Path Planning, Obstacle Avoidance using Stereo Camera, Rapidly-Exploring Random Trees and Convolutional Neural Network

Utkarsh Ghime
Department of Mechanical Engineering
Northeastern University
Boston, Massachusetts, 02115
ghime.u@northeastern.edu

Abstract

Depth Estimation in Computer Vision is commonly done with the help of stereo cameras which uses disparity, camera calibration to triangulate the depth of feature of interest. Some models have been made using a monocular camera and neural network instead of stereo camera setup. In this paper we discuss how we can use computer vision and stereo camera to triangulate the position of object of interest to generate map of the scene. Then we will discuss how we can implement Rapidly Exploring Random trees(RRT) to find a optimal path for our bot. We will also discuss how we can use a trained network to detect road signs and find another optimal path using these constraints

I. Introduction

Consider a situation where a robot is being used for search and rescue, firefighting operation, bomb detonation. Most of the time a human has to control the movement of the bot looking at a small screen without any prior knowledge of the depth of obstacles in front of the bot. Most of the times the task of finding the object of interest through obstacles such as rubbles, building components, etc takes a lot of trial and error and is extremely time consuming. Although some of these systems involves tasks which can not be fully automated, having prior information about optimal path and distance of obstacle is quite handful for control. Rapidly Exploring Random Exploring Random Trees is an optimal algorithm which involves designing motion strategies for obstacle avoidance and finds optimal path once you have mapped the entire course.

II. Related Work

A. *A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms – Daniel Scharstein, Richard Szeliski*

The paper discusses stereo correspondence and their performance to assess different algorithms and decision design in stereo vision algorithms. The paper discusses several methods of stereo vision implementation to evaluate each individual component and their effect on performance. Their work on generalizing interpretation of disparity influenced the triangulation method discussed in this paper. Their implementation of disparity allowed adaptation of search space to geometry of input cameras. The approach discusses stereo camera setup which takes images on linear path with optical axis perpendicular to camera displacement. The (x, y) coordinates of the disparity space are taken to be coincident with the pixel coordinates of a reference image chosen from our input data set. The correspondence between a pixel (x, y) in reference image r and a pixel (x', y') in matching image m is then given by

$$x' = x + sd(x, y),\ y' = y, \qquad (1)$$

The paper used a similar algorithm which uses d(x, y), focal length, angle of view of camera.

B. *Rapidly Exploring Random Trees: Progress and Prospects- Steven M LaValle, James J.Kuffner*

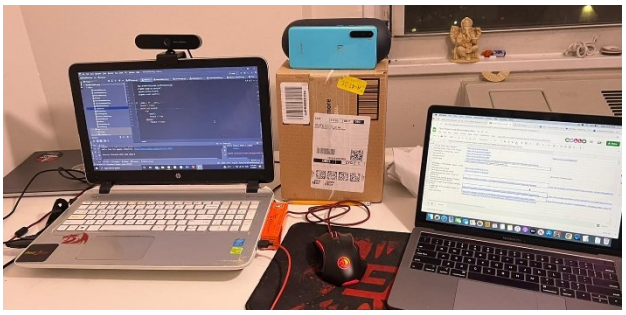
The paper discusses current progress on design and implementation of Rapidly Exploring Random Trees for path planning. They discuss incremental construction of search trees and implementation on RRT on search spaces which contains differential constraints. Further their discussion on properties of RRT to include convergence of RRT to a uniform coverage of non-convex spaces. The paper also used similar classes for state space, collision detection, incremental simulator and a Euclidean distance as distance metric. The project involved Holonomic Path Planning as the constraints were only simulations. The key idea of RRT is to bias the exploration towards unexplored portions of state space which we will further discuss in the paper

C. *High quality Monocular Depth Eestimation via Transfer Learning*

Even though the project involves stereo camera, this paper introduced me to some restrictions of using monocular camera. In general, Monocular Camera have no reference to estimate depth by their own. They depend on datasets and machine learning algorithms to estimate depth. They are quite successful in some cases but we most of the time we have no idea about the scene to be mapped. Most solutions provide blurry approximations of depth. The paper discusses how they can overcome some of the challenges using transfer learning on CNN which works on image encoders for classification. The algorithms mostly require a dataset like KITTI. We can use transfer learning algorithms on stereo cameras instead of more precision. They can also rectify some challenges of stereo vision on scenes with less texture.

III. Problem and Methodology

Most Bots nowadays come with a lot of sensors, the most known sensor for depth estimation is the LiDAR which works on the principle of light ranging. The problem with the sensors is that is extremely expensive and bulky. The cost goes on increasing, the smaller the size gets. The problem with monocular cameras is failure to estimate depth without the help of neural networks and unreliability. We can surely use LiDAR's in expensive products like self-driving car but it makes sense to use stereo camera with computer vision algorithms to estimate depth.



