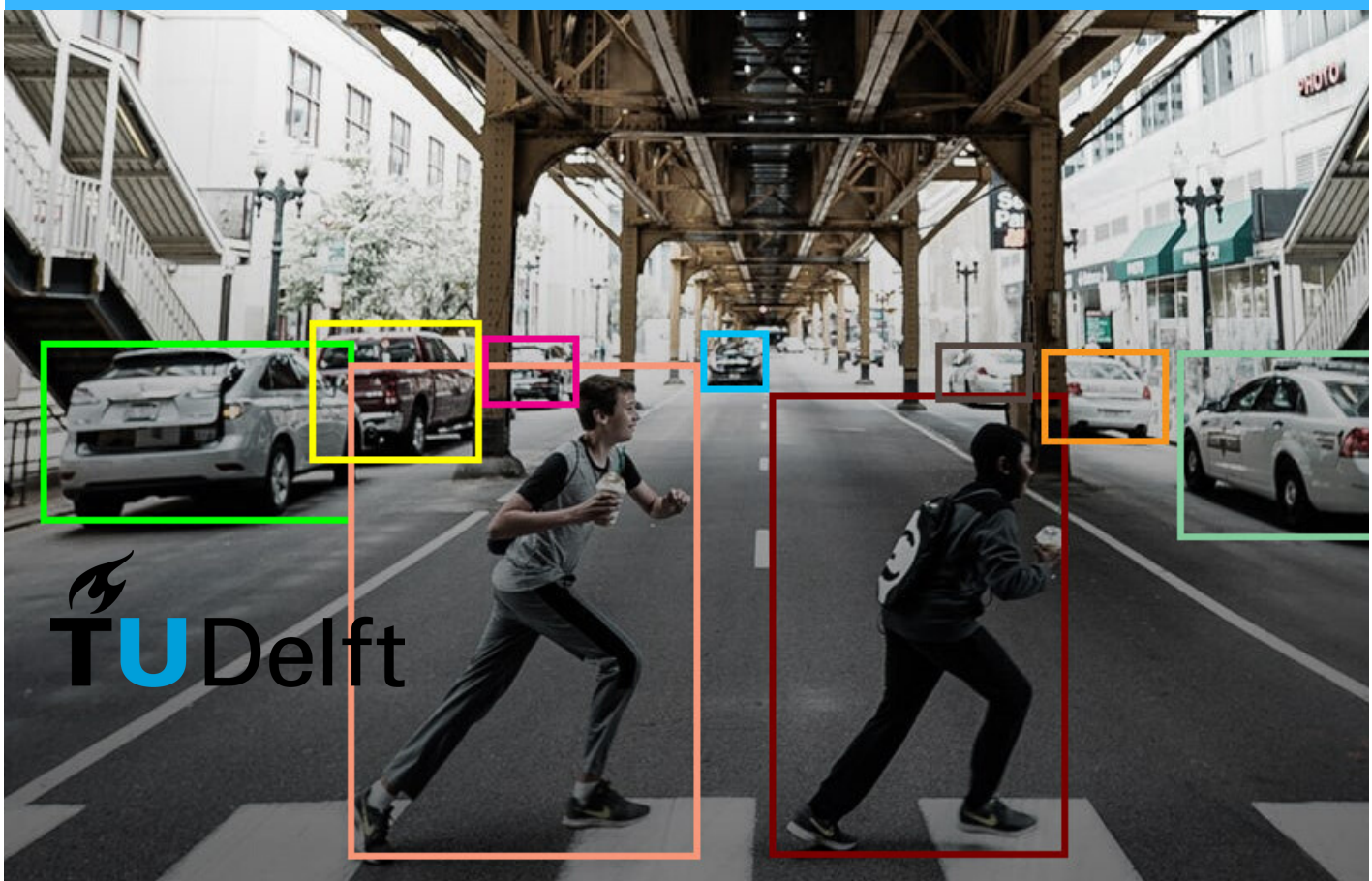
MSc Thesis in Embedded Systems

Multiple Objects Detection and Tracking Using Stereo Cameras

By
Diwakar Babu



MULTIPLE OBJECTS DETECTION AND TRACKING USING STEREO CAMERAS

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science in Embedded Systems

by

Diwakar Babu

September 2021

Diwakar Babu: *Multiple Object Detection and Tracking using Stereo Cameras* (2021)

The work in this thesis was made in the Computer Engineering Group from EEMCS, TU Delft

Supervisors: Dr. Arjan van Genderen

ABSTRACT

The idea of autonomous driving was merely just a dream about 5 years ago but now, with the advancements in technology, it has become prevalent. The aim of this thesis is to provide a low-cost approach for detecting and tracking moving objects from a moving platform. This could be used for an autonomous vehicle to automatically avoid moving objects. Our low cost approach will use a raspberry-pi processor board as computation platform and stereo cameras as sensors. The process of multiple moving objects detection is performed by initially calculating the disparity maps from the stereo image pairs. Following this is the generation of point cloud data from the disparity map which is followed by semantic segmentation and generation of object proposals. An EKF based tracker is used to track the moving objects across the frames.

Keywords: tracking, stereo vision, low-cost, openCV, computer vision, semantic segmentation, pcl, object detection.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Arjan van Genderen for his patience and his supervision during the project. Thank you for accepting me under you as a master thesis student and guiding me in every way possible to complete this project. It goes without saying that this thesis wouldn't have been possible without your invaluable contributions during the project.

A huge thanks to Dr. Stephan Wong and Dr. Rene van Leuken for accepting to be a part of my thesis committee.

A big thanks to my family back in India for their continuous contribution and support during this tough time. I would like to specially give my gratitude to my wife as well for having the trust and the immense support that I received during Covid times to complete this thesis project.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The emergence of autonomous vehicular systems began in the early 1980s. Initially developed automated vehicles depended on modified highway systems with embedded magnets along with vehicle-to-vehicle communication. With evolving technologies, the dependence on highway infrastructure was replaced by vision guidance. In the early 21st century began the usage of sensor systems and adept algorithms to carefully navigate the vehicles on roads.

Experiments on self-driving vehicles have been conducted since the early 1920s as mentioned in [Dri]. It was in 1984 when the first self-efficient and truly autonomous vehicle was developed as mentioned in [Thea]. Research and experimentation have only increased with time and progressing technology. In the figure 1.1 history of the evolution of the autonomous vehicles can be seen. Based on the level of automation, there have been various advancements over the past few decades and the fully automated vehicle is still a dream for a lot of aspiring researchers.

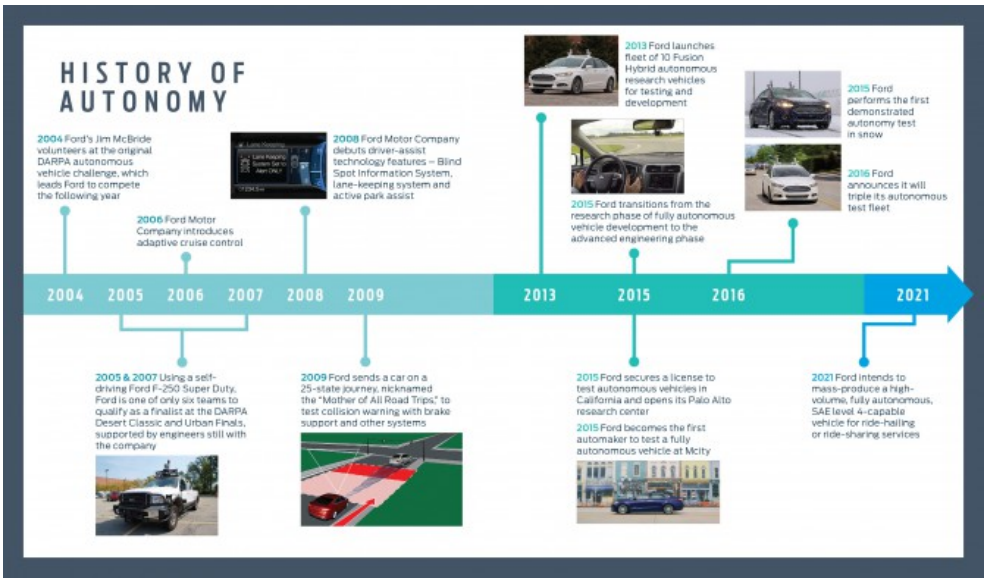


Figure 1.1: Timeline of autonomous driving cars as mentioned in [Aut]

The science of automation came into existence when the need to replace human labor arose, thereby increasing the efficiency many folds. Basically, unification of machines into a system which can function without human intervention. The precursor for automation is the science of mechanization. This uprooted from humans' ability to fabricate tools to leverage their muscular power. The invention of steam engines marked the beginning of the Industrial revolution. Machine tools were mechanized for the betterment of production with humans only managing the machines with little to no

physical exertion like before. Slowly, even the little human interventions required to run the machinery were phased out with the advent of feedback systems. These controlled feedback systems propelled efficiency by ensuring consistent operating levels. With advancements in technology, intervention of a computer processor took automation to the next level. As evolution progressed, the monotonous repetitive tasks in any process has become an obstacle in achieving an efficient system. Applying the process of automation can deliver significant benefits for the overall efficiency of any system. Unification of machines into a single working unit reduces human intervention. But unification of tasks done to reduce human's operational tasks plays differently in the world automation.

Development of fully automated vehicles (AVs) with no human intervention was challenging given the complexity of our dynamic environment. To tackle the said complexity, researchers greatly relied on the capacity of innumerable sensors to gather and process data for the AVs. Efficient algorithms were developed to control the sensors and collect data on impending obstacles which were further fed to the AVs for safe navigation in an independent manner. Although automated vehicles or robots can outperform humans when it comes to consistency and flawlessness, the sensory systems of humans are more sophisticated. Humans' perception of shapes, distances and an overall cognition to classify the objects cannot be matched by robots.

Talking about sensory systems, closed-circuit television (CCTV) systems came into place when surveillance systems were developed. Object detection and tracking are the two main crucial factors for surveillance, which are being done in many cities recently. But due to blind spots in CCTV systems, this compromises coverage and this led to the development of Unmanned Aerial Vehicles (UAV) called a drone. It uses variety of individual or combination of sensory systems like cameras, radar, LiDAR etc for object detection and tracking. Initially developed to be controlled by an user for navigation, the development in automation sector only made the drones smart with auto-flying features. Mainly used in military sector, can now be seen in private sector as well.

In the subject of aerial vehicles, the fundamental challenge was to make the AVs/drones fly and also to program them with the ability to fly them reliably and regularly as needed. The ability to fly reliably and with basic equipment was achieved at a very early stage of the industry. Later came the advancement which led to the research and development in the industry that led to continued usage of the drones in everyday life. From the Figure 1.2 we can see that the introduction of AVs started as early as 1890s when the technology's idea was new and engaging to research and mainly used in military sector.

1.1 MOTIVATION

Autonomous Vehicles represent a major innovation for the automotive industry. However the doubts and challenges to overcome are still huge as the implementation of an autonomous driving environment encompasses not only complex automotive technology, but also human behaviour, ethics, traf-

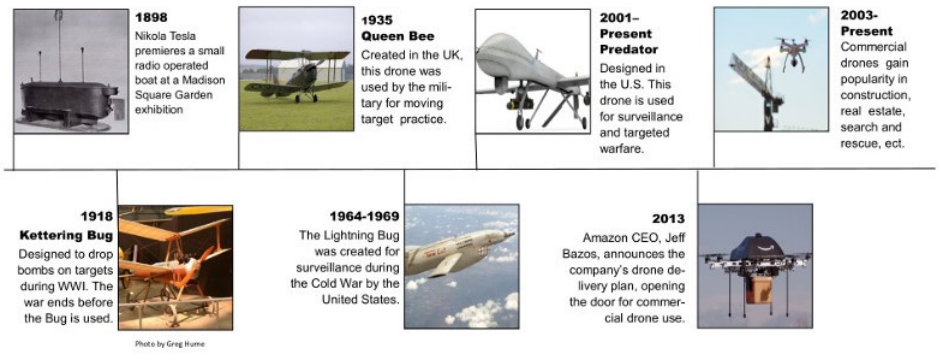


Figure 1.2: Timeline of unmanned aerial vehicles as seen in [Thec]

fic management strategies, policies, liability, etc. The automation of vehicles can be for different purposes. For operation of auto cranes, the automation is done for picking and dropping marked containers. For auto mowers, the automation is done for movement within a marked region and many more. Regardless of the purpose, navigation is key and more so in an unknown environment with many obstacles.

The most important challenges in unmanned aerial or land vehicles: designing a robust real-time obstacle detection and designing collision avoidance program. The two major constraints in system design: type of vision and the platform which runs the system. Systems like Radar, LIDAR, Camera, Sonar, etc. can be used to obtain the information of the environment. This information can later be used to position the obstacles and navigate without collision.

1.1.1 Sensor systems in vogue

Radar has the benefits of being able to detect objects through bad weather conditions, while the resolution decreases over range. The ultrasonic sensor is mostly used as a secondary sensor alongside the primary sensors.

On the other hand, Light detection and ranging (LiDAR) systems use laser technology to calculate the distances to objects by sending out light beams and collecting reflections returned by the objects in the environment. The laser range and reflectivity of the objects are the two key features that determine the efficiency of LiDAR systems. LiDAR though has high resolution and efficacy, it is not well suited for a low cost approach.

Camera-based systems are widely utilized to simulate humans' visual judgement. Despite the cost-effectiveness of cameras, their visual cognition is highly dependent on the environmental conditions (eg., rain or other natural phenomena will impose difficulties in gathering data).

1.1.2 Object Detection

Object detection is a computer vision technique to spot and position objects in a given image or video. The main approach concerning towards object detection uses machine-learning based detection methods. The concept of

object detection works hand in hand with image recognition and image segmentation. Image segmentation helps to discern the elements of the scene on a pixel-level and image recognition helps in labelling the detected object in an image/scene. The distinctive feature of object detection is the propensity to locate or spot objects within an image or video. The importance of object detection can be explained with the aid of the myriad range of its applications. A few unique applications are as follows; autonomous vehicles, surveillance purposes, face detection and so on as described in [Obj]

1.2 PROBLEM STATEMENT AND THESIS GOAL

An important aspect of a good navigation system is the detection of obstacles that may hinder the movement of the vehicle in which the navigation system is implemented. This detection is aided by sensory systems like cameras, 360°, LiDAR, RADAR, etc,.

