Kinect Enabled Monte Carlo Localisation for a Robotic Wheelchair

Theodoros Theodoridis, Huosheng Hu, Klaus McDonald-Maier, and Dongbing Gu
School of Computer Science and Electronic Engineering
University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK
{ttheod, hhu, kdm, dgu}@essex.ac.uk

Abstract. Proximity sensors and 2D vision methods have shown to work robustly in particle filter-based Monte Carlo Locali-sation (MCL). It would be interesting however to examine whether modern 3D vision sensors would be equally efficient for localising a robotic wheelchair with MCL. In this work, we introduce a visual Region Locator Descriptor, acquired from a 3D map using the Kinect sensor to conduct localisation. The descriptor segments the Kinect's depth map into a grid of 36 regions, where the depth of each column-cell is being used as a distance range for the measurement model of a particle filter. The experimental work concentrated on a comparison of three different localization cases. (a) an odometry model without MCL, (b) with MCL and sonar sensors only, (c) with MCL and the Kinect sensor only. The comparative study demonstrated the efficiency of a modern 3D depth sensor, such as the Kinect, which can be used reliably for wheelchair localisation.

Keywords: Monte Carlo Localisation, Particle Filter Localisation, Region Locator Descriptors, Kinect sensor

1 Introduction

The need for reliable wheelchair localisation has become a high priority need for the elderly and disabled. To provide adequate autonomous mobility, robotic wheelchairs can be used for transportation. This would require an efficient localisation method, and a depth sensor for coverage. In our effort to tackle the localisation problem with a robotic wheelchair, the Monte Carlo Localisation (MCL) method was used. The perception was undertaken by the Microsoft's Kinect depth sensor, for the coverage of a 640×480 depth pixels in ~60º wide lens angle. The sensor's resolution along with its low cost, compared to 1D laser range finders, makes it an attractive apparatus for localisation.

MCL is one of the most popular probabilistic methods for online robot localization, which is based on a particle filter algorithm. MCL was primarily introduced by [2], on their work in mobile robot localisation. MCL in its purest form tackles the global position estimation, and the local position tracking problem. To perform localisation, the robot's state space is represented by a sample (particle) density, which approximates the robot's position. The whole process is conducted in two phases; the *prediction* and *update*. In the prediction phase a motion model predicts the robot's

current pose with a Probability Density Function (PDF). Similarly, in the update phase a measurement (sensor) model is incorporated to evaluate each particle in the density, and sample with replacement those particles evaluated as good predictors. In [3], an alternative MCL method is proposed based on an adaptive particle density, which showed improved accuracy and less computation effort. A stereo vision based MCL was presented by [4], using a scale invariant feature transform descriptor, given a 3D map. The measurement model incorporated 3D features extracted from landmarks. The proposed method appeared to solve the 6 DOF motion, and perform accurate localisation with a motion model having sensor measurements only. Another MCL method that exploits visual features was introduced by [5], for the localisation of the Aibo dog-robots. Two MCL variants were proposed; a landmark, and a field line-based extraction. The MCL using the visual features performed fast and reactive localisation within a miniaturised football pitch. In [7] a landmark tree model is employed for self-localisation of an autonomous wheelchair. The method utilises an image retrieval technique and the Bayes rule to localise the wheelchair. In addition, a path planning algorithm is introduced, which exploits a treelike structure to locate landmarks and destination locations. Landmarks are recognised through an image by extracting the shape and structure, and localisation is conducted by traversing within the tree nodes to elicit an optimal path. A probabilistic odometry (motion) model was introduced by [8] for an autonomous wheelchair. The method constructs a set of frequency tables of the wheelchair's pose stored in bins. A particle filter advises these tables to make predictions for localisation.

The rest of the paper is organized as follows. Section 2 describes the classical Monte Carlo Localisation method, given a brief algorithm. Section 3 presents the Kinect sensor and a depth-based region locator descriptor used in MCL. The probabilistic models that implement the MCL's structure are given in Section 4. Section 5 demonstrates a tri-case experiment using a mobile robot, and Section 6 points out conclusions and future directions.

