MAP CONSTRUCTION AND LOCALIZATION USING LEGO MINDSTORMS NXT

Submitted: 7th August 2012; accepted: 17th May 2013

Jasmeet Singh, Punam Bedi

Abstract:
Maps are very useful for understanding unknown places before visiting them as maps represent spatial relationships between various objects in a region. Using robots for map construction is an important field these days as robots can reach places which may be inaccessible to human beings. This paper presents a method to use the data obtained from a single ultrasonic sensor mounted on a robot, to construct a map and localize the robot within that map. Map of the previously unknown environment is created with the help of a mobile robot, built using Lego Mindstorms NXT assembled in a modified TriBot configuration. The robot is equipped with an ultrasonic sensor and is controlled from a computer system running a MATLAB program, which communicates with the NXT over a USB or Bluetooth connection and performs complex calculations that are not possible for the NXT itself. After the map construction, the robot finds its position in the map by using a particle filter. Implementation has been done in MATLAB programming environment using RWTH – Mindstorms NXT Toolbox and has been successfully tested for map construction of a room and localization within that room with the use of a TriBot.

Keywords: *Lego Mindstorms NXT, TriBot, ultrasonic sensor, map construction, localization, particle filter, MATLAB, mobile robot, RWTH – Mindstorms NXT Toolbox*

1. Introduction

Maps are used in a variety of applications for our day-to-day needs. These maps can be considered as macro-scopes which are used primarily to represent spatial relationships between various objects in a region, on a smaller-scale. Constructing a geometrically consistent map is a tedious process that requires determining spatial relationship between various objects, which in turn requires a lot of measurements. The presented work describes a method to construct a map using a Lego Mindstorms NXT based TriBot (referred here-on as simply NXT or TriBot), which has been successfully tested for creating an indoor map of a room.

This paper is divided into two main sections of Map Construction and Localization and the Kidnapped Robot Problem. The map construction and localization section describes how the robot is able to take readings of its surroundings, move in the environment, and also how the constructed global map is displayed based on the readings obtained by the ultrasonic sensor of the TriBot. The latter section describes how the robot's position is found within the constructed map, after it has been placed at a random position within the mapped environment, by using the readings obtained from its sensors and its knowledge of the map.

Previous work in localization of robots has been carried out with the use of more than one ultrasonic sensor at once [1–3]. This requires the robot to know which sensor returned the reading and hence the robot needs to distinguish between several ultrasonic sensors that were mounted on it. With the use of multiple sensors, the cost of the robot increases as well. The work carried out in this paper requires the use of only a single ultrasonic sensor for localization, placed at the origin of the robot reference frame thereby eliminating additional computations for finding the relative positions of all the ultrasonic sensors, with respect to the reference frame, as in [1, 3]. When using particle filters for solving the global localization [1, 4], the map given as an input to the particle filter algorithm, was not built by the robot itself [1] and was constructed by the user via other techniques. But here the TriBot itself constructs the global map that is later used providing a solution to the kidnapped robot problem [4] described in the localization s7th ection.

Mapping with ultrasonic sensors has previously been reliant on the use of multiple sensors in ring formation [5]. With the use of a single ultrasonic sensor it becomes possible to successfully map the environment surrounding the NXT, while keeping the computational complexity to a minimum. It also helps in specifying the number of readings that must be received from the surroundings for each position of NXT, without having to change the number of ultrasonic sensors in the robot's hardware.

For global map construction and localization, the concept of occupancy grids has also been previously employed in [6, 7] which, though effective, depends on the size of the grid cells. A robot when moved to a frontier is able to see the unexplored area of the environment. By constantly moving to successive frontiers the robot is able to increase its knowledge of the surrounding environment. Frontier based exploration have been coupled with occupancy grids for the map construction and localization in [6] but in the implementation of such a system multiple laser range finders, sonar sensors and infrared sen-

sors were used to make the resulting map as accurate as possible. The amount of computation and complexity increases with multiple sensors, but with the use of a single ultrasonic sensor amount of computation is low. In this work frontier-based exploration strategy has been employed without using the occupancy grids and only the readings obtained at the current position of the robot are used for deciding the next frontier for the robot.

In other works of map construction and localization, Simultaneous Localization and Mapping (SLAM) has been used to localize the robot's position as it moves within the environment, while mapping is being performed [3, 4, 8]. In the current system, each robot position is relative to the origin position of the map and the robot has no knowledge of a prior map. It is also possible to implement map construction with a swarm of robots in the same environment [8] to speed up the process but the implementation has been limited to a single robot so as to ensure that a single robot is able to perform the task of map creation efficiently. It is also easier for the computer to keep track of a single robot.

2. Background

2.1. Environment

The environment considered for the implementation is partially observable, with a single robot working in a continuous state space. The actual environment is an empty room created by temporary partitions that is available specifically to the robot and has no human interference, for the purposes of the given experiments.

2.2. Assumptions Taken

- The walls of the boundary of room and the objects within the room are immovable.
- Friction between the floor of room and the wheels of robot is just enough so that, during the execution of a turn at a position, the stationary wheel of the robot does not slip.
- For map construction, the TriBot's first location is assumed to be (0, 0) with a 0 degree orientation.

2.3. Software

MATLAB is used for the purposes of moving the robot, taking readings of the local surrounding environment, performing calculations for map creation and localization, and displaying the created global map.

The RWTH – Mindstorms NXT Toolbox [9], version 4.04 for MATLAB, establishes a connection from the program, running on the computer, to the TriBot. With the use of this toolbox it possible to control all the motors and sensors attached to the TriBot. Hence, it is possible to move the robot by controlling its wheeled motors, use the ultrasonic sensor to detect distances from objects, and also rotate the motor on which the ultrasonic sensor is mounted, through the MATLAB program.