The objective of this thesis is **to research and implement a method to detect multiple moving objects from a moving platform and track their trajectory that can be used in *inexpensive* unmanned vehicles like drones and land vehicles**. The sub goals of the thesis are:

- Study different types of sensors that are best suitable for use in the current scenario.
- Study different algorithms implemented already with respect to the decided sensor for detection and object tracking.
- Implement a robust algorithm that can detect the objects by using an array of image frames as dataset through simulation.
- Run the same algorithm for a real-time scenario and research on further applications.

Some of the boundary requirements can be defined as:

- The system should be developed with low latency due to the high speed movement of vehicles and the rate of decrease of distance between obstacle and vehicle is high.

Additional boundary requirements will be defined based on the type of sensory system and processor that will be used in implementing the system.

1.3 THESIS DOCUMENT OUTLINE

- Chapter 2 describes the related works that are done in focus with detection, tracking moving objects and collision prediction. The chapter 2 also briefly explains the different vision sensors and based on pros and cons how a particular vision sensor is chosen for this research.

- Based on the works previously done, Chapter 3 explains the methodology overview of the algorithm developed to answer the framed research question. This chapter also explains the implemented algorithm/method in the form of a flowchart/UML chart.
- Chapter 4 explains the part of the implemented algorithm that deals with the image acquisition part from the cameras and calculating the disparity map from the pair of images from the cameras.
- Chapter 5 briefly explains the task that is undertaken to detect the objects using the disparity maps obtained from the previous step. Before the object detection step, image segmentation is done to reduce the area of computation in the images for detecting the objects. And also the disparity map is converted to point cloud which is explained in detail in chapter 5.
- Chapter 6 explains in detail the step that performs the tasks of tracking the detected moving objects. Ego-motion compensation is done before tracking the detected objects due to the fact that the vision system is also placed on a moving platform.
- Chapter 7 shows the tabulated results obtained from running experiments on different scenarios. Along with the results, various scenarios and their respective image observations are also explained in detail in chapter 7.
- Chapter 8 concludes the thesis with the summary of the implemented algorithm. The problem statement defined in chapter 2 is evaluated with respect to the results from previous chapter. Along with this, the future directions of this work are also listed and explained.

2 RELATED WORKS

This chapter deals with the in-depth literature review that was carried out to understand and discuss the various processes involved in object detection and tracking algorithm. The object detection and tracking involves various steps in producing the desired output Therefore this chapter is divided into sections, explaining and reviewing the previous existing algorithms for each steps. This chapter will also discuss on deciding the type of sensor and board in which the developed algorithm will run.

2.1 SENSORS AND SYSTEMS FOR IMAGE PROCESSING

For an autonomous land or aerial vehicle, navigation through an unknown environment will be impossible without sensors to read the environment and provide the flight controller with required information. The sensors are chosen based on the following attributes:

- Weight, size and power consumption.
- The frequency of signal and gathering information.
- The required field of view.
- The compatibility of the sensor with the decided platform.
- The range of signal transmission

The controller board also plays a major role in processing the input and output. But for this thesis, the processor board will be chosen based on the type of sensor that will be used. But for basic requirements, the board should be able to handle the sensor input and output and should also be light weighted as discussed previously for sensor's requirements.

2.1.1 Ultrasonic Sensor, LIDAR and RADAR

For detecting short range obstacles, ultrasonic sensor is found to be very useful. Nowadays it is not very common to find ultrasonic sensor being used alone for navigation of autonomous vehicles. They are rather used in combination with other sensors. The main reason being the limitation their sensing range is small. Ultrasonic sensor is used as the chief sensor in detection of obstacles in the path of a fast moving robot where the speed of the moving robot was limited to 0.78m/s as seen in [BK89]. It was successful in detecting obstacles at a close range. Comparing this system to current scenario, by the time sensor detects fast moving obstacle from a moving

vehicle, it would be too late and the probability of collision increases. This leads to deciding the range within which obstacles must be detected and the frequency at which the algorithm must run.

The need for 3D imaging and the expansion of drone technology has led to rapid growth in LiDAR technology. Light Detection and Ranging (LiDAR) is a remote sensing method that can detect objects in real space with relatively high precision and accuracy, depending on the LiDAR unit's specifications. The sensor is fast when compared to any other detection sensors like ultrasonic, as it sends and receives pulses in nanoseconds.

In [XXW19] a LiDAR sensor is used in fast obstacle detection and tracking where a 3D point cloud is obtained from the sensor to map the environment and a Kalman filter based tracker is used to track the moving objects.

Based on the requirements, the cost of LiDAR can vary and it might seem very expensive for developing such a low-cost approach for a object detection and tracking system.

2.1.2 Monocular vs Stereo Camera

The use of cameras for object detection has been around for a long time. Long before the introduction of stereo cameras in the field of object detection and tracking, monocular cameras were being used in detecting objects. The figure 2.1 describes the steps involved for the object detection and tracking process as proposed by [WHN17]. This paper proposed a computationally effective algorithm where the detection of moving objects is done accurately and robustly on a 3D scene.

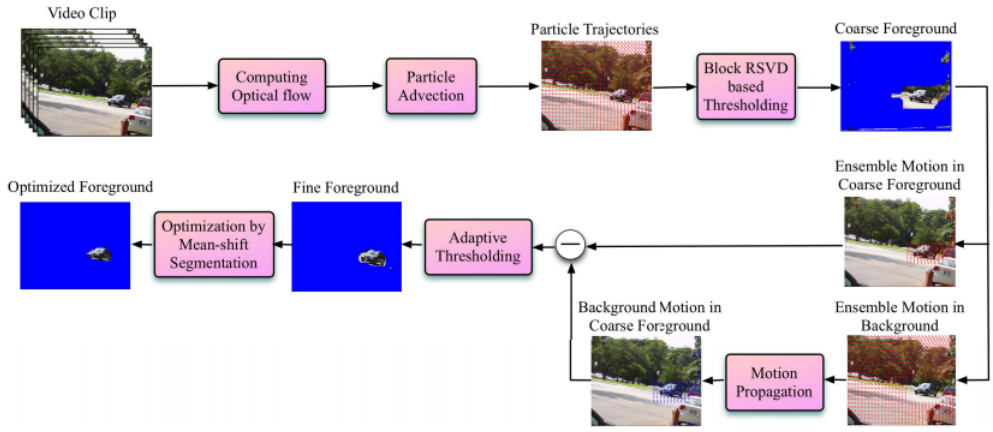


Figure 2.1: Proposed algorithm for object detection and tracking using monocular system.[WHN17]

In [KKS09] an algorithm was developed for detecting independently moving objects in an image sequence from a monocular camera which is mounted on a moving platform. Various geometric constraints such as epipolar (relative movement of the object) and the moving platform's motion is used to estimate the position of the objects on the images. The paper also uses a Bayesian Framework in determining the mobility status of a detected object and this estimation is tracked in the following images.

Before experimentation on object detection using moving stereo cameras, static stereo cameras were widely popular. In figure 2.2, we can see that the researchers in [WSV09] have explained Target Tracking using static stereo cameras' images and the tracking system was developed in combination with depth estimation.

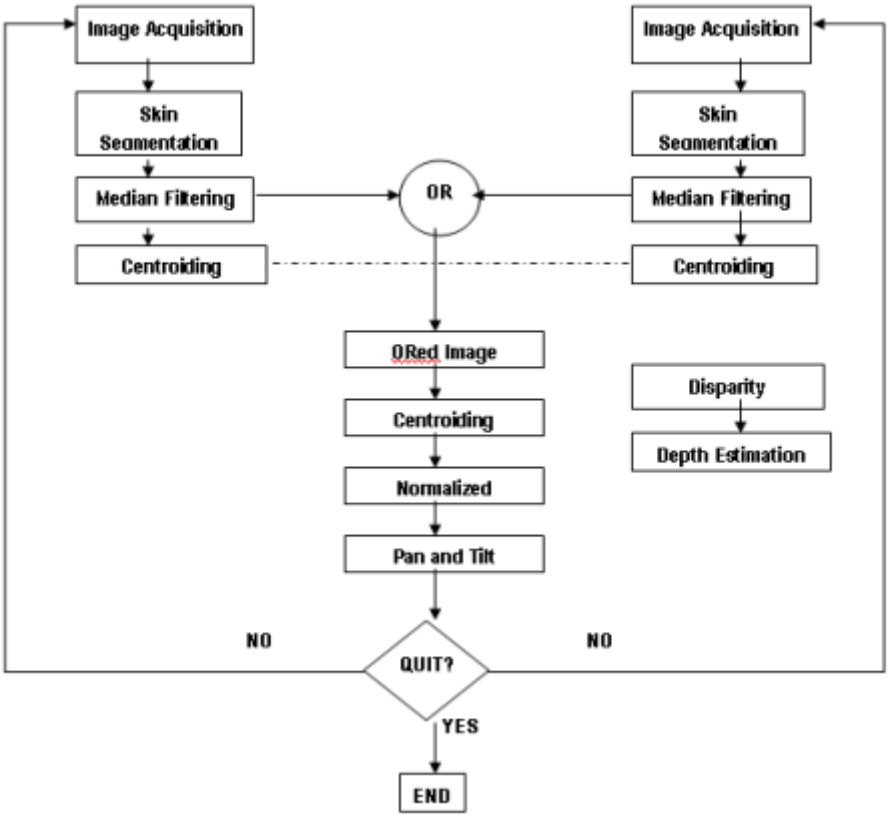


Figure 2.2: Process Flow from the paper [WSV09]. It is seen that the pre-processing of images (left and right pairs) obtained from stereo cameras are done as a parallel process.

Apart from using only stereo vision, combination with ultrasonic sensor is used by [YTLW18]. The proposed algorithm here is where the stereo vision is used to detect any moving objects and calculate a path for the UAV and the ultrasonic sensor is used whenever the obstacle detection algorithm fails.

A new process of obstacle detection algorithm is developed for unmanned land vehicles by [BMPK18] where they use stereo cameras for obstacle detection and use IMU sensors for semantic segmentation process. Inertial Mesaurement Unit (IMU) sensor can measure a variety of factors including speed, direction, acceleration, angular rate, etc. local to the device. This is very useful in case of determining the odometry of the device on which the cameras are mounted. The principle disadvantage of an IMU sensor is that they are prone to errors which accumulates over time. And also, they are considered to be an expensive addition to stereo cameras when just the stereo camera system can be used to determine the same.

In conclusion, combination of various compatible sensors can result in a proper function system for object detection. Due to the limitation of devel-

oping a low-cost system, stereo cameras will be used. Because the stereo cameras not only provides the disparity map which contains the depth information, but this disparity map can be used to generate the point cloud.

2.1.3 Controller Board

Using a stereo camera system can be done on any board, FPGA, SoC and controller. To consider the requirement of low-cost approach, this thesis project resorts to Raspberry-Pi boards. The raspberry-pi boards have their own Stereo-Pi setup with stereo cameras attached to a board that runs with a raspberry pi compute module as seen in [Steb].

2.2 STEREO CAMERA – OBJECT PROPOSALS GENERATION

The process of segmentation, performed on a point cloud is used to partition any image into multiple segments. These segments are later then classified into object classes which is used to differentiate objects from background. Now that we have different objects of different classes, they are bound by bounding boxes to generate what is called an object proposal. [CKZ+15] uses this approach where the proposed algorithm uses object size, detection of ground plane, depth information from point cloud, position for the bounding boxes for the detected objects, visibility and distance to the ground.

One of the main objectives in determining object proposals from point cloud is also the estimation of ground plane. Ground plane detection is done because it provides useful information such as the region through which the objects are moving and also the 3D location of the objects. Ground plane detection is also used for estimation of pitch angle compensation and improves the object detection and free space estimation [WQC+17]. The point cloud data contains objects (feature points or object proposals) not only traversing above the ground but present on other objects. This causes objects occlusion, which means that due to some property of the sensor system setup, the detected objects are unable to present themselves. Mostly it occurs when an object is being tracked, it is hidden by another an object. The algorithm should perform fine even when occlusion occurs. The proposed algorithm in [CPBY14] (as seen in figure 2.3) detects the ground plane using RANSAC (Random Sample Consensus).

The architecture of determining object proposals from a point cloud proposed by paper [SWL19] is seen in figure 2.4. This is done in two stages. Stage-1 consists of segmentation of the whole image and point cloud into foreground and background which then generates a high quality 3D proposals and this is done in a *bottom up* manner. The stage-2 consists of collecting and transforming the object proposal points which is used in combination with the world space information (depth and therefore the position) to learn better about the detected points for accurate bounding box refinement and prediction.

The 3D descriptor matching works by projecting the 3D point cloud onto the 2D images and transforming the 3D detection problem into the 2D space.

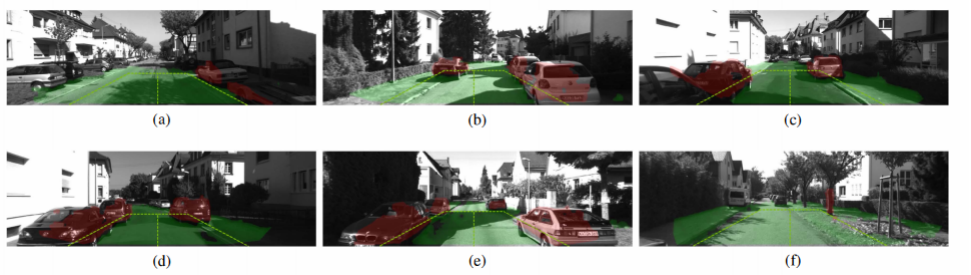


Figure 2.3: The proposed algorithm when applied to these dataset, the traversable regions (ground plane) is depicted in green and the objects are depicted in red [CPBY14].

