ESE 650 - Learning in Robotics
PROJECT 4
Simultaneous Localisation and Mapping

Varsha Shankar
vasha@seas.upenn.edu

Introduction

This report outlines my approach to 2D simultaneous localisation and mapping and is structured as follows :
Section I : Brief summary of methods
Section II: Occupancy Grid
Section III: Motion Model
Section IV: Sensor Model
Section V: Particle filter, Importance weights and Resampling
Section VI: Ground plane mapping
Section VII: Results
Section VIII: References

Section I - Brief Summary

The aim of this project is to map the structure of an indoor environment using odometric data, IMU orientation and LIDAR ranges. Further, we are to map the colours of the ground plane onto this map. The map is in the form of a 2D occupancy grid of the walls and other obstacles.
We begin by constructing a motion model of a differential drive robot. This gives us an initial estimate of the path taken by the robot. However, we need to correct the errors introduced due to slippage of the wheels and errors in the IMU orientations. The robot's pose at a given timestamp is estimated using the aforementioned motion model, incorporating IMU orientation and encoder data. The robot's lidar scans at that time stamp are transformed into global coordinates assuming this pose and used to build a map of the world. This map is correlated to the map at a previous timestamp to gauge the quality of the current pose.
Beginning with multiple arbitrary robot poses, it is then possible, using the above method to build up a map of the robot's environment and track its motion along this path using the best local map built at any given timestamp based on its correlation to the map at a previous time stamp. This is the essence of the particle filter method outlined in the later sections.

Section II - Occupancy Grid

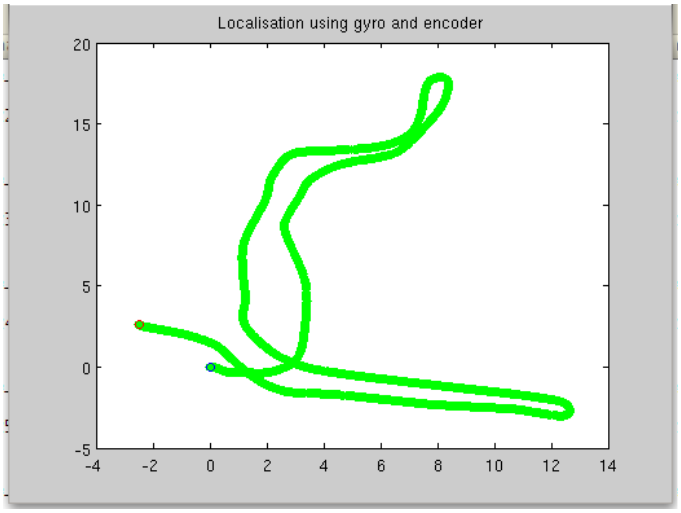
An occupancy grid is the structure used to maintain the map of the world. It consists of cells with a resolution of 10 cm (as per my implementation). Each cell is given a

value according to our belief of it being occupied or free. Given a robot pose and the lidar scans at that time, it is possible to designate cells as being occupied or free. Each time, an obstacle is encountered in a particular cell, its value is incremented by 1.
To compensate for times at which the robot stays still, we clamp the cell values at a maximum (127 here, since we assume the map to be uint8).
When the robot goes through the same area of the map (for example, going up and then down the same corridor), the drift in motion model and errors in the lidar scans, sometimes cause the built up map to be shadowy (multiple walls for the same corridor).
To deal with this, we use a process of decrementing the previous grid map before adding it on the present one.

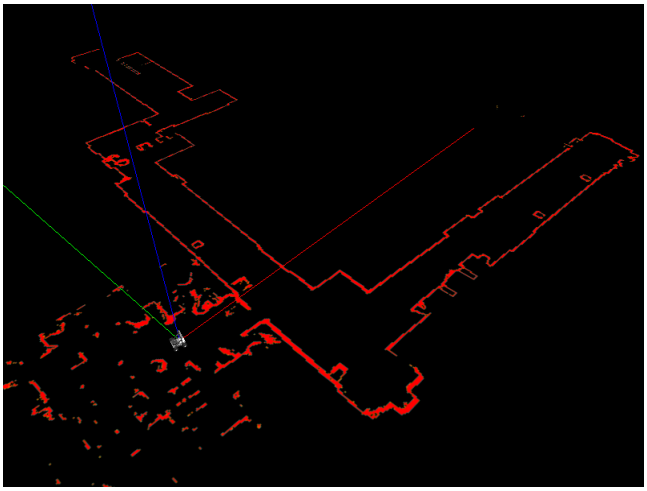
Section III - Motion Model

The motion model of the robot follows the forward kinematic model for a differential drive. It is approximated by assuming that turns are made around a circular path, allowing us to find the radius and angular displacement between two timestamps. The next pose of the robot is then calculated as the next point on this circular arc [1].

