VISUAL OBSTACLE DETECTION AND AVOIDANCE FOR INDOOR MOBILE ROBOTS

İBRAHİM K. İYİDİR
B.S., Computer Engineering, Işık University, 2010

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Engineering

IŞIK UNIVERSITY
2012

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

VISUAL OBSTACLE DETECTION AND AVOIDANCE FOR INDOOR MOBILE ROBOTS

İBRAHİM K. İYİDİR

APPROVED BY:

Assist. Prof. F. Boray Tek (Thesis Supervisor)	Işık University	____________________
Assoc. Prof. Hasan F. Ateş	Işık University	____________________
Assoc. Prof. Olcay Taner Yıldız	Işık University	____________________

APPROVAL DATE: ..../..../....

VISUAL OBSTACLE DETECTION AND AVOIDANCE FOR INDOOR MOBILE ROBOTS

Abstract

This study is a part of a joint team effort to transform a small-scale model car into an autonomous moving robot. This transformation includes several routines that are essential and attached together. Integration of various equipment on the model car is the first step of that routine which is shared among different thesis studies conducted at RAVLAB (Robotics and Autonomous Vehicles Laboratory) of Işık University. Hence, the resulting hardware system which is explained in this thesis is mostly the co-produce of RAVLAB team.

The integration is followed by implementing the software required to establish control and communication links between different units. During this thesis, the author has developed a multi-threaded main control software to facilitate obstacle avoiding movements of the robot while reading and analyzing sensory inputs.

This thesis study mainly focuses on detection of the obstacles with visual information collected from an ordinary color camera. The main obstacle detection algorithm that is proposed in this thesis is adapted from a powerful background subtraction algorithm ViBe [1] (Visual Background Extractor). The proposed algorithm uses the model of the ground plane in order to detect obstacles. A different ground plane model is kept for each pixel location as in ViBe. The proposed algorithm is robust against illumination differences, shadows and the changes in the appearance of the ground plane. A comparison is provided (using ground truth data) with another obstacle detection algorithm [2] which also uses a ground plane based model to detect obstacles. The results of the proposed algorithm under different conditions compared to a counterpart.

In addition to the obstacle detection, during this study two obstacle avoidance algorithms are developed to facilitate navigation of the robot in indoor environments. The experiments show that the robot is able to move while avoiding obstacles by using ultrasonic sensors as well as using the visual camera input.

ROBOT ARAÇLAR İÇİN KAPALI ALANDA GÖRÜNTÜYE DAYALI ENGEL TANIMA VE ENGELDEN SAKINMA

Özet

Bu çalışmadaki amaç küçük boyutlu model bir arabanın otonom hareket edebilen bir robot haline getirilmesidir. Bu dönüşümü sağlamak için birbirine bağlı ve önemli aşamalar vardır. Bu aşamaların ilki model arabanın üzerine gerekli ekipmanın yerleştirilmesi kısmıdır ve bu kısım Işık Üniversitesi Robotik ve Otonom Araçlar Laboratuvarı araştırmacıları tarafından birlikte gerçekleştirilmiştir.

Araç donanımsal olarak hazırlandıktan sonra robot kontrolü ve robotla iletişim kurma amacıyla gerekli programların kodları yazılmıştır. Tez çalışması sırasında, yazar ana programı aynı anda engel tanıma, engelden sakınma ve sensör bilgilerini almak amacıyla çok iplikli bir yapıda tasarlamıştır.

Bu tez çalışmasında ele alınan en önemli konu normal bir kamera kullanarak engelleri robotun hareket ettiği zeminden ayırt edebilmektir. Geliştirilen engel tanıma algoritması, ViBe (VIsual Background Extractor) [1] isimli hareket algılama algoritmasından uyarlanmıştır. Geliştirilen algoritma engelleri ayırt edebilmek için hareket edilen zeminin modelini kullanmaktadır. Zemini modellemek amacıyla her piksel lokasyonu için bir zemin modeli saklanmaktadır. Geliştirilen algoritma ortamdaki ışık düzeyi farklılıklarına, gölgelere ve zemindeki görüntü değişikliklerine karşı gürbüzdür. Bunun yanı sıra, söz konusu algoritma engelleri ayırt etmek için zemin modelini kullanan başka bir algoritmayla [2] elle işaretlenmiş datalar vasıtasıyla karşılaştırılmıştır. Yapılan detaylı deneyler sonucunda ortaya çıkarılan algoritmanın stabil olduğu, değişik koşullarda çalışabildiği ve diğer algoritmadan daha üstün olduğu gözlenmiştir.

Engel tanımanın yanında, tez çalışması sırasında robot için kapalı alanda engelden sakınma algoritmaları da geliştirilmiştir. Yapılan deneyler robotun ses-ötesi sensörler kullanarak veya sadece normal bir kameradan elde ettiği görüntüyü işleyerek kapalı alanda dolaşabildiğini göstermiştir.

Acknowledgements

Firstly, I would like to thank my thesis supervisor Assist. Prof. F. Boray Tek. It was a great experience to work under his supervision. His valuable thoughts have been the keystones of this research. During this study, he never let me to loose my concentration and he's always there whenever I've faced with a problem.

Also, I would like to thank Doğan Kırcalı who is more than a teammate and also a valuable friend. He helped me to built the robot, and he always came with solutions to all hardware related problems during this thesis.

Many thanks to a special person, Semiha Özdağ whose deepest understanding and support has been a priceless contribution to this study.

Last but certainly not least, I am very grateful to my family. I thank to my parents, Zerrin and Hasan, my sisters Berna and Bihter for their patience and encouragement.

To my family…

Table of Contents

List of Tables

List of Figures

List of Symbols

H	Image height
M	Similar pixel threshold
N	Number of samples
R	Distance threshold
S	Set of sample sets
t	Ultrasonic distance update threshold
W	Image width

List of Abbreviations

BC	Ultrasonic sensor on the back
CCD	Charge-coupled device
CMOS	Complementary metal–oxide–semiconductor
CRI_DIST	Critical distance threshold
FC	Front center ultrasonic sensor
FL	Front left ultrasonic sensor
FN	False negative
FP	False positive
FR	Front right ultrasonic sensor
FPR	False positive rate
GHz	Gigahertz
KB	Kilobyte
kHz	Kilohertz
LCD	Liquid crystal display
M_TH	Move threshold
OFR	Obstacle free region
RGB	Red-Green-Blue
SDTA	Safe distance to act threshold
TN	True negative
TP	True positive
TPR	True positive rate
USB	Universal serial bus

Chapter 1

Introduction

In the last decade the mobile robot technologies have made a significant progress. Beside big budget commercial efforts, we see that small projects which are conducted by individuals or small groups who have limited budgets provide a noticeable addition to its momentum. This thesis is such a work that is completed at Robotics and Autonomous Vehicles Laboratory (RAVLAB) of Işık University, and aims to transform a small-scale model car kit into an autonomous navigating robot using a camera, electronic sensors and an onboard PC. We aimed to produce such a system at low cost.

Autonomy for the robot can be defined as giving the decisions without human intervention and autonomy needs perception as a first step. There are two main techniques in order to build such an autonomous robot which are sensor based techniques and vision based techniques. These two can be considered individually whereas these techniques can be combined to form another concept.

Our robot can sense the environment using ultrasonic distance sensors, and it is capable of analyzing the visual information which is acquired by the camera. Both approaches have pros and cons which will be discussed in further chapters.

The first step of a successful navigation is detecting the obstacles in the environment. At that point, the definition of the obstacle became significant. An obstacle may be a bumper or a hole on the road, it can be a cable lying on the ground, it can be a wall or a chair. There's not a generic definition for obstacles. Simply, we can say the definition could be adjusted considering the capabilities of the robot. For some robots, an oil slick on the floor might be an obstacle or lying cable on the floor, but for others they may not be considered as obstacles. Generally speaking, for a ground vehicle, an obstacle may be defined as a protrusion or extrusion of the ground surface, which segments the current planned path.

In the last 30 years, there's a significant progress on obstacle detection and robot navigation tasks. However, the previous works showed us in practice it is not simple to construct a generic method for obstacle detection and navigation tasks due to huge variations in environmental conditions [3].

In line with human perception the most used sensor for obstacle detection is visible light camera. Though analyzing two dimensional images for obstacle detection is challenging, in the last two decades there are significant amount of attempts [3–5]. The pioneering works which use visual information have been done by Moravec [6], Horswill [7, 8] and Lorigo [9]. Sensor based obstacle avoidance and navigation techniques were discussed earlier in such papers [10, 11]. Also Discant et al. [12] investigated different types of sensors in their work such as radar (RAdio Detection And Ranging), ladar (LAser Detection And Ranging) or lidar (Light-Imaging Detection And Ranging) and sonar (SOund Navigation And Ranging).

Contrary to that approaches which use one type of sensor, authors of [13, 14] combined ultrasonic sensing and visual information in order to navigate their robots.

Unlike the sensors (i.e. sonar) that are used for obstacle detection purposes, visual input contains more than one information such as color, texture. Also, visual input supplies knowledge about the characteristics of the environment. These are the reasons that vision based methods are preferable over the techniques which only use sensor information. In this thesis, we used both of the approaches for obstacle detection which are ultrasonic sensor based detection and visual input based detection.

1.1 Outline of Thesis

In this thesis, we covered a broad range of topics for a robot navigation from scratch. In Chapter 2, we discussed the hardware platform of our robot. We investigated the parts that form our robot. Also, we explained the working principles of the sensors on the robot (e.g. ultrasonic sensor, electronic compass).