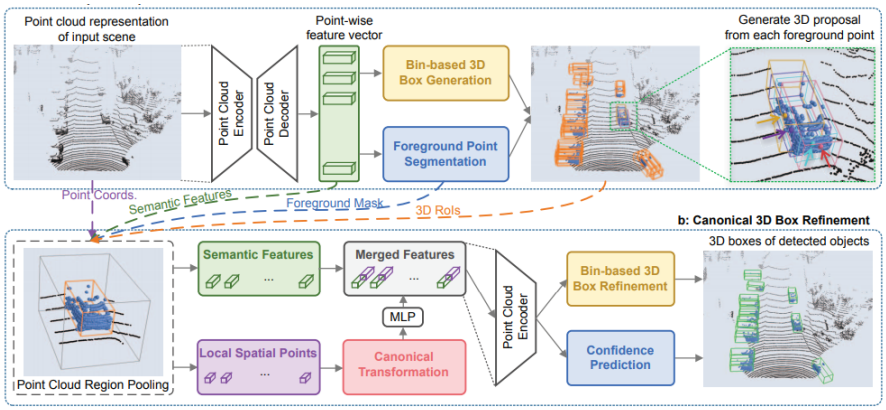


Figure 2.4: The architecture of the proposed algorithm from [SWL19]

The paper [PN16] does the projections at multiple views and use convolutional neural networks (CNN) for the 2D detection tasks as it is possible for the CNN to handle the multiple viewpoints and rotations for objects belonging to the same class. The proposed algorithm has proven to be competitive on overall performance when compared to other 3D point cloud object detection methods. The results of this experiment can be seen in figure 2.5 and the table 2.1 depicts the comparison of different object detection methods with the proposed multiview CNN based object detection (all the result values obtained from [PN16]).

Time	Multi-level CNN	Multi-View	3D-Scan	FPFH
6-Class 3D	28s	350s	450s	2400s

Table 2.1: Time Comparison for Detecting 6 Object Class

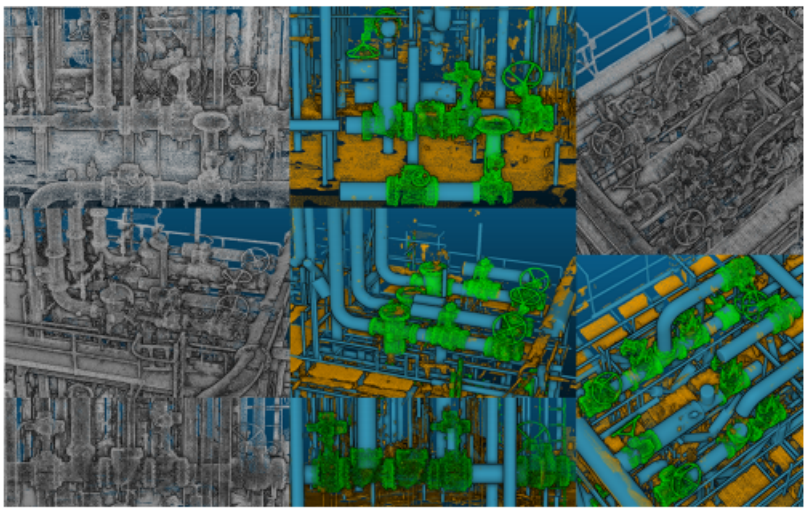


Figure 2.5: Result of the proposed 3D object detection algorithm on a large-scale 3D industrial point cloud [PN16]

2.3 PRE-EXISTING LIBRARIES

As seen from the previous section, the pre-processing stage involved in object detection is to obtain the image frames from the cameras, estimating the disparity map from image pairs and using the disparity map to generate a point cloud.

The researchers of [ZL10] have used the OpenCV library [Ope] for the camera calibration (the process of estimating the intrinsic and extrinsic parameters of the cameras), image rectification (the proces of reprojecting the image pairs on to a common image plane) and stereo matching (depth estimation).

The low-cost approach of generation of a point cloud will ultimately rule out the use of LiDAR and will make use of pseudo LiDAR point cloud generation process. The researchers of [RC11] have discussed in detail regarding the usage of Point Cloud Library [Poi] but this thesis will focus more on generation of point cloud using the disparity map.

2.4 POINT CLOUD – OBJECT TRACKING

After the object detection along with it's 3D information, the detected objects had to be tracked across other frames.

Paper [DRWC+19] uses LIDAR based detectors for vision only object detection. The 3D object detections for the developed tracking algorithm is done by initially computing a depth map, which is obtained from a sequence of stereo images. This is then followed by the generation of 3D point cloud, which is analogous to that of a LiDAR point cloud. This is why it is called a *pseudo LiDAR*. Finally the 3D detections are created using the widely used LiDAR detectors. The paper later compares the performance between LIDAR and pseudo-LIDAR detection across multiple sequences. The performance was evaluated based on the metrics: accuracy (in terms of tracking

the detected objects), precision (error in meters) and false positives(marking objects at wrong places) and negatives (inability to identify an object that was previously identified). The table 2.2 shows that when using a stereo camera (pseudo LiDAR) for object detection, the accuracy reduces which is because of the ability of LiDAR to detect objects even at a farther distance. The stereo camera based tracking also suffers from more missed detections (FN). The figure 2.6 shows when the vehicle is driving straight with a single car in front (same lane), and two cars in the opposite lane.

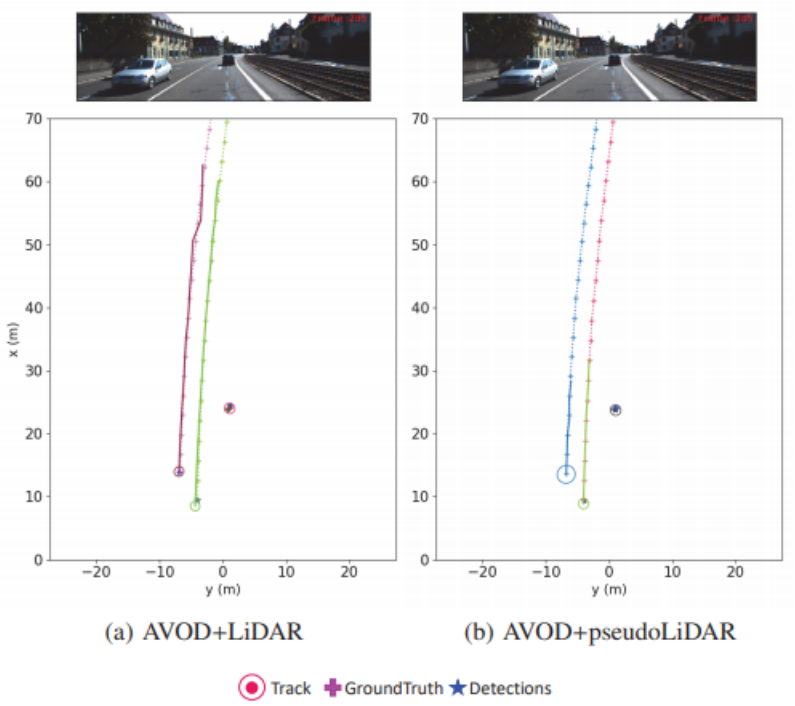


Figure 2.6: Trajectory tracks generated using LIDAR (a) and pseudo LIDAR (b) detectors. Dashed lines with cross markers denote the ground truth tracks. The detections received on the latest frame are denoted by blue stars. Solid lines denote estimated tracks. Ellipses denote two standard deviation bounds on position estimate [DRWC+19].

Detection	**Accuracy**	**Precision**	**FP**	**FN**
LIDAR	61	0.528	975	1924
Pseudo LIDAR	33.5	0.384	336	4729

Table 2.2: Tracking Performance on *KITTI* dataset

One of the most common and majorly used tracking algorithm in image domain is Kalman filter based tracker. The paper [SM11] provides a detailed evaluation of the three different kalman filters: *linear (LKF), extended (EKF) and unscented (UKF)*. Filtering is done mainly to estimate the next state of the system using the previous state and measured observations and parameters. Kalman filter is an iterative prediction-correction process that predicts and corrects the state of the system. If the system had nonlinear dynamics, a sub-optimal estimation can be achieved by using the EKF and UKF whereas LKF is mostly used for systems with linear dynamics. Comparing the Root Mean Square Error (RMSE) in terms of accuracy between the three filters,

we see that in figure 2.7, UKF has smaller errors compared to EKF. But the paper also mentioned that the UKF had the largest computational time when compared to EKF because of the unscented transformation. While this thesis can't use LKF due to it's nature of nonlinearity, EKF based tracker will be used to reduce the computational time.

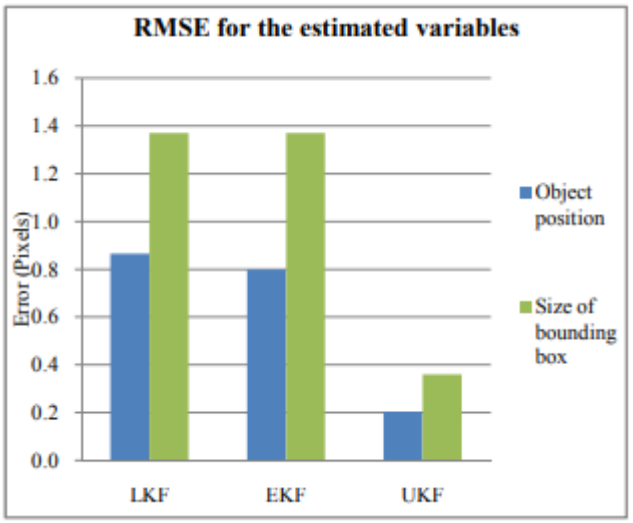


Figure 2.7: Estimation error (RMSE) for object location and bounding box

2.5 CONCLUSION

The thesis topic, considering the low-cost approach of detecting and tracking moving objects, will make use of stereo cameras for sensor and the algorithm will run on Raspberry-Pi (Stereo-Pi) processor board. The resolution and frame rate of the input will be decided based on the processing frequency and power of the board. The algorithm will initially involve the pre-processing stage of estimating the disparity maps from the stereo image pairs for which the *OpenCV* will be used. The generation of point cloud from disparity maps will be done using the library *PCL*. This is later followed by detecting the object proposals, which is done on a segmented image. Before the image segmentation process, a ground plane is introduced using the algorithm developed in [ZEF16] and then follows the Voxel Cloud Segmentation proposed in [PASW13]. This is later followed by the object proposals generation, for which an algorithm developed by [OMML17] is used.

For the tracking stage, object proposals from successive frames is used. As seen previously, the most commonly used algorithm for tracking is based on Kalman filter and the computation time of EKF based filters is found to be less than other kalman filter based trackers.

3 METHODOLOGY

There have been many significant researches done on different ways to detect objects in the path of an autonomous vehicle and track the detected objects. As seen from chapter 2 this thesis will use *Stereo Pi* stereo cameras with *Raspberry Pi 1* compute module. The stereo cameras will obtain both left and right pairs of images and will run the implemented software on the compute module. The proposed method for Object detection and tracking using stereo camera images is a systematized process consisting of a combination of several open source algorithm as seen in the figure 3.1

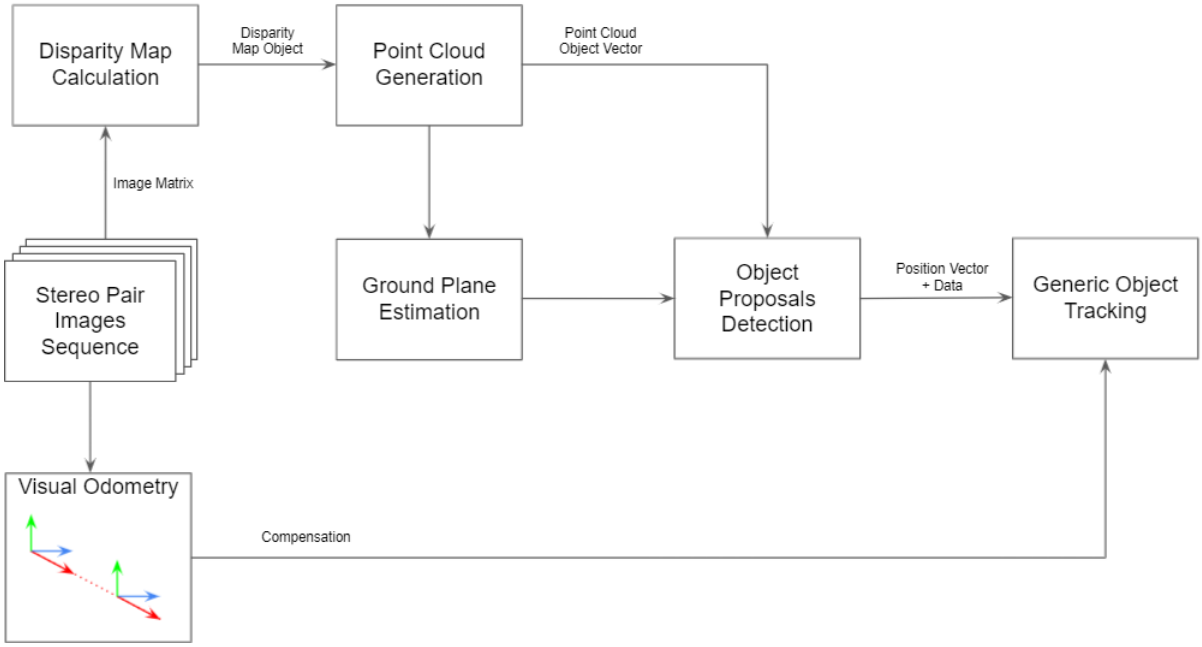


Figure 3.1: Proposed Method for Object Detection and Tracking

The first stage of the process is to obtain the *left* and *right* pairs of images from stereo cameras. This splicing process of images is done on the incoming joint images and stored separately in different folders which are accessed for the further stages. In a real-time scenario, the video captured by the stereo cameras are instantaneously sliced and the images are stored for further process. After the splicing process, these image pairs are used for Stereo Matching or Disparity mapping process. Stereo Matching (as an example seen in fig 3.2) will compare the surrounding of a pixel p in the left image to slightly translated positions q in the right image to estimate the disparity of the pixel p. Each pixel is processed separately, without taking the full image context into account, which often results in noisy disparity [SLD18]. Brief explanation of the process is provided in the chapter 4