2 Monte Carlo Localisation

The general idea of MCL is the deployment of a particle density, designated to predict a robot's position. The particles can be seen as virtual copies of the robot's existence, incorporating 2D kinematics for locomotion (motion model), and a beam array for perception (measurement/sensor model). As stated, MCL is conducted in two main phases to perform localisation. In the *prediction* phase at every time step the particles' kinematics is updated when the robot transits from state x_{k-1} to x_k . Each particle is assigned with a weighting fitness value that assesses how well the actual position of the robot is approximated. This accounts the actual (the robot's) and the expected (the particle's) sensory perception. In the *update* phase a subset of particles is drawn probabilistically with replacement to represent the new particle population. The resampling is based on the Darwinian principle of natural selection. Hence, high fitness particles have higher probability to be selected multiple times. Progressively, the whole particle population converges near the actual position of the robot.

Algorithm 1 Monte Carlo Localisation

1: Generate Particles $\mathrm{X}_{\mathrm{x}} \sim N(\mu_{\mathrm{x}}, \sigma_{\mathrm{x}}^2)$

2: Initialise Particle Weights $w_k = 1/Q$

3: Prediction $\mathrm{x}_k \sim p(\mathrm{x}_k \mid \mathrm{x}_{k-1})$

4: Weight Estimation:

$$w_k = p(z_k \mid \mathrm{x}_k, u)\, w_{k-1}, \quad w_k = w_k / \sum_{q=1}^{Q} w_k^{[q]}$$

5: Update $\mathrm{X}_k \sim \left\langle \mathrm{x}_k^{[q]}, w_k^{[q]} \right\rangle_{q=1}^{Q}$

Algorithm 1 describes briefly the MCL method in five steps. In the first, a particle distribution X is created with randomly chosen state vectors $\mathbf{x} \equiv \langle\langle x, y, \theta\rangle, w\rangle$ from a Gaussian distribution *N*. The randomly chosen parameters are the robot's pose at coordinates *x*, *y*, θ, and the weight factor *w*. Step 2 initialises the particles weights with the factor 1/*Q*, where *Q* is the total number of particles. In step 3, the prediction takes place of the state x_k given the initial state x_{k-1}. At this step, the robot's kinematics is applied at each particle. The weight estimation in step 4 is done by the product of the measurement probability shown as z_k, given the state x_k after applying a control command *u*, for particle *q*. Normalisation follows to scale the particle weights in the interval {0, 1}, so all add to 1. The update phase in step 5 samples with replacement a subset of particles based on their weight values w_k. The process is repeated recursively by looping back to step 2.

3 Kinect Depth-Based Perception

3.1 Kinect Sensor

The main sensor apparatus utilised in this work is the Kinect sensor (see Fig. 1), produced by Microsoft. Kinect is a vision sensor providing a RGB and a depth image in a single device. It is being used by Microsoft's game industry for games that do not require physical control devices such as joysticks. Basically, the user becomes the controller by using gestures and body movements. For using the sensor without the X-Box, as it is the primary device to work with, several drivers have been released to access the device from a PC. For the current application we have used the CL-NUI-Platform driver for Windows 7 (version 1.0.0.1121).

Figure 1. Microsoft's Kinect sensor.

The Kinect's video frame rate is up to 30 Hz, with an 8-bit VGA resolution of 640×480 pixels. The device provides 11-bit depth with 2,048 levels of intensity. The sensor's approximate ranging limit is between 0.5 to 6 m distance, while the angular field of view is 57° horizontally, and 43° vertically. There is also an embedded quadruple microphone array with a 16-bit audio and sampling rate of 16 kHz [6].

3.2 Region Locator Descriptor

The sensor coverage of a significant angular range is a crucial factor for autonomous and safe transportation. Kinect's specifications allow us to cover a large field of view from which information can be acquired. Such information regards the extraction of depth in local regions in this field. The local depth regions can play a twofold role; initially to harness them for the MCL's sensor model to conduct localisation. Secondly, for obstacle avoidance and safe navigation.

A local depth region acquisition method is being proposed to work for localisation purposes. For this reason, we generate a 6×6 depth matrix, which discretises the image in 36 depth regions as depicted in Fig. 2. The region matrix is given by matrix **A** in Eq. 1, with entry elements the regions represented by the matrix **B**. The elements of matrix **B** are the individual depth pixels indexing 107×80, as divided by 6 from the overall image resolution. For each region **B** the average depth is estimated, therefore matrix **A** now consists of the mean values for each region. The final region locator descriptor (*rld*) is a vector with 6 elements reflecting to the minimum distance from each column *n* (0, ..., 5) of matrix **A**.