Chapter 3 investigates our software system. We explained two programs designed to facilitate control of our robot in this chapter; Multi-threaded main program and Arduino program.

Chapter 4 is the most comprehensive chapter in this thesis. We mainly focused on two types of works in this chapter; obstacle detection by obstacle modeling and obstacle detection by ground plane modelling. In obstacle detection by obstacle modeling section, we aimed to detect the traffic cones in the indoor environment using the models of the traffic cones. We tried several methods for cone detection purpose, and we used only color information. The following section in this chapter discusses two different obstacle detection algorithms that use ground plane models. The algorithm in Section 4.2.2 is developed and implemented by us whereas we implemented another algorithm in Section 4.2.1 which is originally owned by Ulrich and Nourbakhsh [2]. We have done slight modifications on the algorithm in [2] when implementing it. In addition, we compared the performances of these three algorithms in Section 4.2.3.

In Chapter 5, we present our obstacle avoidance algorithms. There are two different avoidance algorithms; ultrasonic sensor based and vision based. Ultrasonic sensor based avoidance is also separated into two parts which are basic obstacle avoidance and avoidance using electronic compass.

Chapter 2

Hardware System

In this chapter the components that comprise our hardware system will be discussed (Figure 2.1).

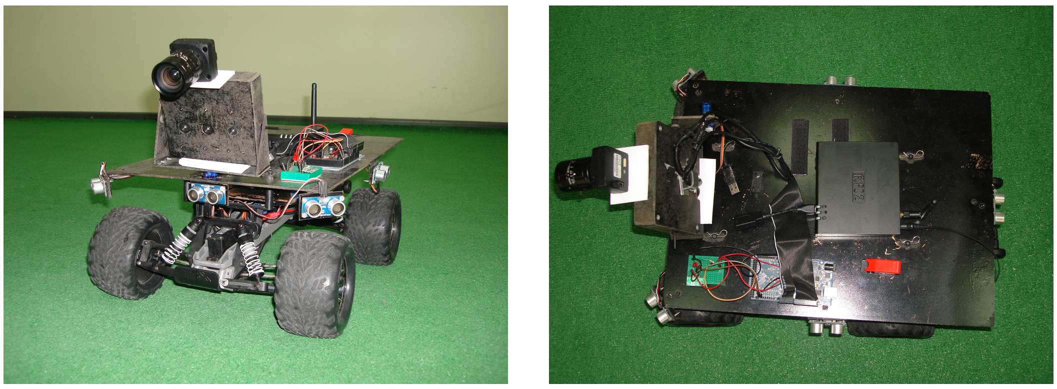


Figure 2.1: Robot with installed components

2.1 Components of the System

2.1.1 Model Car

Our hardware system is built on a Traxxas Stampede model car (Figure 2.2). It is equipped with a waterproof steering servo and an electronic speed controller. Stampede is a powerful model car, its maximum speed is more than 30mph. Also, it is capable of carrying extended equipment such as useful payload (e.g. PC, battery). From now on, the model car will be referred as “robot” during this thesis.

Figure 2.2: Traxxas Stampede

In the beginning of this study we realized that the robot was too fast for our needs. In order to slow down the robot and increase its torque (it becomes easier to control), we have replaced its spur gear (driven gear) with a bigger one. Equation 2.1 denotes the calculation of the gear ratio. As the gear ratio increases, the torque of the robot increases. Stampede’s original spur gear has 86 teeth while the one that we mounted on it has 90 teeth. Also, we replaced its pinion gear (drive gear) that is turned by the motor of the robot with a smaller one for the same purpose but the gears were stripped due to a possible incompatibility with each other. Figure 2.3 shows the placement of gears.

$$GearRatio = \frac{DrivenGearTeeth}{DriveGearTeeth} \tag{2.1}$$

2.1.2 Arduino Board

Arduino [15] is employed as an interface between the robot’s servos, sensors (e.g. magnetometer, ultrasonic sensor) and the onboard computer. Arduino is a microcontroller card that communicates with computer through serial port via

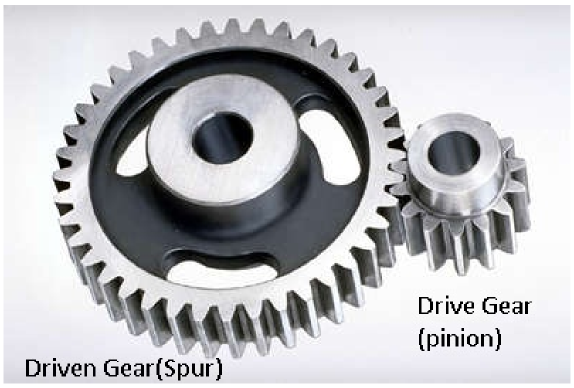


Figure 2.3: Gear placement

USB connection. Arduino Mega[1] has ATmega1280 processor on it where its flash memory is 128KB.

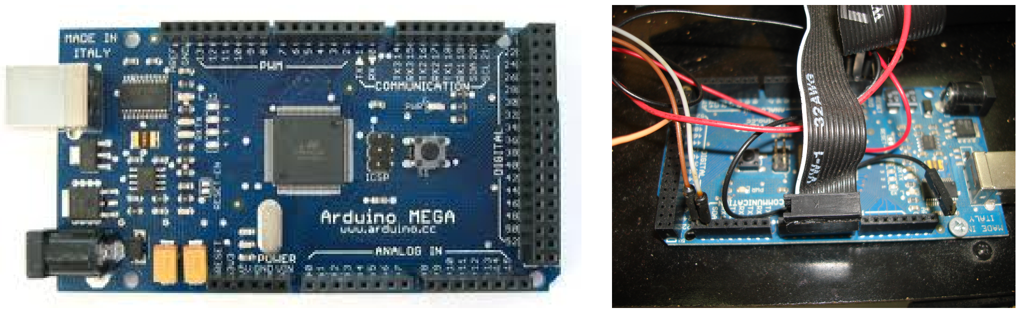


Figure 2.4: Arduino Board

Arduino board has both digital and analog pins placed on it to connect sensors, servos, leds or even LCD panels. The sensors and servos have fixed pin locations that are assigned by us. For example, the motor servo is connected to eleventh pin whereas the direction servo is connected to tenth pin of the Arduino board. The board gets its power through USB connection or it can be connected to a 5V external power source.

In our hardware design, the steering servo and motor servo of the robot are connected to the Arduino board. In addition, 6 ultrasonic distance sensors, a 3-axis magnetometer and a led is connected to the board. The program that we implemented for Arduino runs on the board and it will be investigated in detail in Section 3.2.

[1]http://arduino.cc/en/Main/ArduinoBoardMega/

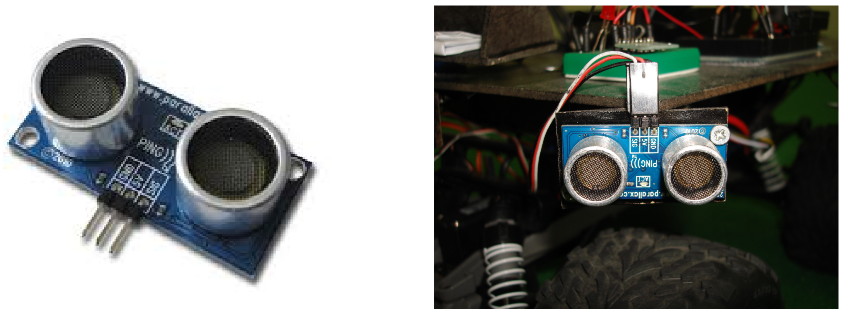


Figure 2.5: Ultrasonic range finder

2.1.3 Ultrasonic Range Finder

Ultrasonic range finder is used to detect obstacles in an environment. It generates and transmits a high frequency wave (40kHz) then wait for the reflected wave from the surface. When the transmitted wave returns (i.e. echo), the sensor calculates the distance to the obstacle by using the elapsed time that has passed while waiting for the echo. In this thesis, Parallax Ping[2] ultrasonic sensors are used (Figure 2.5).

The reason why we have used ultrasonic range finders in this thesis is that they are considerably low-cost comparing to its counterparts (e.g. lidar, laser scanner), and there is no need to do much processing to obtain the output.

2.1.3.1 Limitations of Ultrasonic Range Finders

The Parallax Ping sensor is capable of detecting the objects in the range of 2-300cm. However, the placement of the sensor is an important issue. If it is mounted too low on the robot, the sensor can detect the floor.

Moreover, the angle between the sensor and the obstacle is important in order to achieve a successful detection. With an inappropriate angle, the echo does not return to the sensor (Figure 2.6). Also, the objects that we are aiming to

[2]http://www.parallax.com/Portals/0/Downloads/docs/prod/acc/28015-PING-v1.6.pdf

detect must have reflective surfaces in order to send the echo back. In [10], Borenstein and Koren investigated the limitations of the ultrasonic distance sensors for obstacle avoidance task.