For controlling of the wheeled motors and determining the robot's movements, it is important to find the number of degrees the motors must rotate to be able to travel a certain distance. The diameter of the wheels used is 5.5 cm and hence for a 360 degree revolution of the wheels the robot moves forward by 5.5*π centimeters. The degree of revolution, for the wheeled motors, is calculated according to the distance in centimeters that the robot is required to move.

To use the ultrasonic sensor it is necessary to first establish a serial connection to the NXT brick and to its port on which the ultrasonic sensor is attached. By using this port, the distance readings recorded by the sensor are returned to the MATLAB program via the USB or Bluetooth connection.

2.4. Sensor and Motors

The ultrasonic sensor used with the TriBot measures the time-of-flight of a burst of high frequency sound wave. The value returned by the sensor is a single number representing the straight line distance of an object from the sensor. These values are in centimeters, and are read by the MATLAB program. The ultrasonic sensor is able to detect distances ranging from 4 centimeters to 2.3 meters with an average precision of 3 centimeters and returns better readings when the object to be detected has a hard surface. If a wave hits the surface at a wide angle from the surface normal [5] it may not return to the sensor, in which case a value of 255 is received from the sensor, indicating an error. The wave may also bounce off a separate surface and create a ghost image.

The interactive servo motors of the TriBot are equipped with an integrated encoder which records a count value of 360 for a single circular revolution i.e., when the motor rotates by 360 degrees, a count of 360 is recoded. This encoder has an accuracy of 1 degree. The motor is able to return this count to the program via the serial connection. These motors have a default maximum speed of 1000 encoder counts per second which means the robot can travel in a straight line at approximately 48 centimeters per second. For the purpose of experimentation, the wheeled motors have a fixed speed of 800 encoder counts per second (approximately 38.4 centimeters/second) and maintain a constant torque during motion. The program can also set the *tacho limit* of the motor, which specifies the integer number of encoder counts (or degrees) that a motor executes before coming to an abrupt stop when this limit is reached. In order for the NXT to move one centimeter in the forward direction, both the motors must move in the same direction by a tacho limit of approximately 21 encoder counts. The two wheels are 11 centimeters apart and so to turn the robot by 1 degree, while moving one wheel and keeping the other wheel stationery, the motor must move by a tacho limit of 4. The motor on which the ultrasonic sensor is mounted rotates at a speed of approximately 400 counts per second. So, a 360 degree rotation of the ultrasonic sensor and detection of distances after every 6 degrees takes 30 seconds, on average.

2.5. Robot Used

The robot used for the purposes of experimentation is based on the Lego Mindstorms NXT TriBot,

with a few modifications. The TriBot is equipped with one ultrasonic sensor, mounted on an interactive motor. It is possible to create a ring of 60 sensor readings in one complete circular turn by rotating the sensor anticlockwise by 360 degrees and taking readings after every 6 degree interval. The ultrasonic sensor has been placed at the origin of the robot reference frame [2].

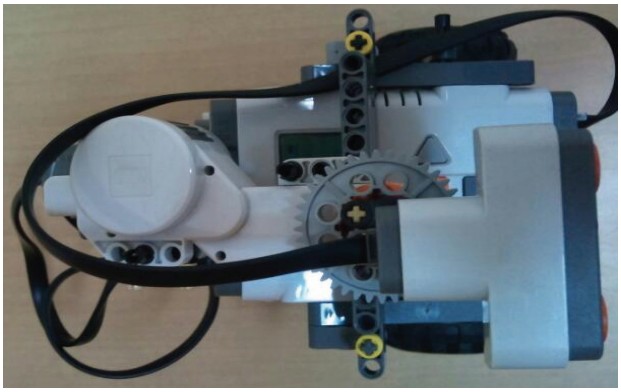

Fig. 1. Top view of the Lego Mindstorms NXT TriBot based robot with an interactive servo motor and the ultrasonic sensor attached on top

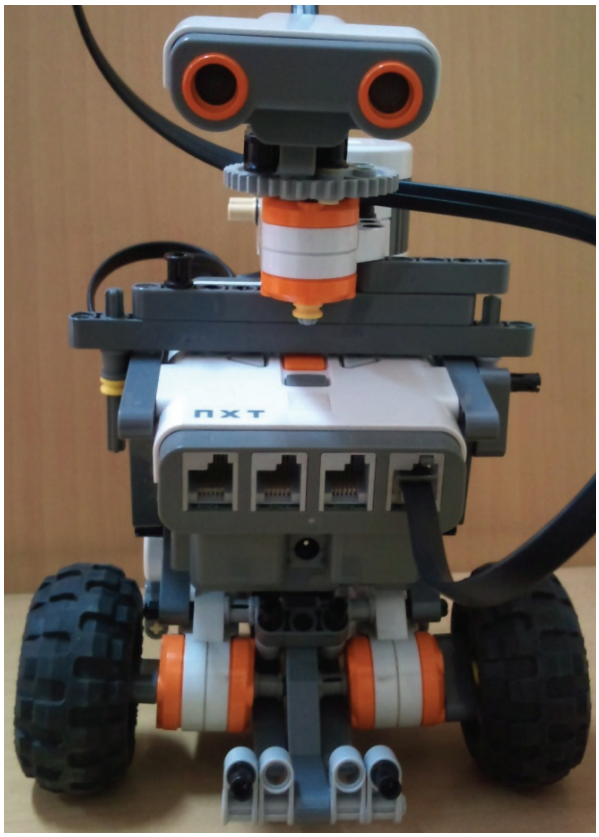

Fig. 2. Front view of the Lego Mindstorms NXT TriBot based robot with the ultrasonic sensor pointing forward

The central component of the TriBot is the NXT brick which is essential for establishing a connection with the computer and controlling the sensors, and motors. Figure 1 shows the TriBot used for experimentation purposes.