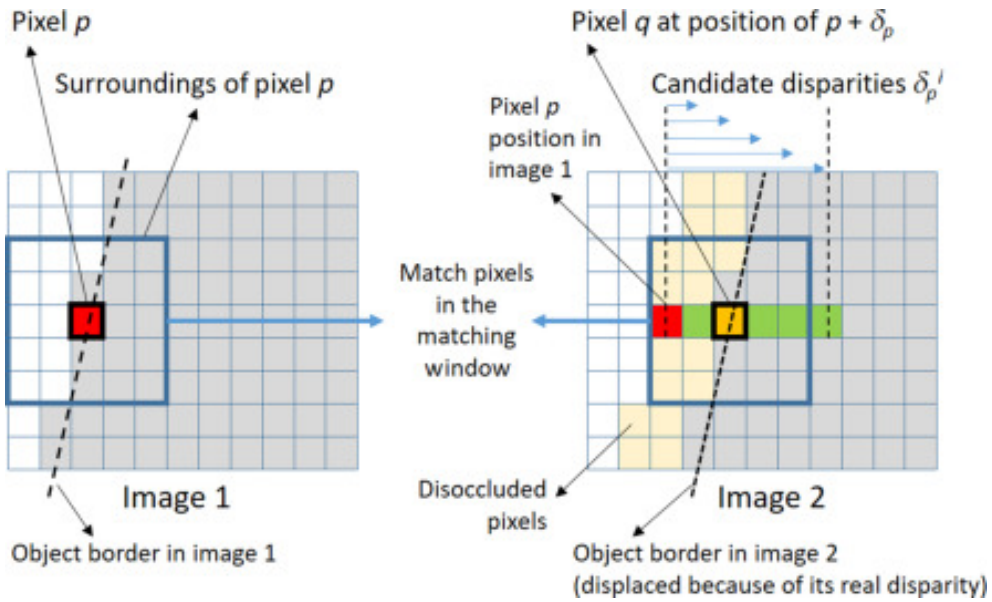


Figure 3.2: Stereo Matching of Image Matrices [SLD18]

Before the object detection algorithm is applied, from the disparity maps, point clouds are generated. After the generation of Point Cloud from the disparity map, a Ground Plane is introduced to detect the objects resting on the plane. Later it is subjected to an object extraction algorithm to detect feature points and segmentation which undergoes clustering to fuse different groups of points belonging to the same object. These cluster of points, belonging to a single object are called 3D proposals where each proposal is a cluster that contains an object. The 3D object proposal is a *class - object* which also contains the 3D information of the detected object. These clusters are then bounded by a box and its geometric centre is obtained. The reason for segmentation is to remove points of cluster within the image that does not contain any value of interest. Brief Explanation of this process is provided in chapter 5.

Tracking is when a detected object is moved, we have to find the displacement of it. In this case, when the cluster of points, along with its geometrical center moves, we find the displacement. At each time step, these objects (cluster of points) are rectified with respect to the relative motion of the platform 6). During this process, the detected 3D object proposals are transformed to a common coordinate frame using the process of visual odometry. The 3D information, relative to the camera, of these clusters is provided as an input to Extended Kalman Filter for tracking the said objects. Brief explanation of the process is provided in chapter 6.

4 STEREO VISION

This chapter describes in detail about how the images are obtained and pre-processed before the implemented object detection and tracking algorithm is applied. The first step in pre-processing stage involves obtaining images from video frames, followed by splicing the image into left and right pairs. A stereo camera provides a single image with both left and right pair attached to each other. After splicing process, disparity map is calculated from the obtained left and right pairs.

For this thesis, the pre-processing stage readies the initial images for the conversion to point cloud, which is done during object detection phase. This is an important stage because in a real-time scenario, when there is any delay in this phase, it can lead to a huge latency in the final phase, which is the tracking. The different phases in this stage are discussed below.

4.1 OBTAINING IMAGES

Images are nothing but matrices ***n x m*** (n rows and m columns) which can be stored in the form of a matrix with the class Mat from OpenCV Library as discussed in section 2. The StereoPi cameras that is used in this project directly obtains the image in grey scale instead of converting color images to grey scale as a separate process. Scenario where people are continuously moving will be the test case and therefore in real-time, the video frames are used as input images. The video frames are obtained at different frame rates such as 30 and 60 and have a resolution of 640x480 for both cases. The total duration of the test case video is 3 seconds, which produces 90 frames for 30 fps and 180 frames for 60 fps. Considering the limited processing power and Raspberry Pi 2's compute module, a limited number of total frames are processed to avoid latency and get average performance of the implemented software.

4.2 SPLICING IMAGES

Images are obtained together, where both the left and right cameras' images are produced jointly, where the image matrix has both the left and right pairs' matrices. So, for disparity processing, both the left and right images must be spliced and stored separately. This is done by using the function, from OpenCV library, ***Rect*** which takes in the parameters start and end coordinates and the width and height of the resulting images after splicing. These image pairs are stored to different folders for further processing. These im-

age files are stored with incremental numbered file names, which makes the job to read these images later for disparity mapping easier.

4.3 DISPARITY MAP CALCULATION

A disparity map tells us the relative difference in the location of the object with respect to the viewer. The reason why human eyes can perceive depth is because of the alignment of the eyes. Before the eyes could perceive the depth, the difference in images from the left and right are automatically processed by the brain. Similarly, in stereo vision, the two cameras are placed in such a way that cameras are pointed to same object but from different angles. This is used to calculate disparity map as seen in figure 4.1 which contains the depth information of the whole scene.

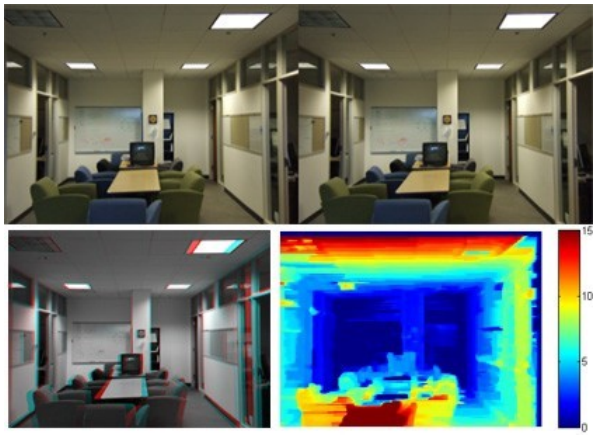


Figure 4.1: An example of Disparity Map calculation obtained from [Theb]. (The top left image displays the left pair and top right image displays the right pair. The bottom left image shows the rectified image based on left and right pairs, which is used to calculate the disparity map as seen in the bottom right image.)

In this project, the disparity map containing the depth information is calculated using the block matching (BM) algorithm as discussed in chapter 2. But before applying block matching algorithm to calculate depth information, we need the rectified image calculated from left and right pair. An additional step of individual camera calibration should be done before the rectification process to recover parameters like optical center, radial distortion, orientation rotation etc,. Take the example as seen from figure 4.2. The two cameras (left and right) are aligned vertically and therefore the observed object $\boldsymbol{P}$ will be observed in the same coordinates vertically (y coordinate) and only the horizontal coordinate x will be different and we can focus on that to calculate the depth and if the object $\boldsymbol{P}$ is closer to both cameras, the difference in x coordinate will be high and vice versa. To calculate depth, we need to calibrate the cameras and rectify the image pairs which will result in a rectified image containing the same object $\boldsymbol{P}$ lying on the same y coordinate with two different x coordinates.

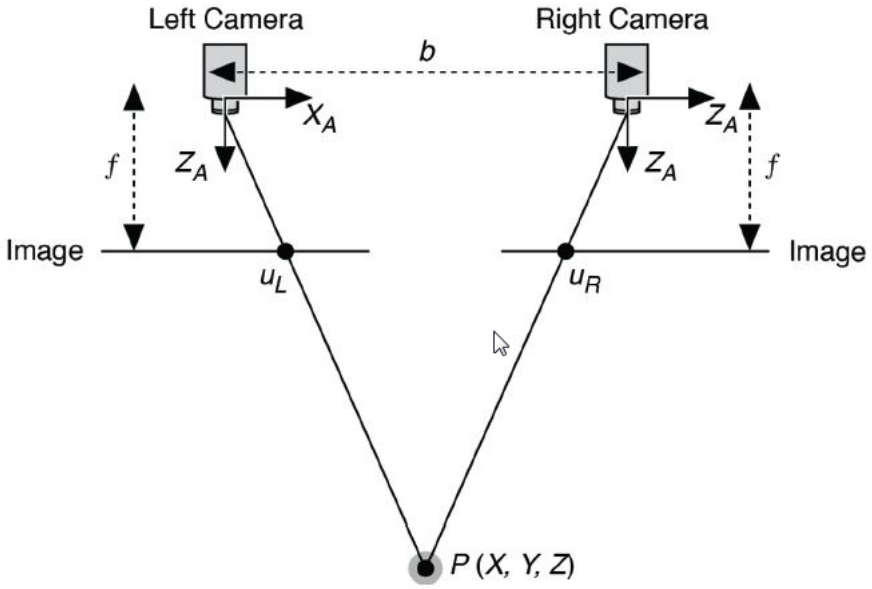


Figure 4.2: An overview of Stereo Camera Setup obtained from [Theb]

Camera Calibration: As seen above, the camera calibration is the process of obtaining information about the camera, which is required to determine an accurate relationship between a 3D point in the real world and it's corresponding 2D project, mainly the rotation and translation parameters. This thesis follows the most commonly used calibration method, the checkerboard method. This is done by defining the real world coordinates of the 3D points using the checkerboard pattern of known information. The reason behind using checkerboard pattern is that it is distinct and easy to detect in an image. By obtaining multiple images of the checkerboard from different angles, the first step is to find the checkerboard corners using the ***findChessboardCorners*** function from OpenCV Library. Once a checkerboard has been identified, ***calibrateCamera*** function is used to pass 3D points in world coordinates and their 2D locations in all images. Now these values can be stored in a *yaml* file which can be accessed easily, instead of running calibration for every frame.

Rectification: Since the cameras are aligned at an angle to focus on an object, we need to align the obtain image pairs parallel to each other. This is done by the stereo image rectification process, where the two images from left and right cameras are re-projected on to a common plane parallel to a line that passes through the optical center (camera parameter). This ensures that the corresponding points have the same y coordinate and are related by a horizontal translation as discussed earlier. Using the camera's intrinsic and extrinsic parameters, rotation and translation, stereo rectification can be applied. Stereo rectification applies rotations to both camera images to bring them to same plane as seen in figure 4.3. This is done by using ***stereoRectify*** function from OpenCV library. This function also returns the projection matrices in the new coordinate space. The next step is to calculate the undistorted and rectified left and right images using the ***initUndistortRectifyMap*** and remap functions, separately for left and right images.

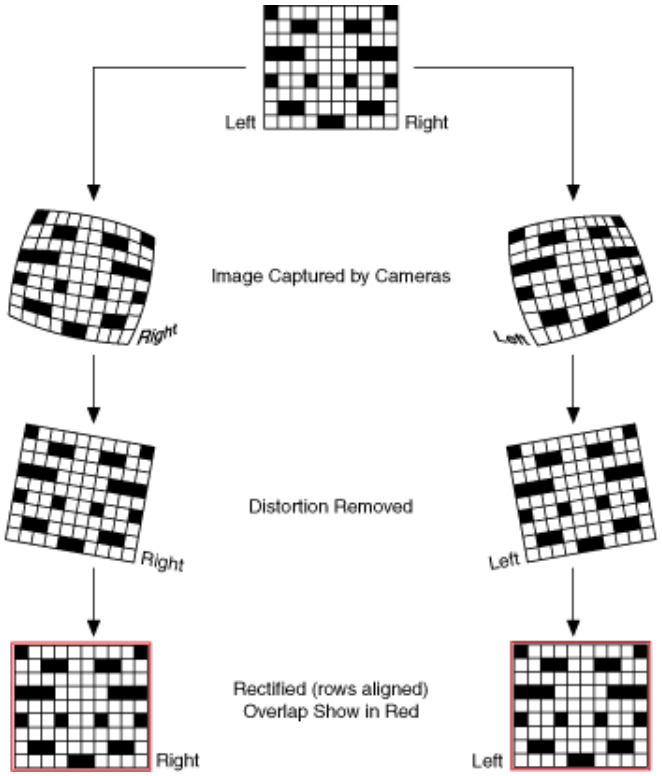


Figure 4.3: Flowchart of Rectification process as discussed in [Theb]

Block Matching: After rectification process, we understand that the single object can be seen with two different x coordinates for left and right cameras separately. The difference between the x coordinate should give us the disparity value but this changes with change in location of object. This can result in a slow process and increase in error due to improper rectification process. This is the reason why Block Matching algorithm is used. Instead of comparing just the pixel value in the same row of the stereo image pair, using the neighboring pixels will result in reducing the computation time. Also, sometimes multiple pixels corresponding to different images can have the same pixel intensity.

As seen in figure 4.4a from the left pair (or right), a block is chosen and it is searched and tried to be matched in the right pair as seen in figure 4.4b. The matching method used here is the *Sum of Absolute Difference*. It is calculated by taking the absolute difference between each pixel in the original block (the block from left pair) and the corresponding pixel in the block from right pair (being used for comparison). In figure 4.4b, a pixel block on the right image pair is chosen (the white box) as the same position on the selected original block on left image pair (as seen in figure 4.4a, the black box) and its neighboring pixel blocks (the white and green boxes) are compared with the original block from the left pair. In figure 4.5, we see that the two selected pixel blocks produce a sum of absolute difference of 938, which means that if the same pixel block from left pair image is compared with a difference pixel block (say the white box) from right image would have produced a sum of absolute difference more than 938.