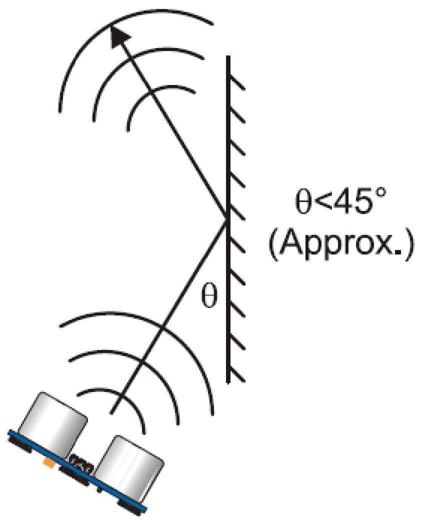


Figure 2.6: Inappropriate sensor reflection angle[2]

2.1.4 Electronic Compass

In this thesis, we used an electronic compass which is named as "tilt compensated electronic compass" (Figure 2.7). This compass produces a value in range [0-360] which denotes the angle between north and its position. Also, this heading value does not get affected by changes from tilt variations. Tilt compensated electronic compass is capable of measuring the acceleration and the magnetic field in 3 axes (x,y,z). The accelerometer is responsible for measuring the tilt angles of pitch and roll axes in order to provide tilt compensation. Magnetometer is responsible to measure the earth's magnetic field to calculate heading angle.

However, earth's magnetic north and geographical north are not the same concepts. There's a declination angle up to 20 degrees between them. We did not consider that difference because we aimed to move the car in a steady angle. Locating true north was not our primary concern.

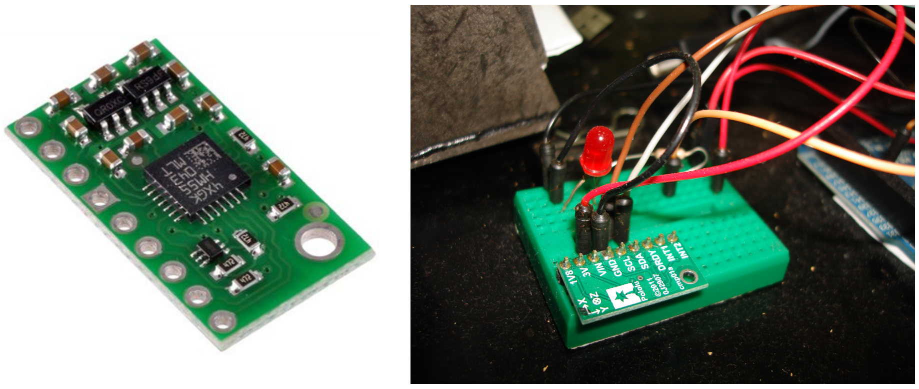


Figure 2.7: Tilt compensated electronic compass

2.1.5 Fit PC

Onboard PC is the core component of our hardware design. We installed a lightweight PC (Fit PC) on the robot platform (Figure 2.8). Arduino board and cameras are connected to that PC. Fit PC has 4 USB ports, 2 ethernet ports and a wireless adapter that allows us to connect it remotely. Fit PC has an Intel Atom Z530 processor working at 1.6GHz and its installed memory is 1GB.

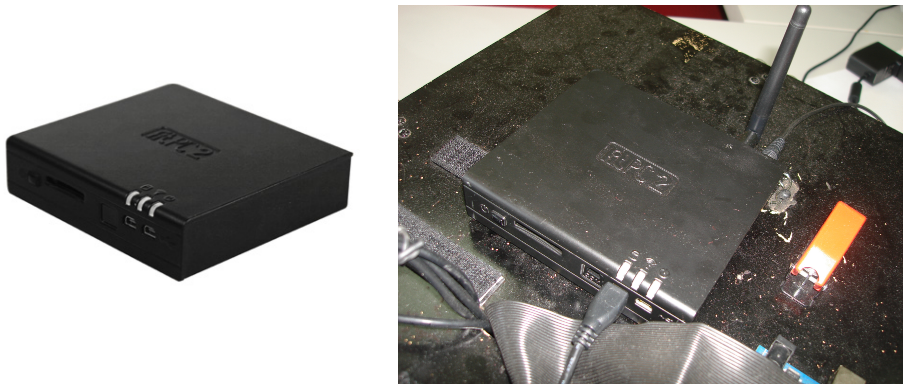


Figure 2.8: Fit PC

2.1.6 Power Source

Lithium-ion polymer (LiPo) batteries are used as power source. The battery supplies power for both the PC and the car where it has 3 cells and its capacity is 4000mAh. Fit PC consumes 6W under low CPU load and 8W under full CPU load. Uptime of Fit PC varies between 2.5 hours and 8 hours depending on the CPU load. The robot's power consumption is difficult to determine since it varies under different speeds.

LiPo batteries are lighter than other batteries which have the same capacity. This is a major advantage since it decreases our load on the robot. Also, LiPo can feed robot's motor with high amount of amperes. Moreover, LiPo batteries do not loose charge significantly when they are stored for long periods (e.g. up to 4 months).

2.1.7 Cameras

Cameras are the most critical components of our system since our method relies on visual information (Figure 2.9). During this thesis study, different models of cameras are used which are A4 Tech Webcam, Kinect, Sony DSC N1, Point Grey Chameleon. In the end, we see that CCD cameras produce much quality images than the CMOS cameras (e.g. webcam). Figure 2.10 shows images that are acquired by different cameras. Operations based on feature matching [16, 17] require the detection of the interest points on the image, or optical flow [18] requires the detection of the same points (interest points) in the consecutive frames. The webcam is not so successful for the detection of such points, while a CCD camera is preferable.

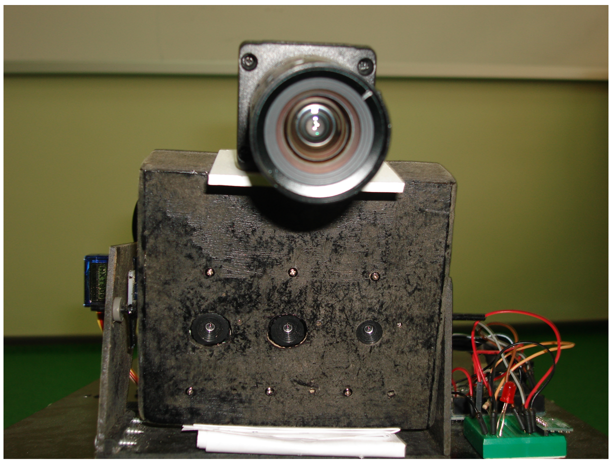


Figure 2.9: Cameras on the robot

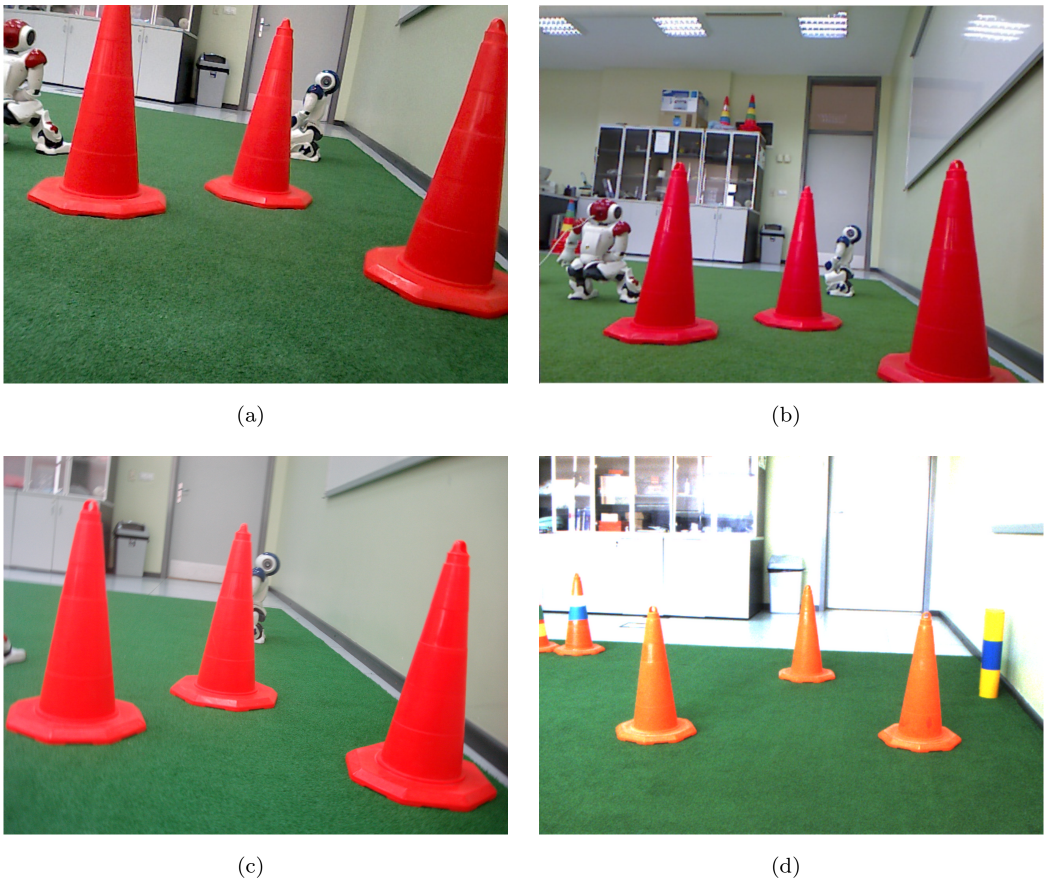


Figure 2.10: Images acquired by different cameras: (a) A4 Tech Webcam (CMOS) (b) Kinect (CMOS) (c) Sony DSC-N1 (CCD) (d) Point Grey Chameleon (CCD)

Chapter 3

Software System

In this chapter, our software structure will be discussed. There are two parts of our software system which are Arduino program (Section 3.2) and multi-threaded main program (Section 3.1). Arduino source code runs on the Arduino board where our main program runs on the onboard computer (Figure 3.1). The programs are needed to control the robot by sending commands to its servos and to get readings from the sensors which are installed on the robot.