Figure 2 shows the front view of the TriBot used. The wheeled motors, on either side of the robot, are synchronized so that if one wheel moves forward the other also moves forward by the same degree and at the same speed. Hence both the wheels move in unison and it is not necessary to control the power and tacho limit for each motor separately. This is important when the robot moves forward or backward to ensure that it follows a straight line and does not drive in a curve. It is also possible to control a single wheeled motor separately when the robot executes a turn.

Figure 3 shows the schematic diagram of the TriBot. The ultrasonic sensor is represented by a yellow triangle, and the orange circle in the center of the robot represents the motor on which the sensor is mounted. The intersecting perpendicular lines within the robot represent the robot reference frame. Origin of the robot reference frame is at the point where these lines intersect. The two wheels on either side of the robot and a swiveling wheel are also shown in the figure using round edged rectangles.

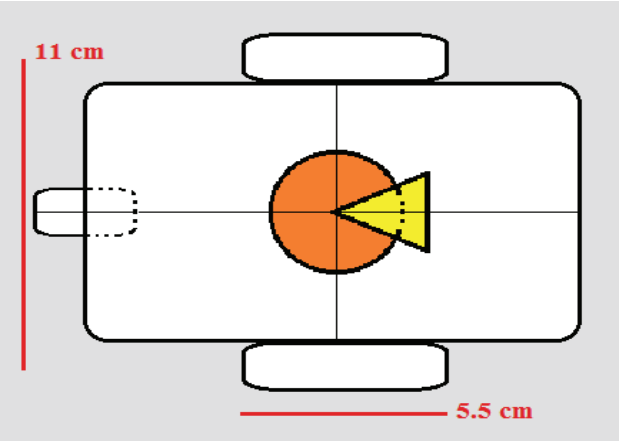

Fig. 3. Schematic representation of the Lego Mindstorms NXT TriBot based robot depicting the robot reference frame

3. Map Construction and Localization

Map construction process involves recording and displaying the robot's surrounding environment. It is implemented as a continuous cycle of two steps namely, sensing and motion, as shown in Figure 4.

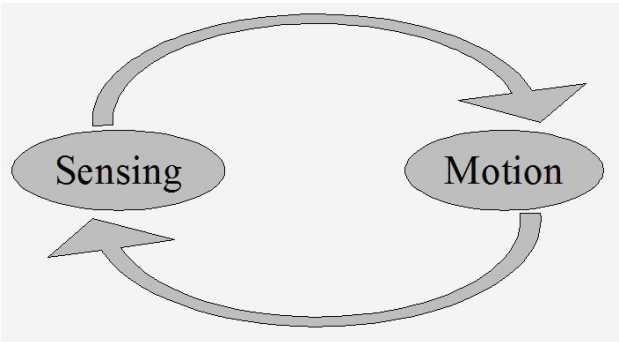

Fig. 4. Map construction process used by the TriBot in implementation

The robot is able to navigate around in an area and record the features of that space. It initially starts at the origin position around which the global map is constructed. In this system, the origin of the robot reference frame is initially assumed to coincide with the origin of the map and the Cartesian coordinates of the robot in the map are (0, 0) with an orientation of 0 degree with respect to the X-axis. A position of the robot in the map is the single set of values, of its X and Y coordinates and its orientation, with respect to the origin of the map. These positions are recorded as distinct rows in a matrix in the MATLAB program; where origin of the map is saved as the first position of the robot. The ultrasonic sensor mounted on the robot is initially assumed to be pointing in the 0 degree direction with respect to the X-axis of the map.

3.1. Sensing

In this step the robot senses its current local surroundings by using the ultrasonic sensor. The TriBot takes a set of 120 ultrasonic readings of the area around its current position by rotating the motor by a 6 degree interval in the counter-clockwise direction, for a total of 720 degrees, instead of a revolution of 360 degrees, which decreases the probability that the recorded readings are representative of a ghost image. The values thus obtained are relative to the origin of the robot reference frame and denote the robot's immediate neighborhood.

Angles, at which these readings are recorded, are also saved along with the distance values which depict the recorded straight line distance between the robot and an object at that angle. The motor is rotated till the sensor points at an angle of 714 (or 354) degrees with respect to the positive X-axis. These 120 readings, saved in a two-column matrix, represent the local map of the robot at that position and the columns in the matrix represent the radius and angle of the Polar coordinates of the objects from the current position of the robot.

After obtaining a set of readings, for each distance reading of 255 the value is deleted and the corresponding angle of that reading is also deleted, so that these readings are not considered for map construction. The sensor is then rotated back to the 0 degree position, as it was before the readings were taken. To display the map, Polar coordinates of the objects taken with respect to the robot reference frame are converted into Cartesian coordinates. The map is initially empty and the set of readings taken at the origin of the global map, are then added to the map.

For all the next sets of readings representing the TriBot's local map, at some known position relative to the origin of the map, the calculated Cartesian coordinates of the readings are shifted by the X and Y values of the TriBot's current position before being added to the existing map. The robot's current position, with respect to the global origin, is also plotted onto the map. This process continues till the robot keeps exploring the room, and the then existing map is treated as the final map of the room.

