Robot Localization and Navigation

Ariel Feliz, Vijayrahul Raja, Xibeizi Ma, Xiang Cao

CPE 593 Applied Data Structure and Algorithms

May 10th, 2016

"I pledge my honor that I have abided by the Stevens Honor System"

- *Ariel Feliz* - *Vijayrahul Raja* -*Xibeizi Ma* - *Xiang Cao*

Contents

Abstract

This project is to help the robot navigate to a particular location and also identify its own position as time changes.This project is based on analyzing random errors and implementing concepts of Kalman filtering and particle filtering for best approximations.

The first part is to implement localization. It starts with Kalman Filter to track the robot's trace and observer the filtering result. We use constant velocity model (CV) and simplified the state transition matrix to one dimension. For the state noise and measurement noise, their means are zero, and the variance are set to be constant. Then use Particle Filter to track the trace of robot. Instead of using Monte Carlo, we used Parens Window in the PF algorithm. Compare the filtering result of KF and PF.

The second part is to implement navigation. We created many random shapes to simulate the obstacles in the real world. We developed our own algorithm to make robot navigate from the departure location to destination with random obstacles.

The processing software is very suitable for implementing several sections (according to several different operations of the robot) of the code and checking the simulated output in realtime. The code will be written completely in Java language.

Mission(objective)

The objective of this robot localization project is to have a robot find its location, utilizing beacons distance information with set locations to triangulate the robot's position. The ultimate mission of this project is to make our robot navigate to a specific location encountering several random obstacles and also identify/ know its own position as time changes. This project is based on analyzing random errors and implementing algorithm concepts of both Kalman filtering and particle filtering for best approximations. Finally, a simulation was projected using processing GUI to track and show the real location and also errors will be simulated in navigational sensor data.

Several key mathematical concepts were implemented like the Gaussian Kernel, Parzen-Window, Density Estimation. Furthermore the project consisted of prenty of data structure algorithm learned throughout the semester; such as linked list, queues, stacks, as well as mapping in form of lists. The challenging portion of this project were producing the localization techniques, implementing the parzens window from a mathematical model, generating random shapes, and the multilayerd mapping structure, and detecting direct line of sight and using that information to navigate. For example, a technique that works well for a

certain robot in one environment may not be suitable for another robot in the same environment. So all the localization techniques, generally provide two information which are the current location of the robot in some environment and the current orientation of the robot in the same environment. If the robot is uncertain, that information needs to be combined in an optimal way. Two most common filters which were effective for our robot localization were the Kalman and Particle Filters.

Methods

Overview

Software

We used Processing programming language which is purely based on Java programming and also contains classes used for creating the graphical user interfaces and for painting, images, sound and other advanced features.

Algorithm

Several complex mathematical concepts were implemented for our robot to Kalman filter, Particle filter, Gaussian kernel, Parzen Window Density Estimation, Uniform Random distribution

Data Structure

Stack, Queue, List, Matrix, Recursion, Mapping

Kalman Filter(xibeizi ma)

Introduction

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.

It uses a system's dynamics model , known control inputs to that system, and multiple sequential measurements (such as from sensors) to form an estimate of the system's varying quantities (its state) that is better than the estimate obtained by using any one measurement alone. As such, it is a common sensor fusion and data fusion algorithm.

There are five core equations we used based on Kalman filter as below, and we separate them into two steps

Time update equations

1. Project the state ahead

$$\hat{X}_{k,k-1} = \Phi_{k,k-1} X_{k-1}$$

2. Project the error covariance ahead

$$P_{k,k-1} = \Phi_{k,k-1} P_{k-1} \Phi^{T}_{k,k-1} + \Gamma_{k,k-1} Q_{k-1} \Gamma^{T}_{k,k-1}$$

Measurement update equations

3. Computer the Kalman gain

$$K_k = P_{k,k-1} H_k^T [H_k P_{k,k-1} H_k{}^T + P_k]^{-1}$$

4. Update estimate with measurement Zk

$$\hat{X}_k = X_{k,k-1} + K_k [Z_k - H_k X_{k,k-1}]$$

5. Update the error covariance

$$P_k = P_{k,k-1} - K_k H_k P_{k,k-1}$$

The time update equations can also be thought of as predictor equations, while the measurement update equations can be thought of as corrector equations. Indeed the final estimation algorithm resembles that of a predictor-corrector algorithm for solving numerical problems as shown below in Fig. 1.