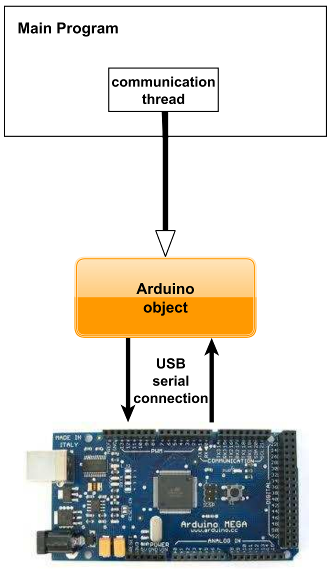


Figure 3.1: Software system

3.1 Multi-threaded Main Program

In our main program, several threads are employed to run simultaneously. The main threads in the program are; heartbeat thread, communication thread, vision thread, driver thread and move thread. All the threads in the program are integrated into each other and work coherently. Figure 3.2 shows the block diagram of our main program.

In the program, we keep a queue called "command queue" in which the threads add command-codes of their desired operations. These commands are move forward, move backwards, steer and move, send heartbeat and read-log commands.

Communication thread: The communication thread is responsible for executing the commands in the command queue by signalling the corresponding threads. When there's no command to execute in the command queue, the communication thread signals the Arduino object functions to read sensor data. Using this approach, our main program always have the most recent sensor readings that is possible.

Heartbeat thread: Our Arduino program awaits for a special code to continue its operation which we call it as heartbeat. The heartbeat thread is utilized for keeping alive the heartbeat mechanism. Heartbeat thread writes the heartbeat command in the command queue at every 1000ms. Arduino board executes its functions as long as it receives the heartbeat command. For that reason, the heartbeat thread has crucial importance in order to keep the Arduino program alive.

Driver thread: The driver thread receives necessary input data (e.g. free directions in the scene) from vision thread and determines the optimum steering angle for the robot. It adds moving and steering commands in the command queue based on the angle values that it is calculated. Calculation of these angles will be discussed later in Chapter 5.

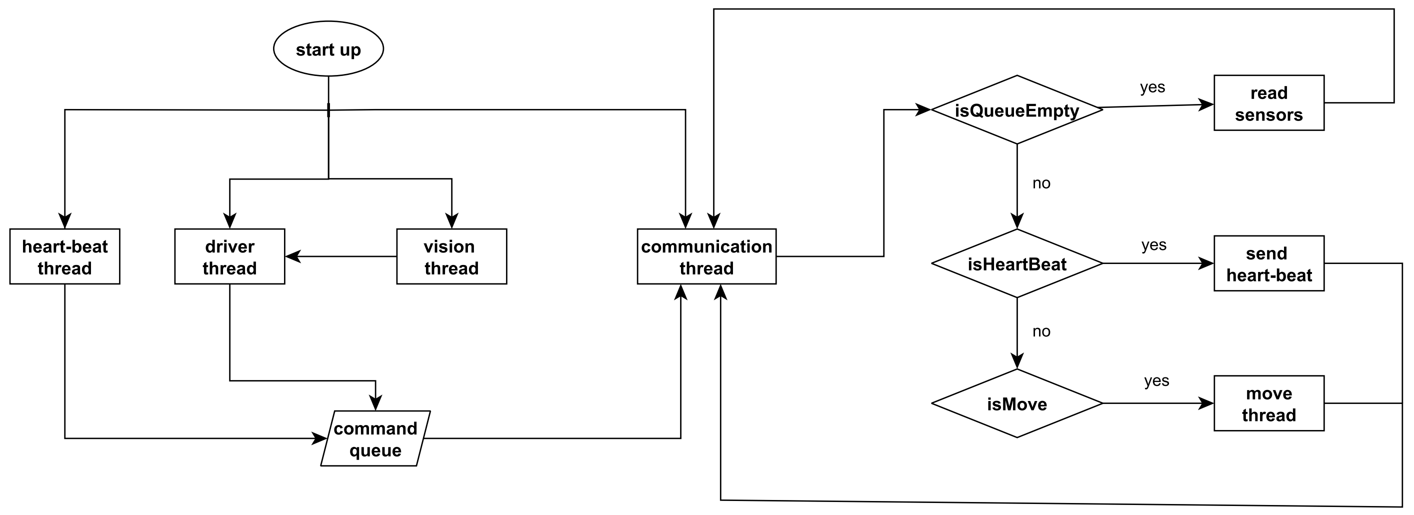


Figure 3.2: System block diagram

Vision thread: The vision thread handles the visual processing (obstacle detection) and generates an output that indicates the free directions (left, center, right) in the scene. It works in coordination with the driver thread. Detailed explanation of visual processing will be discussed in Chapter 4.

Move thread: The move thread is responsible for keeping the track of the robot's states (e.g. stopping or moving). Also, it moves the robot along the given steering angle that is determined by the driver thread. This thread uses predefined speeds for forward and backward directions while moving the robot. A move operation comprises one step of the robot with given steering angle during a constant step time (500ms). Move thread is signaled by the communication thread.

3.1.1 Lock Mechanism

Some of the variables and data structures are shared by several threads in the program (e.g. command queue). Since the threads are working simultaneously, it is possible that they try to access to the same data fields at the same time. However, these conditions cause deadlock and the program becomes unresponsive or even it may crash. For example, heartbeat thread and driver thread push commands into the command-queue where communication thread pops these commands from command queue. To prevent multiple accesses to the command queue, we employed a lock mechanism. We defined a variable named "queue_locked" and set its initial value as false. A thread can only access to the command queue if the value of "queue_locked" is false. This variable is also declared as "volatile" to prevent the compiler to make optimizations on this variable [19]. Figure 3.3 denotes the access to the command queue for a single thread.

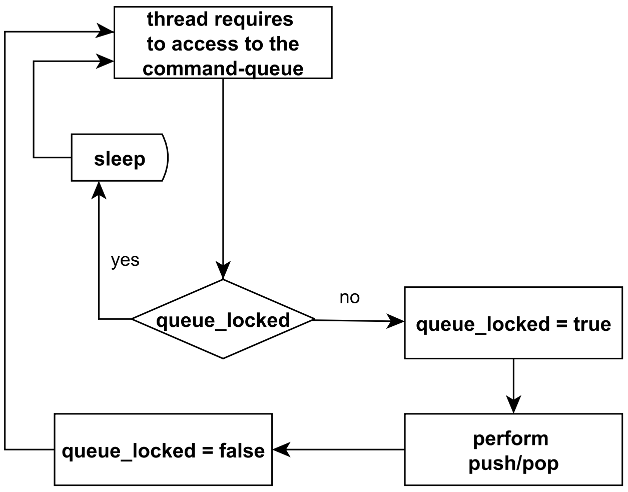


Figure 3.3: Lock and access mechanism of the command-queue

3.2 Arduino Program

The Arduino programming language is based on C,C++ programming languages. However, the syntax of the language is slightly different from C and C++. Arduino has its own development environment [15]. The development environment (Figure 3.4) is responsible from verifying the correctness of the source code and uploading the verified source code into the Arduino board via USB serial connection.

Arduino SDK has many libraries to support different functions for reading sensors, controlling servo motors and communicating with computers over the serial port. For example, Servo library contains functions to control servo motors and Wire library provides functions to send, receive data from sensors such as electronic compass [15].

Our Arduino program reads ultrasonic range finders' outputs, electronic compass' outputs (Figure 3.5) and send these data to the onboard computer using USB serial connection (Figure 3.1). In addition, it controls the motor servo and direction servo of the robot. First it reads the direction and speed values from serial port which are sent by the main program then sets the speed or direction

Figure 3.4: Arduino development environment

accordingly through assigned digital pins on it. The pin values are encoded via a PWM (pulse-width modulation) signal [20]. For once, we calibrated the robot's ESC[1] (electronic speed controller) with speed values that are sent from Arduino by setting the value to its corresponding pin.

3.2.1 Arduino-Main Program Communication

At the initialization step of the Arduino program, pin numbers of the components that are connected to the board must be declared. As an example, the ultrasonic range finder on the center is connected to the second digital pin where the speed servo is connected to the eleventh digital pin on the board.

Arduino board communicates with the onboard PC over the serial port (Figure 3.1). Arduino is capable of reading/writing one byte at a time from/to the serial port. Every action to be performed by Arduino is transferred as commands according to our design. We designed specific codes for different commands. All

