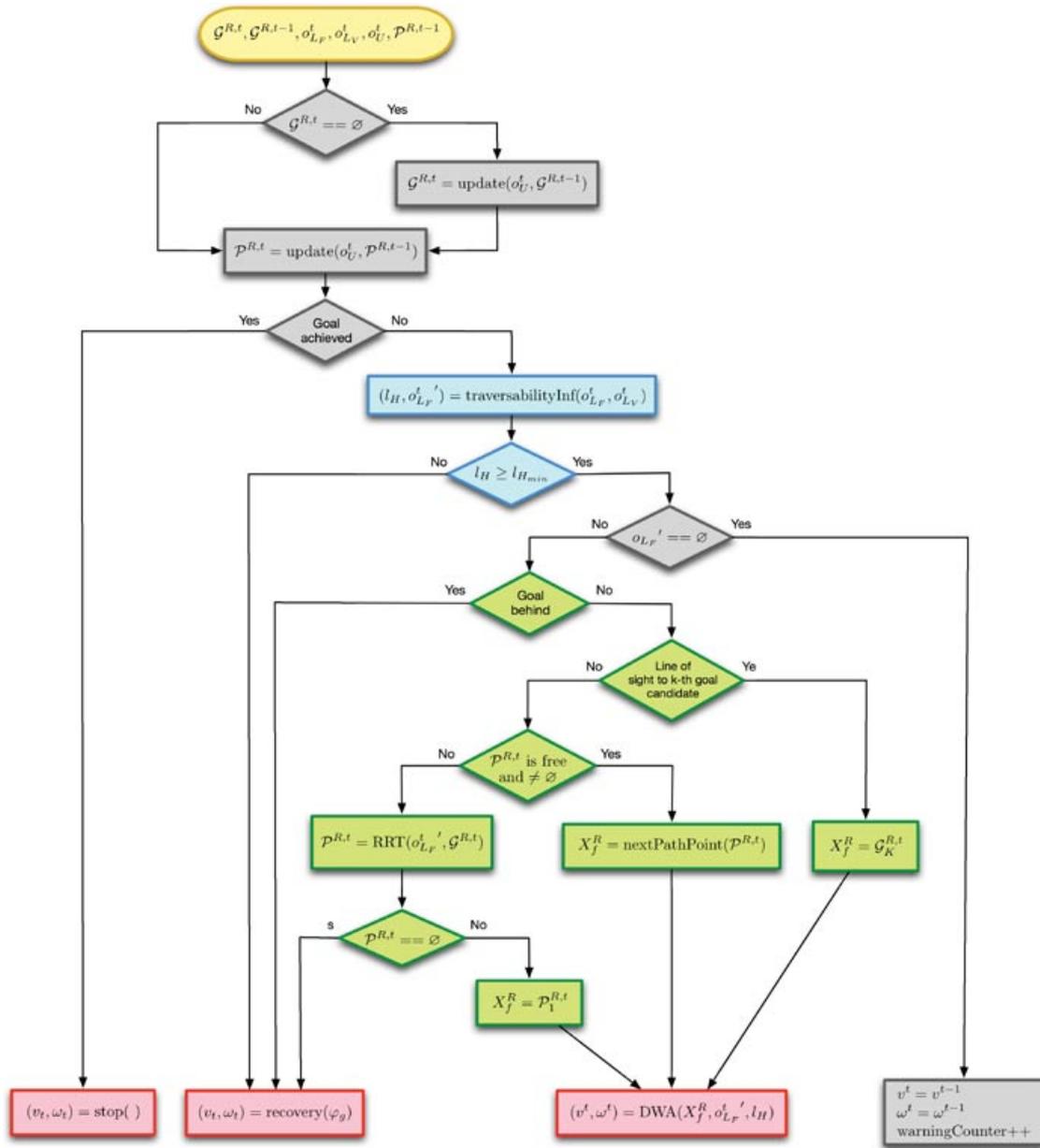
Figure 14. Obstacle avoidance module diagram. Parameters are goal candidates from the path execution module $\mathcal{G}^{R,t}$, laser scans $o^t_{L_F}$ and $o^t_{L_V}$, and odometry updates o^t_U. The process stores the last set of goal candidates $\mathcal{G}^{R,t-1}$ and the last path computed by the local planner $\mathcal{P}^{R,t-1}$. Blocks painted blue, green, and red belong to the traversability inference algorithm, the local planner, and the motion controller, respectively. Gray blocks are process logic.

laser scanners and lower than the vertical scanner. Otherwise we proceed to local planning. If the goal is behind, we want the robot to rotate until it lies within sensor range, using again recovery mode. If the goal is within sensor range, we attempt to find a path to a goal candidate in three ways, stopping when a valid solution is found:

1. First, we check for line of sight to any goal candidate, in the order determined by the path execution module.
2. If a path, updated by odometry data, exists and is valid for the current laser scan, we follow it.
3. Otherwise, we try to find a new path to a goal candidate.

