Autonomous Environment Mapping Robot

Yash Sahijwani, Raghav Soni, Vikhyath Venkatraman

Abstract

The project aims to simulate an autonomous robot that is designed to go to open locations that are inaccessible to humans, traverse that location while simultaneously mapping it and looking for a target object. We use pybullet [1], a Real-Time Physics Simulator to make a model arena which serves as an example. The aim is for a 'husky' car to be able to detect another husky. A camera configuration is attached to the front of the searcher robot. The searcher robot can run in a completely autonomous mode, but can also be manually controlled as necessary. We use a DIY Stereo Camera and OpenCV [2] to create a depth map of the environment so that obstacle can be detected and avoided as necessary. We also use the YOLOv3 model [3] of the ImageAI library [4] in order to not only classify objects into obstacles and targets, but also detect their positions in the environment. To label our training data for the images, we use the LabelImg [5] tool.

Problem Addressed

With the goal of tackling the problem of autonomous navigation and searching robots, the project aims to create a small robot with flexible movements and strong computation power to navigate open areas autonomously while mapping them, avoiding obstacles and looking for a pre-defined target which it would have been trained to detect.
Such a robot can serve as a rescue or search robot in hazardous environments, operate at spaces which are hard for humans to navigate or work as a spy bot if its size can be compressed to a decent level.
Overall, the problems of object detection, obstacle avoidance, autonomous navigation and mapping are tackled throughout the project.

Our Approach

Different approaches to tackle each of the mentioned problems were tried to find the best suited one and then efforts were made to combine and execute them with least possible computational power and inexpensive hardware. The robot was successfully simulated and it performed the task in an efficient manner.

The main aim of the project was to simulate the robot, keeping in mind the cost, computational power and feasibility of such a robot in a real-world environment. Every attempt was made to make robot as inexpensive and as feasible as possible.

Key Concepts Used

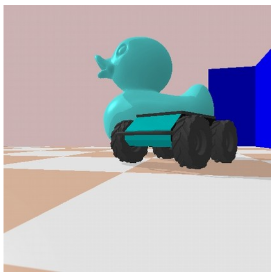
The entire code was written in python. The project was divided into three primary parts.

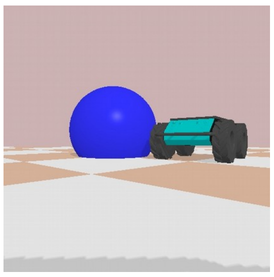
1. Object Detection of Target Object using the YOLO Algorithm

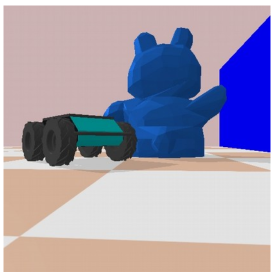
As the aim of our robot was not only to classify objects into targets and obstacles but also detect their position in the environment, we used the **YOU ONLY LOOK ONCE (YOLO)** algorithm of deep learning for object detection.

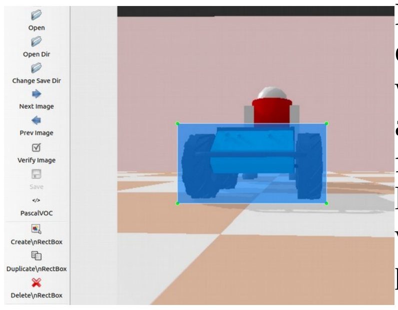
Training Dataset -

In order to acquire the many training images in different environments, orientations and distances that Deep Neural Networks invariably need, we used the **computeViewMatrix** function of pybullet to calculate view matrices for different cases. Using this we were able to generate around 2000+ images of the target object.







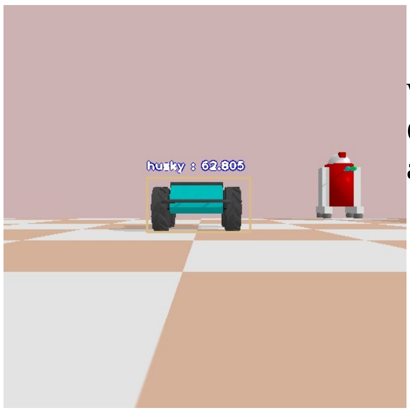


However, to use the YOLO algorithm, bounding boxes around each of the targets (in this case, only the husky) were required. For this, we used the LabelImg tool to (painstakingly) draw bounding boxes around the husky in each image and generate the corresponding xml files containing the annotations of the images. As this was done locally, the source path had to be changes for each xml file, which was done using the ElementTree function [6] of the xml library in python.

Training the Model -

The creation and training of the model was done on Google's colaboratory [7] using the training data uploaded to Google Drive. To create the model, we used the ImageAI library, which provided to us the Convolutional Neural Network used in the YOLOv3 paper.