[1]http://traxxas.com/support/Programming-Your-Traxxas-Electronic-Speed-Control

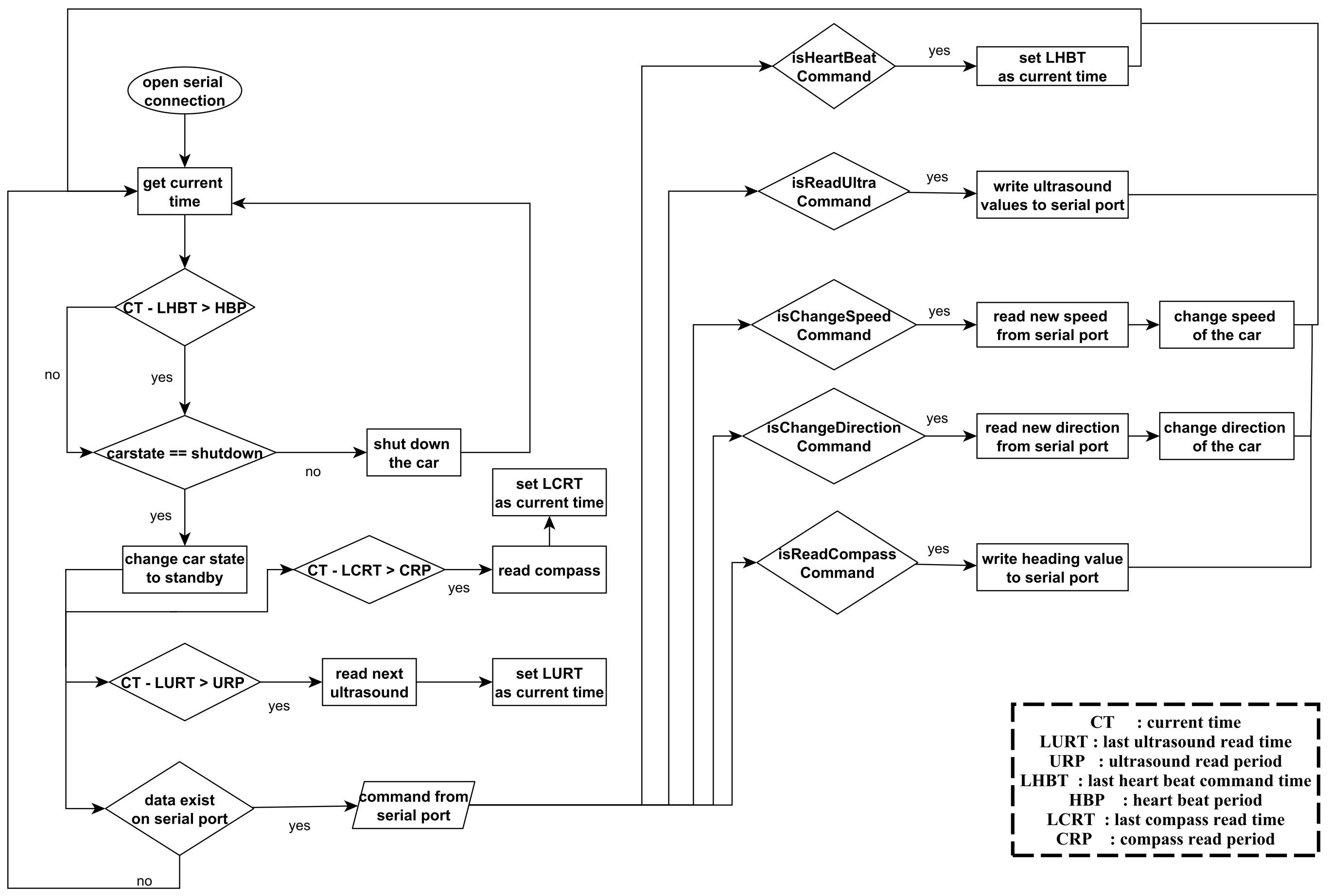


Figure 3.5: Arduino program block diagram

of the commands are in the range of byte data type which vary between 200-212. For example, the command 206 is assigned for changing the speed and the command for reading ultrasound values is 202.

There are two different types of actions that our Arduino program performs. These are reading and writing operations. Reading operations are reading ultrasonic range finder and electronic compass values. Writing operations are writing the speed and the direction into the robot's servos. We used a two step mechanism to implement write procedures. When Arduino board receives a write command value (e.g. 207 for write direction) from main program then it reads two bytes afterwards from the serial port. These two bytes are converted to an integer value that indicates the direction or the speed value which will be sent to the servos. Similarly, the board cannot send sensor values directly because the values are sometimes greater than 255 which is the upper limit for byte data type. Therefore, the integer value is split into two bytes and then sent over the serial port byte by byte.

3.2.2 Heartbeat Mechanism

Normally, the uploaded source code inside the Arduino board starts to run when it is plugged into a power source (e.g. USB or external power). However, we figured out that this continuous operation is unnecessary and developed a mechanism called "heartbeat" which is also used in similar systems such as MAVLink[2]. Our Arduino program awaits for a special code from the main program to continue its operation. This code is designed as the heartbeat signal and must be sent by the main program every 1000ms. Otherwise, Arduino system goes to sleep state. Heartbeat mechanism prevented continuous uninvoked sensor readings. Arduino executes its functions (e.g. reading sensors) as long as it receives the heartbeat command-code from the serial port. Otherwise it does not perform any uninvoked operations.

[2]http://qgroundcontrol.org/about

The status of the Arduino board can be tracked from a led that we have placed on the robot. The red led blinks continuously if there's no heartbeat command. This mechanism is quite useful considering two situations. Firstly, the sensors on the robot work only at desired times and this increases the lifetime of the sensors. Secondly, Arduino board resets itself at start up and during the termination of the serial port connection. At this reset procedure, the board writes some random values to the digital pins and the robot moves out of control (e.g. immediate acceleration or move backwards). Using heartbeat mechanism prevents this failure that is caused by resets and provides us a safe and a totally controlled navigation of the robot.

3.2.3 Debug Mode of Arduino

Many times we wanted to make sure that Arduino program is operating correctly. For that purpose, we implemented a log mechanism. In the debug mode of the Arduino, we keep a buffer (log-buffer) and fill that buffer with the values which are read from serial port by Arduino. When the read-log request (command 212) is received from the main program, we send that log-buffer over the serial port. When the values in the log-buffer are received by the main program, they are written to a text file. We can trace the problem by examining this text file. The debug mode on the Arduino program can be activated by changing the value of the "debug" variable to "true".

Chapter 4

Vision

Obstacle detection is one of the keystones of this study. In order to navigate the robot without crashing, the obstacles should be detected and avoided during the navigation. During this thesis, we focused on two different methods for obstacle detection. In the first approach, we tried to model the obstacles using their appearances. In Section 4.1, we will discuss the obstacle detection method that uses models of obstacles. Thus, we aimed to detect only orange colored traffic cones in the environment whose colors are significantly different from the ground plane. Ground plane is a term that we use instead of the passable regions for the robot.

The second major approach that we studied is to detect the obstacles using ground plane models. In this approach, the assumption is that ground plane differs from the obstacles in terms of their appearances. However, the obstacles are not required to be in the same color. In Section 4.2, we will discuss 2 different obstacle detection algorithms which uses models of the ground plane. The first algorithm we implemented belongs to Ulrich and Nourbakhsh [2] which is based on histograms of a reference area. The reference area is a region in front of the robot which is assumed to be free of obstacles and has the ground plane characteristics in terms of the appearance. The other algorithm in Section 4.2.2 is developed by us which uses individual pixel models for each pixel location to model the ground plane.

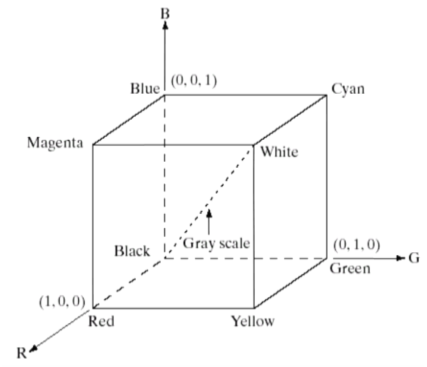


Figure 4.1: RGB color cube (taken from Gonzalez & Woods [21])

4.1 Obstacle Detection by Obstacle Modeling

In RAVLAB, we have several orange colored traffic cones. We use these traffic cones for landmark detection, obstacle detection purposes, or we form paths for robot navigation by placing them on the testing pitch. From now on we will use traffic cone and cone words interchangeably. The color of the cones are significantly different than the ground plane. The carpet on the laboratory's floor is green, floor tiles are grayish where the traffic cones are light reddish or orange. We aimed to detect the orange colored traffic cones in the environment.

Initially, we split the RGB input images into red, green and blue channels. Each RGB color pixel is represented by a color triplet (r, g, b) [21]. Figure 4.1 shows the color cube of the RGB color space. Since the traffic cones seem reddish in appearance, the simplest approach for detecting cones is to apply a threshold value (i.e. 200) on the intensity of the red channel of the input image. The reddish appearance of the traffic cones is due to higher intensity in red channel. Employing a threshold value on the red channel of the image results in a successful detection of the traffic cone (Figure 4.2), but bright areas in the image are also marked as traffic cone. Because, white pixels also have higher values on the red channel. Pure white is expressed with (255,255,255) triplet in RGB color space using 8 bits. Figure 4.2(b) shows the similarity between bright areas and the cone pixels on the red channel.

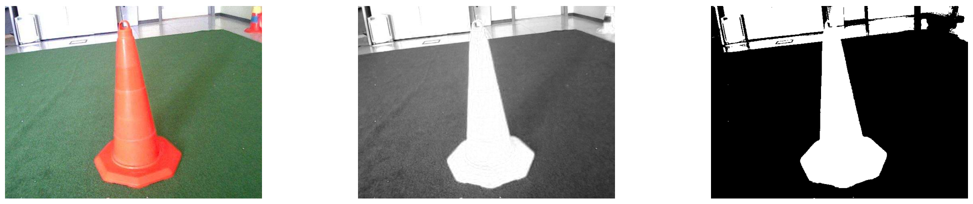

(a) (b) (c)

Figure 4.2: Applying threshold on the cone image: (a) Input image (b) the red channel of the image (c) binary image obtained by thresholding red channel by $th = 200$

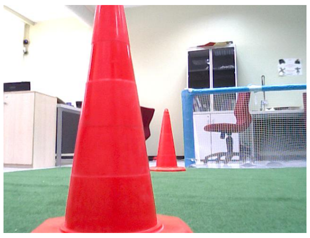
4.1.1 Modelling Traffic Cones

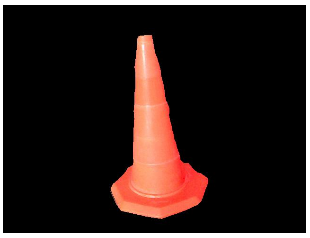
We observed that the first approach (applying threshold on the red channel) was quite successful to extract the cone pixels in the image. However, as shown problems existed related to the bright pixels in the image. We decided to model the traffic cones based on their appearances. As a first step, we took several photos of the traffic cones in indoor environment. We intentionally changed the shooting angle and tried to change the illumination conditions at every image that is taken. After shooting 56 photos which include traffic cones, we produced manually segmented cone masks for every image (Figure 4.3). After isolating cones from the environment, we calculated the mean values of the masked-cone images for 3 channels(red, green, blue). The mean values (mR, mG, mB) were 224, 56, 61 for red, green and blue channels respectively. We call these mean values (mR, mG, mB) as "reference mean values". As it's expected, the mean value of the red channel significantly greater than the other channels.