Table II. Experimental results (1).

Place and date	D (m)	t_{nav} (s)	t_{free} (s)	t_{OA} (s)	t_{stop} (s)	t_{rot} (s)	R_{OA} (%)	$\hat{v}$ (m/s)
Gràcia, 20 May 2010	777.7	1,107.9	978.9	40.1	15	73.9	13.2	0.71
Campus, 3 Jun 2010	858.5	1,056.2	903.3	91.9	19.9	41.1	14.5	0.81
Campus, 22 Jun 2010	2,481.8	3,426.3	2,481.1	541.3	174.3	229.6	27.6	0.72
Campus, 23 Jun 2010	2,252.5	2,727.3	2,325.8	186.8	75.4	139.3	14.7	0.83
Accumulated	6,370.5	8,317.7	6,689.1	860.1	284.6	483.9	19.6	0.77

If the local planner cannot find a solution, we enter recovery mode. This happens especially when a pedestrian walks right in front of the robot and remains too close for the robot to move. In this case, we allow it to rotate.

9. EXPERIMENTS AND ASSESSMENT OF RESULTS

The navigation system was validated over the course of four experimental sessions, one on the Gràcia site and three at the campus. An external computer, connected to the onboard computer via wireless, was used to send manual *go-to* requests (XY coordinates over the map) to the navigation system and for online monitoring using our graphical user interface (GUI). Note that these are high-level requests, equivalent to "send a robot to the southeast door of the A5 building." Goals in the experiments include both long distances across the campus (the longest possible path between two points being around 150 m) and goals closer to each other to force the robot (and thus the path planning algorithm) through more complex situations such as around the trees in the square or around the columns in the A5/A6 buildings. Requests were often chained to keep the robot in motion, sending the robot to a new location just before reaching the current goal. We typically chose closer goals to keep some control over the trajectories and have the robot explore all of the area.

Run time for all experiments added up to 2.3 h, with more than 6 km of autonomous navigation. We set a speed limit of 0.75 m/s for the first session and increased it to 0.85 m/s for the following three sessions—note that this is a soft limit, and the robot often travels faster due to its self-balancing behavior. We used Tibi and Dabo without distinction.

Results are displayed in Tables II and III. Table II lists the navigation distance D, as estimated by the localization module, and the total navigation time t_{nav}, understood to be that spent with the robot attending to a *go-to* request. This measure is divided in time spent on obstacle-free navigation t_{free}, active obstacle avoidance t_{OA}, safety stops t_{stop}, and rotation on recovery mode t_{rot}. The ratio R_{OA} is a measure of the time spent avoiding obstacles, computed as $R_{OA} = (t_{OA} + t_{stop} + t_{rot})/t_{nav}$, and $\hat{v}$ is an estimation of the average translational speed computed using the previous values, $\hat{v} = D/t_{nav}$. Table III displays the number of requests and failures and success rate, as well as the average navigated distance per request $\hat{d}_{req}$.

We were allowed 3 days (mornings only) to conduct experiments at the Gràcia site, the last of which was dedicated to a public demonstration, and so the scope of that session is limited, totaling less than 1 km of autonomous navigation. Even so, it must be noted that, due to time constraints, these experiments were conducted with little to no prior on-site testing. Moreover, although part of the area was fenced, many pedestrians and bicyclists disregarded instructions and crossed the area anyway. This proves the robustness of our navigation system in new environments under similar conditions.

The four runs are plotted in Figure 15. For the session at the Gràcia site, we fenced the rightmost passageway and allowed pedestrians and bicyclists to use the one on the left. The rest of the area was left as is except for four fences placed below the monument, at $y = 20$ (Figure 15, top left), as a safety measure. The second session, already at the campus site, starts at (90,38) and ends at (17,69) when the robot encounters a large section occupied by public works and thus unmapped. In the third session, we ventured once to the passageway between the C and D buildings, which is on the verge of the experimental area and was roughly mapped, and hence we did not revisit. We also had the opportunity to navigate the narrow passageway to the right

Table III. Experimental results (2).