Generated Motion Model



Reference Map



Section IV - Map Correlation and Sensor Model

The inputs to the sensor model include the known robot pose and the lidar scans. The output of the sensor model is the probability of the measurement (lidar) given the current pose. It is assumed for this step that the map is known.
The lidar scans are first transformed to global coordinates using the given robot pose. A map correlation is calculated of the current known map and the present lidar scans (already transformed). This correlation serves as a weight on the probability of this current map being adopted for subsequent iterations.

Section V: Particle filter, Importance weights and Resampling

Using a particle filter to maintain multiple possible robot poses at any one given point in time helped to smoothen out the uncertainty caused due to the sole use of the motion model. In my setup, I used 50 particles, but didn't notice too much of a difference in my results as i varied the number of particles between 30-100.

My noise model for encoder values i.e. for x/y locations of the robot pose is given by :

encoder_noise = (enc_prop*theta + enc_constt)*randn(1,1);

where enc_prop = standard deviation of all the delta theta values i.e. change in orientation values for all states
and enc_constt = a constant value that is added to the noise (set to 0.005)

My noise model for orientation of the robot is given by :

gyro_noise = (gyro_prop*theta + gyro_constt)*randn(1,1);

where gyro_prop = standard deviation of all the delta theta values i.e. change in orientation values for all states
and gyro_constt = a constant value that is added to the noise (set to pi/150)

The noise model required a substantial amount of tweaking in order to get a good map.

Since the noise models are randomised, at each iteration, every particle is placed in a new position and orientation, slightly perturbed from where the original motion model state indicates it to be. For each particle, we then build a map and check its correlation against the map at the previous time step. The correlation value is multiplied by its old weight in order to update the particle.
The particle with the highest weight has its generated map then selected as the one that we will use on the next iteration to correlate the next timestamp's map against.

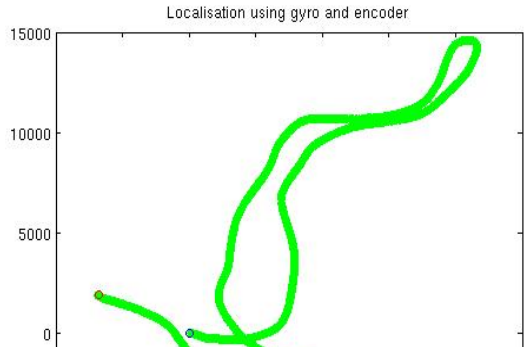
There are time instances at which the number of particles falls below an allowable threshold (again, a parameter that can be played with). At such instances, we resample the particles based on their current weights. The newly resampled particles are initialised to uniform weights again.

Section VI: Ground plane mapping

By this section of the project, we had a SLAM generated occupancy grid map of the robot's path along with its states that define the said path. Using these two values, the objective was to integrate Kinect depth and RGB values onto the projected ground plane on the 2D occupancy grid map.
The depth map obtained from the kinect sensor was interpreted as a disparity image and used to determine (x,y,z) values of the pixels in 3D space. This conversion is done on the basis of kinect parameters such as focal length.
Given a point cloud of 3D points, I wrote a RANSAC algorithm to implement plane fitting wherein the plane that accommodated the largest number of points from the point cloud was selected. The parameters for RANSAC were as follows : a distance threshold which determines how close a point needs to be in order to be considered a part of the plane and percentage threshold, which is the minimum number of points that need to be considered as part of the plane, expressed in terms of a percentage of the total number of points in the point cloud.
I used a distance threshold of 0.1meters and a percentage threshold of 50%.
As a naive way to reduce the number of points in the cloud going into the RANSAC algorithm, I threshold out those 3D points whose z-value is greater than -0.3meters.
RANSAC returns those set of points from the point cloud that would best fit a plane given the parameters I chose. In this process, we assume that the largest plane so found will be the ground plane. This assumption might not hold up in cases where the horizon is very low and the amount of ground visible is significantly small.
The ground plane points so obtained are remapped into the global coordinate frame of the map and its corresponding rgb values are pasted down as a ground plane in the corridor.