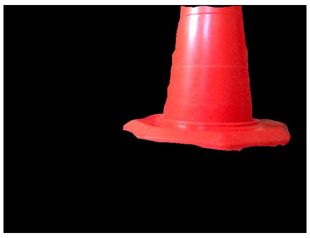
In addition, we calculated the histograms of both the segmented cones and the rest of the environment (Figure 4.4). The only useful information that we can extract from histograms is, red component of the cone pixels is usually above a threshold. We cannot develop any discriminative rules related to the green and blue channels by looking at their histograms since the values are spread in a large interval. These histograms are accumulated from 56 images including traffic cones in every image.

(a)

(b)


(c)


(d)


(e)

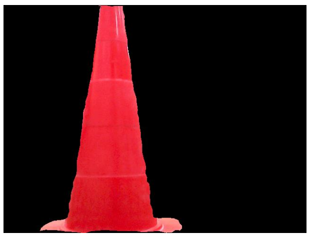

(f)

Figure 4.3: Segmented cones: (a)-(b)-(c) are input images (d)-(e)-(f) are corresponding segmented outputs.

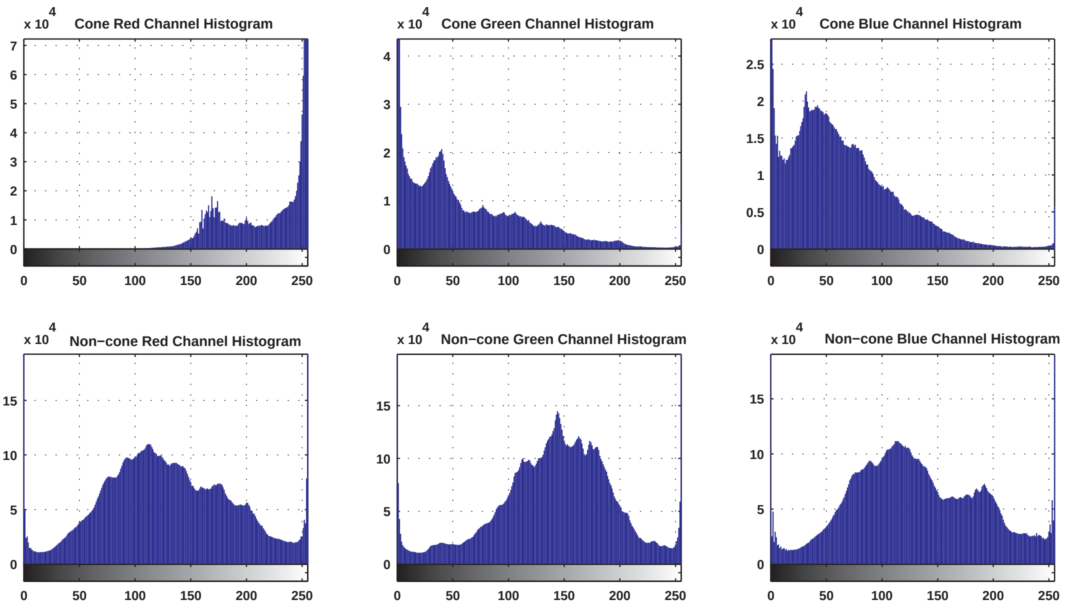


Figure 4.4: Cone and non-cone histograms: The first row shows the histograms of traffic cones. The second row shows the histograms of surrounding environment.

4.1.2 Detecting Traffic Cones by Using the Cone Models

We developed an algorithm based on the assumption that cone pixels are closer to their mean values than a threshold. This closeness is determined by a similarity metric. To mark two pixels as similar, all of the three channel's values must be closer to the reference mean values than a distance threshold R. Algorithm 1 shows the steps of producing binary cone map from the RGB input image where **rCh**, **gCh**, **bCh** denote red, green and blue channels of the input image respectively.

Algorithm 1 Cone pixel detection using pixel similarity

```
$W \leftarrow width\,of\,image$
$H \leftarrow height\,of\,image$
for $i = 0$ to $WxH$ do
  if $|\mathbf{rCh}[i] - mR| < R$ and $|\mathbf{gCh}[i] - mG| < R$ and $|\mathbf{bCh}[i] - mB| < R$ then
    $output[i] \leftarrow cone\,pixel$
  else
    $output[i] \leftarrow non - cone\,pixel$
  end if
end for
```

As it can be seen in Figure 4.5, that method works quite well, but sometimes we can classify other non-cone pixels (e.g. red chair) as cone. Also, determining R value is a challenging task. If the distance threshold R increases, the accuracy of detecting cone pixels increases. On the other hand, the chance of classifying non-cone pixels as cone pixels increase if we increase R. R is dependent on the overall illumination of the scene, so it's not possible to choose a single distance threshold value for all the images under different conditions. We set R as 60 in our experiments. However, this R value can be different for each channel with respect to the standard deviation of the cone models.

Another approach that we have tried for cone detection purpose is to use the distribution of the cone models. We inspired from the theory of the discrete random variables and probability mass function (pmf) [22].

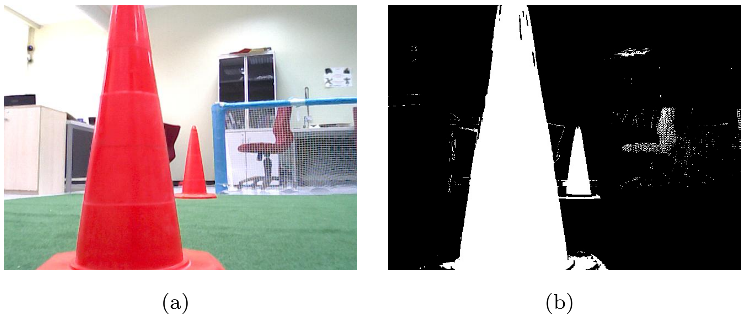


Figure 4.5: Cone detection using distance threshold: (a) Input image (b) corresponding binary output

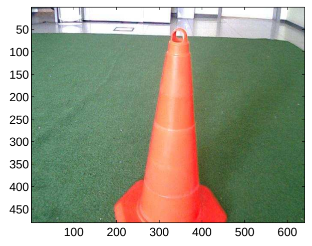
We treated every pixel value $P(x)$ in the histograms as a probability of a discrete variable and set of probabilities can be expressed in terms of pmf. For discrete random variables, the probabilities $P(x)$ must satisfy the following conditions.

$$P(x) \geq 0 \qquad and \qquad \sum_{x \, \epsilon \, X} P(x) = 1 \tag{4.1}$$

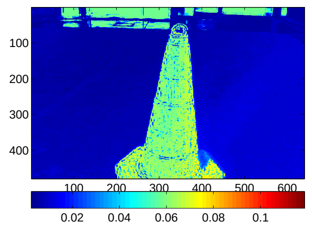
To satisfy the conditions above, we normalized the histograms using Equation 4.2 where *hred*, *hgreen* , *hblue* denote red, green, blue channel histograms of the cone model and P_R, P_G, P_B denote pmf of each channel.

$$\begin{aligned} P_R &= hred/sum(hred) \\ P_G &= hgreen/sum(hblue) \\ P_B &= hblue/sum(hblue) \end{aligned} \tag{4.2}$$

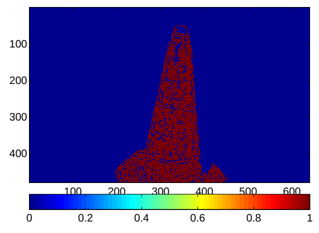
In order to create probability images, we use Equation (4.3) where I represents the input image and PI is the probability image. The bin value of the normalized histograms approximately represents the probability density of that pixel belongs to the cone. Figure 4.6 shows the probability image and the corresponding output image. The output image is obtained by applying a threshold on the probability image. However, determining the threshold value is very challenging.



