REAL-TIME LANE DETECTION AND MOTION PLANNING FOR AUTONOMOUS VEHICLE

by

Nadim Ahmed (152419)

Sultanus Salehin (152431)

Tashfique Hasnine Choudhury (152452)

Alfa Rossi (152471)

A Dissertation Submitted to the Academic Faculty for Partial Completion of the Requirements for the Degree of

BACHELOR OF SCIENCE IN ELECTRICAL AND ELECTRONIC ENGINEERING

Department of Electrical and Electronic Engineering
Islamic University of Technology (IUT)
Gazipur, Bangladesh

November 2019

REAL-TIME LANE DETECTION AND MOTION PLANNING FOR AUTONOMOUS VEHICLE

Approved by:

Dr. Golam Sarowar
Supervisor and Associate Professor,
Department of Electrical and Electronic Engineering,
Islamic University of Technology (IUT),
Boardbazar, Gazipur-1704.

Date:

Table of Contents

List of figures

List of Acronyms

OpenCV	Open source Computer Vision
LDWS	Lane Departure Detection System
GPS	Global positioning system
ROI	Region Of Interest
ADAS	Advanced Driver Assistance Systems
CNN	Convolutional Neural Network
NUBS	Non-uniform B-spline
SCARF	Status Certainty Autonomy Relatedness and Fairness
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
ANU	Australian National University
JFC	John F. Canny
MATLAB	Matrix Laboratory
PPHT	Progressive Probabilistic Hough Transform
LQE	Linear Quadratic Estimation
MSE	Mean Square Error
PID	Proportional – integral – derivative
SP	SetPoint

PV	Process Variable
CNC	Computer Numerical Controller
RPM	Revolution Per Minute
RAM	Random Access Memory
BLE	Bluetooth Low Energy
USB	Universal Serial Bus
SD	Secured Digital
HDMI	High Definition Multimedia Interface
GPIO	General Purpose Input Output
DSI	Data Set Interface
NOOBS	New Out of Box Software
OS	Operating System
RGB	Red, Green, Blue
UART	Universal Asynchronous Receiver-Transmitter
ASCII	American Standard Code for International Interchange
PWM	Pulse Width Modulation
CPU	Central Processing Unit
RANSAC	Random Sample Consensus

FPGA	Field Programmable Gate Array

Acknowledgment

We have our wholehearted admiration for the support, assistance and guidance that many people have showered us with throughout our undergraduate effort on countless occasions. We would like to have the honor to express our heartfelt gratitude to our beloved advisor, our guidance Dr. Golam Sarowar sir in the first place for his continuous assistance, inspiration and invaluable suggestions from time to time. Without his support, our relentless work towards our research would not have resulted in success. We would also like to thank the other members of our thesis committee for their contemplative advice. We also wish to express our appreciation to our family members for the support and love throughout our whole life. Above all, our glorification to ALLAH subhanahu wa ta'la, THE ALMIGHTY would always fall short for giving us strength and perseverance in the process of this accomplishment.

Abstract

Our dissertation describes in-depth the algorithm intended to identify lane lines on streets and highways in different conditions. Lane detection enhances the protection of the independent system to some extent. The autonomous or self-sufficient vehicle is an independent device that senses conditions and determines accordingly without any human intervention. Our emphasis was on the use of a vehicle prototype to recognize lanes that could be used later on a standard vehicle. To capture real-time video, PiCamera is embedded with a RaspberryPi 3.0 Model B for processing purposes. RaspberryPi along with battery, motors are mounted in a prototype constructed using CNC machine with sheer perfection. The real-time video captured by Picamera is sufficient enough to evaluate the performance of the algorithms used. The algorithms that have been used are the concepts of OpenCV, Hough transformation, canny edge detection algorithm and elementary algebra to compute and draw the lines. Python 2.7 was used to write the code for the relevant algorithm. The accuracy level of the algorithm used is quite astonishing. Our research works significantly in the field of autonomous vehicles. The comprehensive approach shown here offers a fairly precise and efficient solution for tracking lines. This can make a big difference in the driving experience in every way possible if used properly.

Chapter 1
Introduction

People use their optical vision for vehicle wheeling and dealing during the driving process. The marking of the road lane is a constant reference for the navigation of vehicles. A self-driving car, also known as an autonomous car or a robotic car, is a vehicle that can sense and interpret the environment and make appropriate decisions without needing human feedback or commands. Often known as a driverless car, a self-driving car can drive itself using multiple techniques such as radar, GPS, computer vision, etc. to sense the surroundings [1]. The basis of a driver support system is a vision module detecting road lane boundaries to determine the ego vehicle's position. Several approaches have already been implemented to complete long-distance road tests. The key reason for building smart vehicles is to improve safety conditions by full or partial driving tasks automation [2]. In the field of driving support systems, road detection has played an important role between various tasks, providing information on lane layout and vehicle location relative to the lane. Nevertheless, car crashes continue to be the worldwide leading cause of death and injuries, especially in Asian countries, where tens of thousands of lives are lost and millions are injured annually [3]. The majority of these deaths and injuries were caused on the roads of the country. Several experiments on the robotization of vehicles and driver support systems are carried out. Autonomous cars ' potential advantages include lower system costs, increased security, enhanced consumer satisfaction and vital reduction in automotive accidents. Some also believe that bringing robots to the car would destroy more traditional wrongdoings such as insurance scams and automotive theft in favor of autonomous vehicles[4][5]. Because of the above-mentioned advantages, most countries have leaped forward in getting self-driving cars to the public roads. In 2014, ‘Lutz pod’, a prototype of autonomous vehicle was introduced in UK for the very first time

and later on in 2017,Christchurch airport in New Zealand initiated the first-ever self-driving car[6][7][8].Despite significant benefits, the complete acceptance of self-driven vehicles poses many unforeseen challenges. Some critics claim that widespread adoption of driverless cars would result in a lack of driving jobs and also endanger the safety of passengers. It is known that there may be a lag in allowing autonomous cars on the road by the disparity in people's beliefs of the necessary government interference. To make their driving experience safer and easier, thus, it is important to look for alternatives. Our dissertation provides one such solution: Real-time lane detection for autonomous vehicles using RaspberryPi.

A Lane Detection algorithm for real-time detection of road lane lines is described in this study. The whole hardware set up was done via a CNC machine, a prototype of autonomous vehicle. In this study, the Lane Detection algorithm applies the principles of probabilistic Hough transform, Canny edge detection and Kalman filter to locate the lane lines in the road frames and enhance the accuracy. The algorithms were implemented in Python 3.7.4 with OpenCV.

Chapter 2

Literature Review

2.1 Review of Lane Detection

A vast number of researchers work with the goal of building autonomous vehicles, and many government institutions have also initiated numerous projects around the world. Such activities produced several models and ideas focused on slightly different strategies.

1. P. Mandlik and A.B. Deshmukh have installed a Lane Departure Detection System (LDWS) to alert the driver when the vehicle attempts to leave its lanes. LDWS is based on the lane recognition and tracking algorithm and uses the' Canny Edge Detection' and' Hough Transform' OpenCV implementations to discern vehicle lane departure on a Raspberry Pi. The test was done using a toy vehicle prototype and USB camera, Intex IT-305WC webcam. Hough transform was used and the result collection was done via Intel Core i3 1.80 GHz processor. [9]
2. At Universitat der Bundeswehr M¨unchen researchers used road as a clothoid for lane detection. Several camera modules were used for video capturing and better focusing.[10]
3. Clanton et al introduced a multi-sensor lane departure warning strategy that used a roadmap with a GPS receiver to sense road surfaces.[11]
4. In [12], the researchers proposed an approach using a lidar with a monocular camera that could track the lane in an area in real-time. The lidar and sensors directly retrieved the information from

the environment, and the instruments are not based on the weather conditions that are the key advantages of these approaches.

5. As far as the work performed using vision-based techniques is concerned, the methodology in [13] obtained the likely results on the dataset of the Caltech path. They also implemented primarily three-lane detection strategies. This methodology often involves the configuration of the design, but here the system's performance and productivity are abandoned.
6. Lee and Moon [14] suggested a real-time lane detection algorithm with a Region of Interest (ROI) capable of working in a shorter time with a low level of noise and response. For lane tracking operation, the system used the Kalman filter and a minimum square approximation of linear flow. The machine also identifies the lane and monitors the road.
7. Song et al.[15] developed a system capable of detecting and classifying the lane using the principle of stereo vision for Advanced Driver Assistance Systems (ADAS) core. They suggested a model for detecting the lane using the concept of Region of Interest (ROI) and used the architecture of the Convolutional Neural Network (CNN) to identify the right or left lane with the KITTI dataset. However, because the difference was noisy, the device could not distinguish the lanes.
8. Yoo et al.[16] developed a lane detection technique based on the approximation of the vanishing point. The program initially used the probabilistic voting technique to identify the lane segment vanishing points. The actual segments of the lane are calculated by setting the disappearance threshold as well as the line direction.

9. Jung et al.[17] used spatiotemporal images obtained from the video to establish a lane marking modality. In order to detect lanes, the Hough transform is applied to the collected frames. For short-term noises such as mislaid lanes or vehicle interference, the device is very successful.
10. A.A.M. Assidiq et al. [18] had a vision-based approach to lane detection to tackle sometimes changing lighting and shadow conditions. To remove the lane lines, a few hyperbolas are connected to the edges of the lane with Hough Transform. It was also reported that the proposed lane detection system could be used as curved and straight roads on both painted as well as unpainted roads.
11. Q. Truong and B.R. Lee [19] used the key method to identify street and lane boundaries using the vehicle's vision-based system. For the construction of road lane boundaries, a vector-lane-concept and non-uniform B-spline (NUBS) interpolation method is used. The pictures were taken using a monocular lens for experimental purposes. As described in the journal, the experimental results are based on real-world road pictures.
12. The new tracking system for autonomous vehicle navigation guided by vision was developed by Yuan et al. [20]. The likelihood between all particles with edge pictures is determined using an algorithm for ' particle sorting, ' which then estimates the three parameters of the real road line state. This approach is checked for reliable results in real road pictures.
13. The researcher used a linear filter piece by piece to detect lanes in reference [21], but it gives little false alarm. In the presence of clamor, shadows, lane paint absence and various environmental conditions, the algorithm is powerful.
14. The researcher used local gradient characteristics to detect lane in a referral [22], but this approach produces more false alarms and poor performance.

15. The writer [23] used the extraction and grouping of features together with a Hough transformation, but it does not identify a high traffic lane and confusing street textures.
16. The researcher has used the segmentation method for the lane in reference [24] but shows a high false-positive rate. Hough Transform senses straight and curve roads and shows good quality under different conditions of illumination.
17. In the case of a homogonally colored street, the SCARF [25] and the MOSFET [26] device used color camera data. In the earlier method, the Bayesian classification was appraised for the determination of the road surface, while in the latter the color segmentation was used for deciding lane markings.
18. A single-camera design is the basis of Australian National University (ANU), [27]. To determine the state of the vehicle the vision module uses different image filter data. A collection of hypotheses produced by a probabilistic particle filter describes possible system states.

2.2 Research Objective and Outline

The main objective of our research work is to ease the process to an extent. Considering the present constraints in this field as well as the merits of self-driving vehicles we had some objectives before we kicked off our undergraduate research work. Thus the objectives are:

1. Using Computer Vision for Lane Detection: The goal was to use a technique named canny edge detection, Hough Transform for detecting lane lines in road to figure out how real-time digital videos can be used to achieve a broad level of understanding.

2. Building a Hard Real-Time system: The need to meet deadlines characterizes a real-time system. In hard real-time systems, missing deadlines can be very risky in real-time situations, process control is one such example. We wanted our work to be as real-time as possible.

3. Consistent output for better efficiency: In addition to keeping performance parameters satisfactory, we worked hard on getting consistent output and used Kalman filter and Past accumulated Average method for eliminating any delay.

4. Hardware Implementation of our research work: We built a prototype of an autonomous vehicle using plywood which will be elaborately talked about later on, for implementing our proposed methodology.

Chapter 1 provides an introduction to our dissertation. Chapter 2 is comprised of a Literature review that includes the current knowledge, including substantive findings, and input on a particular subject theoretical and methodological. Chapter 3 talks about the basics of Lane detection and motion planning algorithms used in our research in detail which includes Canny Edge detection, Hough transformation, segmentation, Kalman filter and PID controller. In chapter 4 our hardware and software implementation is described elaborately. The overall methodology is exquisitely demonstrated in chapter 5. Chapter 6 exhibits the experimental results and comparison among different methods. The conclusion of current work is found in Chapter 7. It provides information about a future query to improve the performance of the methods proposed.

Chapter 3

Lane Detection and Motion Planning Algorithms

3.1 Canny Edge Detection

Edges: Defined:

Edges represent boundaries and are thus a fundamental issue in image processing. In pictures, edges are areas with clear contrasts in color – a change of brightness from pixel to pixel. The general purpose of edge detection is to decrease the amount of information in an image substantially while maintaining the structural properties used for additional image processing. It comprises a set of computational methods designed to identify points in a digital image where illumination of the image changes drastically or has discontinuities in more formal terms. In general it is organized to a set of curved segments called edges at which image brightness changes sharply. Edge detection, particularly in feature detection and functional extraction, is a fundamental tool in image processing, machine vision and computer vision [28].