The robot stops at each new position and records the local map readings of that position, a process which takes 60 seconds to obtain 120 readings. The error in the readings obtained is dependent on the ultrasonic sensor and so an average error of 3 centimeters is possible in the readings, which is a fairly acceptable value for this kind of sensor.

3.2. Motion

After sensing the environment surrounding the robot, the distance in centimeters and the direction of the next frontier from the robot's current position is computed and the robot then proceeds to move toward this new frontier location.

The robot moves to a new frontier location so that it is possible to detect the objects in the previously unexplored area of the room. To ensure that the TriBot is able to explore the complete room efficiently, the 120 distance readings obtained for the current position of the robot are sorted and one of the four farthest values of the recorded distances is randomly selected. Random selection is chosen, as it is possible that the distances recorded are erroneous readings due to a ghost image.

The TriBot turns according to the angle of the direction of the farthest point selected with respect to the robot reference frame and it is then moved half the distance to that point. The distance is halved to counteract the possibility that the point selected represents a ghost image. Every new position of the robot depends only on the relative displacement from the previous position which in turn depends on the readings taken on that position. The robot's relative displacement from the previous position to the new position is stored in a matrix. The relative orientation from previous position is the angle which the robot turned, found during the new frontier location calculation. The X and Y coordinates representing the new position achieved by displacement from previous position are found by converting the distance moved and the angle rotated with respect to the previous position into Cartesian values. The current position of the robot relative to the origin of the map is calculated by taking the arithmetic sum of all previous relative displacements and adding it to the displacement from last known position to the current position. Hence the robot's position is localized by calculating the sum of all the displacements.

The sensing and motion steps are repeated continuously, many times, to construct the final map by the robot through exploration of the environment and to find the position of the robot after every relative displacement. The readings taken at each position contribute to the calculation of the next frontier for the robot and the robot has no knowledge of the readings taken previously. The robot also has orientation information relative to the X-axis of the global map, though it may have approximately 0.25 degrees of error from the previous position of the robot.

All the Cartesian coordinates of the objects observed and the robot positions are stored in separate matrices which are later stored as files, so that it becomes possible to recreate the map after the robot has finished collecting the readings from the environment. A single reading, consisting of X and Y coordinates values of the room takes roughly 16 bytes of file space. So, for 120 readings taken at 100 posi-

tions, for a total of 12000 records, the file size will be approximately 188 kBytes.

4. Kidnapped Robot Problem

The algorithm implemented and discussed here is used by the TriBot to find its approximate location in an existing map by taking the readings of its surroundings. During the map construction process, the robot finds its position through the relative displacements from the origin of the map. In a situation where the robot only knows the coordinates of the objects of the map and does not know its position relative to the origin, the robot's position is localized within the environment by taking measurements from its immediate surroundings and by moving it within the area for which the global map is available. Figure 5 depicts these two steps.

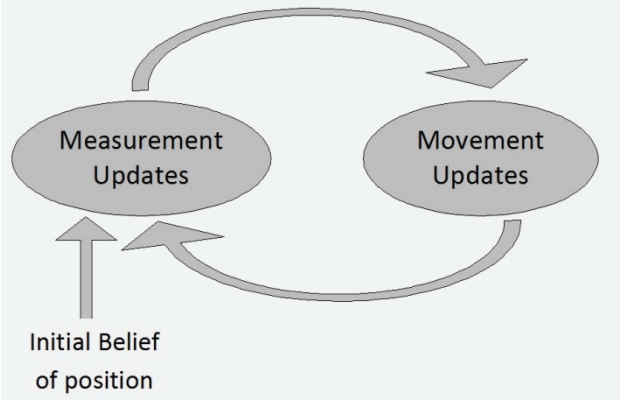

***Fig. 5. Localization process** used by the TriBot*

The position of the particles which represent the guesses of the robot's possible position within the map, are taken as the initial belief for the particles. The particles then update their belief of the robot's position by using the measurements taken by the robot's ultrasonic sensor which leads to survival of only a few of the particles with probable initial beliefs. This selection of the particles with probable beliefs is repeated continuously, by moving the robot and the particles by some distance and taking subsequent measurements, till all the particles are clustered around a single position within the map which can safely predict the robot's approximate location.

4.1. Particle Filters

Particle filters are used in this work as they are easier to program for continuous value problems, as compared to Kalman Filters [10, 11]. Particle filters also work better than Histogram Filters [11] in higher-dimensional spaces. Table 1 compares the Histogram, Kalman, and Particle filters on the basis of two characteristics.

***Table 1.** Comparisons between Histogram, Kalman and Particle Filters*

Filter Name	State-Space	Belief
Histogram	Discrete	Multi-Modal
Kalman	Continuous	Uni-Modal
Particle	Continuous	Multi-Modal

4.2. Localization using Particle Filters

After satisfactory map construction of the room, the robot is kidnapped i.e., removed from the environment and then placed randomly somewhere inside the area represented by the global map. The current position of the robot within the map is unknown and is determined by finding distances to nearby objects with the use of the ultrasonic sensor, and calculating a good posterior distribution of where the robot is based on those readings.

Each particle represents the possible position and orientation of the robot in the map. Set of hundreds of such (N) particles together comprise an approximate representation of the posterior distribution of the robot [11]. In the beginning, all the N particles are uniformly scattered, and the filter allows them to survive in proportion to how consistent these particles are with the sensor readings, such that the particles which are more consistent with the sensor measurements are considered fitter and are more likely to survive. Measurements of the robot's surroundings are recorded and applied to all the particles individually. As a result, closer the particle is to the correct position, more likely the possibility that the measurements belong to that particle, and higher the likelihood of its survival. The particles which are highly consistent with the measurements form clusters around a single position in the map which approximately represents the position of robot as it localizes itself within the map.