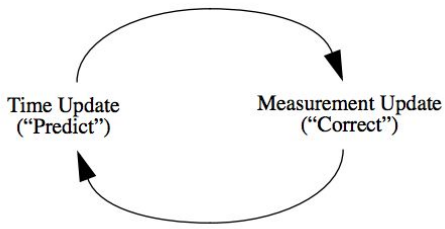


Fig. 1

This is the ongoing discrete Kalman filter cycle. The time update projects the current state estimate ahead in time. The measurement update adjusts the projected estimate by an actual measurement at that time.
We assume that process and environment noise are constant in our project.

Parameters and supporting functions(discuss our code)

The code of Kalman filter in our project as below

```
x1=1.0*x2;
px1 = px2+1.0;
kx = px1/(2.0+px1);
x2=x1+kx*(tempx-x1);
px2=(1-kx)*px1;
y1=1.0*y2;
py1=py2+1.0;
ky=py1/(2.0+py1);
y2=y1+ky*(tempy-y1);
py2=(1-ky)*py1;
```

The matrix in the difference equation relates the state at the previous time step to the state at the current step , in the absence of either a driving function or process noise. Note that in practice might change with each time step, but here we assume it is constant.

Researched weaknesses and strengths

The main advantage of the Kalman filter is its ability to provide quality of the variances, and with relatively low complexity. However, its disadvantage is that it provides the accurate results only for Gaussian and linear models. For Gaussian models with limited nonlinearity, extended Kalman filter is appropriate. For non-Gaussian and nonlinear models, particle filtering is the most appropriate approach, because it is able to provide arbitrarily posterior probability distribution.

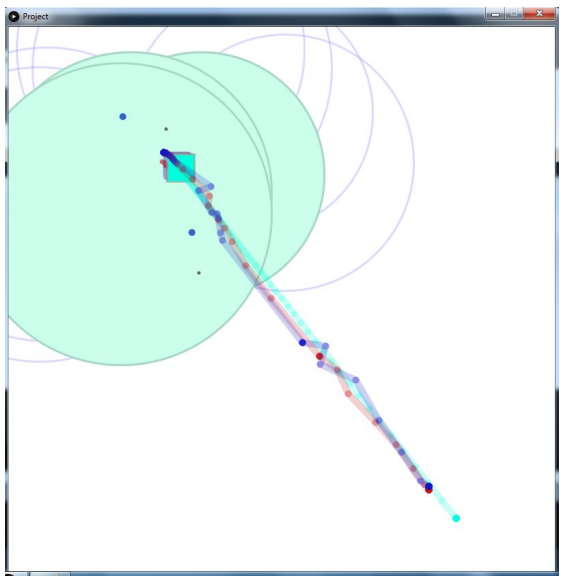
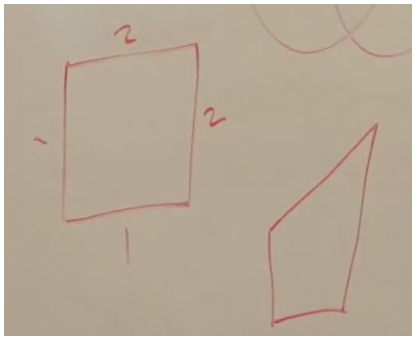


Fig. 2



Gain determines the size of all of our obstacles

Fig. 3

A function was designed namely the line by line function in which the procedures followed were to compare all the vertices on the shape with the beginning point and check if there are any intercepts between those two and the location for the next point is random(The algorithm is discussed below in the code section). In this project, the obstacles could have been generated randomly but our team decided to assign specific number (around 30 shapes) of obstacles for our robot to encounter and take necessary steps.

The very challenging aspect of this obstacle class is what happens when the robot encounters the obstacle? Initially, it was displaying like the robot was moving through the obstacle and not avoiding it ,but after several errors, discrepancies and analysis, the robot was successfully identifying its way past the generated obstacles.Moreover, if the obstacles are very small and aligned next to each other, it would be impossible for the root to go pass through, so the spaces between the obstacles were arranged so that the robot will be able to perform its programmed tasks successfully without thinking about minor issues.