There are several algorithms available, and we focus on one that John F. Canny (JFC) developed in 1986. Canny Edge Detector is a multi-stage algorithm used to detect a large variety of image boundaries. Its founder John F. Canny, also developed an edge detection computational theory to explain why the technique works. Although it is quite old, it has become one of the standard methods for edge detection and is still used in analysis. This is also known as the optimal edge detector. [29]

Basis of the Canny Edge Detector:

The goal for Canny was to improve the image with many edge detectors. Canny found that the edge detection criteria for various vision systems were fairly similar. The target of JFC was to develop an optimal algorithm for the criteria mentioned below:

- Criterion 1: Identification :

  It is necessary to maximize the possibility of detecting real edges while minimizing the risk of incorrectly detecting non-edge points. The initial criteria must have a low error rate and useless data should be filtered out while the useful information is retained. It means that as many edges in the image as possible should be accurately identified and the signal-to-noise ratio should be maximized.

- Criterion 2: Limiting :

  The operator's edge point should be precisely positioned at the center of the edge which means the edges identified should be as close to the real edges as possible. We need to keep the difference between the main image and the produced image as small as possible.

- Criterion 3: Avoiding False edges :

  The third criteria exclude multiple edge responses. A provided edge in the image should be marked only once, and noise should not give false edges where possible.

In brief, the Canny edge detector first smoothes the picture to remove noise based on these parameters. It then uses the object gradient to illustrate regions with large space derivatives. After that the algorithm measures these regions and uses non-maximum suppression to remove those pixels that are not at the peak [30]. To avoid streaking or to thin the edges the gradient set is now further decreased by hysteresis. Canny used the calculus of variations to meet these requirements – a method that identifies the function that optimizes a specific function. Canny edge detection algorithm is one of the most strictly defined methods that provide good and reliable detection among the edge detection methods that have been developed so far [31]. It became one of the most common algorithms for edge detection due to its optimality to satisfy the three requirements for edge detection and the ease of method for implementation. [32]

Canny edge detection algorithm:

The Canny edge detection algorithm can be divided into five different steps:

Step 1: Gaussian Blur

Step 2: Finding Gradients

Step 3: Non-Maximum Suppression

Step 4: Double Thresholding

Step 5: Hysteresis Edge Tracking

The following parts explain every step -

3.1.1. Step 1: Gaussian Blur:

This step can be subdivided into two more steps. These are-

- Grayscale Conversion
- Image Blurring using Gaussian Filter

First the image is needed to be converted to grayscale. Grayscale is a collection of black-to-white monochromatic colors. A black-scale picture therefore only includes shades of gray and no light. Although digital images can be stored as images of a grayscale (or black and white), still color images contain information of a grayscale. [33]

All images taken from a camera may necessarily involve a certain amount of noise. To prevent the noise from being confused for edges it is necessary to reduce noise. By using a Gaussian filter, the picture is first smoothed [34]. In MATLAB the intensity values of the pixels are 8 bit and range from 0 to 255. [35]. After RGB to graysacle converision, the image looks like this-

Fig. 3.1 Grayscale image [36]

The primary step to canny edge detection needs some method of filtering out noise which can be found, then the preservation of the useful image is assured. Convolution is one of the simplest methods for many common image-processing operators[36]. This is performed by masking over a grayscale image starting from the top left corner through the entire image within boundaries. Each value is multiplied with the cell value of kernel. After that, all the values are added together. [37]

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

Fig. 3.2 An example of an image

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

Fig. 3.3 An example of a kernel

Mathematically it can be expressed :

$$O(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(i+k-1, j+l-1)K(k,l)$$

A Gaussian filter kernel is used for smoothing purposes. The kernel of a Gaussian filter with a standard deviation of σ = 1.4 is shown in Equation [38]. The equation for a Gaussian filter kernel of size $(2k+1)\ x\ (2k+1)$ is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2} exp^{\left(-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}\right)}; 1 \leq i,j \leq (2k+1)$$

An example of 5 x 5 Gaussian filter is provided here with a value of σ = 1.4. The increasing standard deviation blurs the intensity of the noise.

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

The size of the Gaussian filter kernel will work as a variable of the performance of the detector. As the size of the filter increases the signal to noise ratio reduces. Moreover, as the size increases localization error for edge detection increases as well. [39]

Fig. 3.4 Smoothed Image(Blurred)[36]

3.1.2. Step 2: Finding Gradient:

This step can be divided into two sub-steps.

- Sobel Operation & Finding Gradient angle
- Tracing the edge using the angle

Sobel Operation & Finding Gradient angle:

The Canny algorithm actually determines edges of the places where the grayscale intensity of the image varies the most. Those portions can be found by observing the gradients. This is where Sobel operator comes into action. After blurring and noise elimination, determination of gradient has to be done for recognizing the edge strength. A 2-D spatial gradient measurement on an image can be done by using a Sobel operator. This operator uses a pair of 3x3 convolutional matrices one of which estimates the gradient in x-direction (columns) and the other in the y-direction (rows) [40]. Sobel Gx and Gy for approximating the gradient in the x- and y-direction respectively by applying the kernels are shown below:

$$K_{Gx} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$K_{Gx} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Edge strength can be found from gradient magnitude as a Euclidean measure by applying Pythagoras theorem. Simplification of this outcome can be made by applying Manhattan distance.The gradient magnitude is as follows:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

$$|G| = \left|G_x^2\right| + \left|G_y^2\right|$$

Fig. 3.5 Gradient Magnitude[36]

Tracing the edge using angle:

An image of the gradient magnitudes indicates the edges most of the time.But, the edges are quite wide and so do not indicate the exact placement of the edges[41]. For the improvement of this situation edge direction should be determined by the following equation:

$$\theta = \arctan\left(\frac{|G_y|}{|G_x|}\right)$$

After determining edge direction it should be related to a direction that can be tracked down in an image. So for the calculation of the angle of edge, the matrix size is proportional to the angle if a 5x5 matrix is used.

x	x	x	x	x
x	x	x	x	x
x	x	a	x	x
x	x	x	x	x
x	x	x	x	x

If we look at ‘a’ in the middle, there are four possible directions when describing the pixels around it:

1. 0 degrees (in the horizontal direction)

2. 45 degrees (along the positive diagonal)

3. 90 degrees (in the vertical direction) or 135 degrees (along the negative diagonal)

4. 180 degrees region is just a mirror region of 0 degrees region.

So, all the edge direction calculated will automatically be round up to the closest angle.

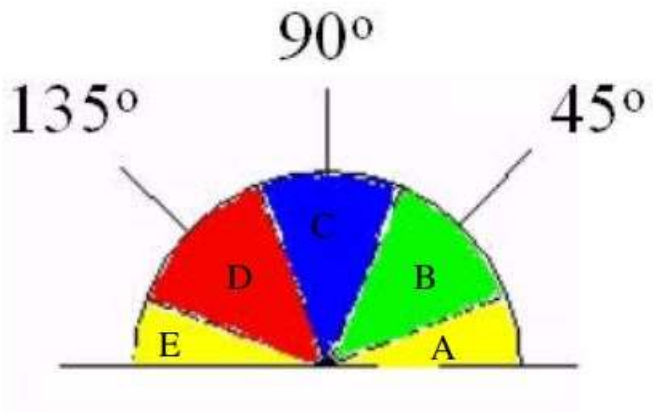

Fig. 3.6 Angle Tracing

Now, edge direction within A & E is round off to 0 degrees (0 to 22.5 & 157.5 to 180 degrees). Edge direction in B is round of to 45 degrees (22.5 to 67.5 degrees), Edge direction in region C is set to

90 degrees (67.5 to 112.5 degrees) and finally Edge direction in the region D is set to 135 degrees (112.5 to 157.5 degrees).

3.1.3. Step 3: Non-Maximum Suppression

This is the step where we sharpen the edges which were blurred. This is achieved by keeping all local maxima in the gradient image and canceling out everything else [42]. The algorithm can be demonstrated in the following manner—

- When pixel q has a larger value of intensity than p and r, the pixel with the maximum value in an edge can be found.
- If this condition is valid, then we preserve the pixel, otherwise we set the pixel to zero and eliminate it (make it a black pixel).
- Non-maximum suppression can be achieved by interpolating the pixels for greater accuracy:

$$r = \alpha b + (1 - \alpha)a$$

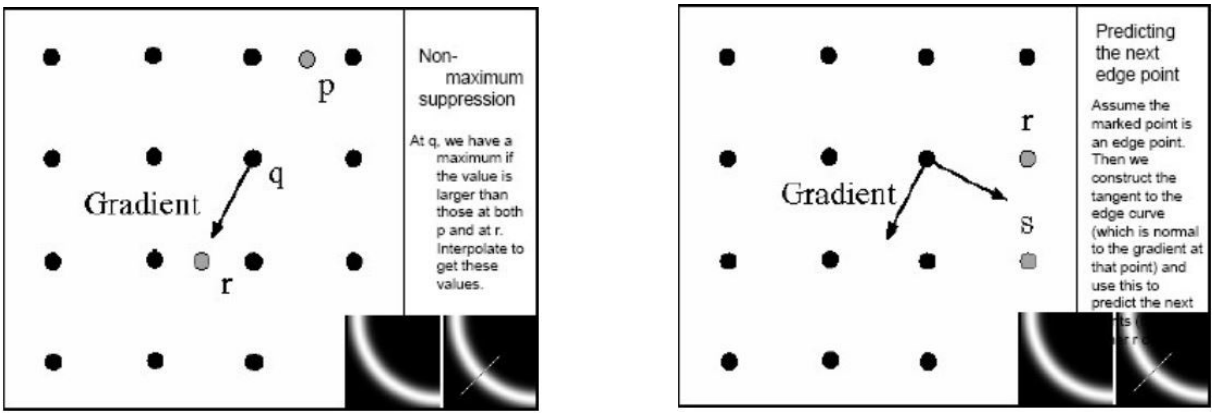

Fig. 3.7 Non Maximum Suppression

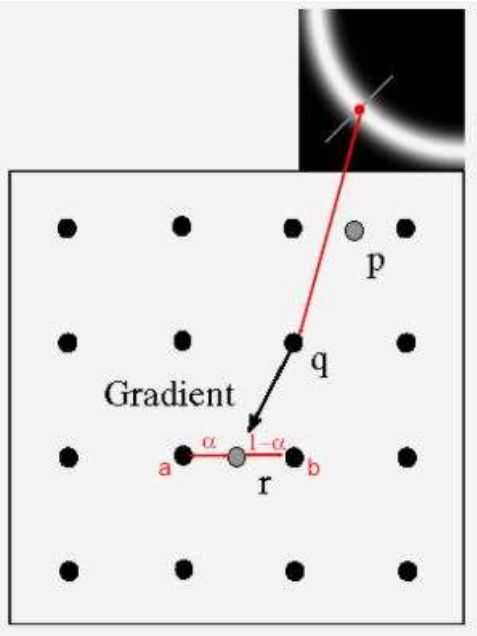

Fig. 3.8 With interpolation[36]

Fig. 3.9 Non-Maximum Suppression with Interpolation[36]

3.1.4. Step 4: Double Thresholding

After step 4, the remaining edge pixels are marked with strength pixel by pixel. Even though we would find most of the edges to be true/valid, some of them might be caused by color difference or any other noise. These two cases can be distinguished in a simple way. If we use a threshold, the edges with a higher value than threshold will be kept and others will be eliminated. The canny edge algorithm introduces two threshold values entitled High Threshold and Low Threshold.

Edges with the value higher than High Threshold are categorized as strong, and edges having a value lower than Low threshold are eliminated. Edges having value in between High and Low threshold are marked as weak. The next step will make us clear how we can find which weak edge is an acceptable edge [43]. The image that is found after double thresholding is as follows:

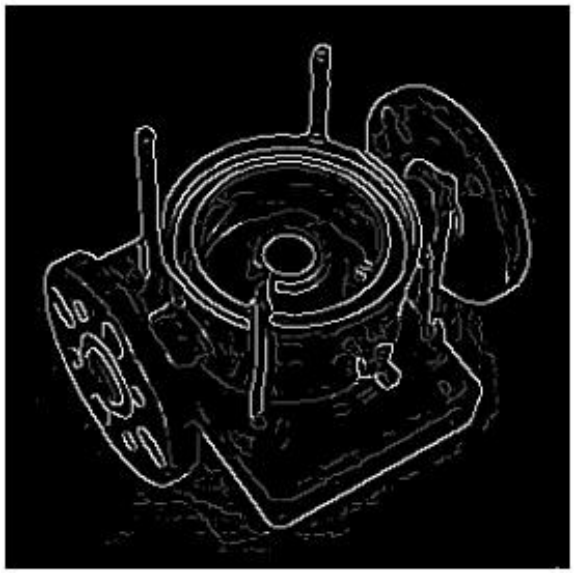

Fig. 3.10 After Double Thresholding[36]

3.1.5. Step 5: Hysteresis Edge Tracking

This step can be demonstrated in two parts. Edge tracking by hysteresis and the final stage is cleaning up. Strong edges are tagged as ‘certain edges’ and can be included in the final edge image right away. The algorithm continuously tracks both the weak and strong edges so that these edges can recursively be iterated through the strong edges and figure out whether there are linked weak edges rather than having to iterate through each pixel in the image.