| Place and date	Requests (#)	$\hat{d}_{req}$ (m)	Errors (#)	Success rate (%)
Gràcia, 20 May 2010	33	23.6	0	100
Campus, 3 Jun 2010	23	37.3	0	100
Campus, 22 Jun 2010	55	45.1	0	100
Campus, 23 Jun 2010	60	37.5	2	96.7
Accumulated	171	37.3	2	98.8
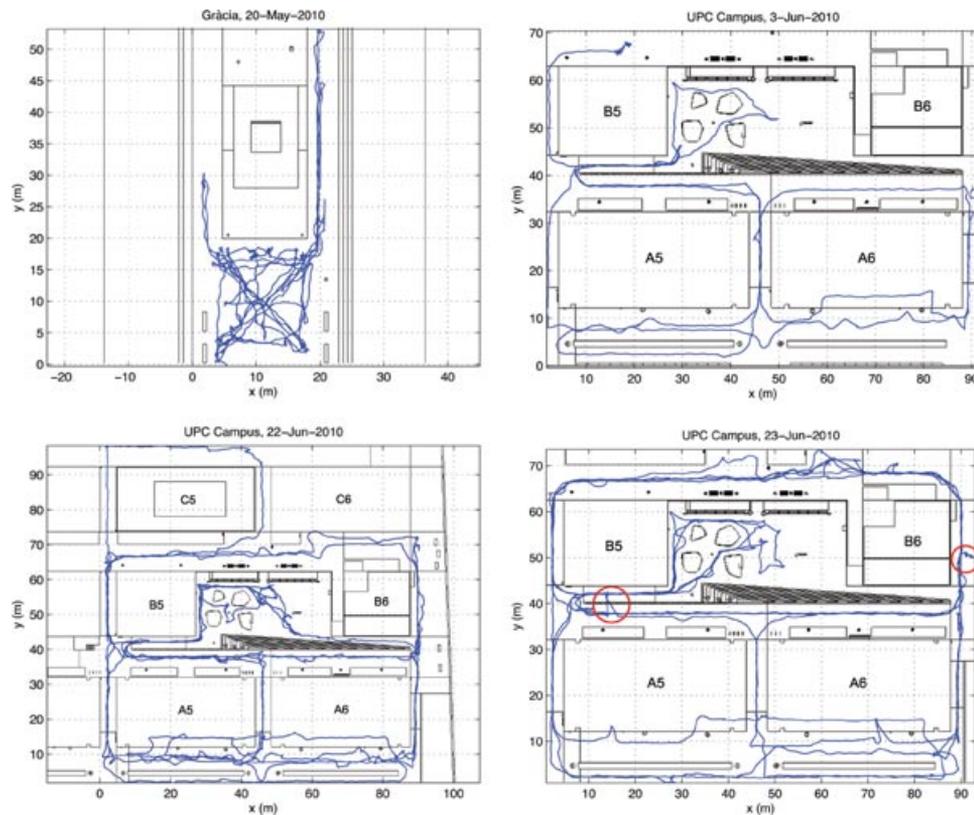
Figure 15. Localization results for the four experimental sessions. Red circles in the bottom right figure mark failure points.

of the FIB square, which is usually occupied by the cafeteria's terrace. Please note that areas where the localization estimate is within a building, such as A5, A6, and C6, are covered (Figure 2, picture number 2).

The fourth run contains the only two errors we encountered. Both are related to the localization algorithm and were mainly due to features of the terrain. These two failures are analyzed in detail below.

The first and second sessions are documented by one video each, available in the online version of this article. The video for the second session contains the session in its entirety. Figure 16 provides a sample screenshot with an explanation of the different data shown.

All navigation processes run concurrently on a single laptop. The localization process runs at 5 Hz, and the obstacle avoidance process runs at 10 Hz. Traversability inference is typically computed in less than 1 ms, whereas the local planner takes 1–10 ms per iteration, up to about 50 ms for the worst-case scenario (five goal candidates, no solution). The computational cost for the dynamic window computation depends on its granularity, taking an average of 10 ms for 175 cells. The path execution module carries a negligible computational load, and the path planning module is executed only upon a *go-to* request and takes

approximately 1 s, which does not interfere in real-time navigation.

Having failures gives us the chance to learn, advance, and improve our system. Therefore, we provide insights into the two localization failures that occurred during the last session, at the campus site. This analysis was made possible by the offline study of the logged data for that session, automatically stored by our software framework. Our localization module can be run offline using dummy sensors, which publish logged sensor data under the same interfaces as during online executions, while keeping synchronization. This allows us to run new real-time, offline executions of the localization process with the data collected during online executions.