Each particle has its own importance weight and survives at random, though the probability of its survival is proportional to its importance weight. Larger the weight of the particle, more likely it is that it represents the robot's position and more likely that it survives. After resampling i.e., randomly drawing N new particles from the old ones in proportion with their importance weights, with replacement, the smaller weighted particles representing less probable robot positions are likely to be discarded.

4.3. Noise

Three types of noise values [11] are considered during the localization of the robot, as follows:

- Forward noise – The error value, in centimeters, that must be taken into account when the robot moves in the forward direction.
- Turn noise – The error value, in degrees, which must be taken into consideration when the robot executes a right or left turn on its current position.
- Sense noise – The error value, in centimeters, that must be taken into account due to the ultrasonic sensor's inability to measure exact distances.

4.4. Particles

To create the particles a class is created in MATLAB, where an object of that class represents a single particle. Each particle created using this class has six attributes, namely – X-coordinate value, Y-coordinate value, Orientation, Forward noise, Turn noise, Sense noise. The position and orientation of a particle are randomly initialized on creation of the particle. The forward, turn and sense noise values for each particle are initially set to 0.5, by default.

4.5. Particle Filter Algorithm

Firstly, 250 particles are created using the class implemented in MATLAB and their noise attribute values are set according to the measured values of the precision for TriBot's wheeled motors and ultrasonic sensor. The values of forward, turn, and sense noise of the TriBot are calculated using the accuracy of the ultrasonic sensor and the motors and are set to 0.05 centimeters, 0.25 degrees and 3 centimeters respectively for the current implementation.

At the current unknown position of the robot a reading is taken by the ultrasonic sensor at a certain angle, between 0 and 45 degrees, with respect to the robot reference frame. Seven more such readings are taken at an interval of 45 degree each, which gives a set of total eight measurements for the robot's current position. For each particle, these angles are used to find the distances between the particle's position and the objects in the map at these angles, with respect to the particle's orientation. The probability that a particle represents the approximate location of the robot is computed by using the equation:

$$\mathbf{p} = \prod_{\propto} \frac{\mathbf{e}^{\frac{(\mu - x)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \qquad (1)$$

Where μ is the distance of the particle from an object, x is the observed distance measurement at that angle with respect to the robot reference frame, σ is the sense noise and α is the set of all angles at which the eight readings were recorded. This probability, p, is eventually treated as the importance weight for that particle.

The particle filter implemented in the current system, is shown in Algorithm 1. RoomX and RoomY are arrays having the values of X and Y coordinates of the room boundary and objects, recorded during map construction. Line 2 creates a set of particles P by creating objects of the class and initializes the respective attribute values for each particle. P is a collection of one-dimensional arrays, viz. P_x, P_y, $P_{orientation}$, P_{sense}, P_{turn} and $P_{forward}$, of size N each, where each array represents an attribute. Array P_x records the X-coordinate value for all the N particles in sequence, P_y stores the Y-coordinate value for all the particles and so on. W is an array of size N that sequentially stores the importance weights of all the particles in P. Line 3 sets the noise values of all the particles.

The constant, *t* is used to define the number of times that the algorithm must be run and a value of 10 is fixed in the implementation. The initially empty array, *select*, is used to save the index of the coordinates of room that satisfy the measurement reading, Z, when applied to a particle. Integer value of the variable *angle* can be initialized manually between 0 and π/4 radians and a value of π/6 radians has been chosen in the implementation.

In line 7, the robot takes a distance reading of its surrounding at the angle specified. The function *polar* takes the arrays of Cartesian coordinates of the room and converts them to Polar coordinates, with respect to the origin of the global map; where *R* represents the array of distances and Θ represents the array of angles of those points, in radians. The *size* function returns the number of rows in an array. The values distX, distY represent the distance between the X and Y coordinates of the particles from the coordinates of the object of the room observed at a certain angle by the robot. These values are used to find the distance of the particle from the coordinates in map, which is then used in line 29 and 30 to find the importance weight of the particle. Line 33 applies the resampling algorithm which randomly draws N new particles from the old ones, with replacement, according to their importance weights. In line 37 implements the movement of the TriBot and all the particles, by turning them by 20 degrees anti-clockwise and moving them forward by a distance of 6 centimeters. A random Gaussian distribution value with zero mean, and the standard deviation set as the forward or turn noise is added when the particles are moved forward, or rotated.

Algorithm 1: Particle Filter Algorithm for Localization.

Input :
$\begin{cases} \boldsymbol{RoomX}, \boldsymbol{RoomY} & \text{: Coordinates of Room} \\ \boldsymbol{N} & \text{: Number of particles} \end{cases}$