(a) Chosen Block from Left pair

(b) Block Searching in right pair on same axis

Figure 4.4: An example of Stereo Block Matching Algorithm Process

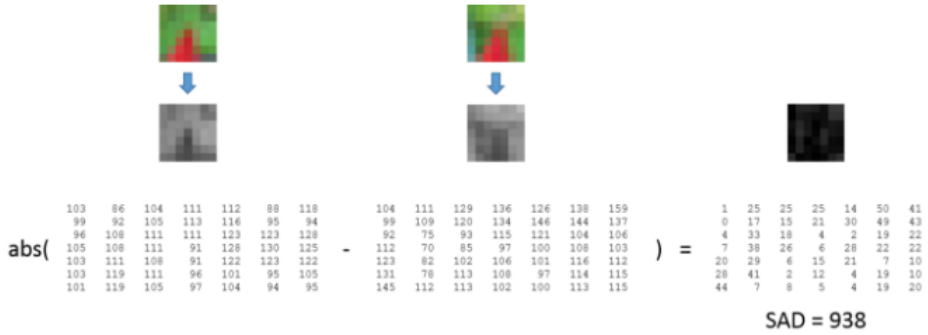
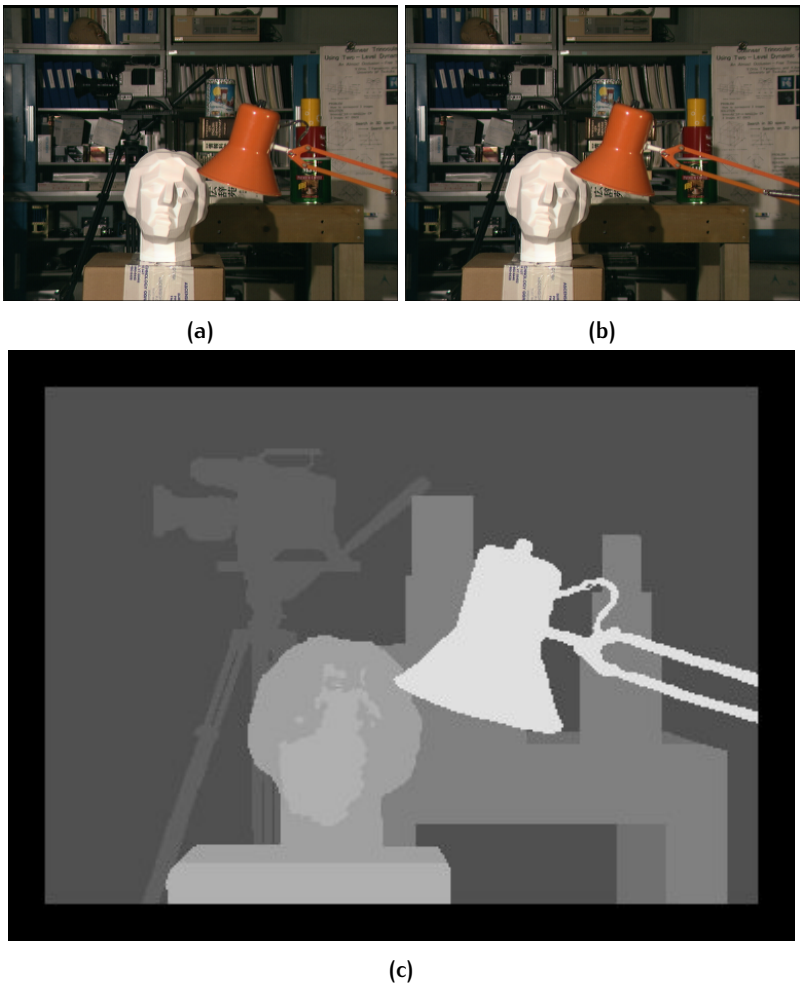


Figure 4.5: The above selected pixel blocks from both left and right image pairs are compared and the resulting sum of absolute difference is calculated.

Searching and matching is done only in a single axis because the images are rectified. The OpenCV library provides the implementation of such a block matching algorithm using the ***StereoBM*** class which is used to obtain the disparity map from a pair of rectified left and right images, just like in figure 4.6.



(a) (b)

(c)

Figure 4.6: The resultant (c)Disparity Map from (a)Left pair and (b)Right pair (This gray scale disparity map explains that the darker area means they are far away from the observer and brighter area means they are closer to the observer.)

4.4 CONCLUSION

The process of obtaining frames from cameras, splicing them, rectifying the image pairs and calculating disparity maps of every frames was discussed in this section. The scenario considered here does not entail high speed moving objects or objects that are located far from the camera. Due to these reasons, frames are obtained at a lower resolution with less frames per second. To make sure that the whole project comes under the low-cost approach, open source library OpenCV is used for these pre-processing stages.

Due to absence of parallel processing in Raspberry Pi compute module, obtaining images and calculating the disparity map from rectified image pairs can result in a slow process, since the next set of frames (from time t+1) will already be available but won't be processed until object detection and tracking algorithm is performed on the current frames. The processing/-computation time details derived from running this pre-processing stage for images obtained from video of duration of 1 second, running at 640x480

resolution for 30 fps (30 frames) and 60 fps (60 frames) can be seen in table 4.1

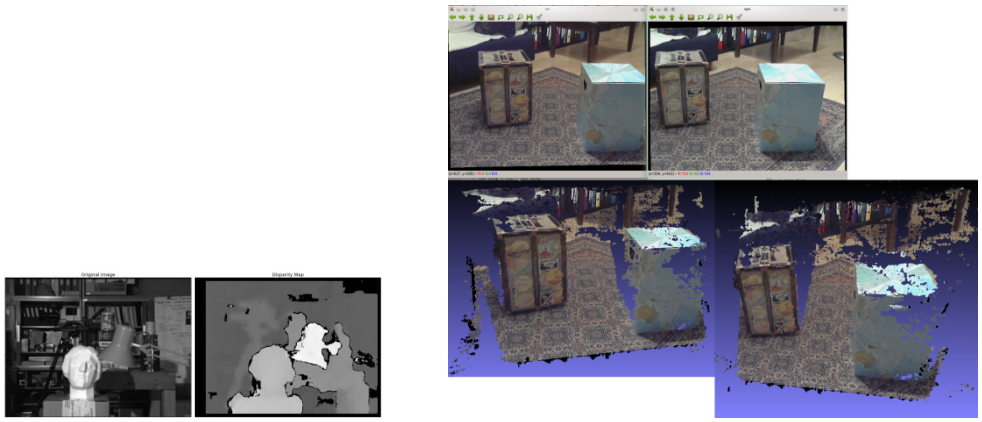
Number of Frames	Time (in ms)
30	10-12
60	19-23

Table 4.1: Different running or processing time for obtaining images and calculation of disparity images

In the next step, the obtained disparity maps are subjected to the conversion to point cloud where object detection in combination with scene segmentation (for faster processing) and clustering (to group clusters belonging to same object) is used to obtain possible sets of objects with 3D information. Let us see that in detail in the next chapter.

5 OBJECT DETECTION

This chapter discusses in detail about the generation of 3D object proposals from the previously generated disparity maps. But before applying the object detection algorithm, the disparity map will have to undergo a bit processing. Remember that a disparity map is an image that is derived from a left image and a right image, and that in the disparity map each pixel also has information about by the distance between the position where it was found in the left image, and the position where it was found in the right image as seen on the figure 5.1 (a). But on the other hand, a point cloud as seen in figure 5.1 (b) is a set of data points, that is suspended in space, where the points represent the shape of the objects that are present in the image.



(a) Disparity Map **(b)** Point Cloud

Figure 5.1: Difference between disparity map and point cloud

Following the generation of a 3D point cloud, a ground plane is estimated and fit into the generated point cloud to reduce the area of processing the image.

The final stage in object detection is generating object proposals, which includes Semantic Segmentation, which is the process of linking each pixel in the point cloud to a class label like person, car, furniture, etc,. Semantic segmentation is a type of image classification at a pixel level.

The algorithms used for conversion of disparity map to point cloud, estimating and fitting the ground plane and the detection 3D object proposals are referred from previous works, which will be mentioned in detail in the upcoming sections.

5.1 POINT CLOUD GENERATION AND GROUND PLANE ESTIMATION

The generated disparity map contains 3D information about the scene but to compute 3D object proposals, we first need to convert the disparity map into a point cloud. The generated point cloud can be visualized using the PCL library functions as mentioned in [Poi].

5.1.1 Generating Point Cloud

The mathematical steps involved in converting a disparity map to a point cloud is as follows

- Every row and column of disparity matrix is processed one by one.
- Current disparity ID is found from the current row (*row*) and column (*col*).
- The disparity value (*disp_val*) of ID is obtained from the matrix.
- Depth is calculated using the equation 5.1.
- The x and y coordinates are calculated using equations 5.2 and 5.3 respectively.
- The translated coordinated are added to the output PCL for the current row and column.
- The above steps are repeated for all the columns in a single row of the disparity matrix.
- The above steps are repeated for all the rows of disparity matrix.

$$z(depth) = ((focal\_len) * (base))/(disp\_val) \tag{5.1}$$

$$x = (((col) - (cen\_x)) * depth)/(focal\_len) \tag{5.2}$$

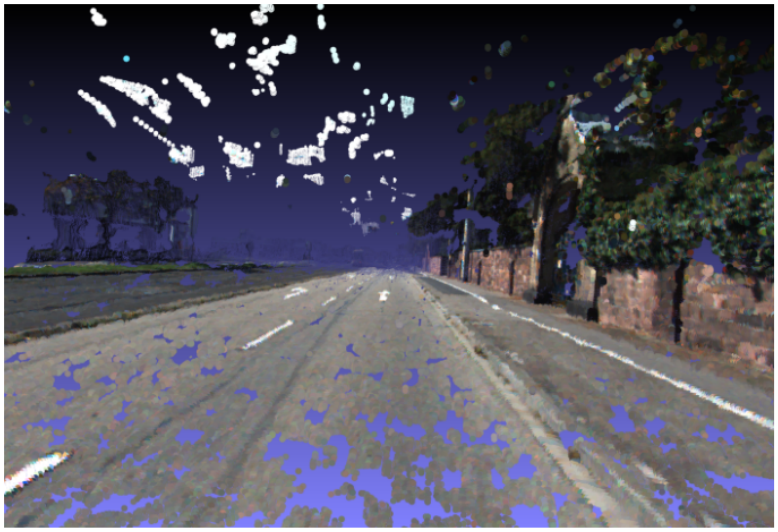
$$y = (((row) - (cen\_y)) * depth)/(focal\_len) \tag{5.3}$$

where *focal_len* represents the focal length, an internal parameter of the cameras (both left and right will have the same focal length, since they both use the same lens) and *base* represents the base line, also an internal parameter which is calculated based on the camera's position. These values are obtained from the camera calibration step, described in the subsection of 4.3

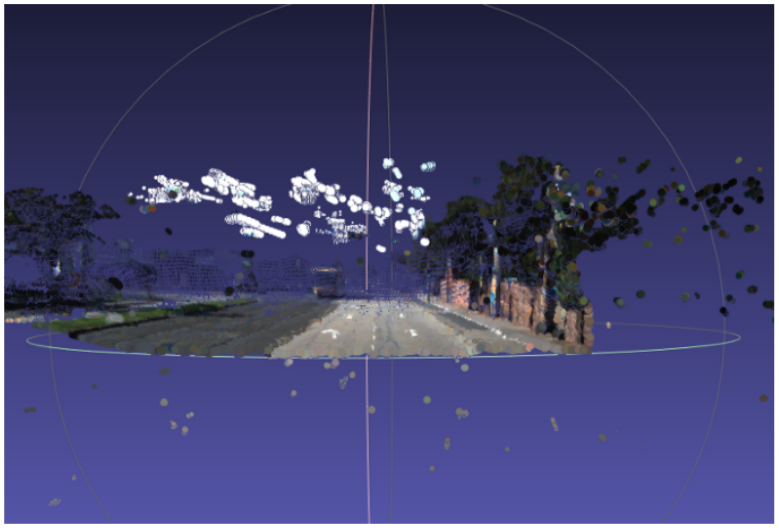
The ***PCL Library*** contains the function ***DisparityMapConverter::compute()*** which follows the above mentioned algorithm in converting the disparity map to a point cloud and this project uses the same. To visualize the generated point cloud, the function ***DisparityMapConverter::getImage()*** is used

to get the image by coloring the generated point cloud and ***pcl::visualization::cloudViewer()*** is used to visualize the point cloud. This is done only to check the output and uses more processing power. Therefore it is not used when tracking module is also running. As seen from the example in figure 5.2, the left and right image pairs (figure 5.2a) are used to calculate the disparity map (figure 5.2b), which when projected onto a 3D space, produces the point cloud as seen in the figure 5.2c.

(a) Stereo vision Pair of images



(b) The resulting Disparity map from the stereo vision pair of images



(c) Visualization of the generated Point Cloud from the disparity map

Figure 5.2: An example of Point Cloud Generation (an example described in [Stea]

5.1.2 Ground Plane Estimation

The ground plane detection is the process of detecting the ground plane in a scenario above which all the objects reside and move. By fitting the ground plane, the object detection process is reduced because of the reduction in the area of image that has to be processed in detecting the objects.

Algorithm 2. Proposed enhanced RANSAC with estimated surface normals

```
WHILE iterations < i
  random_inliers := n random selected points of data
  plane_parameters := parameters of the fitted plane to random_inliers
  consensus_set := random_inliers
  surface_normal := local surface normal to the consensus_set

  FOR every point in data and not in random_inliers
    point_normal := local surface normal to this point
    normal_error := difference between the normal to this point and
surface_normal
    IF point fits plane_parameters with a distance_error < d and
normal_error < m THEN
      add point to consensus_set
    END IF
  END FOR
  IF the number of points in consensus_set is > k THEN
    (A good plane is found)
    this_plane := plane parameters fitted to all points in consensus_set
    this_distance_error := distance error of the fitted points
    this_normal_error := normal error of the fitted points
    IF this_distance_error < best_distance_error and
this_normal_error < best_normal_error THEN
      (the found plane is better than the other previously found planes)
      best_plane := this_plane
      best_consensus_set := consensus_set
      best_distance_error := this_distance_error
      best_normal_error := this_normal_error
    END IF
  END IF
  increment iterations
END WHILE
```

Figure 5.3: Algorithm used for Estimating Ground Plane on a 3D Point Cloud from [ZEF16]

This project makes use of the algorithm that is mentioned in the [ZEF16], which describes the estimation of detecting ground plane on a point cloud using RANSAC, as mentioned in figure 5.3. The depth information for the plane estimation is provided from the disparity map matrix and the generated point cloud library. This method was used in [CPBY14] in detecting traversable ground plane and other obstacles on a street view. Figure 5.4 describes the same.