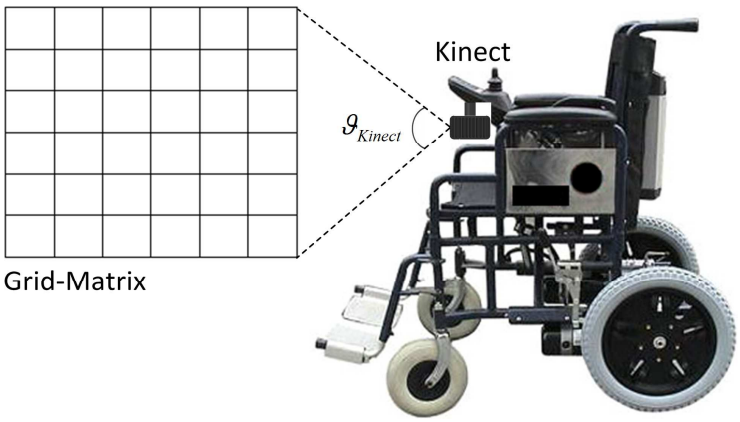

Figure 2. The Kinect sensor attached on a robotic wheelchair. The grid-matrix is extracted by the visual descriptor from the Kinect's depth image.

$$rld = \begin{bmatrix} \underset{m}{\operatorname{argmin}}\, \mathrm{A}_{m,0} \\ \underset{m}{\operatorname{argmin}}\, \mathrm{A}_{m,1} \\ \underset{m}{\operatorname{argmin}}\, \mathrm{A}_{m,2} \\ \underset{m}{\operatorname{argmin}}\, \mathrm{A}_{m,3} \\ \underset{m}{\operatorname{argmin}}\, \mathrm{A}_{m,4} \\ \underset{m}{\operatorname{argmin}}\, \mathrm{A}_{m,5} \end{bmatrix} \Leftarrow \mathrm{A} = \begin{bmatrix} \overline{\mathrm{B}}_{0,0} & \overline{\mathrm{B}}_{0,1} & \cdots & \overline{\mathrm{B}}_{0,5} \\ \overline{\mathrm{B}}_{1,0} & \overline{\mathrm{B}}_{1,1} & \cdots & \overline{\mathrm{B}}_{1,5} \\ \vdots & \vdots & \ddots & \vdots \\ \overline{\mathrm{B}}_{5,0} & \overline{\mathrm{B}}_{5,1} & \cdots & \overline{\mathrm{B}}_{5,5} \end{bmatrix}, \overline{\mathrm{B}} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,80} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,80} \\ \vdots & \vdots & \ddots & \vdots \\ p_{107,0} & p_{107,1} & \cdots & p_{107,80} \end{bmatrix} \quad (1)$$

4 Probabilistic Models

There are four models required for the implementation of a Markov localisation method such as MCL. These models are being employed to represent mainly the particles' pose and perception. A *motion model* expresses the probability for certain actions to move the robot to certain relative positions. A *sensor model* describes the probability for taking certain measurements at certain locations [5]. A *noise model* adds Gaussian noise to the motion model, so as to spread the particle density. Finally, an *odometry model* can be incorporated to conduct a waypoint navigation. The MCL's particle filter obviously acts on the odometry model, refining its direction towards a predefined location.

4.1 Motion Model

The effects of actions on the robot's pose are represented by a motion model [5]. At every time step the robot's kinematics updates the particle's kinematics. The kinematic update is instrumented by Eqs. 2, 3, and 4, with d denoting the delayed linear displacement, and ϑ the delayed rotational one. The noise terms ε are random Gaussian noise explained in Section 4.3.

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + (d + \varepsilon_x)\cos\theta \\ y_{k-1} + (d + \varepsilon_y)\sin\theta \\ \theta_{k-1} + (\vartheta + \varepsilon_\theta) \end{bmatrix} \tag{2}$$

$$d = \sqrt{(x_{k-1} - x_k)^2 + (y_{k-1} - y_k)^2} \tag{3}$$

$$\vartheta = |\theta_{k-1} - \theta_k| \tag{4}$$

In the current work, a priory virtual map M is provided for the interaction of the particles with their environment. The ones which violate the map's boundaries are regenerated with a random pose within the map. The map is represented here by a pair of points, each defining the start-end coordinates of a line. Fig. 3 depicts the line representation.