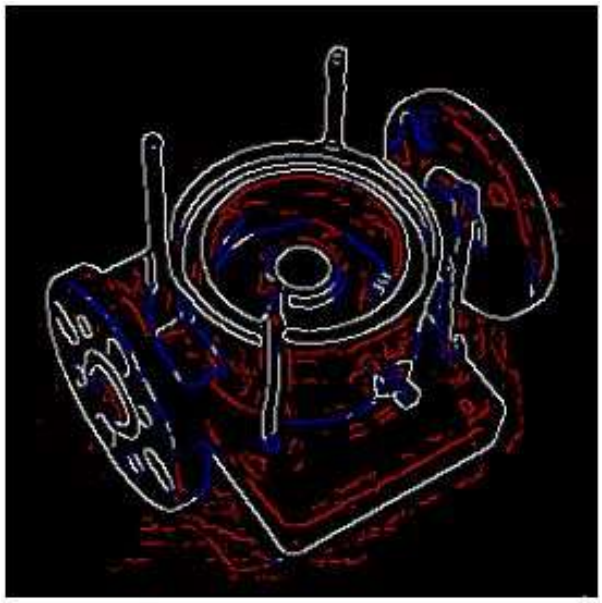

Fig.3.11 Edge Tracking by hysteresis[36]

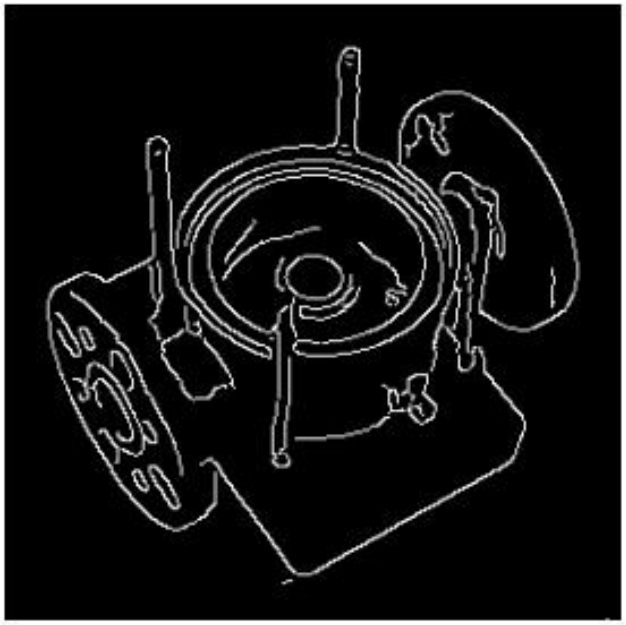

Fig. 3.12 Final Output

At last, the remaining weak edges are iterated through and set to zero.

3.2 Hough Transformation:

Hough transform is a method for extracting features to identify simple shapes in a frame or image such as circles, lines, etc [43]. Only a few parameters can describe a simple shape. As for example, two parameters (slope, intercept) can represent a line, and for a circle — the center coordinates and the radius (x, y, r). Hough transform does a good job of finding these forms in a picture or a video frame.[44]

The purpose of the technique is to locate imperfect instances of objects within a certain category of types by means of a voting procedure. This voting process is performed in a parameter space from which object candidates are extracted as local maxima in a so-called accumulator space that is directly constructed by the Hough transform computing algorithm.[45]

The set of all straight lines in the picture plane forms a family having not more than two parameters. If we fix the parameters for the family, then a random straight line can be represented only by a single point in the parameter space. For such obvious reasons normal parameterization is preferred. As shown in Figure 1, this parameterization specifies a straight line by the angle θ of it's perpendicular and its algebraic distance ρ from the origin (0, 0) [46]. The equation associated with this sort of line is-

$$xcos\theta + ysin\theta = \rho$$

If θ is restricted to the interval [0, π], then we get unique normal parameters for a line. So, with this constraint every line in the *x*-*y* plane indicates a unique point in the θ-ρ plane.

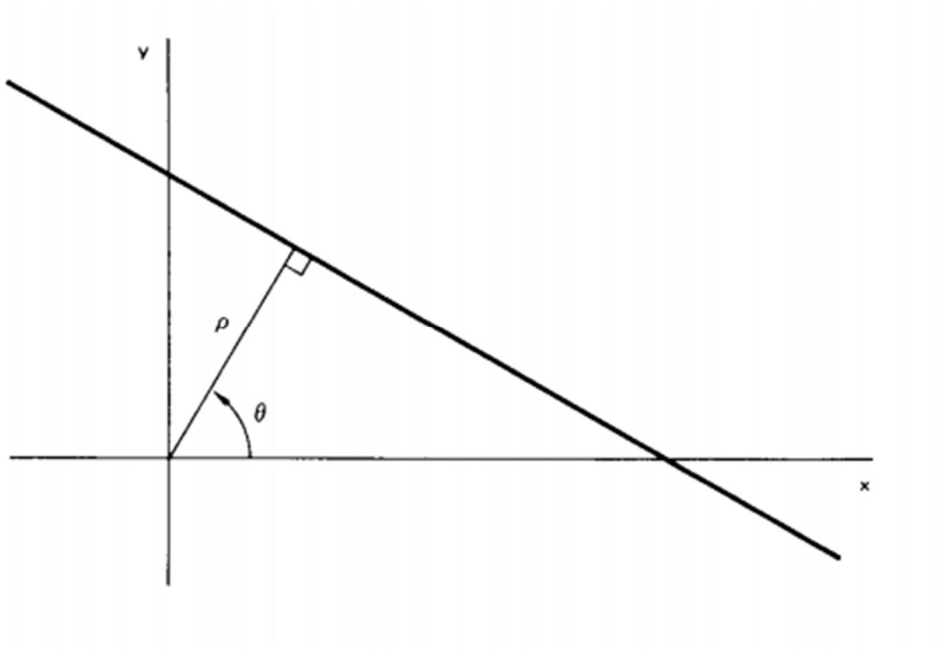

Fig. 3.13. The normal parameters for a line.

When we pick a (ρ, θ), it corresponds to a line in a 2D space which has ρ and θ as its parameters. There are a location and position on this 2D space for every edge point and this space is also termed as an accumulator.

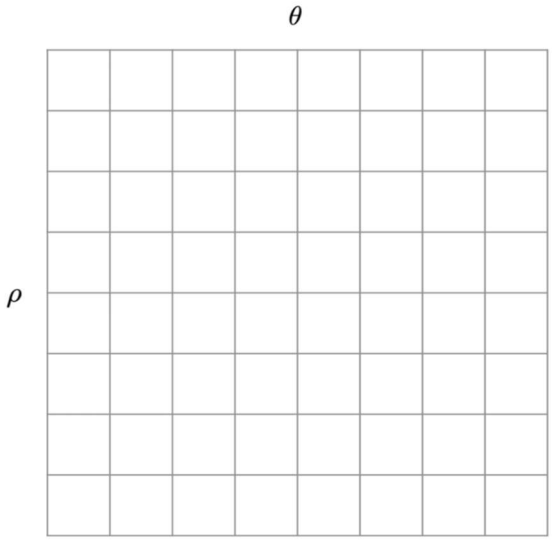

Fig. 3.14 Accumulator

The top left cell indicates (-R, 0) and the bottom right indicates (R, π).

Algorithm:

The algorithm includes several steps. The steps are:

1. Edge detection using canny edge detector
2. Initializing Accumulator and mapping the edge points to Hough space
3. Interpretation of the accumulator to get infinite length lines
4. Conversion of finite length lines from the infinite ones

3.2.1. Initializing Accumulator and Mapping the edge points to Hough space:

Hough transform problem determines the number of dimensions of the accumulator corresponding to the number of unknown parameters. So, for straight lines, there will be two dimensions (r and θ). Therefore, the content of the accumulator can be visualized. Binary edge map is taken as input by the Hough transform and it attempts to locate edges placed as straight lines. All the edge points in the edge map are converted to all possible lines passing through that point. Figure 3 shows this for two such points. An edge map generally contains a lot of points, but the line detection concept remains the same as shown in the figure for two points. Every edge point is converted to a line in the Hough space, and the places where most Hough space lines intersect each other are taken to be true lines in the edge map.[47]

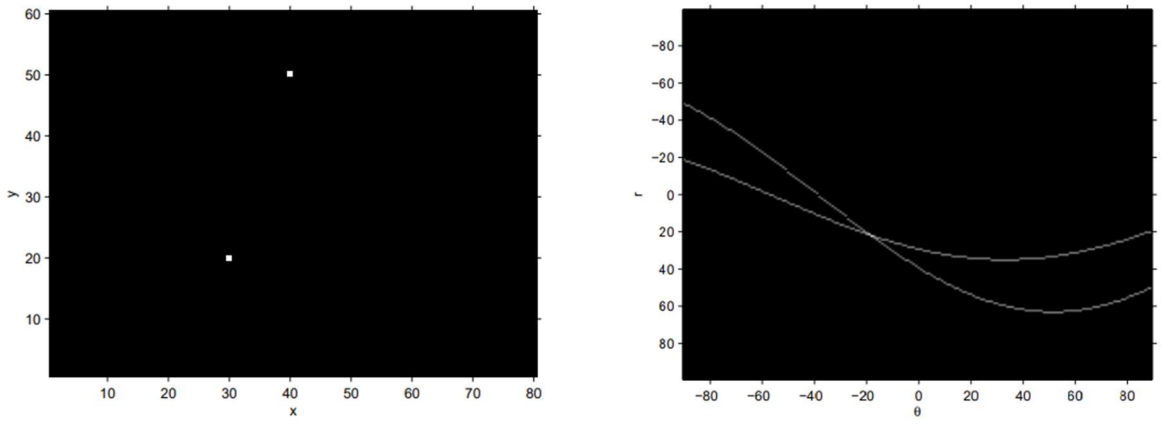

Fig. 3.15 Transformation of two points to two lines in the Hough Space

3.2.2. Interpretation of the accumulator to get infinite length lines:

Once all edge points have been transformed in the edge map, infinite lines are detected by the accumulato r interpretation. Setting some threshold value for the accumulator is the easiest or simplest way to detect li nes and consider all values above the threshold as a line. A simple threshold may be used but it is most likely to detect several (almost similar) lines for each true line. To mitigate this, a suppression neighborhood can be set, so that two lines must be significantly different before both are detected. An example is shown in figure 4 below.[48]

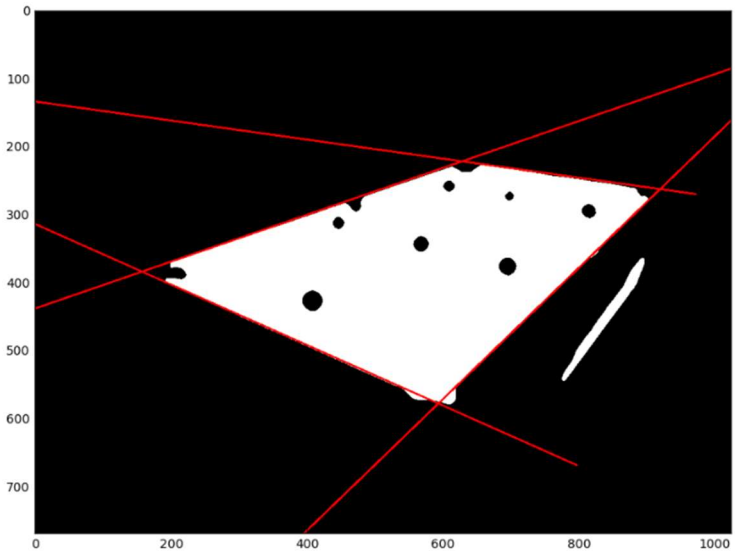

Fig. 3.16 Hough lines of infinite length

3.2.3. Conversion of finite length lines from the infinite ones:

The general Hough transformation detects lines given only by the parameters ρ and θ and caries no length information with it. So, only infinite length lines are identified. If finite lines are required, some extra analysis must be made to identify the areas of the image that has a contribution to each line. For doing so there are several algorithms. One such algorithm stores coordinate information

for all the points in the accumulator, and this information is thus used to set a limit for the lines. But this causes the accumulator to use a lot of memory. Another algorithm is to keep searching along the lines of infinite length in the edge image to finally get the required lines having finite length. An example is shown in figure below.

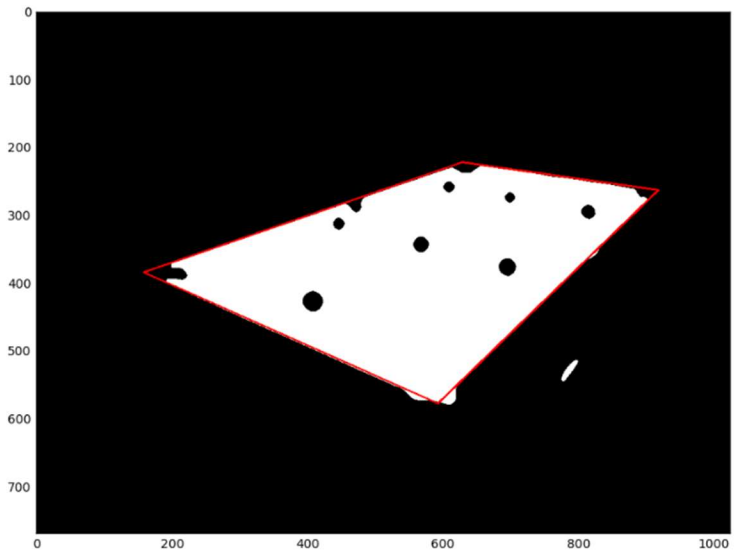

Fig. 3.17 Hough lines of finite length.