(a)



(b)



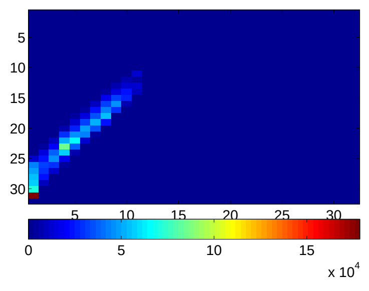
(c)

Figure 4.6: Cone detection using probability: (a) Input image (b) probability image (sum of r,g,b) (c) output image obtained by thresholding probability image by $th = 0.058$

$$PI(x, y) = P_R(I_R(x, y)) + P_G(I_G(x, y)) + P_B(I_B(x, y)) \tag{4.3}$$

Moreover, we run some tests using 2D red-green histograms on normalized red-green images. In order to work on the normalized channels, we used the common equation 4.4 to obtain normalized red, green channels. Note that, normalized blue channel is redundant since it is dependent on the other normalized channels [23].

$$\begin{gathered} r = R\,/\,(R + G + B) \\ g = G\,/\,(R + G + B) \\ b = 1 - r - g \end{gathered} \tag{4.4}$$


(a)

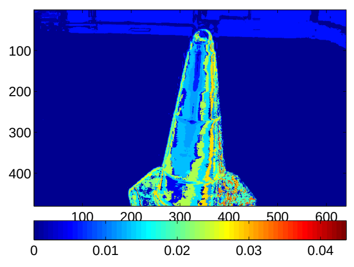

(b)

Figure 4.7: (a) 2D red-green histogram (b) probability image determined by using 2D histogram

In 2D histograms, we keep histogram values for both red and green images together whereas in single dimension histograms a bin corresponds to a single channel's value. 2D red-green histogram has 32x32 bins as shown in Figure 4.7(a). We determined the probability images regarding the same concept above. Figure 4.7(b) illustrates the probability image that is obtained by using 2D red-green histogram.

Although normalized r and g (2D histogram) distinguishes better the cone from homogenous green ground plane, it performs worse in gray colored plane. Figure 4.8 shows different cone images with their probability outputs that are calculated by using 1D and 2D histograms.

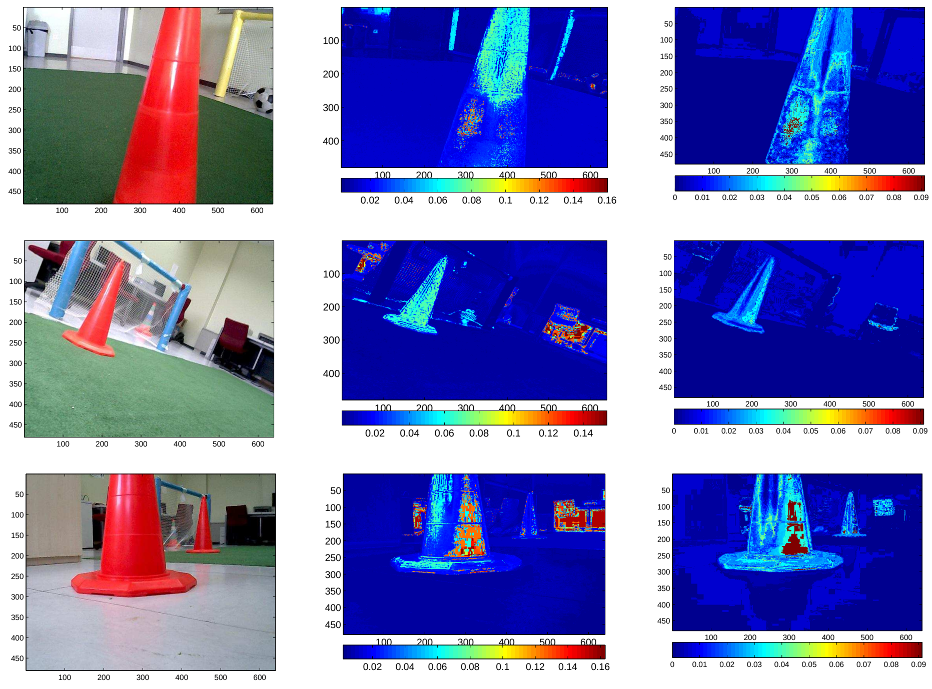


Figure 4.8: Input-1D histogram output-2D histogram output: Input image, 1D histogram output and 2D histogram output triplets are shown in left, center and rights columns respectively.

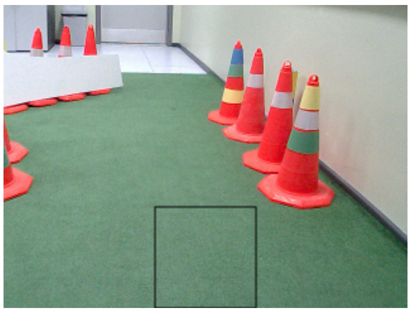


Figure 4.9: Obstacle free region

4.2 Obstacle Detection by Ground Plane Modeling

In this section, two different obstacle detection techniques will be discussed. Both of these techniques use ground plane models in order to detect obstacle pixels. The first method in Section 4.2.1 belongs to Ulrich and Nourbakhsh [2] and based on the usage of ground plane histograms which are accumulated from the obstacle free region (OFR) in front of the robot. The other technique in Section 4.2.2 is developed by us and uses individual sample sets for each pixel location to model the ground plane instead of histograms. As we mentioned in Chapter 1, there is not a generic definition for obstacles. Considering that condition, it is not possible to construct a system without any assumptions. Ulrich and Nourbakhsh assumed an OFR in front of the robot as we did when developing our algorithm (Figure 4.9). The another assumption is that obstacles differ from the ground plane in terms of their appearances. After examining both of these techniques in the following sections, their performances will be compared in Section 4.2.3.

In the implementations of the obstacle detection algorithms, we used two different color spaces which are HSV and normalized RGB. Calculation of H-S-V values can be found in various texts and Appendix A. We implemented the algorithm in Section 4.2.1 in HSV color space, and the algorithm in Section 4.2.2 in normalized RGB color space.

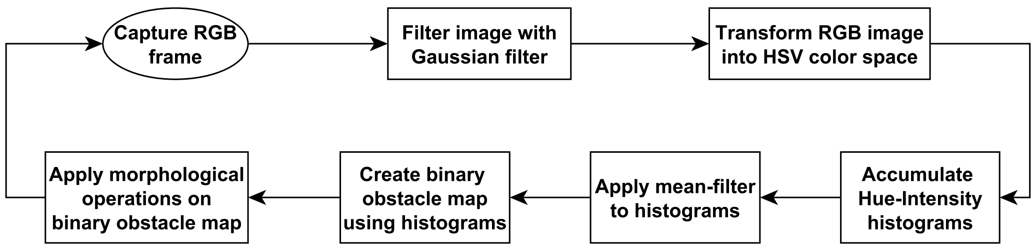

Capture RGB frame
Filter image with Gaussian filter
Transform RGB image into HSV color space
Accumulate Hue-Intensity histograms
Apply mean-filter to histograms
Create binary obstacle map using histograms
Apply morphological operations on binary obstacle map

Figure 4.10: Block diagram of histogram based obstacle detection

4.2.1 Obstacle Detection based on Histograms

In this section we will discuss an obstacle detection algorithm which is a simplified version of Ulrich and Nourbakhsh's work [2]. We used their logic during the implementation, but we did not stick to the original work [2]. The technique is based on the appearance of individual pixels and also have some assumptions. The basic assumptions are:

- Obstacles are not similar to the ground plane in terms of their appearances
- The ground is relatively flat
- There are no obstacles on the reference area

The algorithm is based on Hue and Intensity histograms. These histograms are accumulated from obstacle free region (OFR). To classify a pixel whether if it belongs to the ground plane, a simple check on that pixel's value from the histogram's corresponding bin is enough. If the bin value is above a threshold value, then the pixel is marked as ground plane. The block diagram in Figure 4.10 summarizes Ulrich and Nourbakhsh's technique.

The original algorithm in [2] is implemented in HSI (hue, saturation, intensity) color space while we implemented it in HSV (hue, saturation, value) color space as the library that we have used (OpenCV [24]) provides RGB to HSV conversion function. After acquiring an input frame from camera, the noise on the image is reduced by applying Gaussian filter [21]. Then, the RGB input frame is transformed to HSV color space.

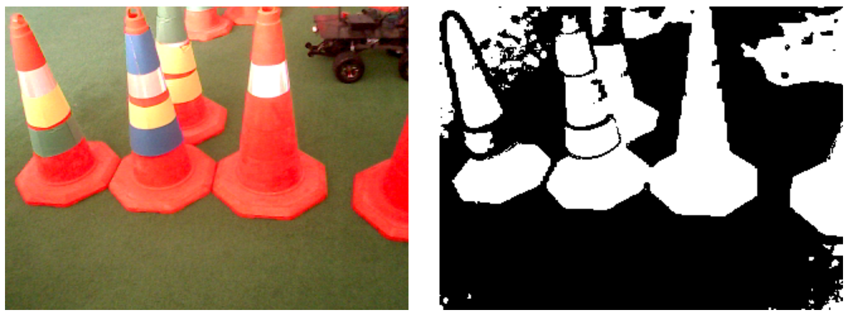
4.2.1.1 Histograms and Obstacle Map Creation Process

Initial hue and intensity histograms are created using only the first frame. At each iteration the histograms are filtered with an average filter using the method in 4.5 where i denotes the index of the histogram and fil_hist is the average filtered histogram, and the average filter size is selected as 3.

$$fil\_hist(i) = [hist(i-1) + hist(i) + hist(i+1)] \; / \; 3.0 \qquad i = 1, \ldots, 254 \quad (4.5)$$

There are two different types of histograms in the algorithm. These are candidate and reference histograms. The algorithm aims to have a background information about the ground plane appearance while keeping candidate histograms. We keep 10 candidate histograms, 5 for hue values, 5 for intensity values. Also, there are 2 reference histograms (hue and intensity).

At every 10 frames, we fill candidate histograms using the pixels in OFR. Meanwhile, we get the heading information from electronic compass and keep it with the corresponding candidate histogram. The heading value is important when forming reference histogram from candidate histograms. At the creation time of the reference histogram, the heading value that is obtained from compass is compared with the heading values assigned to each candidate histogram. The histograms that does not satisfy heading (odometry) condition are eliminated and does not included to the reference histogram. The maximum value for each bin among the remaining candidate histograms forms the reference histogram. The heading values that we collect with candidate histograms inform us about the placement of the robot. If the robot makes a substantial amount of turn after accumulating the histogram, we assume that the appearance might have changed after that point and we do not include that candidate histogram to our model.