Algorithm 2 illustrates the map validation procedure. Step 1 describes a state estimate for the current particle *q*. The mapFlag at step 2 is set to true, as it decides later on whether the state of the evaluated particle violates the map's boundaries. The for-loop in step 3 validates the particle's *x*, *y* state coordinates with each line in the map. A newly random coordinate state is chosen for the particle which violates the boundaries, or, the particle is left intact otherwise.

Algorithm 2 Particle map (M) validator

1: Estimate $\mathbf{x}_k^{[q]}$
2: mapFlag = **true**
3: **for** i = 1 to M **do**

$$j = (i + \mathrm{M} - 1)\,\%\,\mathrm{M}$$

if $\left(\begin{array}{l} (\mathrm{M}_y^{[i]} > q_y) \neq (\mathrm{M}_y^{[j]} > q_y)\ \& \\ q_x < \dfrac{(\mathrm{M}_x^{[j]} - \mathrm{M}_x^{[i]})(q_y - \mathrm{M}_y^{[i]})}{(\mathrm{M}_y^{[j]} - \mathrm{M}_y^{[i]})} + \mathrm{M}_x^{[i]} \end{array} \right)$

$$\mathrm{mapFlag} = \overline{\mathrm{mapFlag}}$$

end if
end for
4: **if** mapFlag = **true**
return rand($q^{[\mathrm{x}]}$)
end if

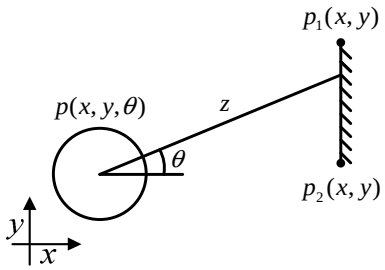

Figure 3. Beam representation of a particle.

4.2 Sensor Model

The sensor model is described by the likelihood probability $p(z|\mathrm{x})$ of Eq. 6. It is a probabilistic error between the particle's virtual beam range z (Eq. 5), as shown in Fig. 1, and the real robot sensor range s, acquired from the visual depth descriptor. The product of the likelihood probabilistic error between virtual and real sensor ranges, for particle q, constitutes the particle's goodness of fit (weight, Eq. 7). The weight is an assessment of how close the particle approximates the robot's state vector **x**.

$$z = \frac{[p_2(y) - p_1(y)][p_1(x) - x_q] - [p_2(x) - p_1(x)][p_1(y) - y_q]}{[p_2(y) - p_1(y)]\cos\theta - [p_2(x) - p_1(x)]\sin\theta} \tag{5}$$

$$p(z\,|\,\mathrm{x}) = \begin{cases} 1, & \text{if } |s - \bar{z}| > \sigma_z \\ 1 - \dfrac{|s - \bar{z}|}{\sigma_z}, & \text{otherwise} \end{cases} \tag{6}$$

$$w_q = \prod_{i=1}^{S} p(z^{[i]}\,|\,\mathrm{x}) \cdot w^{[i]} \tag{7}$$

where $\bar{z}$ is the average of a measurement sample, and σ_z the standard deviation. Ultimately, *S* is the max number of sensors, which has been defined primarily from the visual descriptor as 6 (region matrix **A**, Eq. 1). Here, for consistency the sensor model assigns six beam sensors to each particle with 10° angular difference.

4.3 Noise Model

The noise model is generated using a zero-mean Gaussian distribution. For the translational noise (see Eq. 8) the delayed distance *d* is multiplied with the Gaussian *N* to acquire a random noise scalar, which is added with *d* in the kinematic update of Eq. 1. The exponent *n* is chosen for introducing more (=1) or less noise (>1). Similarly for the rotational noise.

$$\begin{aligned} \varepsilon_x &\sim N(0,\, \sigma_x^2)^n \times d \\ \varepsilon_y &\sim N(0,\, \sigma_y^2)^n \times d \\ \varepsilon_\theta &\sim N(0,\, \sigma_\theta^2)^n \times \vartheta \end{aligned} \tag{8}$$

The random Gaussian noise is sampled from a multivariate distribution *N*, which is characterised by a density function [1] with $\mu = 0$ and $\sigma = 1$ (see Eq. 9).

$$N = \det(2\pi\Sigma)^{-1/2} \exp\left\{-\frac{1}{2}\mathrm{v}^{\mathrm{T}}\Sigma^{-1}\mathrm{v}\right\} \tag{9}$$

where Σ is the covariance matrix, and **v** a random vector.