Besides, for faster implementation of the algorithm Progressive Probabilistic Hough Transform (PPHT) [49] can be used. The main concept of this idea is to convert arbitrarily selected points in the edge image into the accumulator. For a particular infinite line when a bin in the accumulator has achieved a certain number of votes, the edge image is looked along that line to check if one or more finite line(s) are present. Then all points or pixels on that line are erased from the edge image. So, finite lines are returned.

3.3 Kalman Filter

Kalman filter [50] is a classical algorithm that has its application in array of topics such as missile guidance, navigation, and control of vehicles, particularly aircraft, spacecraft and dynamically positioned ships [51]. Originally developed by Rudolf E. Kálmán, it is also known as linear quadratic estimation (LQE). This algorithm takes a series of measurements observed over time, including statistical noise and other, often Gaussian, inaccuracies, and generates estimates of unknown variables which appear to be more reliable than those based on a single measurement alone by estimating a specific distribution of probabilities over variables for each time frame [52]. For many monitoring and data forecasting activities, Kalman filter has long been considered the optimal solution. Its use was widely recorded in the study of visual movement.

3.3.1.Kalman Filter Derivation

Using maximum probability statistics, a more robust derivation can be established. It is achieved by refining the goal of finding the $\hat{x}$ which maximizes the probability or likelihood of y [53]. That is;

$$\max [\mathrm{P}\ (\mathrm{y}|\hat{x}) \tag{3.1}$$

Assuming that the additive random noise is Gaussian distributed with a standard deviation of σ_{k} gives;

$$\mathrm{P}\ (\mathrm{y_k}|\hat{x}_k) = \quad K_k \exp \ - \quad (\frac{(y_k - a_k\hat{x}_k)^2}{2\sigma_{\mathrm{k}}^2}) \tag{3.2}$$

where K_k is a normalization constant. The maximum likelihood function of this is:

$$\mathrm{P}\,(\mathrm{y_k}|\hat{x}_k) = \Pi_k \; K_k \exp \; - \; (\frac{(y_k - a_k\hat{x}_k)^2}{2\sigma_\mathrm{k}^2}) \tag{3.3}$$

Which leads to:

$$log\mathrm{P}\,(\mathrm{y_k}|\hat{x}_k) = -\frac{1}{2}\sum_k \left(\frac{(y_k - a_k\hat{x}_k)^2}{\sigma_\mathrm{k}^2}\right) + \quad constant$$

The driving function of the equation above is the Mean Square Error, which may be maximized by the variation of $\hat{x}_k$. Therefore, the mean squared error function is applicable when the expected variation of $\mathrm{y_k}$ is best modeled as a Gaussian distribution. In such a case the MSE serves to provide the value of $\hat{x}_k$ which maximizes the likelihood of the signal$\mathrm{y_k}$.

Kalman described his filter using state-space techniques, which allows the filter to be used either as a smoother, filter or predictor. The latter of these three, the Kalman filter's ability to predict data, has proved to be a very useful feature. It has resulted in the application of the Kalman filter to a wide range of tracking and navigation issues. Define the filter in terms of state-space methods often simplifies filter implementation in the discrete domain, which is another reason for its widespread appeal.

3.3.2. State Space Derivation

Assume that we want to know the value of a variable within a process of the form;

$$x_{k+1} = \; \Phi x_k \; + \omega_k \tag{3.4}$$

Where x_k is the state vector of the process at time$k$, (nx1), Φ is the state transition matrix of the process from the state at k to the state at $k+1$, and is assumed stationary over time, (nxm). ω_k is the associated white noise process with known covariance, (nx1).

Observations on this variable can be modeled in the form:

$$z_k = Hx_k \ + v_k \tag{3.5}$$

Where z_k is the actual measurement of x at time k ,(mx1), H is the noiseless connection between the state vector and the measurement vector, and is assumed stationary over time, (mxn). v_k is the associated measurement error. This is again assumed to be a white noise process with known covariance and has zero cross-correlation with the process noise, (mx1). For the minimization of the MSE to yield the optimal filter it must be possible to correctly model the system errors using Gaussian distributions. The covariances of the two noise models are assumed stationary over time and are given by;

$$Q \ = E[\omega_k \omega_k^T] \tag{3.6}$$

$$R = e[v_k v_k^T] \tag{3.7}$$

The mean squared error is,

$$€_t(t) = E(e_k^2) \tag{3.8}$$

This is equivalent to;

$$E[e_k e_k^T] = P_k \tag{3.9}$$

Where; P_k is the error covariance matrix at the time, (nxn).

Equation 3.9 may be expanded to give;

$$P_k = E[e_k e_k^T] = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] \tag{3.10}$$

Assuming the prior estimate of $\hat{x}_k$ is called $\hat{x}'_k$, and was gained by knowledge of the system. It is possible to write an update equation for the new estimate, combing the old estimate with measurement data thus;

$$\hat{x}_k = \hat{x}'_k + K_k(z_k - H\hat{x}'_k) \tag{3.11}$$

where; K_k is the Kalman gain, which will be derived shortly. The term $(z_k - H\hat{x}'_k)$ in eqn. (3.11) is known as the innovation or measurement residual, i_k .

Substitution of 3.5 into 3.11 gives;

$$\hat{x}_k = \hat{x}'_k + K_k(Hx_k + v_k - H\hat{x}'_k) \tag{3.12}$$

Substituting 3.12 into 3.10 gives;

$$P_k = E\ [[(I - K_k H)(\hat{x}_k - \hat{x}'_k) - K_k v_k][(I - K_k H)(\hat{x}_k - \hat{x}'_k) - K_k v_k]^T] \tag{3.13}$$

At this point it is noted that $(\hat{x}_k - \hat{x}'_k)$ is the error of the prior estimate. It is clear that this is uncorrelated with the measurement noise and therefore the expectation may be re-written as;

$$P_k = (I - K_k H)E[(\hat{x}_k - \hat{x}'_k)(\hat{x}_k - \hat{x}'_k)^T](I - K_k H) + K_k E[v_k v_k^T]K_k^T \tag{3.14}$$

Substituting equations 3.7 and 3.10 into 3.13 gives;

$$P_k = (I - K_k H)P'_k(I - K_k H)^T + K_k R K_k^T \tag{3.15}$$

Where P'_k is the prior estimate of P_k.

Equation 3.15 is the error covariance update equation. The diagonal of the covariance matrix contains the mean squared errors as shown;

$$P_{kk} = \begin{bmatrix} E[e_{k-1}e_{k-1}^T] & E[e_k e_{k-1}^T] & E[e_{k+1}e_{k-1}^T] \\ E[e_{k-1}e_k^T] & E[e_k e_k^T] & E[e_{k+1}e_k^T] \\ E[e_{k-1}e_{k+1}^T] & E[e_k e_{k+1}^T] & E[e_{k+1}e_{k+1}^T] \end{bmatrix} \tag{3.16}$$

The sum of the diagonal elements of a matrix is the trace of a matrix. In the case of the error covariance matrix the trace is the sum of the mean squared errors. Therefore, the mean squared error may be minimized by minimizing the trace of P_k which in turn will minimize the trace of P_{kk}.

The trace of P_k is first differentiated with respect to K_k and the result is set to zero in order to find the conditions of this minimum.

Expansion of 3.15 gives;

$$P_k = P_k' - K_k H P_k' - P_k' H^T K_k^T + K_k (H P_k' H^T + R) K_k^T \tag{3.17}$$

Note that the trace of a matrix is equal to the trace of its transpose, therefore, it may be written as;

$$T[P_k] = T[P_k'] - 2T\,[K_k H P_k'] + T[K_k (H P_k' H^T + R) K_k^T] \tag{3.18}$$

where; $T[P_k]$ is the trace of the matrix P_k.

Differentiating with respect to K_k gives;

$$\frac{dT[P_k]}{dK_k} = -2(HP_k')^T + 2K_k(HP_k'H^T + R) \tag{3.19}$$

Setting to zero and re-arranging gives;

$$(HP_k')^T = K_k(HP_k'H^T + R) \tag{3.20}$$

Now solving for K_k gives;

$$K_k = P_k'H^T(HP_k'H^T + R)^{-1} \tag{3.21}$$

Equation 3.21 is the Kalman gain equation. The innovation, i_k has an associated measurement prediction covariance. This is defined as;

$$S_k = HP_k'H^T + R \tag{3.22}$$

Finally, substitution of equation 3.21 into 3.17 gives;

$$P_k = (I - K_kH)P_k' \tag{3.23}$$

Equation 3.23 is the update equation for the error covariance matrix with optimal gain. The three equations 3.11, 3.21, and 3.23 develop an estimate of the variable x_k . State projection is achieved using;

$$\hat{x}_{k+1} = \Phi x_k \tag{3.24}$$

To complete the recursion it is necessary to find an equation that projects the error covariance matrix into the next time interval, k+ 1. This is achieved by first forming expressions for the prior error:

$$e'_{k+1} = x_{k+1} - \hat{x}'_{k+1}$$

$$= \Phi e_k + \mathrm{w}_k \tag{3.25}$$

Extending equation 3.10 to time $k+1$;

$$P'_{k+1} = E[e'_{k+1}\, e'^{T}_{k}] = E[(\Phi_{ek} + \mathrm{w}_k)(\Phi_{ek} + \mathrm{w}_k)^T] \tag{3.26}$$

Note that e_k and w_k have zero cross-correlation because of the noise w_k actually accumulates between k and $k+1$ whereas the error e_k is the error up until time$k$. Therefore;

$$P'_{k+1} = E[e'_{k+1}\, e'^{T}_{k}]$$

$$= E\left[\Phi_{e_k}\left(\Phi_{e_k}\right)^T\right] + E[\mathrm{w}_k \mathrm{w}_k^T]$$

$$= \Phi P_k \Phi^T + Q \tag{3.27}$$

This completes the recursive filter.

3.4 PID Controller

A proportional – integral – derivative controller (PID controller) is a control loop device that employs feedback that is commonly used in industrial control systems and a number of other applications that require continuously modulated command. PID control is a well-established way to move a device to an aim or position. A PID controller calculates an error value as the difference between a required set point (SP) and a measured process variable (PV) on an ongoing basis and applies a correction based on proportional, integral, and derivative terms (referred to as P, I, and D), thus the title. In practical terms, accurate and sensitive correction is automatically applied to the control function.

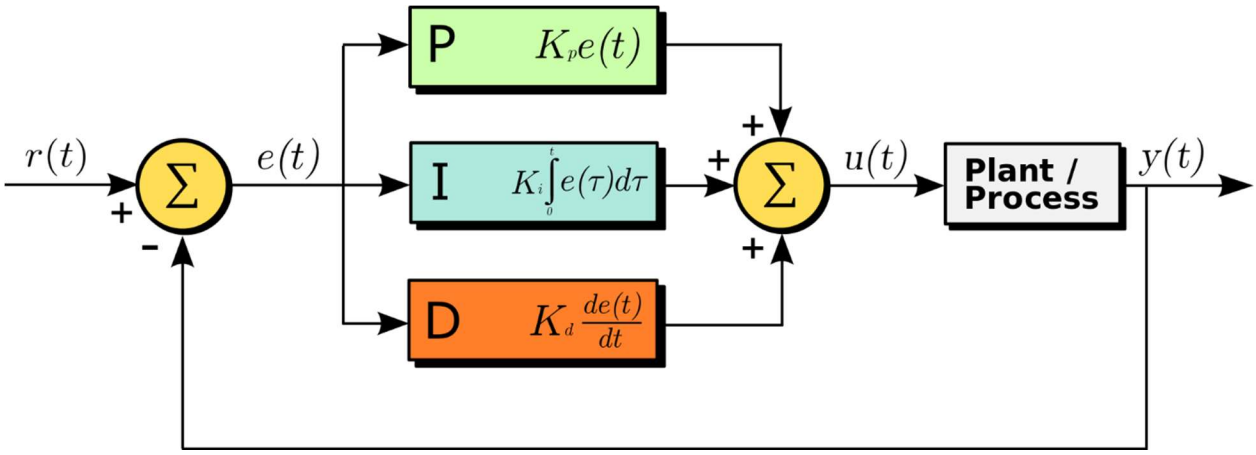

Fig. 3.18 Block Diagram of a PID controller[54]

The complete function of the control:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt}$$

Where K_p, K_i, K_d all non-negative, denote the coefficients for the proportional, integral, and derivative terms respectively (sometimes denoted P, I, and D).

The use of the PID algorithm does not guarantee optimal process access or device stability. In situations when excess delays exist: the measurement of the value of the process is delayed or the action of control is insufficiently effective. The control unit's sensitivity to an error can be defined, the degree to which the system exceeds a set point and the degree of any system shift.