Particle Filter

Introduction

This type of filter is an essential tool for tracking and modeling a dynamic system like in this robot navigation project. For example, it could be used to compare and analyze our expected outcome of how the robot navigation changes in time from the given inputs and the expected state the robot at various time intervals. Particle Filtering is also closely related to Kalman filtering but very efficient in solving and bigger and challenging problems. This particle filter is an useful tool for variety of situations especially in tracking, image processing, smart environments and so forth. In each step in particle filter, we obtain estimate distributions following the equation

Also, only the most recent particles are sampled at time n.

$$x_k = f_k(x_k-1,\ v_k-1)$$

here xk is the vector representing the system's state at an arbitrary time k, vk−1 is the state noise vector, fk is a possibly nonlinear and time-dependent function describing the evolution of the state vector.

$$z_k = h_k(x_k, n_k)$$

where h_k is a possibly time-dependent and non-linear function describing the measurement process and n_k is the measurement noise vector.

Parzen-Window Density Estimation

This Density Estimation is an data interpolation techniques where if an instance of the random sample x is given,Parzen-windowing estimates the PDF(Probability Density Function) from where the x originated or derived. The continuous probability function follows these properties

$P(a < x < b) = \int_a^b p(x)\,dx$ which is non-negative for all real x.

Another way of explaining is that if we want to measure the value of the PDF of a sample x,Then, we could place a window function at x and determine how many observations fall within our window or identify the number of contributions to this window by each different observations x_i. The sum of all of these x_i (observations) contributions will be added to get the PDF P(x) value of sample x.

To have a visual analysis, let's say we have a region R with is a d-dimensional hypercube h, the volume will be h^d. Then to estimate the density at point x, we center R at count the samples in R and input the values in this formula: $p(x) = \frac{k/n}{V}$ where x is inside some region R, k is the number of samples inside R, n is total number of samples inside R and V is the total volume. In this parzens windows, following this equation, (EQUATION) if xi is inside the hypercube, it will be 1 and 0 otherwise.

With infinite number of samples, and appropriate conditions, it can be shown that $p^n{}_\varphi(x)$ is p(x).

$$p(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \frac{K(x-x^i)}{h}$$

In this equation,K is the kernel and d is the dimensional space(3D in this case) h or h_n the window width or bandwidth parameter that corresponds to the kernel's width. The bandwidth is basically chosen based on the number of available observations n.

Convolution

Based on the convolution kernel probability density estimation, particle filter for nonlinear discrete time process is proposed. The best use of this method is when the current particle filters, does not know about the observation variables or the observation noise is nothing or very small, they will be free of limitations. A variant of this particle filter is the convolution kernel approximation technique. This convolution kernel approximation technique is used because other approach

like adding a dynamic noise term to parameter components of the filters like kalman will degrade the filter's own estimation performance.

Kernel function

A kernel function is and there are many different usages like kernel estimation,kernel smoothing and so on. According to statistics, this function commonly refers to the kernel density estimation. The KDE is one of the ways to identify the probability density function of a random variable. In this particle filter, kernels are invoked to form a continuous estimate of the further back in position density function. Particles are allocated based on the information estimated from the kernel density estimate.

Comparing this function with the Parzens windows implemented. For example, if there is a box with sides of length h centered at x, then we define a kernel function to find the number of samples that fall within these boundaries. The total number of points inside the cube will be then:

$$\sum_{n=1}^{N} \frac{K(x-x^n)}{h}$$

Substituting this equation back into the density estimate equation will provide the non-parametric kernel density estimation.

Researched weaknesses and strengths

This particle filter has several strengths and few weakness which makes it one of the most efficient tracking technique. The particle filter has relatively a very high accuracy of identifying the location, it has been tested on a wide range of data sets and it also shows the good description of the embedded methods. The only weaknesses are that there are no analysis of computational complexities of this filter. Computational complexities are that the amount of computing resources needed for a particular type of tasks. Moreover, there seems to be no discussion or theoretical analysis of parameter determination in the model. The parameter estimation(also sample statistic) is the process of using sample data to collect/estimate the values(parameters) of the distribution. The third weaknesses is that it has a very slow frame rate about (0.4- 2fps). This means that the object displayed moves in a continuous and between different frames but in a very slow frame rate scenario. Finally, there are no statistical tests are performed to compare with other types of methods.

Navigation

Introduction

Navigation consisted of a robots ability to navigate in a predefined and set map. However the maps obstacles will initially be generated at random location and random shapes. The robot will then go on to plan and execute a path trajectory that will get robot to set user location.