The first failure happened approximately at the XY point (90,50). The robot was traveling from left to right along $y = 38$ and turned to its left to go up the ramp at $x = 90$ (Figure 15). The turning point can be seen in Figure 2, picture number 3. After turning, the localization uncertainty grew larger, while the position estimate deviated very clearly from the true position as perceived by the team members, finally causing a navigation error. This was due to two causes. First, the robot passed over a big terrain irregularity (a crack on the pavement) just before starting
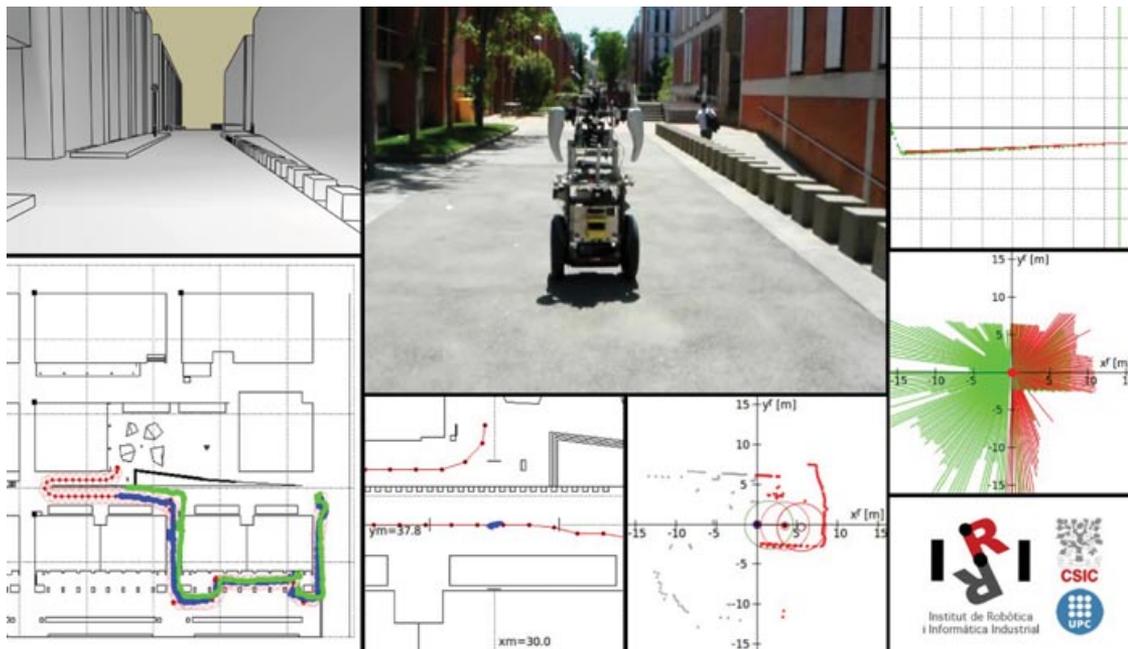
Figure 16. Sample screenshot. Clockwise, starting from the top left: (1) 3D map from the localization viewpoint, (2) handheld camera, (3) vertical laser and regressions, (4) horizontal laser scans in green and expected front laser scan in red, (5) logo, (6) obstacle avoidance and path execution output, (7) path planning output (red) and particle set from the localization algorithm (blue), and (8) path planning (red), accumulated odometry (green), and localization estimate (blue) on the full 2D map, for reference.

the turning maneuver, reported by odometry data as a period with high roll oscillations (Figure 17) and noisy heading increments. Our current approach constrains the roll component to the 3D model, assuming that the platform is a rigid body, so that roll oscillations caused by surface irregularities are not tracked well by the filter, as can be seen in Figure 17. Second, this happened around the bottom-right corner of the B6 building, which has large, floor-to-ceiling glass windows, which we modeled as walls. Offline inspection of the front laser data shows how in many instances the laser beam penetrates the windows before the robot turns to face the ramp (Figure 17). Modeling this behavior would require a complex observation model, because it depends on outdoor and indoor window lighting, as well as on ray incidence, this being one of the main limitations for laser devices. Figure 17 also shows the presence of three pedestrians (team members) close to the robot, blocking three important sectors of the back laser scan. The odometry issue led to noisy particle propagation, and the laser issue led to poor filter correction. The combination of the two events caused a localization error. We have performed 20 real-time, offline executions of the localization filter at this point with the logged data, resulting in a failure ratio of 45%, clearly indicating that this was a challenging situation.

The second localization failure was due to faulty odometry data, again after passing over a big terrain irregularity. Our localization approach can filter noisy data

peaks, but this case was extreme as odometry data were clearly incorrect for both translation and rotation for approximately 1.2 s, providing odometry increments around 0.4 m and −8 deg for an odometry acquisition period of $T_{\rm odo} = 0.1$ s. These data are clearly erroneous as the robot was at this time moving straight ahead at a speed of approximately 1 m/s (Figure 18). This was the first time that such an error was reported on our platforms, and the localization filter did not check the coherency of odometry data before using them for particle propagation [see Eq. (4)]. Using faulty data for about six consecutive iterations caused the localization estimate to advance and turn right with no chance for recovery in the filter's correction step. This can be clearly seen in Figure 15, where the robot jumped from (16,37) to (13,42). After acknowledging the error, we relocalized the robot manually and resumed the experiments; hence the second jump to the correct position. The terrain irregularity that caused this error was another crack in the pavement. These are frequent throughout the campus and occasionally cause the robot to take a small jump and thrash sideways. This behavior can be seen in the campus video.