1 **begin**
2 Create N uniformly distributed particles P, and initialize attributes P_x, P_y, $P_{orientation}$, P_{sense}, P_{turn}, and $P_{forward}$.
3 Set Noise values of P_{sense}, P_{turn}, $P_{forward}$.
4 **for** $t \leftarrow 1$ **to** 10 **do**
5 *angle* ← π/6 ;
6 **for** $i \leftarrow 1$ **to** 8 **do**
7 Z ← Distance reading with TriBot(*angle*) ;
8 *select* ← [ ] ;
9 (*R*, Θ) ← polar (RoomX, RoomY);
10 **for** $j \leftarrow 1$ **to** N **do**
11 W(*j*) ← 1.0 ;
12 **for** $k \leftarrow 1$ **to** N **do**
13 $dcos \leftarrow R(k) * \cos(\Theta(k)) - P_x(j)$;
14 $dsin \leftarrow R(k) * \sin(\Theta(k)) - P_y(j)$;
15 $d \leftarrow \sqrt{\mathbf{dsin^2 + dcos^2}}$;
16 **if** *dcos* ≅ 0 **do**
17 $r \leftarrow \mathbf{sin^{-1}\,(dsin/d)}$;
18 **else**
19 $r \leftarrow \mathbf{cos^{-1}\,(dcos/d)}$;
20 **end if (line 16)**
21 **if** $\lfloor \mathbf{r * 100} \rfloor = \lfloor (\mathbf{P_{orientation}(j) + angle}) * \mathbf{100} \rfloor$ **do**
22 Add *k* to *select* array.
23 **end if (line 21)**
24 **end for (line 12)**
25 **for** $i \leftarrow 1$ **to** size(*select*) **do**
26 $distX \leftarrow \mathbf{Px(j) - RoomX(select(i))}$;

27 $distY \leftarrow \mathbf{Py(j) - RoomY(select(i))}$;
28 $dist \leftarrow \sqrt{\mathbf{distX^2 + distY^2}}$;
29 $\mathbf{p} \leftarrow \frac{\frac{(\mathbf{dist-Z})^2}{\mathbf{e^{(2*(Psense(j))^2)}}}}{\sqrt{\mathbf{2\pi*(Psense(j))^2}}}$;
30 W(j) ← W(j) * *p* ;
31 **end for (line 25)**
32 **end for (line 10)**
33 P2 ← Resampling Wheel Algorithm (W, N).
34 P ← P2 ;
35 *angle* ← *angle* + π/4 ;
36 **end for (line 6)**
37 Move the robot and particles.
38 **end for (line 4)**
39 **end**

For resampling the particles on the basis of their importance weights, a particle that has higher importance weight, must be sampled or picked more number of times than a particle that has low importance weight. This is achieved through the resampling wheel algorithm [11] described in Algorithm 2. First, an initial integer guess of a particle number is taken from the uniform interval (1, N) at random and labeled as *index*. A value *beta*, initialized as zero, is added to a real value drawn uniformly from the interval (0, 2*maxw) to give a new value of *beta*; where *maxw* is the maximum importance weight in W. If weight of the *index* particle is smaller than the current value of *beta*, then the weight of the particle is subtracted from *beta* and *index* is incremented by one. Otherwise, if the *beta* value is lower than weight of *index* particle, then that particle is chosen to survive.

In the given algorithm, U_1 represents a function that returns a uniformly integer drawn value and U_2 returns a uniform real value from the intervals specified. The *max* function returns the maximum value among all the elements of an array. P2 is, initially, an empty set of arrays and stores the attribute values of the particles selected for survival after resampling.

Algorithm 2. Resampling Wheel Algorithm.

Input :
{ ***W*** **: Array of importance weight of particles**
N **: Number of particles**

Output : { *P2* : Particles after resampling

1 **begin**
2 *index* ← U_1(1, N) ;
3 *beta* ← 0 ;
4 P2 ← [ ] ;
5 *maxw* ← max(W);
6 **for** *i* ← 1 **to** N **do**
7 *beta* ← *beta* + U_2(0, 2**maxw*) ;
8 **while** W[*index*] < *beta* **do**
9 *beta* ← *beta* – W(*index*) ;
10 *index* ← *index* + 1 ;
11 **end while**
12 Add the *index* particle to P2.
13 **end for**
14 **end**

If the *beta* value is small, there is a high possibility of a particle with large importance weight being picked more than once during resampling. After resampling, N new particles are obtained, each having a new position and orientation values that have been derived from old particles with high importance weights.

At the end of the process an approximate location of the robot is found by analyzing the X and Y coordinate of the locations where most of the particles are clustered in the global map.

5. Experimental Results

Figure 6 shows the area under consideration for the purposes of experimentation. The cross mark shows the robot's starting position for map construction. The light-gray colored portion in the figure is beyond the environment under observation.

Fig. 6. Room used for Map Construction and Localization by the TriBot

5.1. Map Construction

Figure 7 shows the two dimensional maps created after taking readings on the first two positions. All the objects recorded by the TriBot are displayed as a point cloud of black dots as the ultrasonic sensor returns a single value of the straight line distance of an object from the ultrasonic sensor. In the figure the positions of the robot where the readings were taken are displayed by magenta colored dots. Figure 7a shows the local map created at the origin position by the TriBot during map construction. Figure 7b shows the existing map after adding the readings taken at the second position, to the map created in Figure 7a.