Setup reconstruction for demo

A. The Setup

The first part of project was to establish an obstacle course and stereo camera setup. The stereo camera module was made by using a Logitech webcam and a Sony IMX586 camera module placed roughly 28cm away from each other. The height of both camera sensor from the ground level was approximately equal.

B. Camera Calibration and un-distortion

The first step is to establish a relationship between real world co-ordinates and the 2-D co-ordinates of object position in each frame. The paper used OpenCV's inbuilt function 'findchessboardcorners' and 'caliibrateCamera[1]' to obtain camera matrix and distortion parameters. The camera matrix was useful in finding focal length in x and y axis of each camera.

$$\begin{matrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{matrix}$$

Where f_x and f_y are the focal length in x and y direction.

Note: For stereo cameras to work, you need to use the cameras having same focal length and image parameters like width and height. Here the focal length is different so results are inaccurate

B. Angle of View

The angle of view was around 56 degrees for Sony IMX606 primary camera with undistortion. The field of view in radian was calculated by the following formula:

$$fov_x = 2\tan^{-1}\frac{w}{2f_x}$$

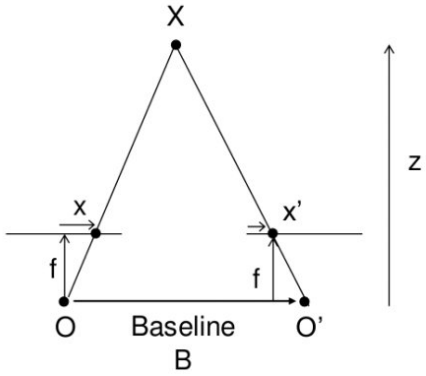
Where w is the frame width

C. Obstacle Detection

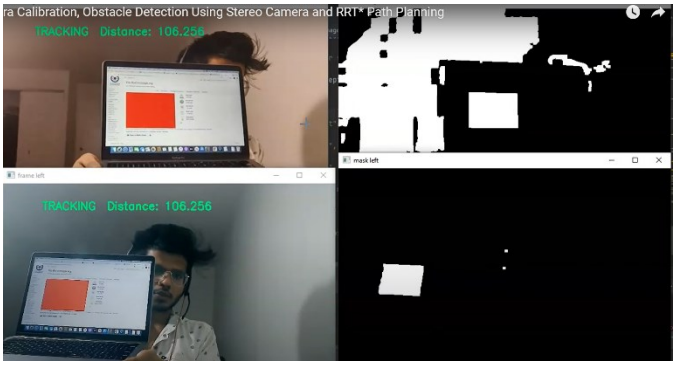
- The Obstacle were red rectangles for simulation, which can easily be replaced by pre-trained models for image classification but for the purpose of this project, red rectangles were used. The colour mask defined HSV colour space for Red. Shape detection was implemented by using contour detection and approxpolydp from OpenCV libraries. The perimeter gave the approximate polygonal and the length of approxpolydp gave the number of sides of the shape. If the length was exactly equal to 4, the object is a rectangle.

D. Depth Estimation:

Extracting the feature points of the center of obstacle. By using the feature points, shape of frame from both cameras, field of view and baseline(distance between both cameras), I approximated depth of the object of interest.
Disparity = frame width (cam A) – frame width (cam B)
Note: x co-ordinate only



Z = (baseline * focal length)/ disparity
Where z is in cm, baseline is in cm, focal length and disparity are in pixel



The Logitech camera has too much noise, but the Sony IMX is accurately detecting the red rectangle

E. RRT Map

For the task of mapping the space, I used pygame module and declared the map dimension, start point and end point at start. We can automate this process by using corner detection algorithms and selecting the last feature as the end point of map and our relative position using depth of the feature. But for this paper, I already declared those points to just show simulation of path planning. The earlier x,y,z co-ordinates of

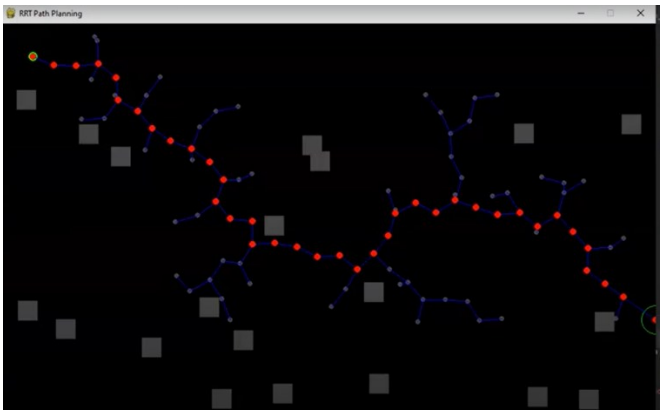
obstacle are used to place them in the RRT map. The obstacles are checked for collision points, if or not they collide. None of the obstacles, I used collided with each other and if they do, they wont be placed in the map. In real world scenario however, you can combine the co-ordinates to create one big obstacle instead of many. I used a flag to describe initial condition of not reaching the goal

F. RRT Path Planning

The path planning algorithm is useful in exploring every point of your map, using any random number generator create random samples(points) across the map. I used the pygame collision function to check if the points are colliding with the obstacle, start or end point, if they do, the points are deleted. Its very easy to create this from scratch as I already have the co-ordinate location, but I decided to use inbuilt function for less computational power. Once the random sample are generated, start from the nearest vertex by selecting a random vertex and calculating the distance between them, I used Euclidean distance. The process is repeated for new random vertex in the direction of earlier vertex and for the same Euclidean distance. I added edges on all the nearest vertex and checked if the edges collide with the obstacle. If they do, the current sample point is omitted, and a new nearest vertex is taken into consideration. This process continues till you reach the end point and the flag updates to True.