These two failures teach us that robust navigation is still an open issue for mobile robots operating in urban pedestrian areas. We are currently working on improving the localization module in several directions:

- Use the translational and rotational velocities provided by the Segway platform instead of the odometry
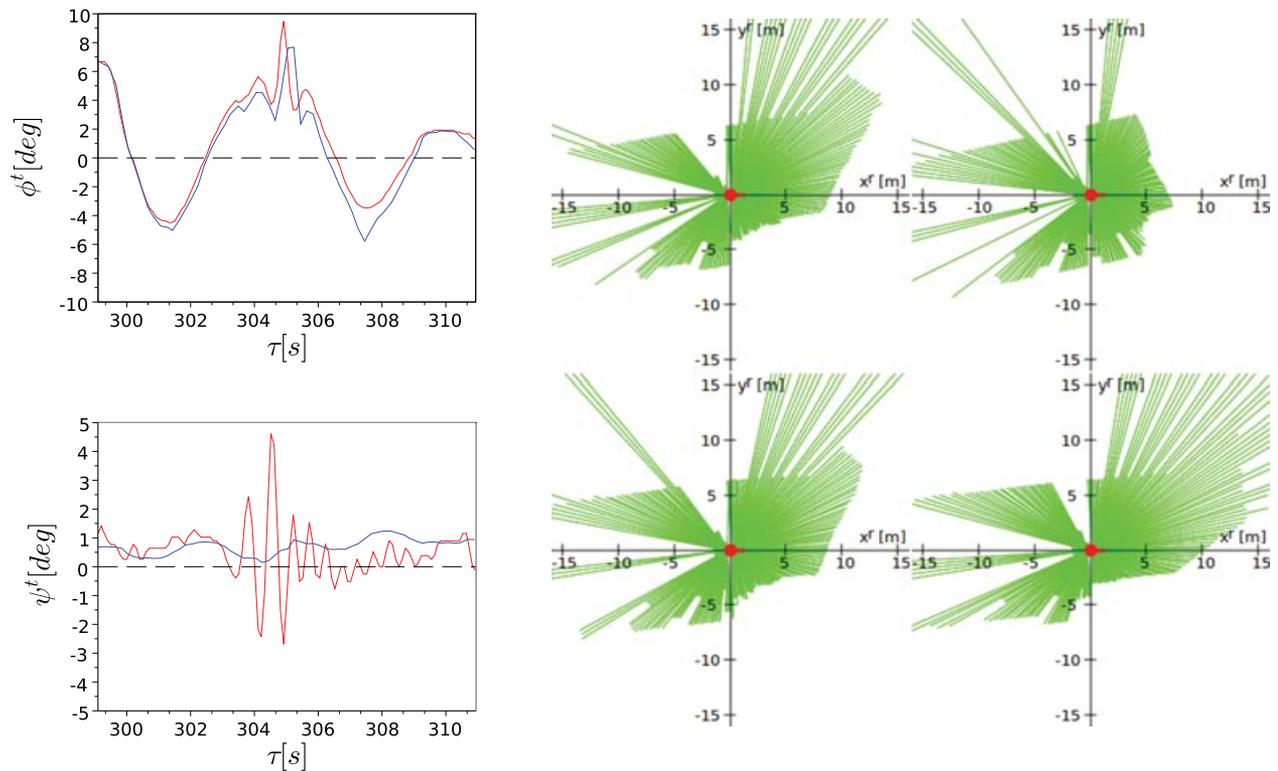
Figure 17. Left: Pitch (top) and roll (bottom) data provided by the platform (red) and estimated (blue) at the surroundings of the first failure. Right: Four consecutive laser scans just before the failure. Note how the front laser beams penetrate in many instances the windows at $y^R \sim 6$ m and the pedestrians blocking three large sectors of the back laser scan.

increments, in order to avoid the integration of faulty odometry data. Velocities are the outcome of an embedded filter and as such are more stable (Figure 18).

- Integrate the roll increment provided by the platform in the propagation step, as is done for pitch. In the correction step, we plan to add a similarity function related to the expected roll, given (x, y, θ) and the gravity constraint on the floor map.
- Investigate the trade-off between the number of particles and the filter rate. Relaxing the filter rate, while keeping the navigation requirements, leads to the possibility of increasing the number of particles, thus exploring a larger area of the state space.