4.4 Odometry Model

An odometry model is particularly useful for waypoint localisation. The model describes the robot's motion in a rotational and a translational displacement. The rotational displacement is estimated by the arctangent between the current (*x*, *y*) and the next (*x*', *y*') coordinate pairs. The translational displacement is given by the difference of the same coordinate pairs, estimated with the Euclidean distance. The model's displacements are described by Eq. 8

$$\begin{aligned} \delta_{rot1} &= \mathrm{atan2}(y'-y,\, x'-x) \\ \delta_{trans} &= \sqrt{(x-x')^2-(y-y')^2} \\ \delta_{rot2} &= \theta'-\theta \end{aligned} \tag{8}$$

The robot's motion at (*t*-1, *t*] is approximated by a sequence of transitions as shown in Fig. 2 [1]. Initially, a rotational transition δ_{rot1} is performed, followed by a translational transition, δ_{trans} and a consequent rotational δ_{rot2}. The robot's transitional displacements over a number of nodes completes the waypoint localisation. The purpose of the MCL's particle filter is to act on the odometry model by refining the robot's heading direction towards a designated node.

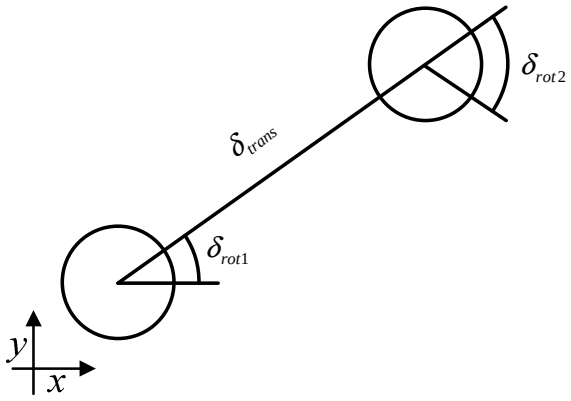

Figure 4. Representation of the odometry model in two rotational, and one translational transition.

5 Experimental Results

5.1 Experimental Setup

The Essex robotic arena was the main experimental hall where the experimental procedure took place. The environmental setup illustrated in Fig. 5(a) was used for the conduction of the experimental cases. The experiments were separated in three different cases so as to show the efficiency of the visual descriptor when applied for MCL, and the Kinect sensor when compared to proximity sensor localisation. In addition, the initial experiments were taken by an Activmedia Pioneer robot (Fig. 5(b)) for testing purposes before the actual wheelchair is used to carry human beings.

We have used an odometry model to navigate the robot through 19 nodes (waypoints) as Fig. 5(a) also depicts. Lastly, Fig. 5(c) illustrates the region depth map using the robot's onboard Kinect sensor.

(a) (b) (c)

Figure 5. (a) The experimental environment, (b) The Kinect-enabled mobile robot, (c) The grid depth map.

5.1 Localisation Cases

a) Odometry Model

Employing an odometry model only, the robot is obviously lost so an incorporated obstacle avoidance undertakes to help the localisation. Fig. 6(a) presents an experimental trial for navigating with an odometry model. The trajectory seems rather unstable due to several maneuvers performed by the robot for the avoidance of the walls. From the poor localisation performance shown in this graph, we see the importance of how useful would be a probabilistic filter for the autonomous transportation of humans.

b) Sonar-Based MCL

The classical MCL method, as initially proposed by [2, 3], employed proximity sensors. In this case, we performed MCL with an array of six sonar sensors. The resultant trajectory of Fig. 6(b) demonstrates a much smoother trajectory when we introduce the MCL method to the odometry model. A diminished use of the obstacle avoidance algorithm is observed, and the path now is drawn clearer.

c) Kinect-Based MCL

The proposed MCL method, enhanced with an incorporated region locator descriptor, shows in Fig. 6(c) to perform better than the sonar-based MCL, and apparently better than the odometry model on its own. The performance enhancement is owed to fact that the Kinect sensor utilises a complete coverage of the frontal view. This turns out to be essentially useful for the sensor model, as now the particles are evaluated more accurately. The Kinect's depth coverage in x and y image coordinates, contrary to lasers acting in x only, is proven also to be safer and more robust for a robotic wheelchair use.