PID CONTROLLER THEORY:

Proportional term:

The proportional term produces an output value commensurate with the real error value. By multiplying the error with a constant proportional response can be modified. This is called proportional gain constant.

$$P_{out} = K_p e(t)$$

A high proportional gain leads to a significant performance change for a specific error shift. The process can become unstable if the proportional gain is too high. A small gain in turn leads to a small response to a large input error and to a control system that is less responsive or less sensitive. If the proportional gain is too low, the control action in response to system disruptions may be too small. [54]

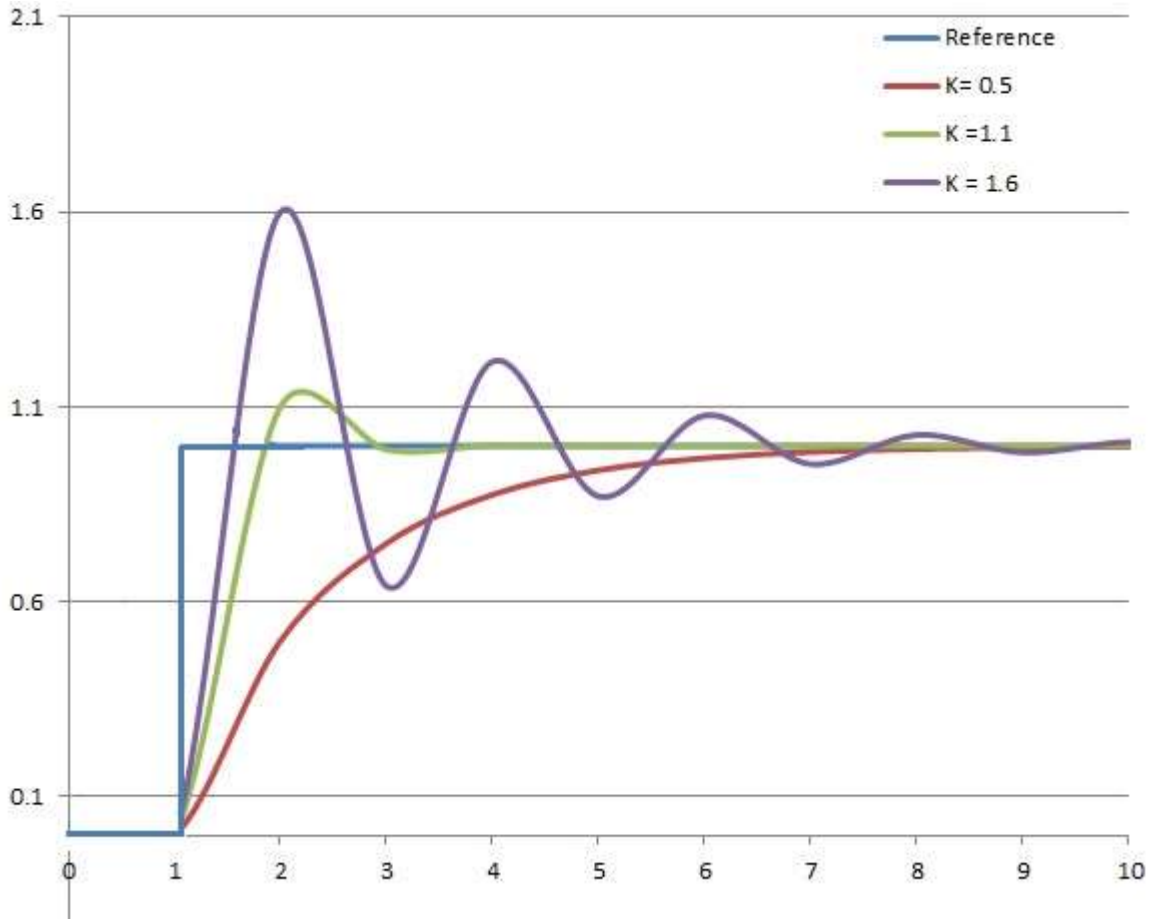

Fig. 3.19 Response of PV to step change of SP vs time, for 3 values of K_p [54]

Integral term:

The integral term contribution is proportionate to the size of the error as well as the length of the error. The integral part of a PID controller is the sum of the instantaneous error over time and the cumulative offset, which should have previously been fixed.[55]

The integral term is given by:

$$I_{out} = K_i \int_0^t e(\tau) d\,\tau$$

The integral term speeds up the process moving towards the setpoint and eliminates the remaining static error that happens with a purely proportional controller. But since the integral term reacts to accumulated past errors, it can override the present value.

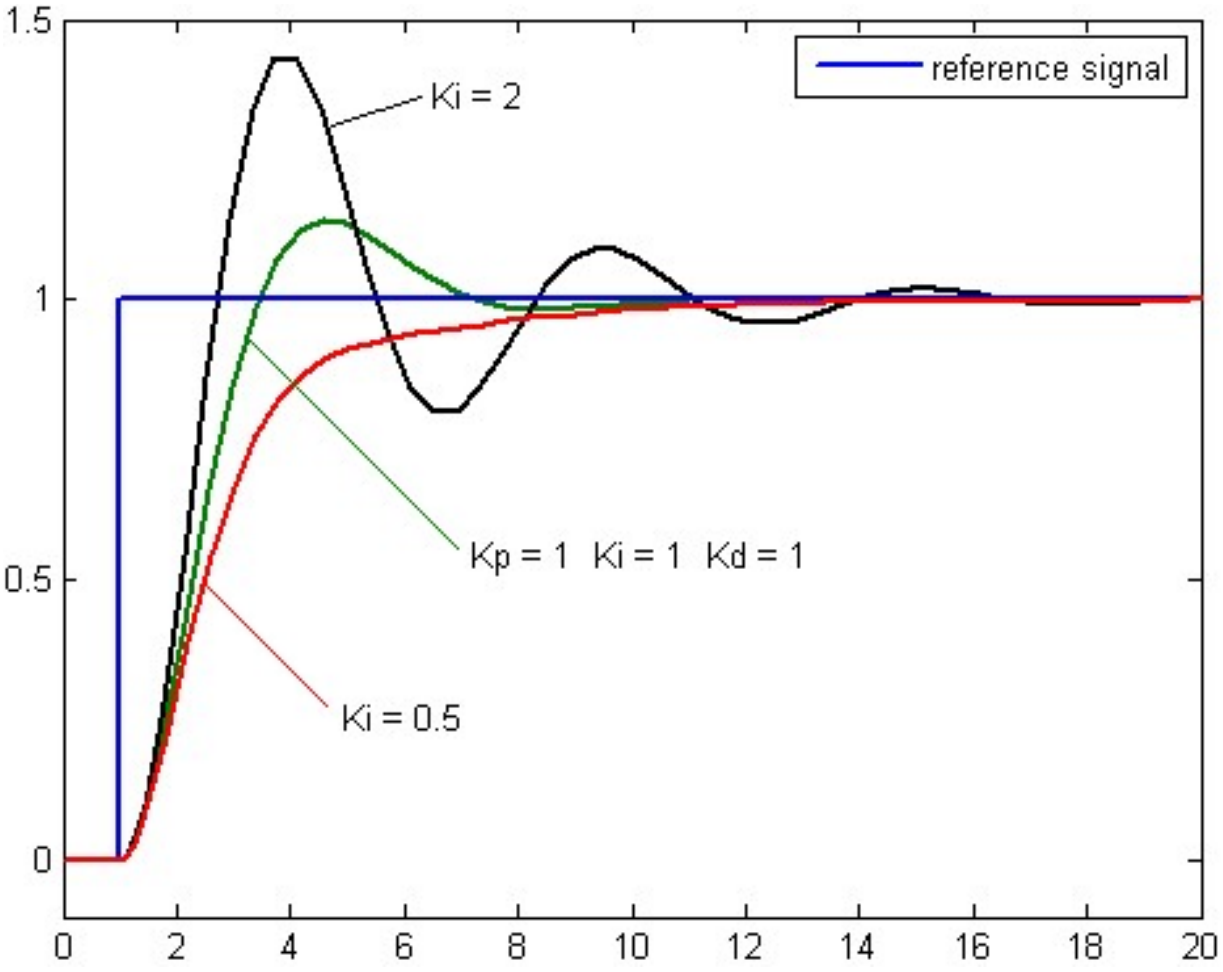

Fig. 3.20 Response of PV to step change of SP vs time, for 3 values of K_i [54]

Derivative term:

Through evaluating the slope of the error over time, the derivative error is determined and this value is compounded by the derivative gain. The extent of the derivative term's contribution to the general control action is known as derivative gain. The derivative term is given by:

$$D_{out} = K_d \frac{de(t)}{dt}$$

Derivative steps predict system performance and thus improve system adjustment time and stability.

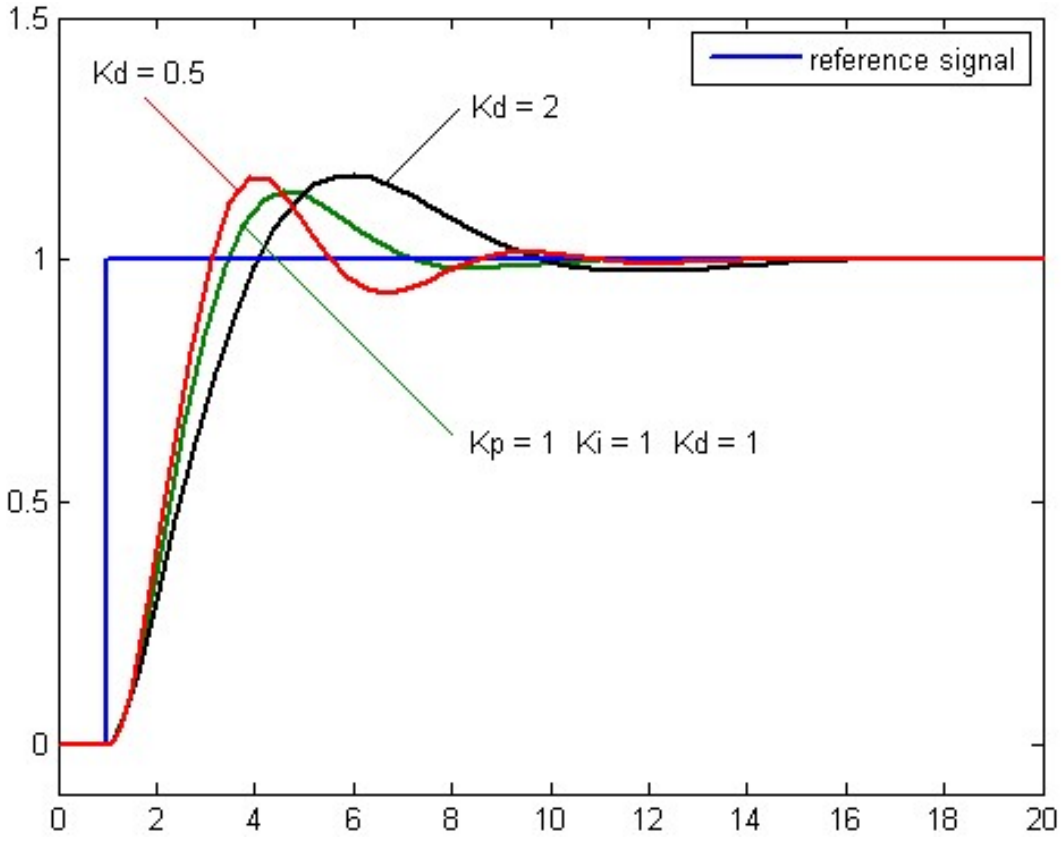

Fig. 3.21 Response of PV to step change of SP vs time, for three values of K_d [54]

Chapter 4

Hardware & Software Setup

4.1 Hardware Configuration:

Our goal was to build a system containing a real vehicle. But due to resource constraints, we couldn't manage it. Hence, we had to build a prototype of a vehicle. Our vehicle contains the following parts:

1. Plywood body: We build a plywood body of 32cmX35cmX12cm. Plywood was used for better isolation of electrical equipment. The plywood was cut using CNC [56] cutter for better measurement accuracy, so it was easy to waterproof the vehicle.
2. Primary Image Processing Unit: Raspberry Pi Model 3 B [57] was used as the primary image processing unit. Raspberry Pi is cost-efficient and a vast library of computer vision is readily available for it.