An early, improved version of the particle filter is giving promising results after a series of real-time, offline executions when the robot passes over the two failure situations.

10. LESSONS LEARNED AND FUTURE WORK

Despite many recent advances, reliable autonomous navigation in urban, pedestrian environments remains a major challenge in robotics. This paper presents a solution geared toward two-wheeled robots, highly mobile and with a small footprint, and deals effectively with problems in both perception and control due to their self-balancing behavior. It has been extensively tested in two different urban scenarios, covering more than 6 km with a 99% success rate.

The main issue to be addressed in the future is one of perception. We rely on 2D laser scanners for localization and obstacle avoidance because (1) they provide an immediate perception stage with no need for feature extraction and (2) they are robust to varying illumination conditions. On the downside, they are by definition limited to a 2D plane. This impairs the visibility of horizontal laser scanners on urban scenarios with ramps, which has been addressed by means of the vertical laser scanner and the traversability inference algorithm. But obstacles outside either plane remain a problem. We find in practice that objects or terrain features lying low on the floor, such as curbs or steps, are the most problematic, as the vertical scanner can sense only forward and the wheels extend approximately 30 cm to either side. 3D laser scanners are costly and bulky, and current time-of-flight cameras are not usable under direct sunlight. We plan to explore the use of multiple laser scanners tilted to the floor to build local obstacle maps, using feature matching and/or in conjunction with the localization estimate. Note that with this configuration the platform's tilt would again pose a problem and the mobility of the robot may be reduced because of blind
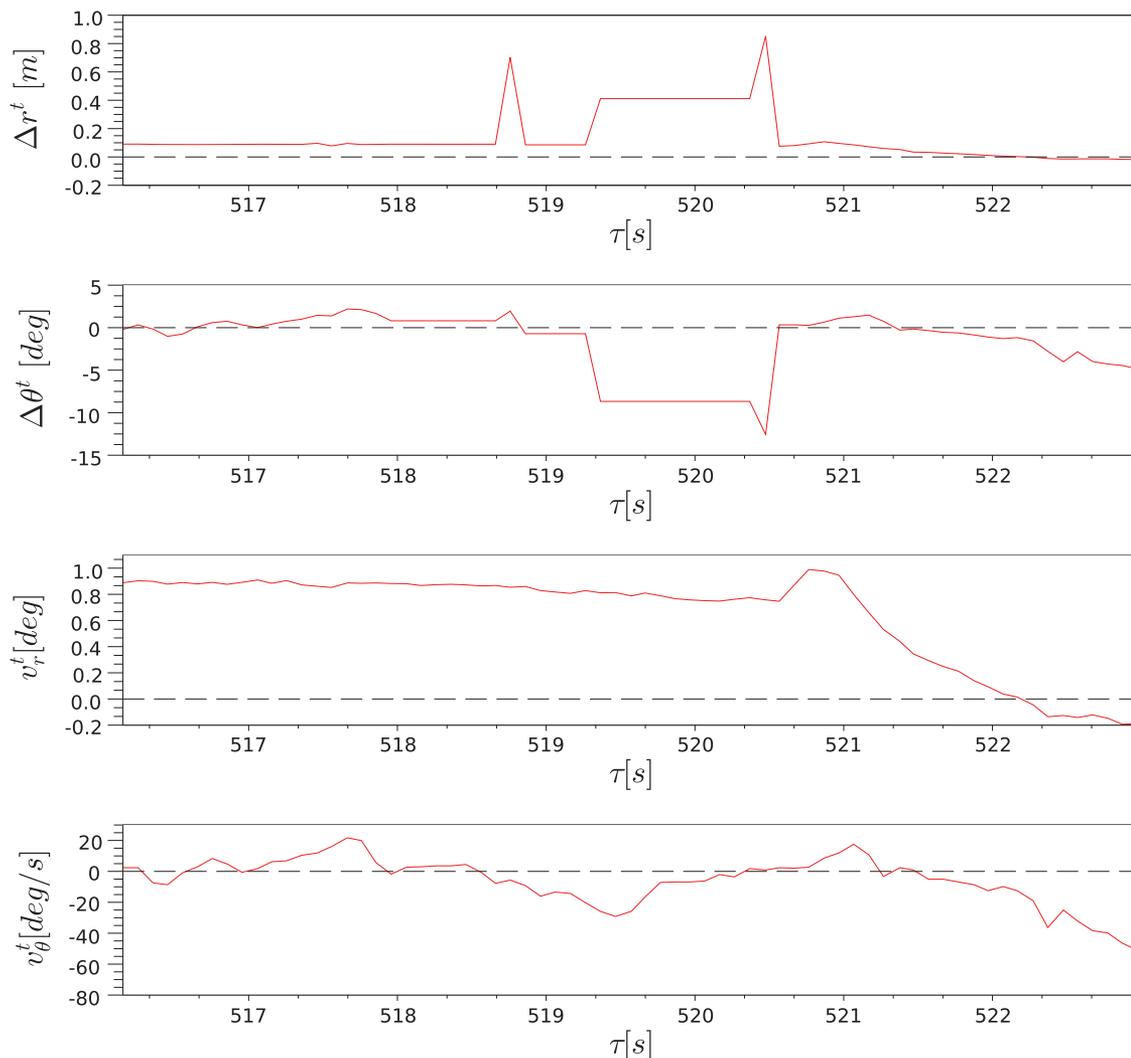
Figure 18. Odometry increments and velocities provided by the Segway platform during the second localization failure. Acquisition period is $T_{odo} = 0.1$ s. After a first peak in translation at $\tau = 518.8$ s, odometry increments are faulty during the time period [519.3, 520.5] s, while platform velocities remain coherent. The robot was moving straight ahead at approximately 1 m/s.