Mapping:

Generating the map and the information gained consisted of several parts. Beginning with generating obstacles at random shapes and places while avoiding any overlaps. A combination of graph theory along with intersects detection techniques were also utilized in order to generate graph that will have the interconnections required for the robot to navigate around those obstacles, as well as avoiding any concaving vertices.

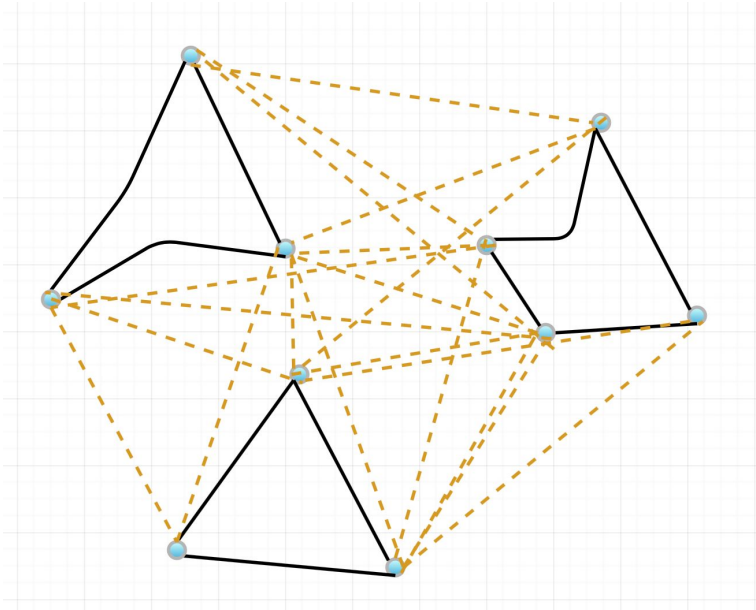


Fig. 4

The graph data structure consisted of two layers. One layer connecting all their vertices that make up an object, and another layer for the connections to other vertices that make up other objects. The mapping was connected in list form as opposed to matrix since we only need to set it up once.