To problems with RRT is that it used random number generator, and they are not truly random. We need to add a bias function to randomize the algorithm even further.

This algorithm does not return the most optimal path according to total distance covered. We need to iterate the algorithm and add a cost function to determine the total Euclidean distance covered from start to end. This distance is compared with all other distances of each iteration and the most optimal is selected



G. Traffic Sign Recogntion using Pytorch and Tensorflow

The objective of training the network is to send the obstacle detected as an image in a classifier trained to detect traffic sign. If the RRT algorithm is implemented in self-driving vehicles, detection of traffic signs can be quite handy. The next step was to create a CNN for traffic signs recognition. I used two different datasets which included the German traffic sign dataset[2]. The first model consisted of 953 images with unevenly distributed weight in each label (a speed limit of 40 sign had around 150 images) whereas a speed limit of 15 had 4. The lack of data and uneven weight contributed to extremely less test accuracy. The highest test accuracy I got was around 63%. The CNN consisted of 2 Convolutional layers of 20 filter 5*5 each, a dropout layer of 0.5, 2 max pooling layer of 2*2 kernel, 2 relu and 1 log softmax function.

The second training model was implemented using keras. The model consisted of sequential compilation as follows:

1) *2 convolutional layer of 32 5*5 filter with relu activation*
2) *Max pooling layer 2*2*
3) *Dropout layer of 0.25*
4) *2 convulational layer of 64 3*3 filter with relu activation*
5) *Max pooling layer of 2*2*
6) *Dropout layer of 0.25*
7) *Flatten layer*
8) *Dense layer of 256 with relu activation*
9) *Dropout layer of 0.5*
10) *Dense layer of 43 with softmax*

The network was complex as the dataset consisted of more than 50000 images and 43 labels, The model was trained using 'Adam' optimizer for 16 epochs and 64 training batch size. The model gave a good test accuracy of 93%

IV. Experiments and Results

The depth detection algorithm worked better than I expected, even though the camera modules were different, and I used distorted matrix, I got an error of approximately 15 cm. To verify the experiment, I manually measured the distance from the camera module using a scale as well as the LiDAR sensor in Apple iPhone. At each instance, the error was around 15 cm given that I don't displace the camera geometry. The program will further be tested using actual stereo camera in order to be able to use un-distorted camera matrix and stereo maps. This will enable me to use Kalman filters for even more accuracy. The front view of the obstacle is detected with an error of around 1 cm due to distortion.

No.	Scale Distance(cm)	LiDAR distance(cm)	Measured Distance(cm)
1	93	92.67	107.561
2	75	75.3	90.254
3	NA	153	177.763
4	56	56.7	71.7

The RRT Path Planning Algorithm worked as expected and planned the right path for all instances. I chose not to add the cost function as the obstacle environment was complex enough to not have any straight paths. RRT* uses a lot of computation power compared to RRT.

The CNN experiment did not go as expected, I expected both models to have high accuracy as CNN should be able to classify images given enough filters. Though it does help understand that label weight distribution is quite necessary in such architecture. You need more and more data as well to get high test accuracy.

V. Summary

Stereo Cameras are a great way to implement depth estimation, SLAM, Path planning, etc. A good alternative to LiDAR due to their cost and size. George Hotz has already developed a SLAM using stereo cameras for his company comma.ai. It's a plug and play autonomous vehicle device which cost around 1500$. The experiment in this paper shows that even self-made stereo camera with different focal length produces close results. In read world, the objects are far more complex, but the progress in machine learning is overcoming all those challenges and soon we will have the technology to convert every car to self-driving for as low as 1500$.

Further, I am going to implement machine learning models for complex objects in this project and combine this with stereo SLAM and implement pose estimation and graph optimization.

VI. References

1) *https://benchmark.ini.rub.de/index.html*
2) *http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf*
3) *http://www.cs.cornell.edu/~asaxena/learningdepth/*
4) *https://www.pygame.org/news*
5) *https://learnopencv.com/depth-perception-using-stereo-camera-python-c/*
6) *https://www.tensorflow.org/api_docs/python/tf/keras*
7) *https://www.youtube.com/watch?v=KOSS24P3_fY*
8) *https://vision.middlebury.edu/stereo/taxonomy-IJCV.pdf*