spots when turning on its axis, as explained in Section 8. We would also have to deal with occlusions and dynamic obstacles explicitly. We also plan to explore the use of stereo cameras in hybrid laser–camera solutions, for both localization and obstacle avoidance.

Future research also includes the identification and tracking of moving objects for better motion planning on the local domain and the extension of the path planning and path execution algorithms from 2D to 3D. The latter would enable us to deal with multilevel environments: whereas the campus setting presents great variations in height, there is at most one traversable surface for a (x, y) point. Higher level extensions include the segmentation of unmapped objects for incorporation to the

3D map, by extending past and current work on mapping the campus area (Valencia, Teniente, Trulls, & Andrade-Cetto, 2009). The robot may then deal with situations such as the one encountered in the second experimental session, in which most of a passageway was blocked by public works (see the end of the campus video). Lower level work includes redesigning the robot to reduce its weight and accomplish a better response, specially when traversing ramps.

APPENDIX: INDEX TO MULTIMEDIA EXTENSIONS

The videos are available as Supporting Information in the online version of this article.

Extension	Media type	Description
1	Video	Session 1: Gràcia, 20 May 2010
2	Video	Session 2: Campus, 3 June 2010

ACKNOWLEDGMENTS

REFERENCES

Arras, K., Castellanos, J., Schilt, M., & Siegwart, R. (2003). Feature-based multi-hypothesis localization and tracking using geometric constraints. Journal of Robotics and Autonomous Systems, 44, 41–53.

Arulampalam, M., Maskell, S., Gordon, N., Clapp, T., Sci, D., Organ, T., & Adelaide, S. (2002). A tutorial on particle filters for online nonlinear/non-gaussian Bayesian tracking. IEEE Transactions on Signal Processing, 50(2), 174–188.

Corominas Murtra, A., Mirats-Tur, J., Sandoval, O., & Sanfeliu, A. (2008, November). Real-time software for mobile robot simulation and experimentation in cooperative environments. In Proceedings of the Simulation, Modelling and Programming for Autonomous Robots (SIMPAR'08), Lecture Notes on Artificial Intelligence, Venice, Italy.

Corominas Murtra, A., Trulls, E., Mirats-Tur, J., & Sanfeliu, A. (2010, November). Efficient use of 3D environment models for mobile robot simulation and localization. In Proceedings of the International Conference on Simulation, Modelling and Programming for Autonomous Robots (SIMPAR'10), Lecture Notes on Artificial Intelligence, Darmstadt, Germany.

Corominas Murtra, A., Trulls, E., Sandoval, O., Pérez-Ibarz, J., Vasquez, D., Mirats-Tur, J. M., Ferrer, M., & Sanfeliu, A. (2010, May). Autonomous navigation for urban service mobile robots. In Proceedings of the IEEE International Conference on Robots and Systems, Taiwan, Republic of China.

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. International Journal of Robotics Research, 29(5), 485–501.

Doucet, A., de Freitas, N., & Gordon, N. (2001). Sequential Monte Carlo methods in practice. New York: Springer Science.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. Robotics & Automation Magazine, 4(1), 23–33.

Georgiev, A., & Allen, P. (2004). Localization methods for a mobile robot in urban environments. IEEE Transactions on Robotics and Automation, 20(5), 851–864.

Haddad, H., Khatib, M., Lacroix, S., & Chatila, R. (1998, May). Reactive navigation in outdoor environments using potential fields. In Proceedings of the IEEE International Conference on Robotics and Automation, Leuven, Belgium.