The model was trained using 70% of the dataset as the training set, and the remaining 30% as the cross-validation set. We used Transfer Learning [8], as it trained faster and gave better results as compared to training it from scratch. We used a small batch size of 4 as bigger batch sizes had a higher computational cost and set the number of epochs to 5. It took around 4 hrs to train the model on the K80 GPU given by Colab. Models after each epoch were saved in different .h5 files. The model after the last epoch gave us the least training loss as well as the least validation loss and hence, we used that for detection.



We evaluated the model and found that on setting the **IOU** to 0.5, **Object Threshold** to 0.3 and **Non-Max Suppression** value to 0.5, the average **mAP** was 0.9932.

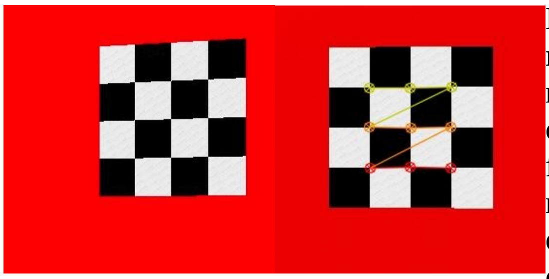
2. Depth Map using Stereo Cameras for Obstacle Avoidance

As the autonomous robot searched through the area for the desired object to be detected, a sensor was required to detect any obstacles that might come in its path to avoid it while simultaneously creating a map of the environment that was being explored. Various options were explored. **LiDAR** was a major candidate but such sensors are expensive and often prone to failure in sunlight. Therefore, an inexpensive and much more reliable approach was thought of – a **DIY stereo camera** to make a **depth map** of the environment.

Cameras Used -

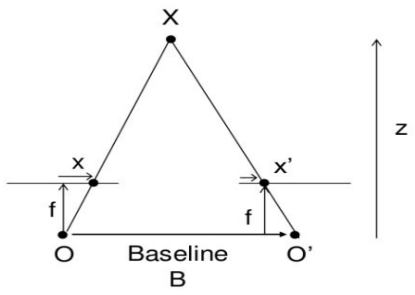
Rather than buying expensive stereo cameras, one can use two simple similar cameras mounted at a distance of **4 to 6.5 cm** from each other and then calibrate them using tools from **OpenCV** to work as stereo cameras. The crucial aspect is to make sure that both cameras produce images simultaneously.

Calibrating the cameras individually -



Every real-world camera has two kinds of distortions - **radial and tangential.** Due to this, straight lines in the real world appear curved in the camera images. This causes problem while creating a depth map. Therefore, first we need to calibrate each camera individually to remove **distortion**. This is done with help of a database of images of chessboards. We detect the corners in the chessboard and then remap the image so that the curved lines of the **chessboard** become straight. This process outputs remapping matrices which are applied to the images afterward in order to remove distortions [9].

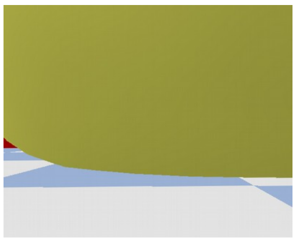
Calibrating cameras to work as Stereo Cameras -

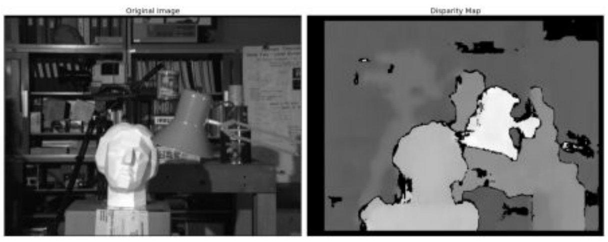


For two cameras to work in tandem as Stereo Cameras, we need to know the distance between them, the angle their axes make and their focal lengths. This is done automatically by the **stereoCalibrate()** and **stereoRectify()** functions in OpenCV. We apply these functions to the two camera matrices produced by the chessboard images of the cameras and they return the **remapping matrices** that remap the left and right images in a manner that they can be fed into the **StereoBM_create()** function [10].

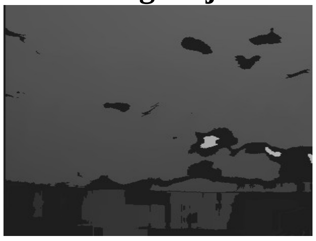
Creating the Depth Map -

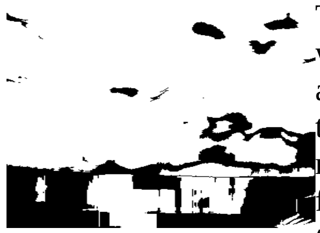
Simultaneous images from the two cameras are remapped to remove distortions and to fit them for the **StereoBM_create()** function that creates a **depth map**. The function also has various parameters such as **minNumDisparities, maxNumDisparities, blockSize, speckleWindowRange** and others which can be calibrated to obtain the best results [11].