Results

Kalman Filter

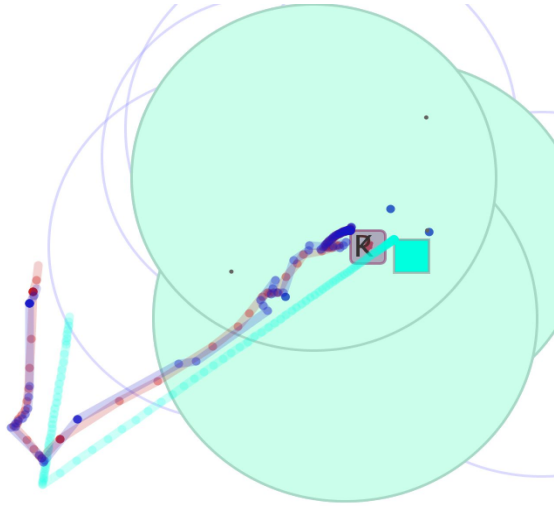


Fig. 5

Particle Filter

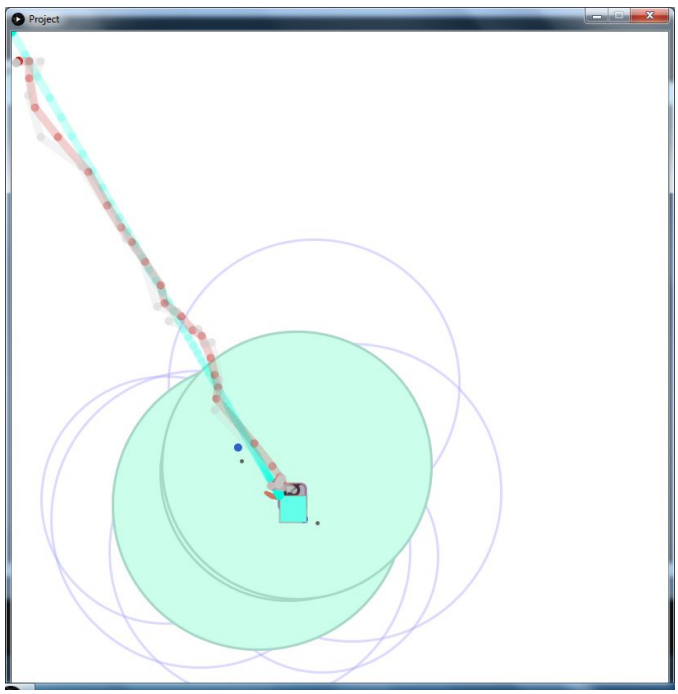


Fig. 6

Navigation

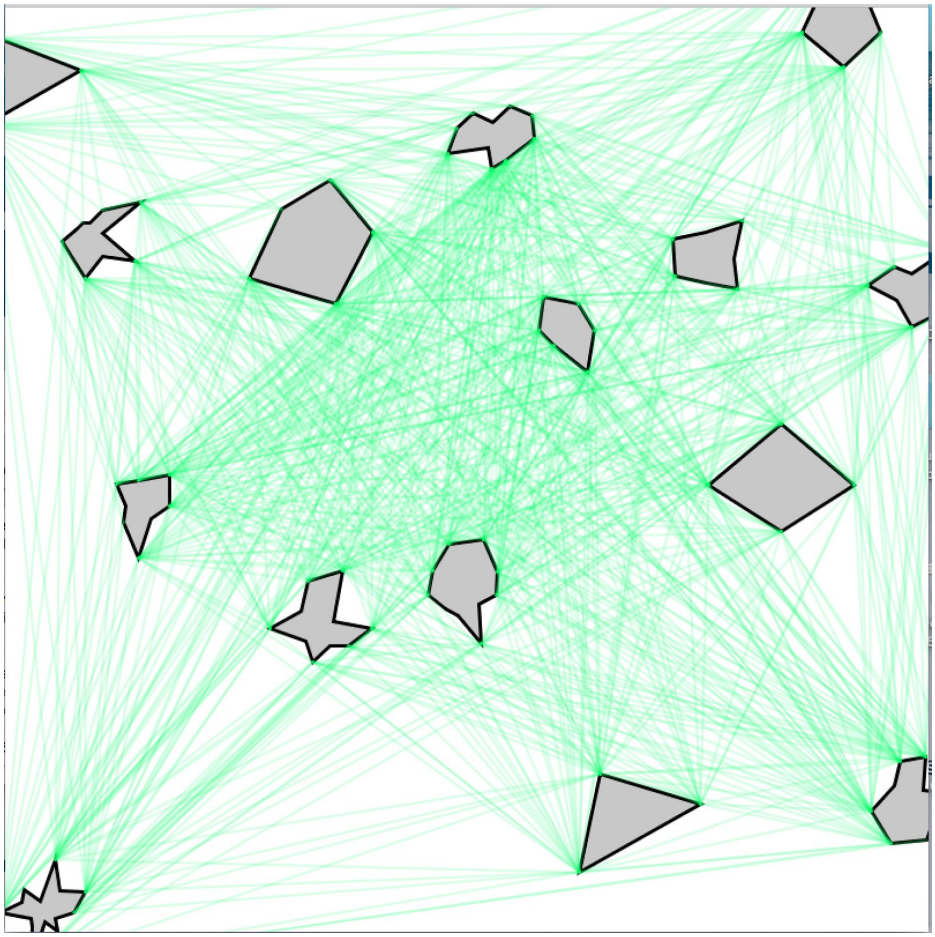


Fig. 7

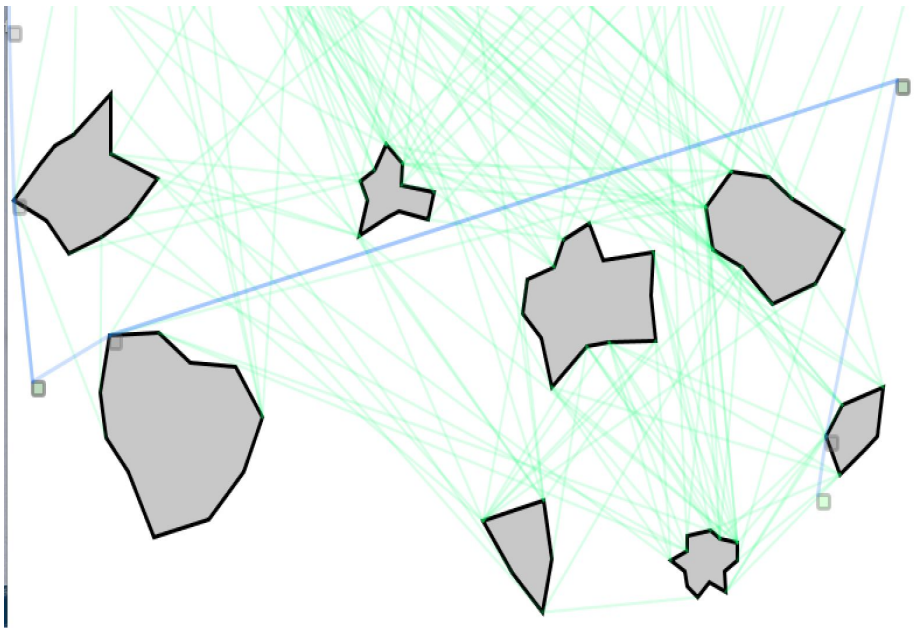


Fig. 8

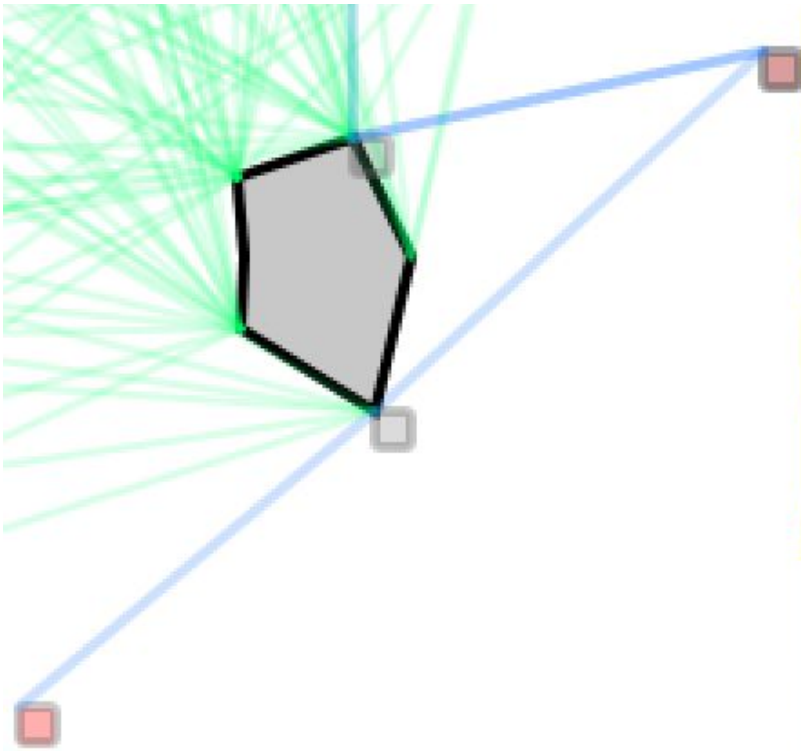


Fig. 9

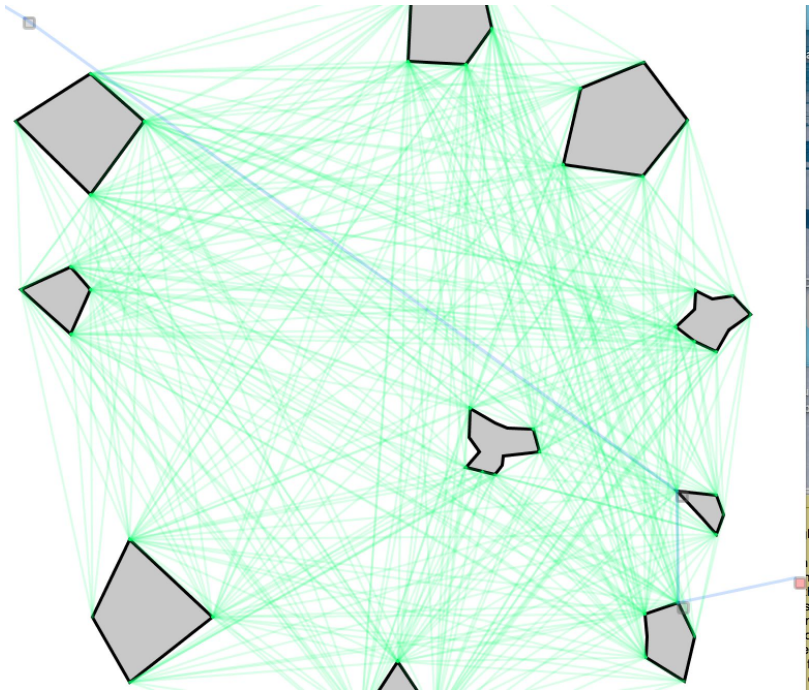


Fig. 10

Analysis

So many different classes were implemented for the expected output of our project. The beacon,bot, EstBot, Intersects,Particle,Project and the TrackBot class. Several other classes have also also being implemented for additional functionality of our robot like separate class for random objects creation , class for robot to identify obstacles, and find best alternatives to reach the goal and so forth. Moreover, all of these different classes were called in Project.pde as an object reference and to show the exact output of the robot in Processing GUI.