Fig 4.1 Raspberry Pi Model B

3. Motion Planning and Steering Unit: Arduino Uno [58] Rev3 was used as our motion planning and steering unit.

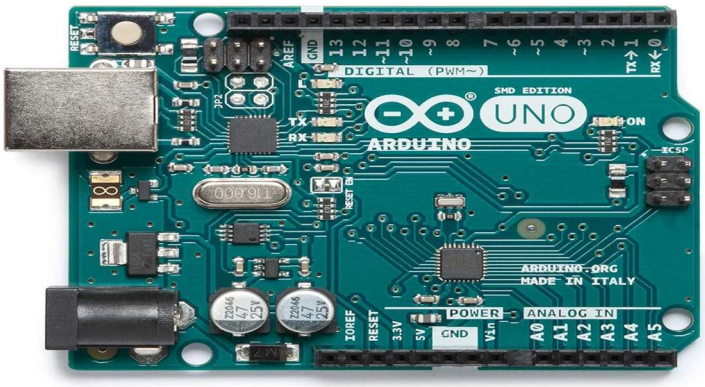

Fig. 4.2 Arduino Uno rev 3

4. On-board camera: Pi Cam 1.3[59] is used for real-time video capture.

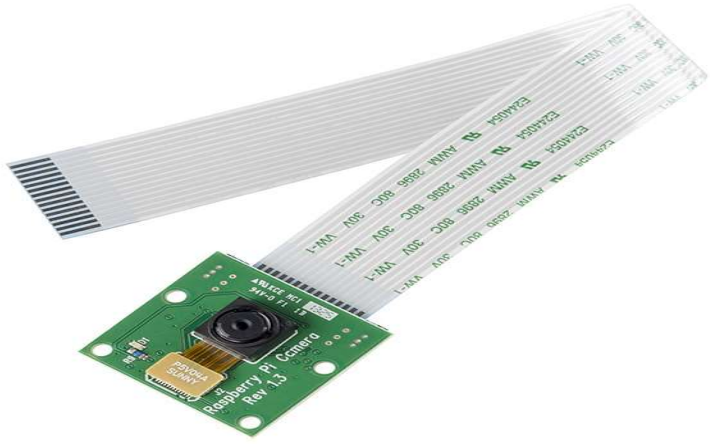

Fig. 4.3 Pi Cam 1.3

5. Wheels: 120 mm diameter wheel was used for our vehicle.

6. Motors: 12 volt, 300 RPM, 10 kg-cm torque motors were used.

7. Battery: 3 cell 3300 mAh LiPo cell was used.

8. Motor Driver: VNH2SP30 [60] full-bridge motor driver was used to control the motor.

Fig. 4.4 VNH2SP30 full-bridge motor driver

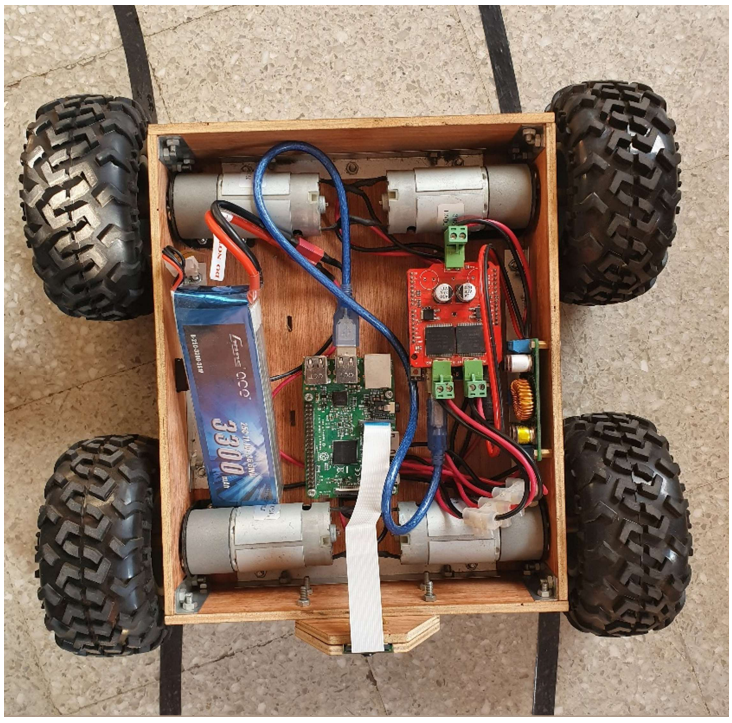

Fig. 4.5 Total hardware setup

4.2. Configuration of Raspberry Pi:

The Raspberry Pi is a low-cost, credit-card-sized device that plugs into a computer monitor or television, using a regular mouse and keyboard. It is a small versatile device that allows people of all ages to explore programming and learn how to program in languages such as Scratch and Python.[61]

Fig. 4.6 Raspberry Pi

Technical Specification:

1. Broadcom BCM2837 64bit ARMv7 Quad Core Processor powered Single Board Computer running at 1.2GHz
2. 1GB RAM
3. BCM43143 WiFi onboard
4. Bluetooth Low Energy (BLE) on board
5. 4 x USB 2 ports
6. CSI camera port for connecting the Raspberry Pi camera
7. Micro SD port for operating system setup and data storage

8. Full-size HDMI
9. 40pin extended GPIO
10. DSI display port for connecting the Raspberry Pi touch screen display
11. Upgraded switched Micro USB power source (now supports up to 2.4 Amps)

4.2.1 Installation of Raspbian:

There are also a couple of ways to download and use the Raspberry Pi operating system. Using the NOO BS (New Out of Box Software) installer is the most user-friendly process.

Simply downloading the ISO operating system, formatting the SD card, mounting the ISO and finally boo ting the Pi we set up the OS. It basically contains the following steps:

1. Downloading the NOOBS package
2. Extracting the downloaded package
3. Copying the contents to the SD card
4. Plugging the SD card in the Raspberry Pi and booting it
5. Choosing the OS option as Raspbian and clicking the install button
6. Once installed we hit okay and the device should reboot

Configuration of Raspberry Pi Camera Rev 1.3

To configure the camera the first task is to connect the camera ribbon to the Pi camera ribbon port. Afterward the following steps were followed:

1. Enabling the camera module by typing ‘sudo raspi-config’ in the command window first
2. Then choose the ‘Interfacing options’ there
3. Afterward choosing the ‘Camera’
4. Pressing ‘Yes’ to enable and rebooting

4.2.2 Installation of OpenCV and Python on the Raspberry Pi:

To deal with image or video processing we had to come across the OpenCV library. In our case we used OpenCV integrated in Python Scripts. The following steps show how to install OpenCV on the Raspberry Pi and how to integrate it into Python:

1. First of all, we updated the package lists by the command and then reboot-

   sudo apt-get update && sudo apt-get upgrade && sudo rpi-update
2. Then we cloned OpenCV from git-

   git clone https*://*github*.com/*Itseez/opencv*.git && cd OpenCV &&git checkout 3.0.0*
3. We installed Python 2.7 afterward. To do so we had to type the following-

   sudo apt-*get install* python2*.7*-dev
4. We also needed the package management tool pip, which installs NumPy right away:

   cd ~ && wget https*://*bootstrap*.pypa.io/get*-pip*.py && sudo python get*-pip*.py*
5. Then we simply installed NumPy-

   pip install numpy
6. Now to compile OpenCV we create a build folder in which the compiled files land:

   cd ~/opencv && mkdir build && cd build

   cmake -*D* CMAKE_BUILD_TYPE=*RELEASE*

   -*D* CMAKE_INSTALL_PREFIX=*/*usr*/local*

   -*D* INSTALL_PYTHON_EXAMPLES=*ON*

   -*D* INSTALL_C_EXAMPLES=*ON*

-*D* OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/*modules*

-*D* BUILD_EXAMPLES=*ON* ..

7. Then we compiled
8. After compilation we installed OpenCV:

   sudo make install && sudo ldconfig

Chapter 5

Methodology

5.1 Data Collection and Preprocessing

Video is captured by PiCam 1.3 at the resolution 640 x 480. The frame is resized to half of its width and height, thus the resolution is converted to 320 x 240. The image must be converted to one color scale in the coming stages of the presented algorithm, edge detection and Hough transformation, also called a grayscale image. So, each frame of the video is converted to Grayscale from RGB. Afterward, to remove the noises Gaussian filter is implemented for image blurring. Then canny edge detector is used for edge detection.

Algorithm 5.1 Canny edge detection

```
cap = cv.VideoCapture(0)
while (cap.isOpened()):
    ret, frame = cap.read()
    h = frame.shape[0]
    w = frame.shape[1]
    frame = cv.resize(frame, (int(w / 2), int(h / 2)))
    frame = cv.flip(frame, -1)
    gray = cv.cvtColor(frame, cv.COLOR_RGB2GRAY)
    blur = cv.GaussianBlur(gray, (5, 5), 0)
    canny = cv.Canny(blur, 50, 150)
```

Fig. 5.1 Actual Frame

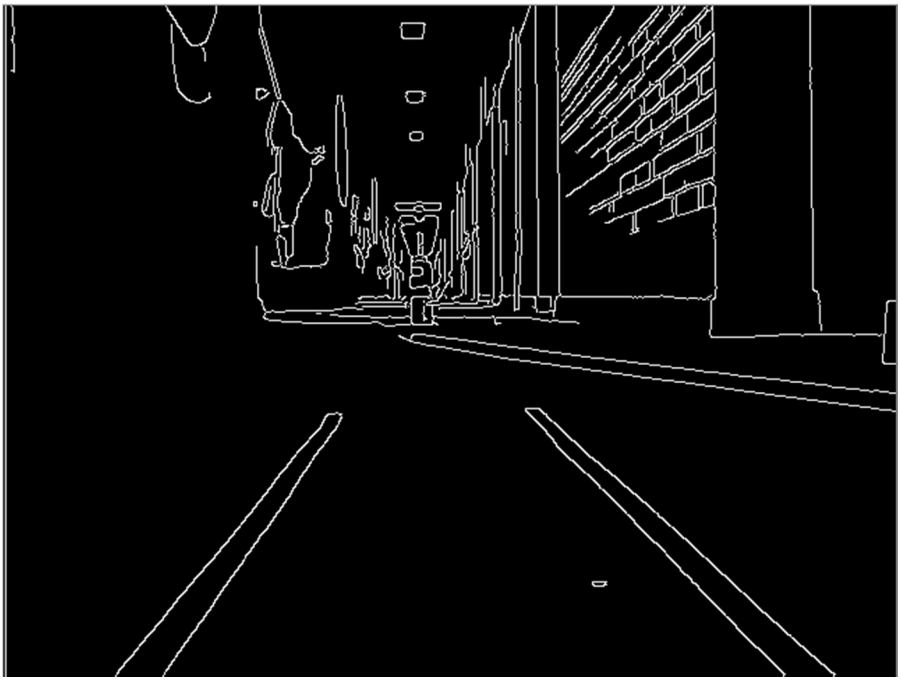

Fig. 5.2 Edge detected Frame

The road surface region of the bottom half of the picture is present in any photo which contains roads (or lane lines). The region of interest (ROI) is calculated on the basis of this data. So, just to keep the ROI masking is done on the edge detected image. The mask used eliminates the upper half of our image in a polygon shape and just keeps the lower half which contains lane lines.

Algorithm 5.2 Masking

```
height = canny.shape[0]
  width = canny.shape[1]
  polygons = np.array([[(0, height), (width, height), (width, 3 / 4 * height), ((3 / 4 * width), height / 3),
                (width / 4, height / 3), (0, 3 / 4 * height)]])
  polygons = polygons.astype(int)
  mask = np.zeros_like(canny)

  cv.fillPoly(mask, polygons, 255)
  segment = cv.bitwise_and(canny, mask)
```

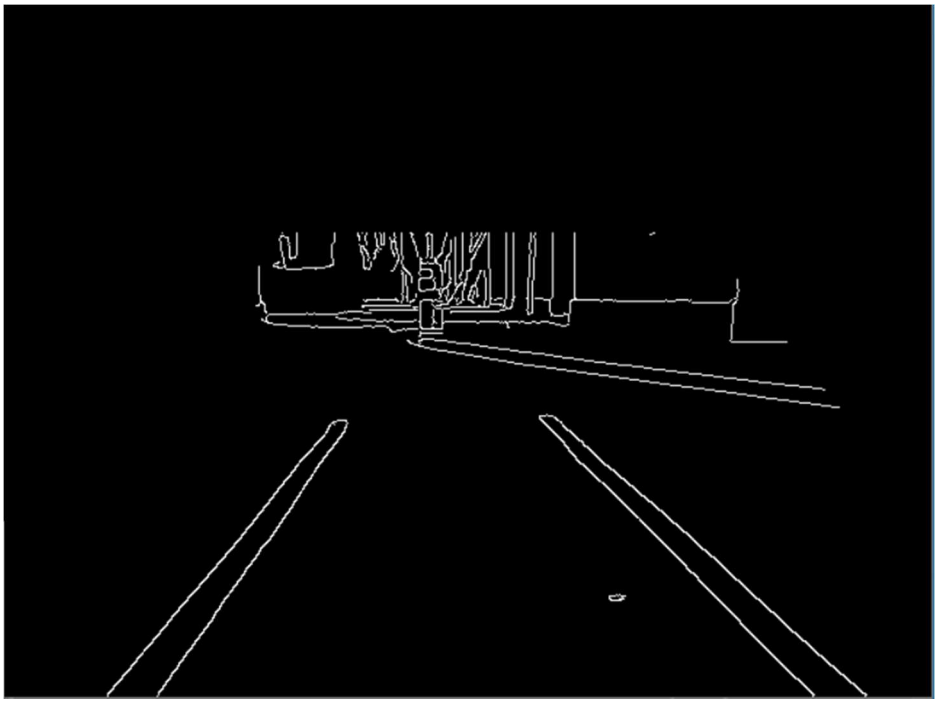

Fig. 5.3 Segmented Frame

5.2 Hough Lines Detection:

First of all from the hough function we obtain two coordinates (x1, y1 and x2, y2) of the line equation. From the coordinates we find out the y-intercept and the slope by polyfit function.