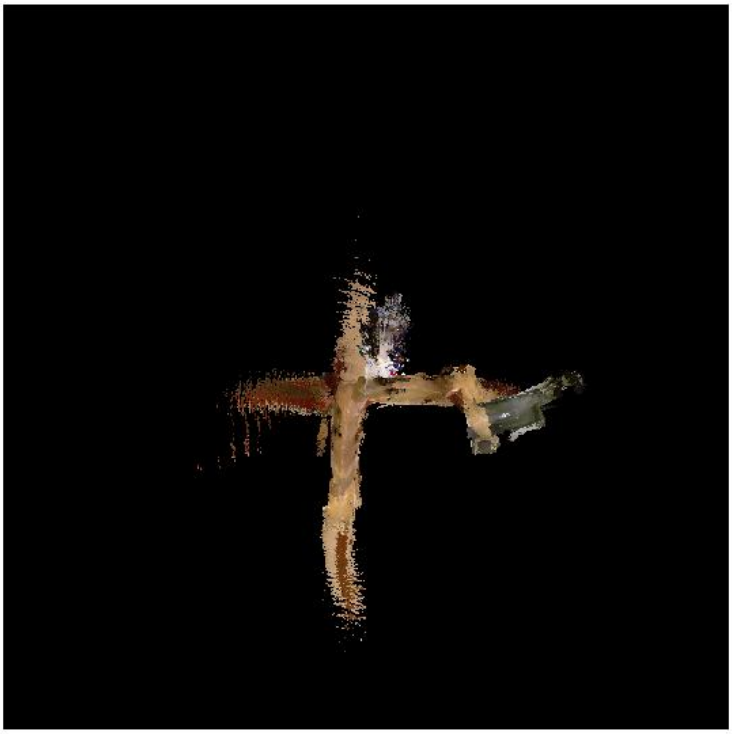
Section VII - Results

DataSet 20
Motion Model



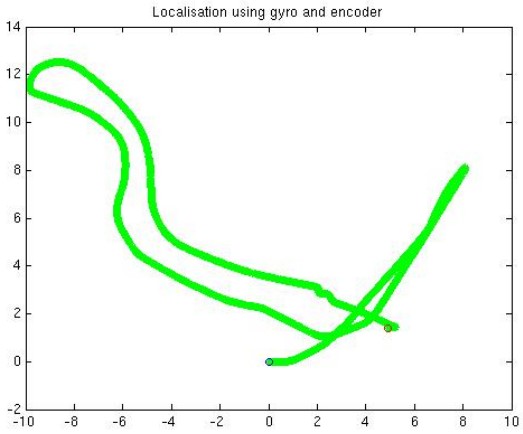
Occupancy Grid

RGBD Mapping

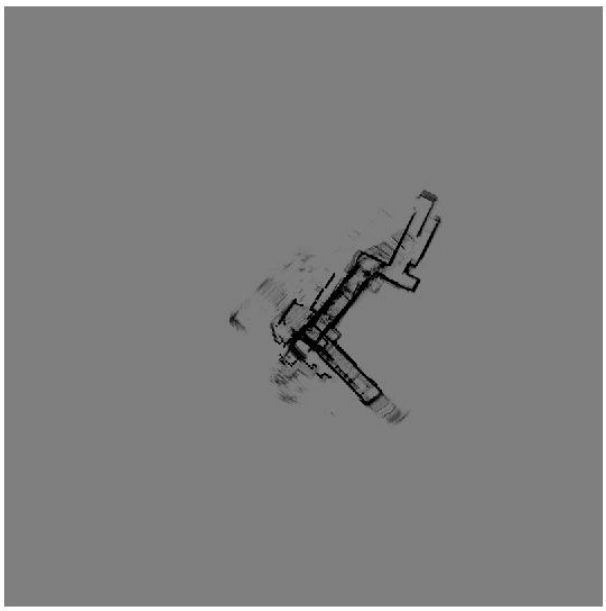


DataSet 22

Motion Model



Occupancy Grid



RGBD Mapping

Section VI - References

[1] www.cs.columbia.edu/~allen/F11/NOTES/icckinematics.pdf
[2] web.mit.edu/16.412j/www/html/.../Eliazar%2BParr-ijcai-03.pdf
[3] cres.usc.edu/pubdb_html/files_upload/514.pdf

[4] www.cs.utexas.edu/~kuipers/slides/L13-occupancy-grids.pdf