In the beginning of this project, one of the first classes created were the Beacon. In simple terms, the Beacon two small circles(shown in the GUI below) that are going to help for the robot localization. The 6 different spots in the beacon will be triangulated. The function of the Beacon is actually used by the Bot class. The Bot could identify or know where the Beacon's are actually located and is doing the necessary calculations accordingly.

Inside the triangulation method(shown below) we are able to triangulate two or three Beacons. There is a Beacon that takes only 2 parameters and a Beacon that takes 3 parameters.

The function " triangulates" when the two beacons are used and it also sets the particle to show on the GUI output. In every single step, it collects 100 particles. This step will be very useful for the particle filter. In Kalman Filter, what the team actually performed for took the average of all the 100 particles to one particle (one of the Kalman entries instead of 2) in order to keep it synchronized and make it time efficient. So the next time around the Kalman captures the second particle and vice versa.

Particle Filter

In the particle filter, with the 100 particles, we were able to create the distribution and then used the Gaussian Kernel functions (for the convolution that it does) so that it is nonparametric (it can be as many as the sigma is set to). This is the process happening inside the Poisson's window. One benefit of the non parametric is that the expected value does not necessarily depend on only the mean and sigma because some of the values may be skewed a little on the sides. The Gaussian kernel fills in the missing values in the particles. The number of particles to be taken per second could be set in the program, but the calculations will be made only after 100 samples. These procedures are very helpful for the robot to estimate with the most accuracy. Noise was added to the number of samples every step of the way.

Kalman Filter

The Fig.2 is the simulation results of our project. The purple particles The cambridge blue box represents the true position of the robot. The dark blue box represents the position calculated by Kalman filter. The blue vertices represent the sensor reading of the cursor's position. If there is no environment noise, these vertices reflect the true position of the cursor. The red path represents the Kalman Filter estimate of the true position. When there is a lot of environment noise, the Kalman Filter estimate is less accurate than a direct reading.

x2=x1+kx*(tempx-x1)
where the new location x2 depends on the previous location x1, the constant speed per time step s. When sighting beacons in our optoelectronic tracker ceiling panels, the noise in measurements of nearby beacons will be bigger than that in far-away beacons.

The main advantage of the Kalman filter is its ability to provide quality of the variances, and with relatively low complexity. However, its disadvantage is that it provides the accurate results only for Gaussian and linear models. For Gaussian models with limited nonlinearity, extended Kalman filter is appropriate. For non-Gaussian and nonlinear models, particle filtering is the most appropriate approach, because it is able to provide arbitrarily posterior probability distribution.
The accuracy of Kalman filter is different with error covariance matrix.

In our processing code, there is a method implemented called K nearest Beacons. It is very important for the functioning of our robot. Our triangulation method doesn't take more than 3 beacons currently. The furthest beacons have been removed and the closest beacon's to the robot will be the most accurate. Following the proportionality, as the accuracy of the beacons decreases when you move further , it will be hard to identify the location and there will be more noise generated.

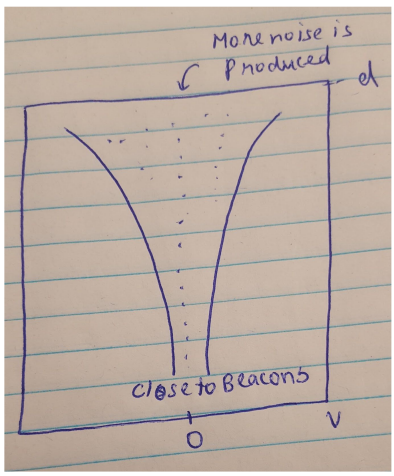


Fig. 11

Separate classes were created for each bot, The 'K' refers to the Kalman Bot and the 'P' refers to the Particle Bot.