Detecting Objects -

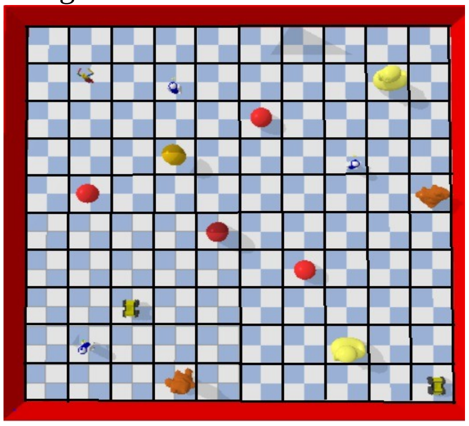


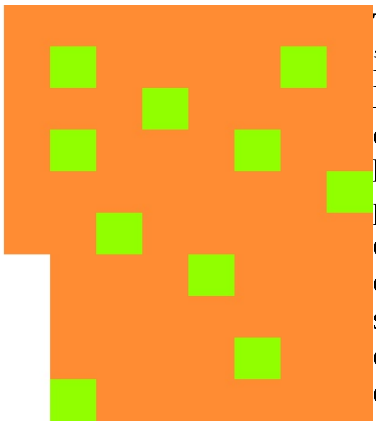


The depth map is **thresholded** to make the objects within a range appear as white and the background appear as black. The image is cropped to remove the floor and then the image is resized to (50,50) to make computations faster. Then, to avoid noise, the fraction of pixels that are white is calculated. If they turn out to be greater than 0.2, it means an object is in front and hence the protocol to avoid that object is executed.

3. Autonomous Movement of the Robot

A very simple approach was taken. The robot functions in an open environment, hence it can get information about its position and orientation through a GPS sensor. In indoor environments, however we can use probabilistic mapping and SLAM along with a LiDAR sensor for mapping and navigation.





The robot is fed information (latitude and longitude) about the corners of the area that it has to search. One major problem that we encountered was that calculating depth map and searching for objects was computationally very expensive. The simulation even started lagging due to it. So, we decided that the whole search area would be divided into a number of blocks that form a grid. The size of a block would be roughly 2x2m as that is the effective range of depth map we are using.
Hence, instead of keeping cameras open all the time, we will only perform optical functions before entering a new block. This way, the robot can work with substantially less computational resources without losing a significant amount of speed.

The robot can start from any one of the corners and then it covers the whole area moving from one block to another lane by lane, until it detects the target. If an obstacle is detected in any block, that block is skipped by moving from the adjacent lane and then coming back to the original lane. The algorithms to improve this method further are being worked on. The position and orientation of the robot with respect to its environment are obtained from the GPS centre at all times.

We also keep a track of the blocks that are empty and of those that have objects in them. In this way, a rough map of the environment traversed is created. More precise maps can be made if computational power is increased and depth map is calibrated properly. Then we may even calculate the shape of the obstacle in front.

Conclusion

The problem was solved in an efficient manner in simulation, while keeping in mind the real world conditions that the robot might face.

We look forward to improving upon the various aspects of this project.

References

[1] - https://pybullet.org/wordpress/

[2] - https://opencv.org/

[3] - J.Redmon and A.Farhad. YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767, 2018.

[4] - https://github.com/OlafenwaMoses/ImageAI

[5] - https://github.com/tzutalin/labelImg

[6] - https://docs.python.org/3/library/xml.etree.elementtree.html

[7] - https://colab.research.google.com/

[8] – C.Tan, F.Sun, T.Kong, W.Zhung, C.Yang and C.Liu. A Survey on Deep Transfer Learning. ArXiv preprint arXiv:1808.01974, 2018

[9] - https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

[10] - https://docs.opencv.org/master/dd/d53/tutorial_py_depthmap.html

[11] - https://albertarmea.com/post/opencv-stereo-camera/

Acknowledgements

Special thanks to The Robotics Club, IIT(BHU), its members and our mentors for guiding us and providing us with the opportunity to learn something new and exciting while executing it.

An Aside - Self-Balancing Bot Using RL

The field of robotics combined with Deep Learning is progressing faster than ever. One of the hot topics is Reinforcement Learning which helps in deriving an optimal policy for different situations. Here, a 2 wheeled robot which was trained using the DeepQ Algorithm of Reinforcement Learning to self-balance is also presented in the hope that in the future we can work on it to learn to move around on its own and help in mapping places which have even less space than our husky robot occupies. It was developed as a side project with the hope that someday such a robot will act as our searcher robot.