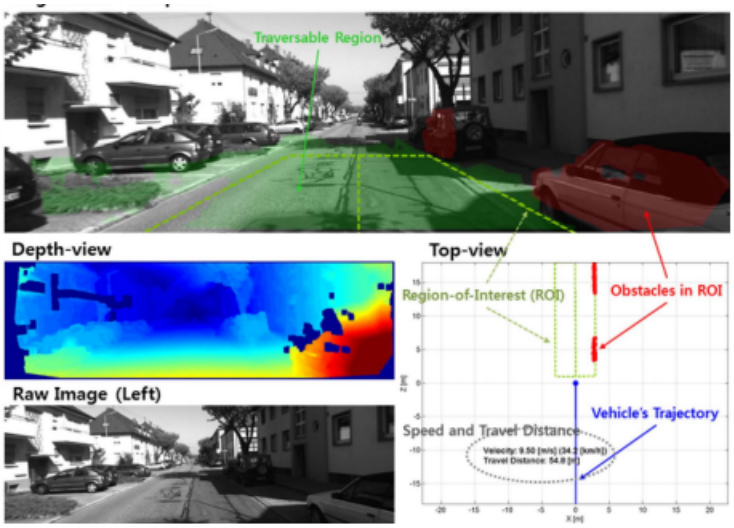


Figure 5.4: Visualization of Ground plane detection as seen in the top image. The middle left image shows the depth view of the same scene, while the bottom left image is the actual scene. This is an implemented algorithm for ground plane detection from [CPBY14]

5.2 DETECTING 3D OBJECT PROPOSALS

The generated point cloud was processed to find and estimate the ground plane on it. The next step involves the process of object detection, where semantic segmentation is performed on the generated point cloud to classify the image segments into different classes. Basically it is the process of classifying all the objects (of interest or not) into different classes such as *cars, pedestrians, buildings, etc,..* This is performed based on the idea of [HK08]. The semantic segmentation process is followed by the generation of 3D object proposals.

5.2.1 Semantic Segmentation

3D point cloud segmentation is the process of classifying point clouds into different homogeneous regions such that the points in the same isolated and meaningful region have similar properties. To be precise, it is the process of linking each point in the cloud to a class label, where these labels could mean a person, car, flower, furniture, building, sky, etc., As an example, [CKZ+15] performs semantic segmentation on a single image, classifying the image into different parts such as roads, pedestrians, trees, etc,. as seen as in figure 5.5.

This project follows the approach developed by [OMML17]. The reason for performing segmentation is to classify parts of images or frames, that are of no interest (like buildings, trees, etc,.) and they are ignored for further processing.

Before the clustering process, the segmentation classifier is initially trained using KITTI dataset [GLU12] and using this classifier, the software filter out regions which are unlikely to belong to an object, where an object is classified based on labels from annotated images like cars and pedestrians.

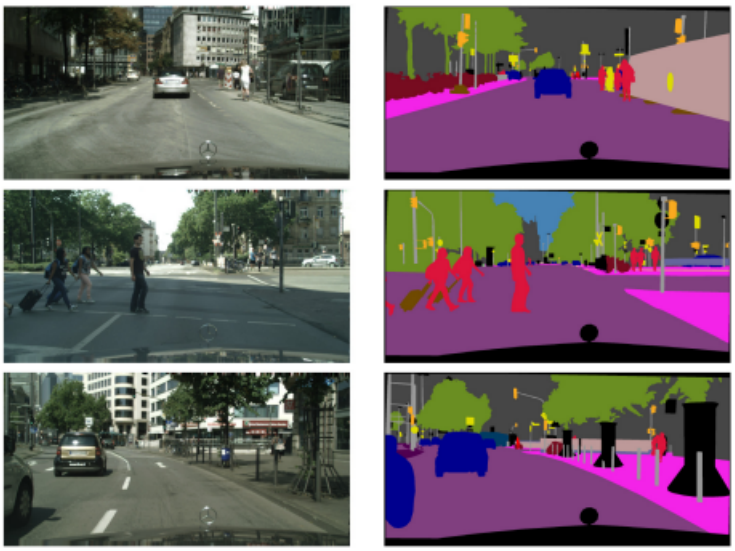


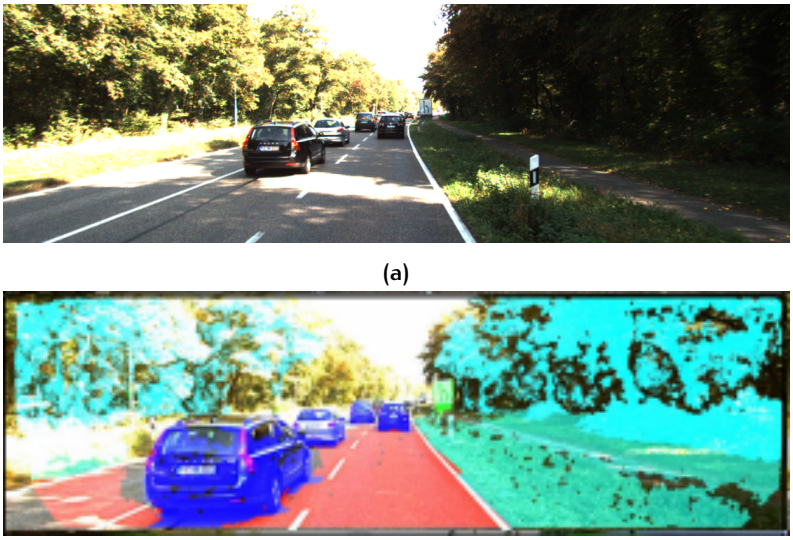
Figure 5.5: The set of images on left, when segmented produces the result as seen on the right side

The segmentation is done by clustering certain points from the point cloud and then use the information to classify the cluster of points into a single object. To be precise, the segmentation algorithm groups pixels in an image into meaningful regions. This is done with the help of steps based on segmentation process by [PASW13] which uses Voxel Cloud Connectivity Segmentation (A voxel is a three dimensional analog to a pixel, representing numerical quantities like color, location in a three dimensional space, etc. A supervoxel similar to voxel but only larger in size).

Voxel Cloud Segmentation generates over-segmented 3D point cloud dataset called *supervoxels*. These supervoxels adhere to object boundaries better than the current 2D methods. One of the main property of supervoxel is that they do not cross the boundary of an object. It is done using the existing class in the point cloud library, ***pcl::SupervoxelClustering*** as mentioned in [Sup]. It is performed as follows:

- Load the input point cloud obtained from previous step.
- Check the input arguments and set default values. Default values refer to the parameters of supervoxels such as the voxel size (determines the leaf size), seeding size (the size of the supervoxels) etc.
- Initialize a supervoxel clustering using the above mentioned class, setting values to the data structure in the created object such as centroid (the centroid of the supervoxel) etc,.
- The 'viewer' function from the library is used to add, get and set rendering properties like *voxel_centroid_cloud* to get the centroid and *colored_voxel_cloud* to add color to the supervoxels according to their labels.
- This process is iterated throughout the point cloud by extracting the previous supervoxel's adjacency list, creating a point cloud of the centroids of each supervoxel's neighbors.

This results in a supervoxel graph that is stored as point cloud data, where each supervoxel clusters is labeled based on the classified objects from segmented regions. For the visualization, the visualizer from *pcl library* is used, to use the function to add color to segmented scene. From the figure 5.6, we see that they implemented their developed algorithm with KITTI benchmark image 5.6a from [GLU12]. The original image frame contains a moving car and other objects like trees and bushes (without taking the ground plane into consideration). The segmented image 5.6b shows that the multiple moving cars are segmented as blue regions and the tree and bushes are classified into cyan regions.



(a)

(b)

Figure 5.6: (a) shows an image obtained from a stereo vision KITTI benchmark, that shows multiple cars moving.(b) shows the semantically segmented image of the input image.

5.2.2 Extracting Object Proposals

An object proposal generation method by [OHE+16] produces a set of object proposals from the region of interest in the point cloud. But this set of object proposals might contain under- or/and over-segmented objects, where a group of pedestrians and cars are classified under same class of object.

Therefore, [OMML17] developed a method, where these classified objects are referred as short 3D proposals and these proposals provide precise 3D information and measurements, mainly the objects' positions with respect to camera and it's size. These 3D object proposals, when combined with 2D detection, provide an observation. The project follows the object detection process by creating a *detection* class, that uses object parameters from the *object_proposal* class.

- Object proposal objects have parameters such as original point cloud indices (the list of point cloud data, from the segmentation process), estimated ground plane (the ground plane point cloud data is used to

visualize the ground plane), bounding box setter (based on the radius of the detected point clouds), etc.

- Every object proposal has a parameter called score which is set based on the detection of voxel points of each object. This parameter is used to reduce the overlapping of feature points that are of an object present in two different clusters. This means that overlapping detected voxel clusters will have the same point cloud data. This value changes over iteration, which is later filtered to get proposals that are well generated.
- Initially the object proposals are filtered based on the score, which refers to the detections that are good to worse (good detections have more score value). These scores are set by iteratively running the steps.
- Following this, a CRF (combined random field) model is used to select suitable proposals from the previous step to reduce the over-segmentation.
- By getting the maximum value of width and length of every proposals generated, bounding box class functions are used to draw the bounding boxes around objects which share the same label.

This results in the list of point cloud objects, that are semantic segmentation and when visualized is bounded by a 2D box. From the previously seen segmented figure 5.7a, the segmented image is then subjected to the developed model in generating the object proposals, which are bounded by boxes as seen in figure 5.7b.

(a)

(b)

Figure 5.7: (a) shows the semantically segmented image of the input image as seen from the previous subsection. (b) shows the generated object proposals that are bounded by a bounding box.

5.3 CONCLUSION

From this chapter, we see that the final set of object proposals with their 3D information such as position and size are generated using various previously implemented approaches. The above developed algorithm for processing the generated disparity map for object detection was run for various scenarios, with one or more than one type of objects and the running time is as shown in table 5.1. This table, gives the value of run time for processing a single frame (right and left images obtained at time t).

Object types used	Time (in ms)
1 (2 Persons)	10
2 (2 Persons + 1 ball)	15

Table 5.1: Different running or processing time for generating object proposals from the calculated disparity maps.

The use of high resolution camera will only increase the running time but will help in detecting objects that are farther from cameras or smaller in size. To compensate the high running time, usage of high resolution cameras can be coupled with using a processor with high processing frequency.

6 TRACKING

The detected object proposals are tracked throughout the different frames. The position of the detected objects in world coordinate domain is obtained by initially finding the orientation and position of the moving camera, which is done using visual odometry. Then later, the detected objects are projected on the world space domain. This is then followed by tracking the objects using a Extended Kalman Filter based tracker.

6.1 VISUAL ODOMETRY

Visual odometry is the process of determining the position and orientation of the moving stereo cameras by analyzing the associated image frames. A calibrated stereo camera pair, which helps compute the feature depth between images at various time points is used to compute the actual motion. This is done with the algorithm that is developed by the [GZS11]. A visual odometry class is created with calibration, bucketing data structures and functions that update and get motion and it is also used to compute the transformation vector to matrix. The step by step process is as follows:

- Bucketing is the process of reducing the number of detected features and this is spread uniformly over the image domain.
- Following this, the feature points from the previous frame are projected on the current frame, using the transformation matrix, computed using the transformation vector. For this, we use the camera calibration parameters from the stereo camera setup.
- This process is iterated to compute the motion which uses the current coordinate system (determined from the transformation matrix) along with the orientation. The transformation parameters are used to obtain the velocity vector also known as ego motion vector.
- Now we calculate the state vector using the equation 6.1.

$$X = \begin{bmatrix} p & v \end{bmatrix}^T \tag{6.1}$$

where the state vector is X, p is the position vector of the moving stereo cameras (represented as $[x \;\; y \;\; z]^T$ which uses 3D space geometry derived from the image as reference) and v is the velocity vector (represented as $[\dot{x} \;\; \dot{y} \;\; \dot{z}]^T$). Now that the position and state vector of the stereo cameras is known, as the camera moves, this position is used as the reference to determine the position vector and compute state vectors of the objects detected,

projected on the world space domain. This state vector is used in EKF based tracker.

6.2 TRACKER

The state vectors of different bounding boxes (which refers to the detected objects) obtained from consecutive frames are then tracked to estimate the movement of the boxes across frames. A filter is used to filter the observations and refine the matching process across different frames. The project follows a EKF based tracking algorithm developed in [OMML17].

The basic algorithm in an Extended Kalman Filter (EKF) is as follows:

- Predicting the State (X)
- Compute the error covariance
- Compute the Kalman gain
- Update the state estimate
- Update the error covariance

The goal is to estimate, update and predict the state vectors of the detected bounding boxes which move with constant velocity across frames. The tracking is performed as follows:

- An extended state vector is calculated using the variance. This variance is computed by subtracting the distance of the same object proposals (bounding boxes) in successive frames. This variance is updated in every frame, as it is used in calculating the correction in position of bounding boxes.
- The current extended state vector is then used to predict the error using the EKF function (based on an example implementation from here) and this process is iterated for subsequent frames.
- The centroids of the bounding boxes are then connected using a line that is drawn.

6.3 CONCLUSION

At the end of tracking process, the detected object proposals are tracked and the position of the objects are predicted using the EKF equations. Missed detection and occlusions are handled by the update, predict and compute model of the EKF trackers and the objects are associated with the individual objects' trajectories.

7 OBSERVATION AND RESULTS

This chapter is used to view the results of the implemented algorithm to detect and track the multiple moving objects and their results are discussed in detail. Various scenarios are also used to run the developed algorithm to check the results.

7.1 DEVICE

The stereo-pi system is obtained from [Steb] on which the developed algorithm is run as seen in figure 7.1. This stereo pi system uses a raspberry pi 1 compute module which contains the guts of Raspberry Pi 1. This uses the BCM2835 processor with a limited memory of 512MB RAM. This supports 2 Raspberry Pi cameras (OV5647 sensors).

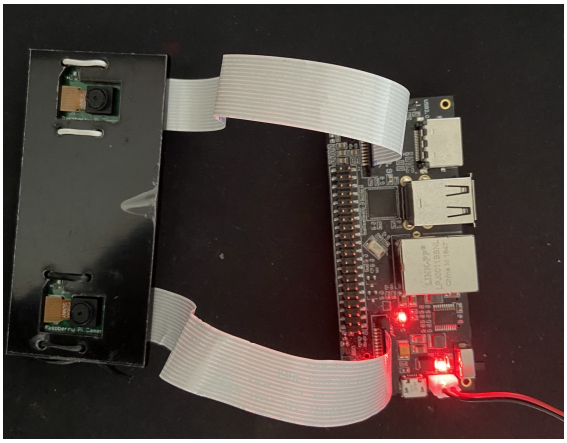


Figure 7.1: The device used for this thesis is the above shown Stereo-Pi with Raspberry-Pi compute module, with the stereo cameras.