In the Intersects class, we created an intersect method as shown here

```
Intersects()
 {
  x = 0;
  y = 0;
  xD = 5;
  yD = 5;
  show = false;
```

```
  inner = false;
  R = 50;
  G = 100;
  B = 200;
 }
```

These are just the vertices that we wanted to create where some of them will “show” meaning that inside the Bot function, the Bot has a file, the bot has a function to determine which of the triangulation point will be used and show with display the dots.

The TrackBot Queue class was created, so that the robot path could be recorded. After a while, it will move back to zero because every single time, one of them will be dequeued since it follows the First In First Out algorithm. The noise seen near the bot is due to localization.

Moreover, in order to challenge the unique actions of our robot, we generated Random obstacles at random places so whenever it is moving, it collides with obstacles and follows necessary and cost/time effective localization techniques. These obstacles will have a random map generator so they are not the same every single time. The random is the number of sides of the obstacle and also the number of gain that we are choosing. For example, if we have a square with all sides equal to 90. Assigning the gain values of 1 1 2 2 will change its shape as shown in the picture below
(we start at one point and start building around that point).The side length will eventually modify the shape of the square.

The 6 different spots in the beacon will be triangulated. The function of the Beacon is actually used by the Bot class. The Bot could identify or know where the Beacon’s are actually located and is doing the necessary calculations accordingly.

Inside the triangulation method(shown below) we are able to triangulate two or three Beacons. There is a Beacon that takes only 2 parameters and a Beacon that takes 3 parameters.

Navigation

We were successfully able to generate obstacles on the map without having any of them collide with each other. Furthermore we were able to generate graph connections at all desired vertices in direct line of sight with no intersects; while excluding concaving vertices along an obstacle as can be seen in figure 4. From figure 7 and 6 you can observe one of the weaknesses illustrating how there are times where the optimal path is not chosen due to no 2nd layer connections within the same object in our designed graph connection structure. However figure 5 illustrates optimal path planning.

Conclusion

Overall, this project was a fresh experience to each and every one of us because most of our group members have never worked in complex projects like this robot localization and

facing difficulties and hurdles every step of the way made us to understand so many complex algorithms and routines which we would have not heard of in the first place. The major goal of navigation our robot was achieved successfully with so many trial and errors and unexpected output's which were fixed eventually applying all the Java concepts and couple of Data Structure algorithms as well. Majority of the time was spent on creating the Bot and applying special filters like Kalman and Particle for precise location of our robot.

Constraints and Opportunities

It is difficult for us not only to understand the algorithms of Kalman filter and Particle filter, but also to archive these algorithms based on our own project. The very difficult and challenging section of our code was analyzing these filters and implementing them on Processing using Java. Moreover, Navigation was also a very tough section since it is one of the major goals of our project and our team members had to analyze all possible ways and program it for the robot to understand, avoid obstacles and reach the specified goal. Our team also had to make sure that our algorithms were working accurately for all sorts of random obstacles and not just one particular set in a map.

Navigation

Future work and lessons learned

Our professor Dov Kruger suggested our undergraduate members to develop this project into a senior design we have decided that, if some of us take relevant Artificial Intelligence courses at Stevens and undertake several other projects, then turning this processing GUI robot into full fledged real life machine will become feasible at the end of the senior design project. On the other hand, programming the core knowledge of the robot requires our computer engineers but the electrical and mechanical components of the life size robot should be handled by mechanical and electrical engineers from our team. So we are thinking about forming a team where engineers from multiple fields incorporate their idea, knowledge and technical aspects along with enormous effort to make a successful smart robot made especially by the Stevens students with excellent support from our professor.

While working on this project, several lessons were learned through obstacles and after several errors. For examples, initially when working with the Kalman and Particle Filters, it was understood that there were several difficult mathematical concepts and procedures involved like the Gaussian Kernel ,Parzen-Window estimation, Convolution, uniform distribution and so forth. Incorporating all of these topics into our robot was extremely time consuming, but a good amount of knowledge were gained from working on this very new project for both undergraduate and graduate team members.

Contributions

Ariel Feliz:

Triangulation

Particle filter

Robot navigation

Map generation

Vijayrahul Raja:

Particle filter

Robot Navigation

References

Max:

Kalman filter

Robot Navigation

Xiang Cao:

Kalman filter

Robot Navigation

Coordinator

https://github.com/aafeliz/RobotLocalizationNavigation