The following line equation was used to find the angle values for each detected line.

$$y = mx + c$$

where *m* is the slope and c is the y-intercept.

m can be obtained from:

$$m = \frac{y2 - y1}{x2 - x1}$$

m is the value of the tangent of the angle that line makes with the x-axis. Hence, *m* can

also be written as,

$$m = tan(\theta)$$

Once the value of *m* is found, the following equation can be used to compute the value of angle that the line makes with the x-axis:

$$\theta = arctan(m)$$

Then depending on the slope values, if the slope is between -10 to -80 the lines are categorized as left lanes and between 10 to 80 the lines are categorized as right lanes. After categorizing all the lines of the left part are converted to a single lane and similar thing is done for finding the right lane as well by averaging the slope and y-intercept values.

Algorithm 5.3 Hough Transform

```
hough = cv.HoughLinesP(segment, 1, np.pi / 180, 10, np.array([]), minLineLength=20, maxLineGap=2)
```

Algorithm 5.4 Slope and y Intercept

```
parameters = np.polyfit((x1, x2), (y1, y2), 1)
```

Fig. 5.4 Lane and Current position Detected Frame

5.3 Calculating Current Position:

We obtained two lanes by the above procedures. So, We take the average of the x coordinates of the farthest most end of the two lanes to calculate and obtain the current position of the vehicle. Since the Pi Cam is mounted exactly at the center of the vehicle the desired position of the bot is always the center of the obtained frame. So we chose the setpoint to be 160 of the x-axis and this should be always the desired position of the vehicle. Now the current position changes with the vehicle movement either in the right or in the left direction and thus the current position have a significant deviation from the set point. The green dot in figure X shows the current position.

Algorithm 5.5 Current position

```
if len(left_avg) > 0:
        left_line = calculate_coordinates(frame, left_avg)
        _, _, left_x2, pos_y = left_line.reshape(4)
```

```
else:
    left_x2=0
if len(right_avg) > 0:
    right_line = calculate_coordinates(frame, right_avg)
    _, _, right_x2, pos_y = right_line.reshape(4)
else:
    right_x2=320

lines = np.array([left_line, right_line])

pos_x = int((left_x2 + right_x2) / 2
```

However,there are some shortcomings in finding the current position. Depending on the weather and light conditions, there exists some inconsistency in this process - sometimes the current position can be determined without any complications and sometimes the current position cannot be easily determined. To overcome this inconsistency we used two different algorithms-

1. Past Accumulated Average Method

2. Kalman Filter Algorithm

In the Past Accumulated Method we initialized an accumulator matrix of n size. The past n values of current positions found will be stored sequentially. The first element of the accumulator will be the value of the most recent or present frame. The next element will be the immediate past position and so on. For our case we took $n = 8$ and the elements of the matrix are averaged to obtain the current position. This algorithm helps us to remove the failure of the system even if the system misses any frame.

Another algorithm that we used was the Kalman Filter prediction algorithm. Here, we calculated the steering direction at any time instant, T. A predefined measurement noise was given which always remains fixed. Kalman filter uses the direction and the measurement noise and predicts the current position for the next time instant T+1. Kalman Filter compensates the measurement noise that might be there.

Figure Y shows the current position point detected by two algorithms.

Fig. 5.5 Past Accumulated Average method Vs Kalman Filter Algorithm

We further modified our algorithm by combining past accumulated average method and Kalman filter prediction. We took the current position from both method and average the combined value to predict our current position.

5.4 Communication:

The current position is sent to the Arduino Uno Rev3 from the Raspberry Pi using UART communication. The transmitting UART converts parallel data from the Raspberry Pi into serial form and transmits it in serial to the receiving UART which is Arduino for our case. The value of current position varies from 0 to 320. These three bytes information take a lot of resources while sending it to the Arduino. So, this range is mapped to a lower range from 1 to 99 and thus we managed the serial communication by restricting the data size within two bytes. The current position information is sent only when there is a transition in the value of 1 to 99. An indicator ASCII value is sent after transmitting the data of each frame so that the Arduino can differentiate the data of the individual frames.

The Receiving UART i.e. Arduino Uno receives the first byte of serial data and shifts its position by 1 decimal place and then receives the second byte and keeps on doing this till it gets the predefined ASCII value. As soon as it finds the predefined ASCII value it breaks the loop and shows the accumulated last value which corresponds to the current position of the vehicle.

5.5 Motion Planning:

PID algorithm was implemented on the Arduino Uno. The algorithm calculates the error by subtracting the obtained current position value from time to time from the set point. This error value is used to plan the motion of the vehicle using PID algorithm. The PID algorithm was tuned by fixing integral constant, derivative constant and proportional constant by trial and error method.

5.6 Steering:

Depending on the PID output the drive function adds the error value of PID to the base value(100 PWM for our case) of left motor and subtracts it from the right or vice-versa depending on the necessity. Then it caps the value to a certain value of PWM which should not be exceeded. Thus the vehicle moves smoothly without any sharp turns or extremely high or low PWM and moves autonomously maintaining the lanes.

Chapter 6

Experiments and Results:

6.1 Comparative Study Between Different Methods:

6.1.1 Steering direction without any filter against steering direction with past accumulating average method:

We compare the desired direction of motion, gained firstly by taking the middle point of the lanes in every frame and by taking the accumulated average of the middle points of the lanes of some past frames and the present frame. We represent our result in the 2-D graph shown below. The Y-axis of the graph shows the horizontal values of the direction of motion in terms of image pixel and the X-axis shows the frame number. The results found with no filtering are depicted in blue and with past average accumulation is depicted in orange.

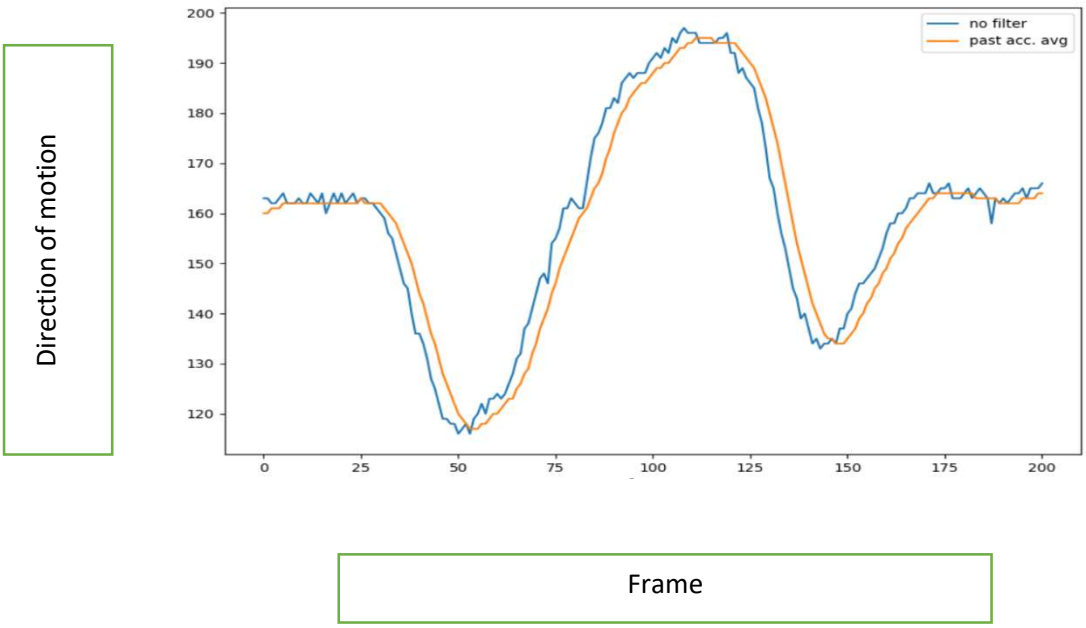

Fig. 6.1 Direction of motion with no filter and with past accumulated average

As we can see, the past accumulated average method indeed makes the changes in the direction of motion smoother. The averaging of the previous frames makes the system less sensitive to sudden changes in direction that might happen due to false positives in our detection phase. So, the system is less prone to failure now.

But it is not without any drawback as we can see a frame-lag being added when we use past accumulating average method. This frame-lag can cause a lack of response when a sudden change in direction is needed.

6.1.2 Steering direction without any filter against steering direction predicted by Kalman filter:

In this section, we compare the desired direction of motion, obtained first by taking in each frame the middle point of the lanes and then by using the Kalman filter to estimate the current position and the desired direction of motion. In the 2-D graph shown below, we represent the outcome. The graph's Y-axis shows the horizontal pixel values of the direction of motion and the X-axis shows the frame number. The tests found without filtering are shown in blue and are shown in orange with Kalman filter.

The figure is given on the next page.

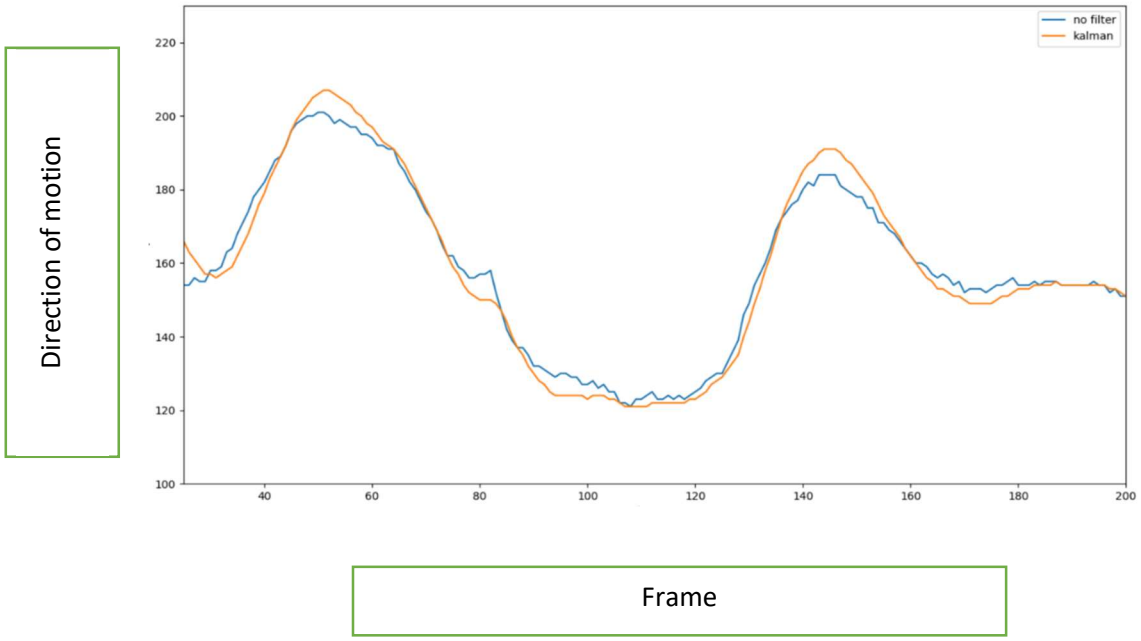

Fig.6.2 Direction of motion with no filter and with Kalman filter

It is evident from the graph that Kalman filter prediction of steering direction doesn't have any frame-lag, which was visible in the case of past accumulated average method. So, Kalman filter predicts the movement direction faster. It also makes the sudden changes in direction smoother than the original result obtained without any filtering but not as smooth as the past average accumulation method and is sensitive to sudden changes that can make the system unstable.

6.2.3 Steering direction without any filter against steering direction calculated by the average of Kalman filter's result and past accumulating average method's result:

This section contains a comparison between the direction of motion desired, obtained first by taking the middle point of the lanes in each frame and then by using the average of the Kalman filter result and the past accumulated average result. We reflect the result in the 2-D graph shown below. The Y-axis of the graph displays the horizontal pixel values of motion direction and the X-axis shows the frame number. The tests found without filtering are displayed in blue and with the average of Kalman filter prediction and past accumulated average result is shown in orange.

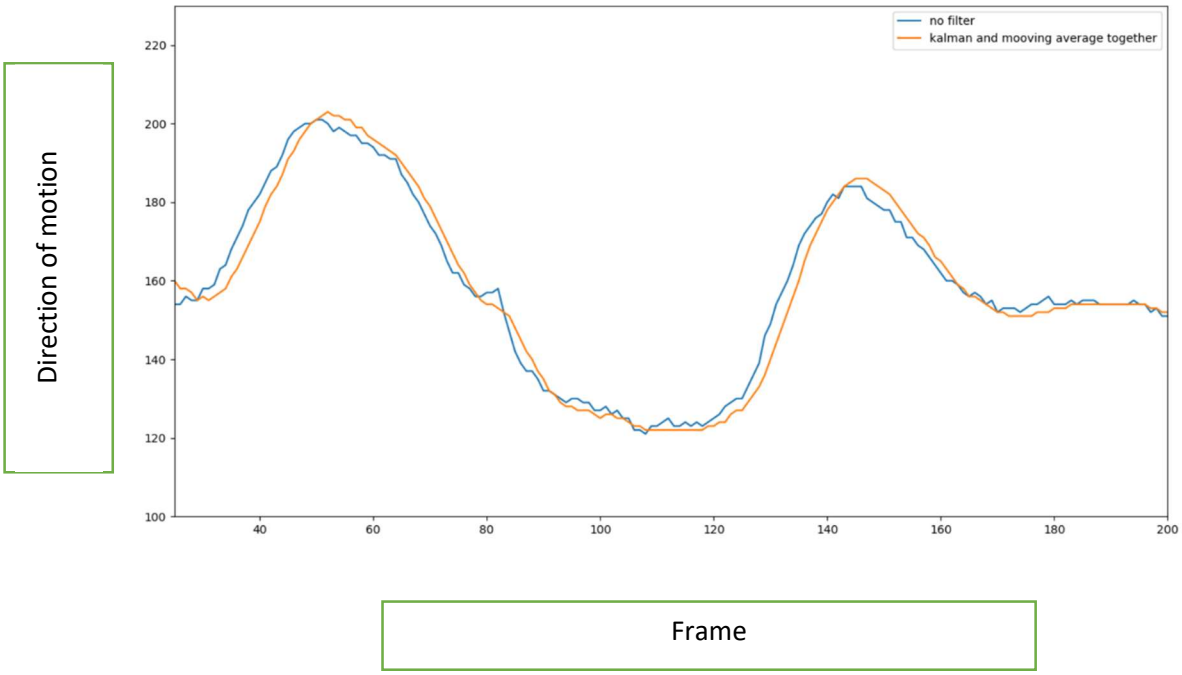

Fig.6.3 No filter and with the avg. of Kalman filter and past accumulated average

The steering position calculated by averaging the Kalman filter result and the past accumulating average method's result reduces the frame-lag found in the case of past accumulating average method as well as makes the system less sensitive and the transitions a lot smoother which was absent in the case of Kalman filtering.