The following sections containing the results were obtained by running the implemented program on the above mentioned device, where the images are directly fed from the stereo cameras to the process. Instead of training the data while processing for semantic segmentation process, the project uses already trained data set due to the availability of minimum processing power of the device. While running this program, we obtain the run time for different steps as seen in table 7.1

Process	Time (in ms for a single frame)
Disparity Map Calculation	2
Point Cloud Generation	18
Object Proposal Generation	5
Tracking	4

Table 7.1: Different running or processing time for different processes that are implemented.

7.1.1 Single type of Object - Person walking

In this section, a single object is used to detect and track across the frames, which is captured by the stereo pi cameras. From the figure in 7.2, it can be clearly seen that a single object type (person) is moving across the frame. Figure 7.2g shows the tracking line, with the bounding box of the detected object type (person).

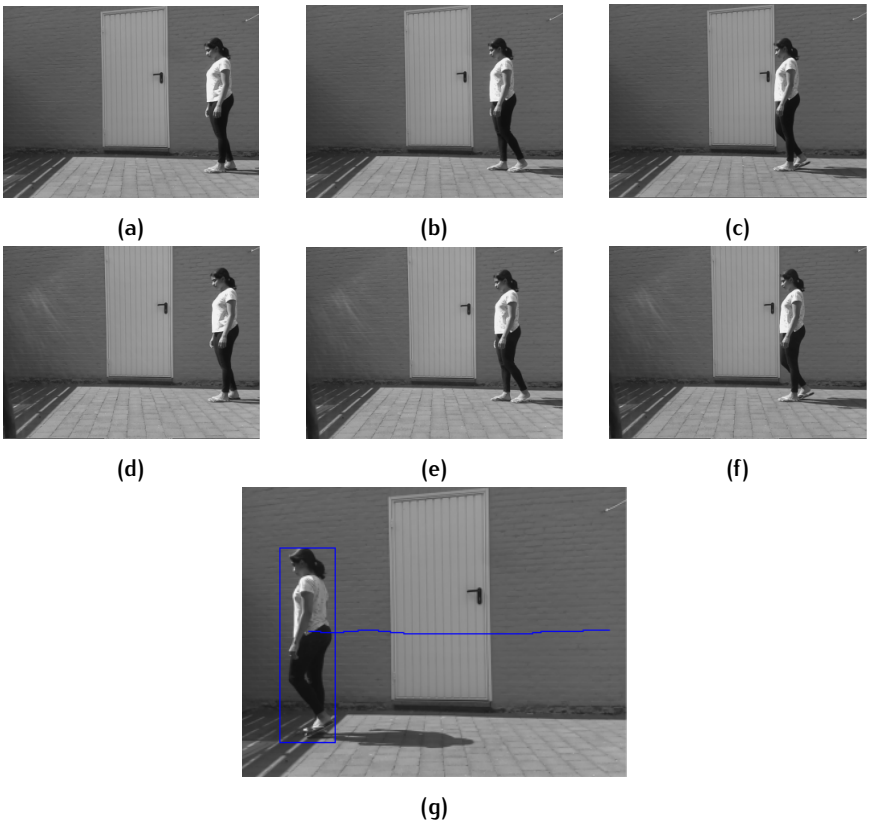


Figure 7.2: (a), (b) and (c) shows the left image frames taken at time t, t+1 and t+2 respectively; (d), (e) and (f) shows the right image frames taken at time t, t+1 and t+2 respectively and finally (g) shows the tracking line, that follows the detected object (person).

7.1.2 Single type of Object - Person walking - Walks back

In the figure 7.3, the person walk across the frame and starts to walk back from the other side of the frame. The tracking line still holds clearly, without any error in this case.

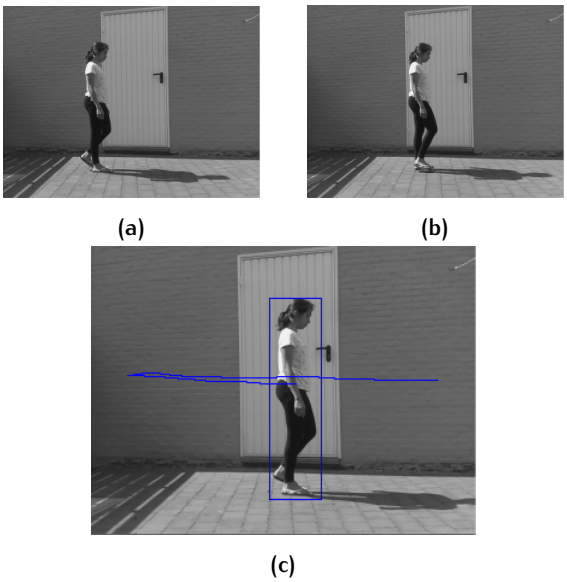


Figure 7.3: (a) and (b) shows the left image frames where the object (person) walks back after reaching one end of the frame; (c) shows the tracking line, that follows the detected object (person), tracking from time t=0 (initial point).

7.1.3 Single type of Object - Person walking and Crossing

In this particular scenario, two objects of same type (2 persons) are seen walking along the same line and this results in following a tracking line which follows along both the objects of same type as seen in figure 7.4. Figure 7.5 also shows that when the objects move back from the other side, the tracking line continues to track them without breaking and without getting affected by any occlusion occurred while the objects pass each other.

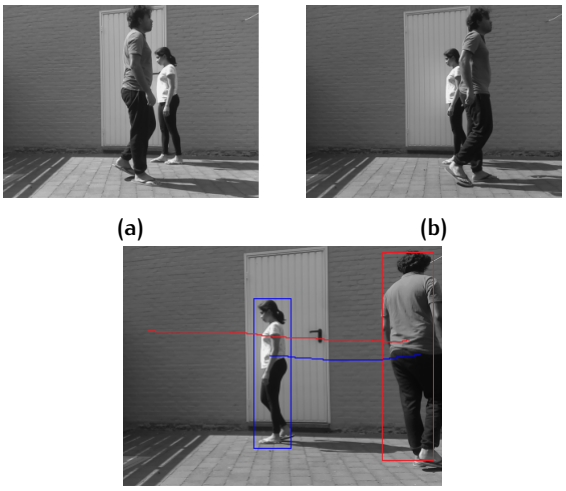


Figure 7.4: (a) and (b) shows the left image frames where 2 objects (of same object type - persons) walk from one side to another; (c) shows the tracking line, that follows the detected objects (persons).

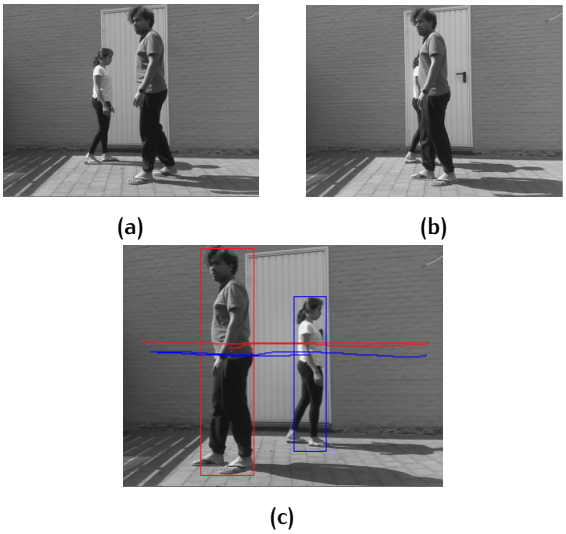
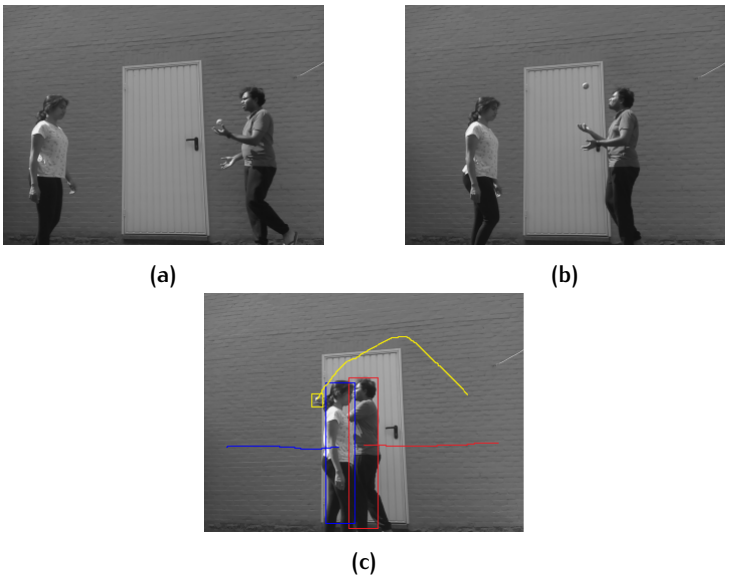


Figure 7.5: (a) and (b) shows the left image frames where 2 objects (of same object type - persons) walks back after reaching one end of the frame; (c) shows the tracking line, that follows the detected objects (persons), from time t=0 (initial point).

7.1.4 Multiple types of Objects

7.1.5 Scenario: Without break in tracking

For this scenario, two objects of same type (person) and an object of a different type (ball) is taken, as seen from figure 7.6. The object (ball) is seen bouncing up and down from the object (person's hand). The figure 7.6c shows that the objects (2 persons) are tracked properly just like it is mentioned in the previous scenario and the ball (object of different type) is tracked just like any other object.



(a) (b)

(c)

Figure 7.6: (a) and (b) shows the left image frames where 3 objects (2 objects of same type - persons and 1 object of ball) walking and throwing the ball over; (c) shows the tracking line, that follows the detected objects (persons and the ball)

Break in tracking

When it comes to tracking the ball in this particular scenario, every time the ball object reaches the other object (person's hand), it is detected as a new ball, instead of considering it as the old ball. Therefore, we see a break in tracking line between the frames when the ball is in person's hand as seen in figure 7.7.

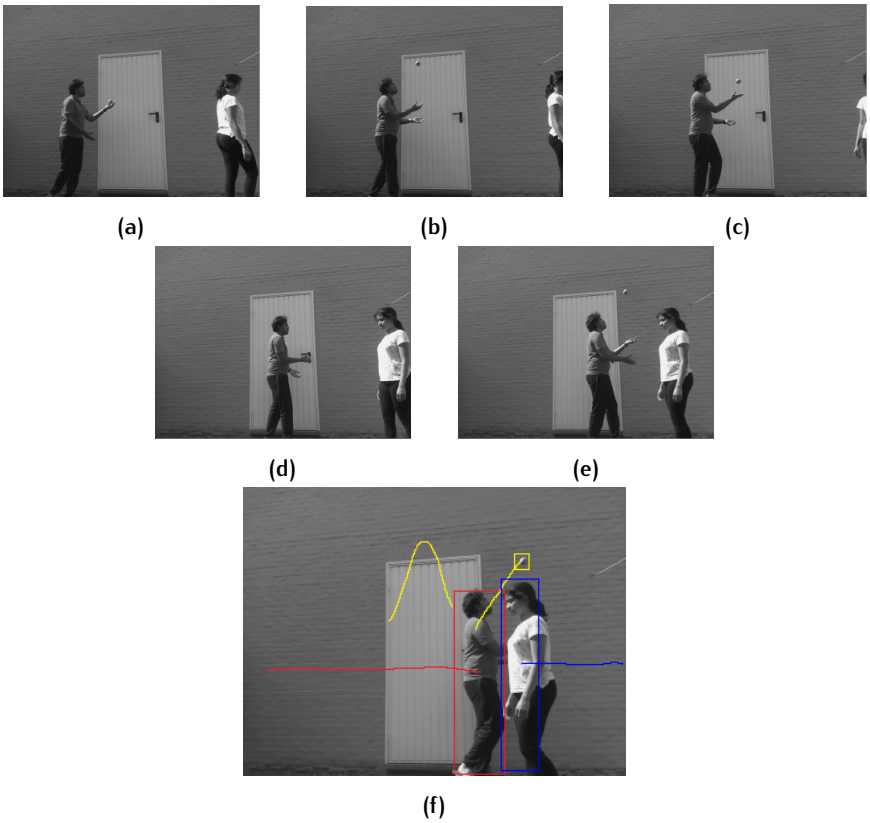


Figure 7.7: (a), (b), (c), (d) and (e) shows the left image frames taken at time t, t+1, t+2, t+3 and t+4 respectively; (f) shows the tracking frame when the ball leaves the hands of the person, without the continuity.

7.2 BENCHMARK INPUT

In this section, the software code is slightly modified to take in images from a directory instead of camera. In this case, the cameras are not used and therefore the following results were obtained using the benchmark images from [GLU12]. And these results were obtained from running the program in a laptop with **Intel i7 6th Gen** processor, with 4 cores and also with **NVidia 960m GPU**. Running the implemented program on such a powerful device can reduce the running time when compared to the run time as seen in table 7.1 which is obtained by running it on raspberry pi.