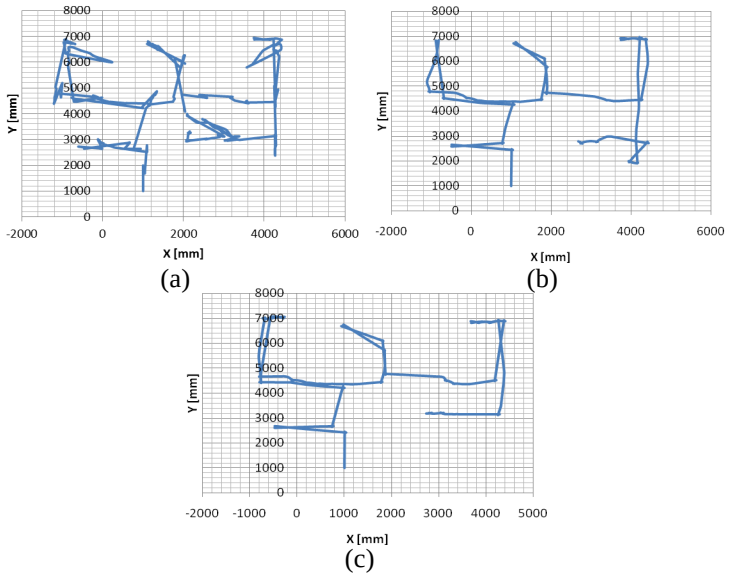

Figure 6. Experimental trajectories. (a) Using an odometry model, (b) Using sonar-based MCL, (c) Using Kinect-based MCL

6 Conclusions

In this work an attempt has been made to introduce a region locator descriptor, through a depth image, to Monte Carlo Localisation method. The Microsoft's Kinect sensor was the main apparatus incorporated to enhance the localisation performance. Indeed, the experimental work showed that the suggested descriptor can improve the localisation accuracy when compared with proximity sensors.

Our future directions will focus on testing colour and edge histogram descriptors for MCL. Eventually, to test the performance of the current method on different wheelchair types for indoor and outdoor localisation.

Acknowledgements

This research has been financially supported by the EU Interreg IV A 2 Mers Seas Zeeën Cross-border Cooperation Programme – SYSIASS: Autonomous and Intelligent Healthcare System. More details can be found at the project's website http://www.sysiass.eu/. We would also like to thank Robin Dowling for the technical support.

References

1. S. Thrun, W. Burgard, and D. Fox, "Probabilistic Robotics," MIT Press, Cambridge, MA, 2005.
2. F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo Localization for Mobile Robots," IEEE International Conference on Robotics and Automation, 1999, pp. 1322-1328.
3. D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," Proceedings of the Sixteenth National Conference on Artificial Intelligence, 1999, pp. 343-349.
4. P. Elinas and J. J. Little, "sMCL: Monte-Carlo Localization for Mobile Robots with Stereo Vision," In Proceedings of Robotics: Science and Systems, 2005, pp. 373-380.
5. T. Rofer and M. Jungel, "Vision-Based Fast and Reactive Monte-Carlo Localization," IEEE International Conference on Robotics and Automation, 2003, pp. 856-861.
6. Wikipedia contributors, "Kinect," Wikipedia: The Free Encyclopedia. Wikimedia Foundation, 2004 [Online]. Available: http://en.wiki pedia.org/wiki/Kinect.
7. X. Zhao, X. Li, and T. Tan, "A Novel Landmark Tree Based Self-Localization and Path-Planning Method for an Intelligent Wheelchair," Proceedings of the 9th IEEE International Workshop on Robot and Human Interactive Communication, 2000, pp. 84-89.
8. T. Yaqub, M. J. Tordon, and J. Katupitiya, "A Procedure to Make the Probabilistic Odometry Motion Model of an Autonomous Wheelchair," Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation, 2006, pp. 526-531.