From the results obtained in the previous sections it is evident that the steering direction obtained by using the average of both the Kalman filter and past accumulating average makes the system more desirable for usage as the frame-lag is minimal and the system is less sensitive than the system having Kalman filter.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

From our experience, we can testify that our modified algorithms such as Kalman Prediction and combining the Kalman prediction and Past accumulated averaging makes steering direction calculation much efficient. But the limited resource we used in our thesis is not enough to make a self-driving car that can run on real roads as we didn't have enough processing power to implement any algorithms to avoid collisions with other cars. Our camera module gives a very narrow field for work, therefore, a wide-angle camera with a better resolution would have helped in our case. Still the results that we have are satisfactory as it shows the efficiency of different algorithms used and the prototype can successfully follow road lanes in confined environmental condition.

7.2 Future work

Instead of using processing power-constrained embedded systems like Raspberry Pi, embedded systems with much stronger CPU can be used to implement sophisticated algorithms like RANSAC [65] and filters like Particle filter [66]. A system with our detection and movement algorithm along with a collision avoidance algorithm [67] can be built and implemented in a real car running in a real environment. Machine learning can be used to remove environmental and lighting constraints, which would make our system more robust. A surveillance system can be built

by implementing some feedback sensors in our system. By tweaking our algorithms, cost-efficient production and assembly systems can be built. A hospital patient monitoring system can be a good sector to use our system. FPGA implementation of our algorithms along with some machine learning algorithms can be a good way to make cost-effective autonomous or semi-autonomous vehicles.

References

[1] Wikipedia: *https://en.wikipedia.org/wiki/Self-driving_car*

[2] A.M. Lutzeler, E.D. Dickmanns, ”Road recognition with MarVEye”, ¨Proceedings of the IEEE Intelligent Vehicles Symposium 98, Stuttgart,Germany, October 1998, pp. 341-346.

[3] T.M. Jochem, D.A. Pomerleau, C.E. Thorpe,’’MANIAC: A next generation neurally based autonomous road follower”, Proceedings of the Third International Conference on Intelligent Autonomous Systems, Pittsburg, PA, February 1993.

[4] Benozzi, M. and A. Broggi, M. Cellario, 2002. Artificial vision in road vehicles. Proceedings of IEEE, 90 (7): 1258-1271

[5] Broggi, B.M., 1998. GOLD: A parallel real-time stereo Vision system for generic obstacle and lane detection, IEEE Transactions on Image Processing, 1998: 4-6.

[6] BBC News. (2014, Jul.) Uk to allow driverless cars on public roads in january.[Online]. Available: *http://www.bbc.com/news/technology-28551069*

[7] R. Burn-Callander. (2015, Feb.) This is the lutz pod, the UK’s first driverless car. [Online]. Available: *https://www.telegraph.co.uk/finance/businessclub/technology/11403306/This-is-the-Lutz-pod-the-UKs-first-driverless-car.html*

[8] M. Hayward. (2017, Jan.) First New Zealand autonomous vehicle demonstration kicks off at Christchurch airport. [Online]. *https://www.stuff.co.nz/technology/88790124/first-new-zealand-autonomous-vehicle-demonstration-kicks-off-at-christchurch-airport*

[9] P. T. Mandlik, \A review on lane detection and tracking techniques," 2016

[10] A.M. Lutzeler, E.D. Dickmanns, ”Road recognition with MarVEye”, ¨Proceedings of the IEEE Intelligent Vehicles Symposium 98, Stuttgart,Germany, October 1998, pp. 341-346

[11] J. M. Clanton, D. M. Bevly, and A. S. Hodel, "A LowCost Solution for an Integrated Multisensor Lane Departure Warning System," in IEEE Transactions on Intelligent Transportation Systems, vol. 10, no. 1, pp. 47-59, Mar. 2009.

[12] Q. Li, L. Chen, M. Li, S. Shaw and A. Nüchter, "A Sensor-Fusion Drivable-Region and Lane-Detection System for Autonomous Vehicle Navigation in Challenging Road Scenarios," in IEEE Transactions on Vehicular Technology, vol. 63, no. 2, pp. 540-555, Feb.2014.

[13] M. Aly, "Real-time detection of lane markers in urban streets," in Proc. IEEE Intelligent Vehicles Symposium, Eindhoven, pp. 7-12, 2008.

[14] C. Lee and J. H. Moon, "Robust Lane Detection and Tracking for Real-Time Applications," in IEEE Transactions on Intelligent Transportation Systems, vol. no.99, pp.1-6, Feb. 2018

[15] W. Song, Y. Yang, M. Fu, Y. Li and M. Wang, "Lane Detection and Classification for Forward Collision Warning System Based on Stereo Vision," in IEEE Sensors Journal, vol. 18, no. 12, pp. 5151-5163, Jun.2018

[16] J. H. Yoo, S. Lee, S. Park and D. H. Kim, "A Robust Lane Detection Method Based on Vanishing Point Estimation Using the Relevance of Line Segments," in IEEE Transactions on Intelligent Transportation Systems, vol.18, no. 12, pp. 3254-3266, Dec. 2017.

[17] S. Jung, J. Youn, and S. Sull, "Efficient Lane Detection Based on Spatiotemporal Images," in IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 1, pp. 289-295, Jan. 2016.

[18] A. Assidiq, O. O. Khalifa, R. Islam, and S. Khan, \Real time lane detection for autonomous vehicles," 2008 International Conference on Computer and Communication Engineering, pp. 82{88, 2008.

[19] Q.-B. Truong and B. R. Lee, \New lane detection algorithm for autonomous vehicles using computer vision," 2008 International Conference on Control, Automation and Systems, pp. 1208{1213, 2008.

[20] Y. Shu and Z. Tan, \Vision based lane detection in autonomous vehicle," Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No.04EX788), vol. 6, pp. 5258{5260 Vol.6, 2004.

[21] C. Mu and X. Ma, “Lane detection based on object segmentation and piecewise fit-ting,” TELKOMNIKA Indonesian J. Elect. Eng., vol. 12, no. 5, pp. 3491-3500, May 2014.

[22] A. Parajuli, M. Celenk, and H. Riley, “Robust lane detection in shadows and low illumination condition using local gradient features,” open J. Appl. Sci. , vol. 3, no. 1B, pp.68-74,Mar. 2013

[23] Y.Li, A.Iqbal, and N.R.Gans, “Multiple lane boundary detection using a combina-tion of low-level image features.”In Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on, pp. 1682-1687, IEEE, 2014.

[24] V.S.Bottazzi, P.V.Borges, B.Stantic, “Adaptive regions of interest based on HSV histograms for lane marks detection”. In Robot Intelligence Technology and Applications 2, pp. 677-687, Springer International Publishing, 2014

[25] J.D. Crisman, C.E. Thorpe, ”UNSCARF, a color vision system for the detection of unstructured roads”, Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA, April 1991,pp. 4962501

[26] L. Michael Beuvais, C. Kreucher, ”Building world model for mobile platforms using heterogeneous sensors fusion and temporal analysis”, Proceedings of the IEEE International Conference on Intelligent Transportation Systems 97, Boston, MA, November 1997, p. 101.

[27] Nicholas Apostoloff and Alexander Zelinsky, ”Vision In and Out of Vehicles: Integrated Driver and Road Scene Monitoring”, Proceedings of the International Symposium of Experimental Robotics (ISER2002), Italy, 2002.

[28] Wikipedia: *https://en.wikipedia.org/wiki/Edge_detection*

[29] Wikipedia: *https://en.wikipedia.org/wiki/Canny_edge_detector*

[30] Canny, J., A Computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[31] Green, B. (2002, January 1). Canny Edge Detection Tutorial. Retrieved December 3, 2014;

[32] Mallat S, Zhong S. Characterization of Signals from Multi scale Edges [J]. IEEE Trans on PAMI, 1992, 14 (7):710-732.

[33] Otsu N. A threshold selection method from gray-level histograms. IEEE Trans Systems, Man and Cybernetics, 9(1):62-66, 1979.

[34] Gebäck1, T. & Koumoutsakos, P. "Edge detection in microscopy images using curvelets" BMC Bioinformatics, 10: 75, 2009.

[35] Moeslund, T. (2009, March 23). Canny Edge Detection. Retrieved December 3, 2014

[36] Canny Edge Detection- CSE IIT, Delhi.

[37] John Canny. A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, PAMI-8(6):679–698, Nov. 1986.

[38] Sergei Azernikov. Sweeping solids on manifolds. In Symposium on Solid and Physical Modeling, pages 249–255, 2008.

[39] F. Mai, Y. Hung, H. Zhong, and W. Sze. A hierarchical approach for fast and robust ellipse extraction. Pattern Recognition, 41(8):2512–2524, August 2008.

[40] R. Boyle and R. Thomas Computer Vision: A First Course, Blackwell Scientific Publications, 1988, p 52.

[41] Gaussian filtering *http://robotics.eecs.berkeley.edu/~mayi/imgproc/gademo.html*

[42] Justin Canny article: *http://justin-liang.com/tutorials/canny/*

[43] *https://en.wikipedia.org/wiki/Hough_transform#Implementation*

[44] *https://www.learnopencv.com/hough-transform-with-opencv-c-python/*

[45] John Canny. A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, PAMI-8(6):679–698, Nov. 1986.

[46] Richard O. Duda and Peter E. Hart. Use of the Hough transformation to detect lines and Curves in pictures. Commun. ACM, 15(1):11–15, January 1972.

[47] *http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/HoughTrans_lines_09.pdf*

[48] *https://stackoverflow.com/questions/42660184/opencv-detect-flawed-rectangle?noredirect=1&lq=1*

[49] C. Galambos, J. Kittler, and J. Matas. Progressive probabilistic Hough transform for line detection. Computer Vision and Pattern Recognition, IEEE Computer Society Conference on, 1:1554, 1999.

[50] Kalman, R. E. (1960). "A new approach to linear filtering and prediction problems". Transactions of the ASME – Journal of Basic Engineering. Series D. **82** (1): 35–45. doi:10.1115/1.3662552.

[51] Paul Zarchan; Howard Musoff (2000).” Fundamentals of Kalman Filtering: A Practical Approach”. American Institute of Aeronautics and Astronautics, Incorporated. ISBN 978-1-56347-455-2.

[52] Wikipedia.Kalman Filter. [Online]. Available: *https://en.wikipedia.org/wiki/Kalman_filter*

[53] T. Lacey, "Tutorial: The Kalman Filter", Computer Vision, *http://www.cc.gatech.edu/classes/cs7322-98-spring/PS/kf1.pdf*

[54] Wikipedia: *https://en.wikipedia.org/wiki/PID_controller*

[55] *https://www.omega.co.uk/prodinfo/pid-controllers.html*

[56] CNC. [Online].Available: *https://en.wikipedia.org/wiki/CNC_router*

[57] Raspberry Pi ModelB. [Online]. Available *https://www.raspberrypi.org/products/raspberry-pi-3-model-b/*

[58] Arduino Uno. [Online]. Available: *https://en.wikipedia.org/wiki/Arduino_Uno*

[59] Pi Camera. [Online]. Available: *https://uk.pi-supply.com/products/raspberry-pi-camera-board-v1-3-5mp-1080p*

[60].Motor Driver. [Online]. Available: *https://www.sparkfun.com/products/10182*

[61] Raspberry Pi Model B. [Online]. Available: *https://thepihut.com/products/raspberry-pi-3-model-b?variant=1138129768*

[62] Raspberry Pi. [Online]. Available: *https://pythonprogramming.net/introduction-raspberry-pi-tutorials/*

[63] Camera Module. [Online]. Available: *https://pythonprogramming.net/camera-module-*raspberry-pi-tutorials/

[64] Installing OpenCV. [Online]. Available: *https://tutorials-raspberrypi.com/installing-opencv-on-the-raspberry-pi/*

[65] Wikipedia .[Online]. Available: *https://en.wikipedia.org/wiki/Random_sample_consensus*

[66] Wikipedia. [Online]. Available: *https://en.wikipedia.org/wiki/Particle_filter*

[67] IEEE explore.[Online]. Available: *https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7451174*