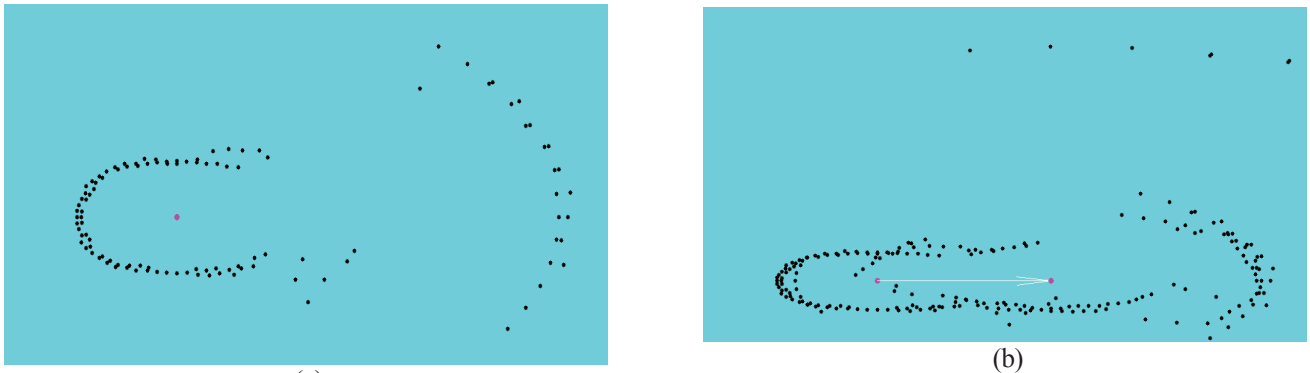

Fig. 7. (a) Map obtained after taking readings at the origin position. (b) Map obtained after taking readings at second position and adding them to the existing map

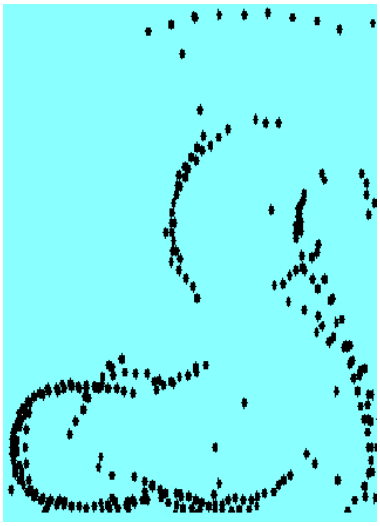

Fig. 8. Map constructed by the TriBot after one scanning of the room where the boundaries of the room can be easily identified

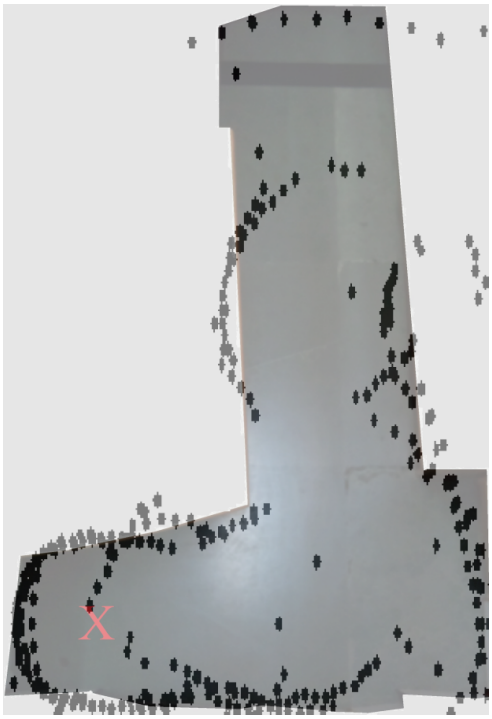

Fig. 9. Image of the room under consideration and the map created by the TriBot superimposed onto it

Figure 8 shows the map created after one scanning of the room. In the map constructed, the complete room is scanned only once and for better results of map construction and localization, it is possible to let the TriBot scan the room continuously till a highly detailed map is constructed. Noisy readings, representing ghost images, can be seen in the map which lie either in the region beyond the room boundaries or within the empty area of the original room shown in Figure 6.

Figure 9 shows the map constructed in Figure 8, superimposed onto the room shown in Figure 6.

5.2. Kidnapped Robot Problem

Figure 10 shows the particles initially distributed, almost uniformly. The green triangles represent the particles created initially. The blue asterisks represent the boundary of the room created during map construction.

Figure 11 shows how the green colored particles have clustered on the right hand side of the map and predict the approximate position of the TriBot.

In later iterations of the algorithm, the particles move within the map as the robot moves in the actual environment. Figure 12 shows how the particles after executing a left turn and moving forwards, and further upwards, in the map as the robot moves.

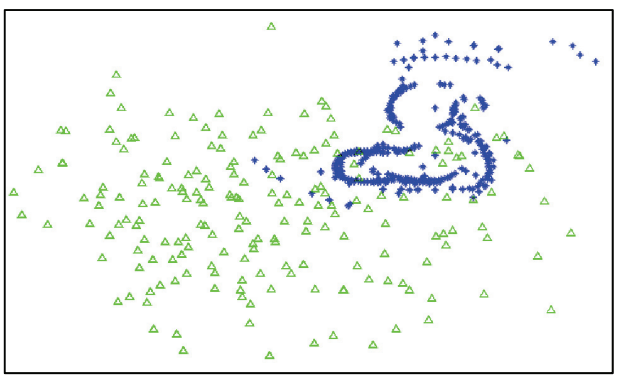

Fig. 10. Particles created for localization of the robot and to solve the kidnapped robot problem

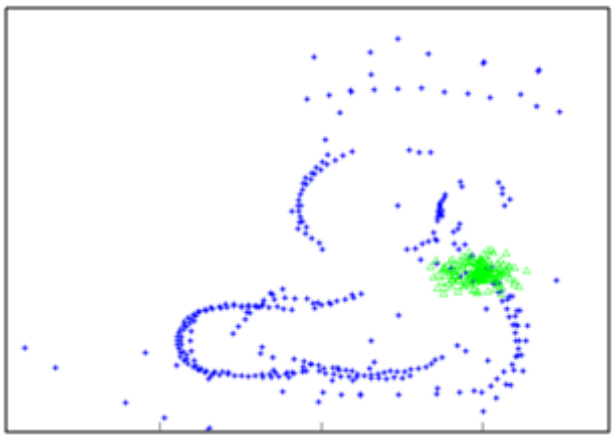

Fig. 11. Particles finally localized to the robot's approximate position within the map