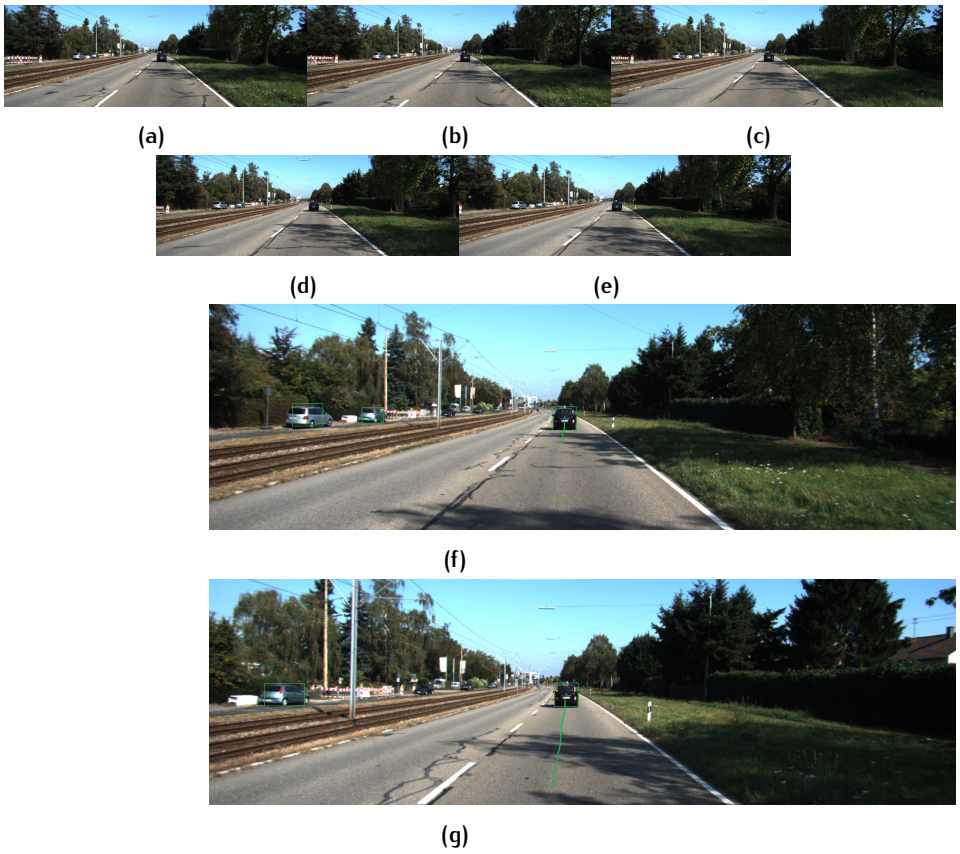


Figure 7.8: (a), (b), (c), (d) and (e) shows the left image frames where (a) corresponds to the 1st frame and so on; (f) shows the tracking frame of the 5th frame and (g) shows the tracking frame of the 15th (last) frame.

The image 7.8f corresponds to the tracking of objects using the 5th frame, where it shows the detection and tracking of 2 cars on a road, where one car is seen moving and another car is seen standing still. This result shows that the still car has no tracking line since it is not moving. The figure 7.8g corresponds to the tracking of objects using the 15th frame (last frame), where it shows the detection and tracking of only one car, since the car moved out of the frame.

7.3 CONCLUSION

The above results were obtained from the board with limited memory and processing power and therefore there were limitations and requirements according to which the algorithm was run. There are as follows:

- Due to limited processing power, the program was able to handle 30 frames per second, where each frame is processed at approximately 30ms as seen from table 7.1. The program was also run for 60 frames per second but the results obtained due to that had huge latency and therefore were not utilized.
- The camera used had a maximum resolution of 1280x480 but the program stopped abruptly while using this. This is due to the fact of availability of limited memory. Therefore the resolution used to obtain proper results was 640x240.
- There shouldn't be any set minimum or maximum number for number of objects that this algorithm can detect and track but due to lack of limited memory, the program was limited to detecting and tracking maximum of only 3 objects.
- At certain points, due to the size of objects, there is a possibility that occlusion occurs. Then the system will stop the tracking abruptly and starts tracking the same object with a break (like in figure 7.7). This can be avoided by having a higher resolution image and higher frames per second. This way, the object can still be detected on the hands of the pedestrian and still be able to track it, in the ideal way as intended.
- When running the developed algorithm on the raspberry pi, the processor gets overheated sometimes, which results to thermal shutdown abruptly in the middle of the process.

8 CONCLUSION

8.1 SUMMARY

This research work is focused on creating a low-cost solution to multiple moving objects detection for land and aerial vehicles. Currently there are a lot of detection and tracking methods being used, which uses various sensor systems. These methods and systems were studied to evaluate how appropriate it would be for this thesis's requirement. After immense research, stereo cameras were used as the sensor system due to the major requirement of low-cost approach.

For the first stage of the process, the algorithm is required to capture the image frames from the camera feed and have to be stored separately as left and right camera images. After which the stored images are used in computing disparity maps to get depth information of the scene.

Following this, the disparity map is used to generate point cloud data on which a ground plane is estimated above which all the objects, that are to be detected, reside. This step is skipped in case of aerial vehicle, due to the absence of ground plane. Image segmentation is performed to classify the regions of images to different classes thereby reducing the area to be processed. The object classes of interest are then extracted as point cloud data which are later tracked using an Extended Kalman Filter based tracker. The final output of this developed algorithm is the detection of objects and tracking them across the image frames.

This whole system is implemented on a Raspberry-Pi 1 compute module attached with a Stereo-Pi board, which is used due to the low-cost approach as seen in figure 8.1. The stereo-Pi board has it own stereo cameras attached, which is used to obtain the image frames for processing.

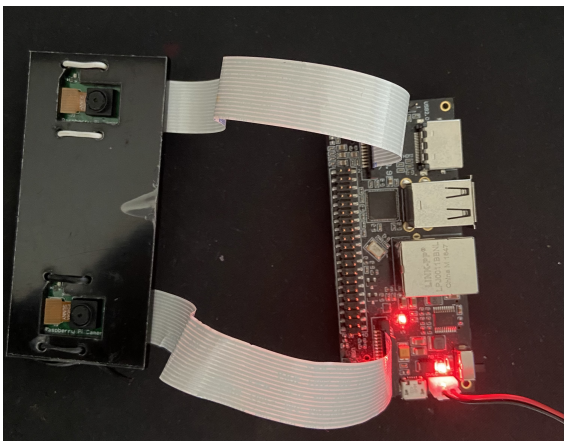


Figure 8.1: The device used for this thesis is the above shown Stereo-Pi with Raspberry-Pi 1 compute module, with the stereo cameras.

8.2 CONTRIBUTIONS

Moving objects detection and tracking is a wide area for a research topic and there are numerous researches happening simultaneously all the time. It would be difficult and a lot of work to develop a new set of algorithms for the current thesis problem statement. In contrast, this thesis makes use of already existing algorithms in an efficient way to develop a low-cost system that runs according to the framed requirements.

Now, addressing the thesis goals mentioned in section 1.2, we answer them as following:

- There are a lot of sensor systems that are currently being used for object detection and tracking. Considering the low-cost approach, stereo cameras were selected to make sure that the program runs without any problem.
- Monocular cameras and stereo cameras were used in a lot of systems for object detection and tracking, but the problem was that a lot of the algorithms were developed on a fast running boards to implement parallel processing. But with respect to this project, the raspberry-pi won't be able to handle parallel processing and therefore had to execute the simplest object detection and tracking algorithms which were developed previously.
- Running the above mentioned algorithm on a low-cost board on a real case scenario, the results were as expected without any error. Although the program had some latency due to low processing power, it can be used a proof of concept to develop the same on a faster board.

This thesis actually works as a set of already developed or existing algorithms. To make them work as expected in a single pipeline, was a tough challenge. The contributions to this thesis are:

- Different algorithms were selected and connected together, to implement a pipeline for object detection and tracking on a raspberry pi board. When working in a real-time environment, the program can run and obtain results without an external display.
- This thesis uses Open CV library to handle the pre-processing stage of this algorithm. Program was developed from scratch to obtain the video from the cameras and process the image frames from left and right cameras. In addition, the program is also used in computing disparity maps from the stored stereo pairs.
- This thesis describes the generation of point cloud from the computed disparity maps. A program was developed to generate the said point cloud data using the point cloud library.
- Following the point cloud generation, a ground plane is fit to reduce the area of processing. A program was developed using the algorithm mentioned in [ZEF16] which uses RANSAC.

- After ground plane estimation, the thesis follows the process of semantic segmentation of point cloud (image segmentation). This was achieved by developing a program that uses Voxel Cloud Connectivity Segmentation from the point cloud library (reference [PASW13]).
- Control flow and data transfer between different algorithms was managed properly.
- During the tracking process, due to the limitation mentioned in 7.3, the developed system is able to detect and track 3 different objects. Therefore, the EKF based tracking program was amended to track the objects in three different colors, instead of having the same colors for all objects.

8.3 FUTURE WORK

The future scope of this thesis remains in using the developed algorithm on a faster board. This thesis uses images of resolution 640x480 and using a better resolution camera will be of great use. Advantages for developing such a solution are:

- Multiple core processor can result in parallel processing, which will result in low latency.
- Higher processing rate can result in processing images quicker than current rate, which can also be advantageous when using high frame rate.
- Higher resolution cameras have the advantage of capturing objects at a far point and this can provide the system enough time to react.
- Using an external peripheral like NVIDIA Jetson Nano module will reduce the overheating of the processor and could also be used for training the data, instead of using the trained dataset.

BIBLIOGRAPHY

Autonomous Driving, Both Close and Far from Ubiquity (https://www.skynettoday.com/editorials/autonomous_vehicles).

Johann Borenstein and Yorem Koren. Real-Time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1179–1187, 1989.

Borja Bovcon, Rok Mandeljc, Janez Perš, and Matej Kristan. Stereo obstacle detection for unmanned surface vehicles by IMU-assisted semantic segmentation. *Robotics and Autonomous Systems*, 104(Figure 1):1–13, 2018.

Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3D object proposals for accurate object class detection. *Advances in Neural Information Processing Systems*, 2015-Janua:424–432, 2015.

Sunglok Choi, Jaehyun Park, Jaemin Byun, and Wonpil Yu. Robust ground plane detection from 3D point clouds. *International Conference on Control, Automation and Systems*, (Iccas):1076–1081, 2014.

Driverless Cars of the 1920s - The Atlantic (https://www.theatlantic.com/technology/archive/2016/06/beep-beep/489029/).

Carlos Diaz-Ruiz, Yan Wang, Wei Lun Chao, Kilian Weinberger, and Mark Campbell. Vision-only 3D tracking for self-driving cars. *IEEE International Conference on Automation Science and Engineering*, 2019-Augus:1029–1034, 2019.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

Andreas Geiger, Julius Ziegler, and Christoph Stiller. StereoScan: Dense 3d reconstruction in real-time. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 963–968, 2011.

Geremy Heitz and Daphne Koller. Learning spatial context: Using stuff to find things. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5302 LNCS(PART 1):30–43, 2008.

Abhijit Kundu, K. Madhava Krishna, and Jayanthi Sivaswamy. Moving object detection by multi-view geometric techniques from a single camera mounted robot. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 4306–4312, 2009.

Object Detection Guide (https://www.fritz.ai/object-detection/#part-basics).

Aljosa Osep, Alexander Hermans, Francis Engelmann, Dirk Klostermann, Markus Mathias, and Bastian Leibe. Multi-scale object candidates for generic object tracking in street scenes. *Proceedings - IEEE International Conference on Robotics and Automation*, 2016-June(3):3180–3187, 2016.

Aljosa Osep, Wolfgang Mehner, Markus Mathias, and Bastian Leibe. Combined image- and world-space tracking in traffic scenes. *Proceedings - IEEE International Conference on Robotics and Automation*, 0:1988–1995, 2017.

OpenCV: Open Source Computer Vision Library (https://github.com/opencv/opencv).

Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Worgotter. Voxel cloud connectivity segmentation - Supervoxels for point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2027–2034, 2013.

Guan Pang and Ulrich Neumann. 3D point cloud object detection with multi-view convolutional neural network. *Proceedings - International Conference on Pattern Recognition*, 0:585–590, 2016.

Point Cloud Library (PCL) (https://github.com/PointCloudLibrary/pcl).

Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). *Proceedings - IEEE International Conference on Robotics and Automation*, pages 9–12, 2011.

Olgierd Stankiewicz, Gauthier Lafruit, and Marek Domański. *Multiview video: Acquisition, processing, compression, and virtual view rendering*, volume 6. 2018.

Yasir Salih and Aamir S. Malik. 3D object tracking using three Kalman filters. *ISCI 2011 - 2011 IEEE Symposium on Computers and Informatics*, pages 501–505, 2011.

Stereo Vision for 3D Reconstruction (https://medium.com/analytics-vidhya/depth-sensing-and-3d-reconstruction-512ed121aa60).

StereoPi (https://stereopi.com/).

Super Voxel Clustering (https://github.com/PointCloudLibrary/ pcl/blob/ master/doc/tutorials/content/supervoxel_clustering.rst).

Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D object proposal generation and detection from point cloud. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:770–779, 2019.

The Carnegie Mellon University Autonomous Land Vehicle Project (https://www.cs.cmu.edu/afs/cs/project/alv/www/index.html).

The Depth I: Stereo Calibration and Rectification (https://python.plainenglish.io/the-depth-i-stereo-calibration-and-rectification-24da7b0fb1e0).

The Evolution Of Drones Timeline (http://www.regimage.org/the-evolution-of-drones-timeline/).

Yuanyuan Wu, Xiaohai He, and Truong Q. Nguyen. Moving Object Detection with a Freely Moving Camera via Background Motion Subtraction. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(2):236–248, 2017.

Kangru Wang, Lei Qu, Lili Chen, Jiamao Li, Yuzhang Gu, Dongchen Zhu, and Xiaolin Zhang. Ground plane detection with a new local disparity texture descriptor. *IEICE Transactions on Information and Systems*, E100D(10):2664–2668, 2017.

Lee Chong Wan, Patrick Sebastian, and Yap Vooi Voon. Stereo vision tracking system. *Proceedings - 2009 International Conference on Future Computer and Communication, ICFCC 2009*, pages 487–491, 2009.

Desheng Xie, Youchun Xu, and Rendong Wang. Obstacle detection and tracking method for autonomous vehicle based on three-dimensional LiDAR. *International Journal of Advanced Robotic Systems*, 16(2):1–13, 2019.

Yang Yu, Wang Tingting, Chen Long, and Zhang Weiwei. Stereo vision based obstacle avoidance strategy for quadcopter UAV. *Proceedings of the 30th Chinese Control and Decision Conference, CCDC 2018*, pages 490–494, 2018.

Ramy Ashraf Zeineldin and Nawal Ahmed El-Fishawy. Fast and accurate ground plane detection for the visually impaired from 3D organized point clouds. *Proceedings of 2016 SAI Computing Conference, SAI 2016*, pages 373–379, 2016.

Ling Zou and Yan Li. A method of stereo vision matching based on OpenCV. *ICALIP 2010 - 2010 International Conference on Audio, Language and Image Processing, Proceedings*, pages 185–190, 2010.

COLOPHON

This document was typeset using LaTeX. The document layout was generated using the arsclassica package by Lorenzo Pantieri, which is an adaption of the original classicthesis package from André Miede.