Khatib, M., & Chatila, R. (1995, March). An extended potential field approach for mobile robot sensor-based motions. In Proceedings of Intelligent Autonomous Systems, Karlsruhe, Germany (pp. 490–496).

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research, 5(1), 90–98.

Latombe, J. (1991). Robot motion planning. Norwell, MA: Kluwer Academic Publishers.

LaValle, S., & Ku, J. (2000, March). Rapidly-exploring random trees: Progress and prospects. In Algorithmic and computational robotics: New directions. Fourth Workshop on the Algorithmic Foundations of Robotics, Hanover, NH. Natick, MA: AK Peters, Ltd.

Levinson, J., Montemerlo, M., & Thrun, S. (2007, June). Map-based precision vehicle localization in urban environments. In Proceedings of the Robotics: Science and Systems Conference, Atlanta, GA.

Luettel, T., Himmelsbach, M., Hundelshausen, F. V., Manz, M., Mueller, A., & Wuensche, H.-J. (2009, October). Autonomous offroad navigation under poor GPS conditions. In Proceedings of the IROS 2009 3rd Workshop on Planning, Perception and Navigation for Intelligent Vehicles, St. Louis, MO.

Metta, G., Fitzpatrick, P., & Natale, L. (2006). YARP: Yet another robot platform. International Journal on Advanced Robotics Systems, 3(1), 43–48.

Minguez, J., & Montano, L. (2004). Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios. Transactions on Robotics and Automation, 20(1), 45–59.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., & Thrun, S. (2008). Junior: The Stanford entry in the Urban Challenge. Journal of Field Robotics, 25(9), 569–597.

Morales, Y., Carballo, A., Takeuchi, E., Aburadani, A., & Tsubouchi, T. (2009). Autonomous robot navigation in outdoor cluttered pedestrian walkways. Journal of Field Robotics, 26(8), 609–635.

Nuske, S., Roberts, J., & Wyeth, G. (2009). Robust outdoor visual localization using a three-dimensional-edge map. Journal of Field Robotics, 9(26), 728–756.

OBJ. (–). OBJ file format. http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/. Accessed 24 February 2011.

OpenGL. (–). OpenGL website. http://www.opengl.org. Accessed 24 February 2011.

Philippsen, R. (2004). Motion planning and obstacle avoidance for mobile robots in highly cluttered dynamic environments. Ph.D. thesis, reference number 3146, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.

Rauskolb, F. W., Berger, K., Lipski, C., Magnor, M., Cornelsen, K., Effertz, J., Form, T., Graefe, F., Ohl, S., Schumacher, W., Wille, J.-M., Hecker, P., Nothdurft, T., Doering, M., Homeier, K., Morgenroth, J., Wolf, L., Basarke, C., Berger, C., Glke, T., Klose, F., & Rumpe, B. (2008). Caroline: An autonomously driving vehicle for urban environments. Journal of Field Robotics, 25(9), 674–724.

Sanfeliu, A., & Andrade-Cetto, J. (2006, October). Ubiquitous networking robotics in urban settings. In Proceedings of the IEEE/RSJ IROS Workshop on Network Robot Systems, Beijing, China.

Sanfeliu, A., Andrade-Cetto, J., Barbosa, M., Bowden, R., Capitán, J., Corominas Murtra, A., Gilbert, A., Illingworth, J., Merino, L., Mirats-Tur, J. M., Moreno, P., Ollero, A., Sequeira, J., & Spaan, M. (2010). Decentralized sensor fusion for ubiquitous networking robotics in urban areas. Sensors, 10(3), 2274–2314.

Simmons, R. (1996, April). The curvature-velocity method for local obstacle avoidance. In Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, MN (vol. 4).

Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic robotics. Cambridge, MA: MIT Press.

Thrun, S., Fox, D., Burgard, W., & Dellaert, F. (2001). Robust Monte Carlo localization for mobile robots. Artificial Intelligence, 128, 99–141.

URUS. (–). URUS project website. http://urus.upc.edu. Accessed 24 February 2011.

Valencia, R., Teniente, E. H., Trulls, E., & Andrade-Cetto, J. (2009, October). 3D mapping for urban service robots. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO (pp. 3076–3081).

Wijesoma, W. S., Kodagoda, K. R. S., & Balasuriya, A. (2004). Road-boundary detection and tracking using ladar sensing. Transactions on Robotics and Automation, 20(3), 456–464.

Yun, J., & Miura, J. (2007, April). Multi-hypothesis outdoor localization using multiple visual features with a rough map. In Proceedings of the IEEE International Conference on Robotics and Automation, Rome, Italy.