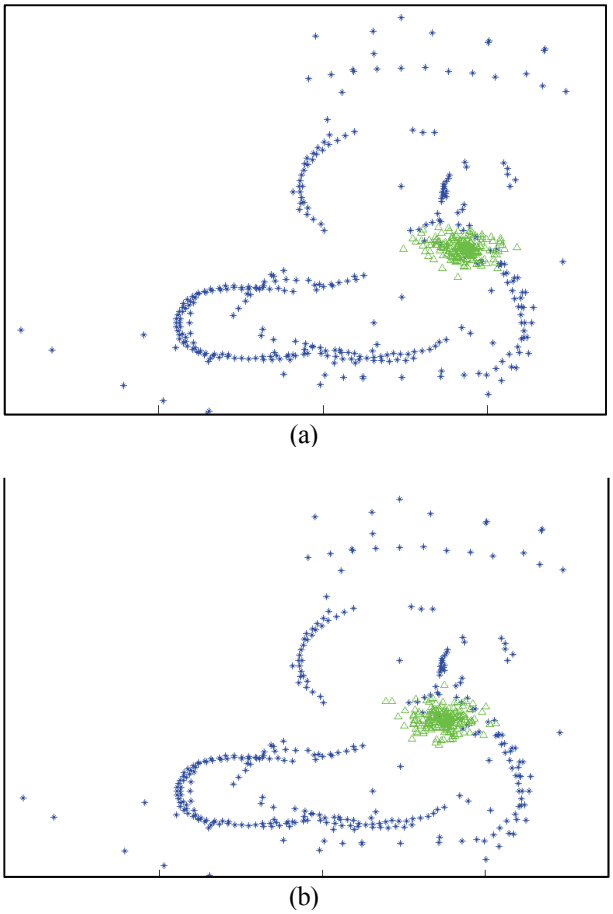

Fig. 12. Particles moving with the robot's actual movement. (a) The cloud of green particles is near the right hand side boundary (b) The particles move upwards and left, further away from the boundary

6. Conclusion

Efforts have been made in this paper to create a map of an environment using Lego Mindstorms NXT TriBot. This work may find applications in creating maps of unknown environments that are inaccessible for humans, by deploying robots into them or in applications of home security systems to detect anomalies when the map changes in the absence of human interference. The MATLAB programming environment with RWTH – Mindstorms NXT Toolbox has been used for the purpose of implementation and testing of the system with the TriBot. The implemented system has been successfully tested with an indoor map creation of a room and for subsequently solving the kidnapped robot problem with particle filter localization of the TriBot within that room. The robot in this work uses a single ultrasonic sensor, making it highly cost effective.

A better representation of the map can be created by making the environment multi-agent with the use of several similar Lego Mindstorms NXT TriBots that are controlled via MATLAB and by overlapping the individual maps created by them all. It is also possible to reduce the number of particles used in the implementation, and achieve almost similar results as the ones shown in this paper.

AUTHORS

Jasmeet Singh*, Punam Bedi – Department of Computer Science, Faculty of Mathematical Sciences, New Academic Block, University of Delhi, Delhi- 110007, India, jasmeetsingh89@ymail.com, pbedi@cs.du.ac.in

*Corresponding author

References

[1] Burguera A., González Y.' Oliver G., "Mobile Robot Localization Using Particle Filters and Sonar Sensors", *Advances in Sonar Technology*, In-Tech: Vienna, Austria, 2009, Chapter 10, pp. 213–232.

[2] Adiprawita W., Ahmad A. S., Sembiring J., Trilaksono, B. R., "New Resampling Algorithm for Particle Filter Localization for Mobile Robot with 3 Ultrasonic Sonar Sensors", In: *Proceedings of International Conference on Electrical Engineering and Informatics*, Bandung, Indonesia, July 17–19, 2011, pp. 1431–1436.

[3] Burguera A., González Y., Oliver G., "Sonar Sensor Models and Their Application to Mobile Robot Localization", *Sensors*, vol. 9, 2009, pp. 10217–10243.

[4] Thrun S., "Particle Filters in Robotics", In: *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, Edmonton, Alberta, Canada, August 1–4, 2002, pp. 511–518.

[5] Howell J., Donald B.R., "Practical Mobile Robot Self-Localization", In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, USA, April 24–28, 2000, vol. 4, pp. 3485–3492.

[6] Yamauchi B., Schultz A., Adams W., "Mobile Robot Exploration and Map-Building with Continuous Localization", In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Leuven, Belgium, May 16–20, 1998, vol. 4, pp. 3715–3720.

[7] Varveropoulos V., "Robot Localization and Map Construction Using Sonar Data", The Rossum Project: 2000. Available online: http://www.rossum.sourceforge.net/papers/Localization (accessed on 17 January 2012).

[8] Howard A., "Multi-robot Simultaneous Localization and Mapping using Particle Filters", *Int. J. Robot. Res.*, vol. 25, 2006, pp. 1243–1256.

[9] RWTH – Mindstorms NXT Toolbox, RWTH Aachen University, 2010. Available online: http://www.mindstorms.rwth-aachen.de/trac/ wiki (accessed on 18 August 2011).

[10] Fox D., Burgardy W., Dellaerta F., Thrun S., "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots", In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, FL, USA, July 18–22, 1999, pp. 343–349.

[11] Artificial Intelligence (CS373) Programming a Robotic Car, Udacity, 2012. Available online: http://www.udacity.com/overview/Course/cs373 (accessed on 26 February 2012).