(a) (b)

Figure 4.11: Histogram based method input-obstacle map pair: (a) Input frame (b) binary obstacle map

In the original algorithm, Ulrich and Nourbakhsh applies a two pass check on the candidate histograms. In the first pass, they eliminate the histograms whose orientation differs than 18 degrees. In the second pass, they include the histograms to model if the robot could successfully moved 1 meter after accumulating the corresponding histogram.

Only reference histograms are considered when constructing the binary obstacle map. A pixel is classified as ground plane if the hue histogram's corresponding bin value is more than hue threshold and intensity histogram's bin value is above the intensity threshold.

4.2.1.2 Morphological Operations on Obstacle Map

Mathematical morphology aims analysing the shape and form of objects [25]. Morphological operations can be applied to gray scale and binary images. We used two morphological operations in order to get rid of noisy pixels or pixel groups on the binary obstacle map. We first applied filling hole (closing hole) [26] operation to remove connected components that do not touch the image border.

After applying hole filling operation (Figure 4.12(b)), we applied area-open morphological operation. Area-open [26] removes blobs in the image if their sizes are below than a given threshold (Figure 4.12(c)). The threshold metric is defined

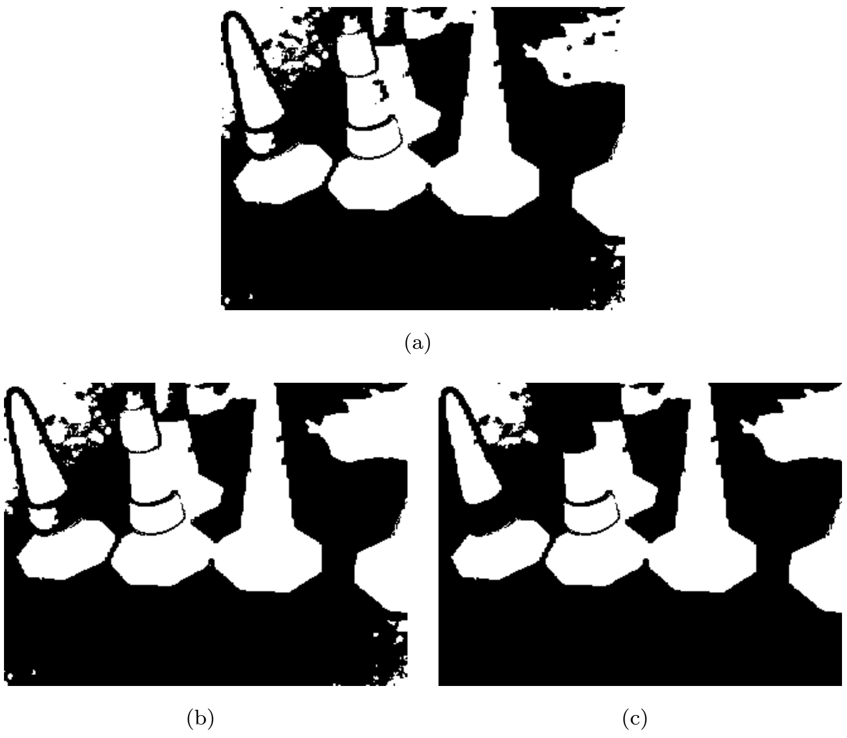


Figure 4.12: Morphology applied binary obstacle map: (a) Binary obstacle map (b) after hole closing (c) after area opening

as the number of pixels that the region covered. We set area-open threshold as 1000 pixels.

Note that, Ulrich and Nourbakhsh [2] did not used morphological operations on their work. However, we needed a cleaner obstacle map as we used this algorithm for the robot navigation in Chapter 5.

4.2.2 Obstacle Detection using Camera and Ultrasonic Sensor

Following the work of [2], our obstacle detection technique is based on the assumption that the square region in front of the mobile robot (OFR) is free of obstacles at the initialization step (Figure 4.9). In addition, our ground plane modeling scheme is inspired from ViBe [1] which is a background subtraction algorithm that is proposed for motion detection. ViBe keeps positive samples for every pixel location and classifies a previously unseen pixel according to its distance from stored samples. In other words, every pixel location has an independent classifier to decide whether a given candidate pixel for that location is from the ground plane or not. Through time, some random samples are updated with some random new pixels collected and classified from the scene. This update mechanism gives ViBe its dynamic adaptive behavior and robustness against local illumination changes, reflections and shadows. We modified this update behaviour by adding ultrasonic distance sensor check. We update sample sets if the ultrasonic sensor reading is free of obstacles. Also, we keep histograms of the sample sets and OFR, and we re-initialize the sample sets if necessary (see Section 4.2.2.3).

In addition, ViBe [1] is simple, efficient and faster than its counterparts which suits well to our robot where we cannot operate with high performance computers due to limitations of size and payload. In our obstacle detection algorithms, we used ViBe's logic to build the ground plane model which is based on the pixel locations.

4.2.2.1 Initialization

As in [1][27], we define and maintain a sample set $\mathbf{S}(x, y)$ for each pixel location (x, y) of the image $\mathbf{I}$ of dimensions WxH where W is width and H is height of the image. Each sample set $\mathbf{S}(x, y)$ stores N samples:

$$\mathbf{S}(x, y) = \{p_1, p_2, p_3, \dots, p_N\} \tag{4.6}$$

For initialization, in the first frame, random pixels are chosen from the obstacle-free square region (OFR) in front of the robot and their values are added to $\mathbf{S}(x, y)$ for every (x, y) in $\mathbf{I}$. The set of sample sets $\mathbf{S}$ whose size is $NxWxH$ forms our ground plane model. The size of OFR must be determined based on the width of the car and camera's field of view. If the region that we collect samples from gets larger, the chance of including obstacle pixels in the ground plane model increases whereas a smaller area would mean less variation in the model. Because the ground plane model **S** is created using only the first frame, the initialization is very fast. We operate this algorithm in normalized RGB color space. Normalization of an RGB image was discussed in previous sections and the process was shown in Equation 4.4.

4.2.2.2 Obstacle Detection

After initialization, each new frame is processed to detect ground plane and obstacle pixels. A given pixel value at location (x, y) is classified as an obstacle or not by examining its $\mathbf{S}(x, y)$. The classification is based on finding enough similar values to the pixel's own value in its $\mathbf{S}(x, y)$. Finding at least M similar pixels in $\mathbf{S}(x, y)$ is sufficient to mark the pixel in location (x, y) as ground plane pixel.

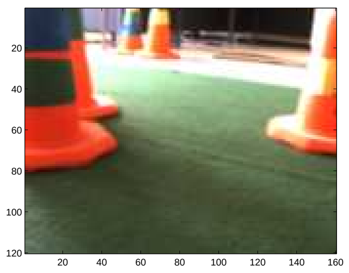
Let the pixel value at position (x, y) be q, then we can express the number of similar pixels (SPC) as follows:

$$SPC = \|dist(q, p_i) < R)\|, \quad where\ p_i \,\epsilon\, \mathbf{S}(x, y)\ , i = 1, ..., N \tag{4.7}$$

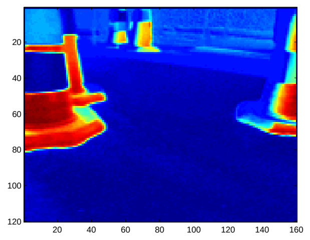
$dist$ is the Euclidean distance between pixels which is calculated as in 4.8, and R is the threshold that is applied on the difference of the pixels in order to decide the similarity of pixels.

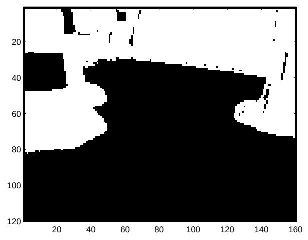
$$\begin{aligned} distRed &= \mathbf{q}_{red}(x, y) - \mathbf{p}_{red}(x, y) \\ distGreen &= \mathbf{q}_{green}(x, y) - \mathbf{p}_{green}(x, y) \\ dist &= \sqrt{distRed^2 + distGreen^2} \end{aligned} \tag{4.8}$$

Figure 4.13(a) shows an example input image where Figure 4.13(b) shows the corresponding distance image which is formed by the pixel values that indicate



(a)





(b) (c)

Figure 4.13: Obstacle Detection: (a) Input image (b) distance image (c) binary obstacle map

the distance between $\mathbf{S}(x, y)$ and corresponding pixel value q at location (x, y). Distance image is obtained by calculating average of N distance values between the samples in $\mathbf{S}(x, y)$ and q. After classifying each pixel of the input image $\mathbf{I}$, a binary obstacle map is generated (Figure 4.13(c)). White pixels denote obstacles whereas black pixels denote the ground plane.

4.2.2.3 The Update Rule of Sample Sets

ViBe applies a random update rule to the background model [1][27]. If the pixel q at position (x, y) is classified as background, its value is used to update a random value in $\mathbf{S}(x, y)$. The update decision depends on a term called "subsampling factor". For each new background pixel found, the probability of inclusion in the sample set is (1/subsampling factor) and decision is made using a random value. Moreover, one of the neighboring pixel's sample set of q is updated using the same probabilistic approach. Motivation here is that the adjacent pixels often share a similar temporal distribution [28].

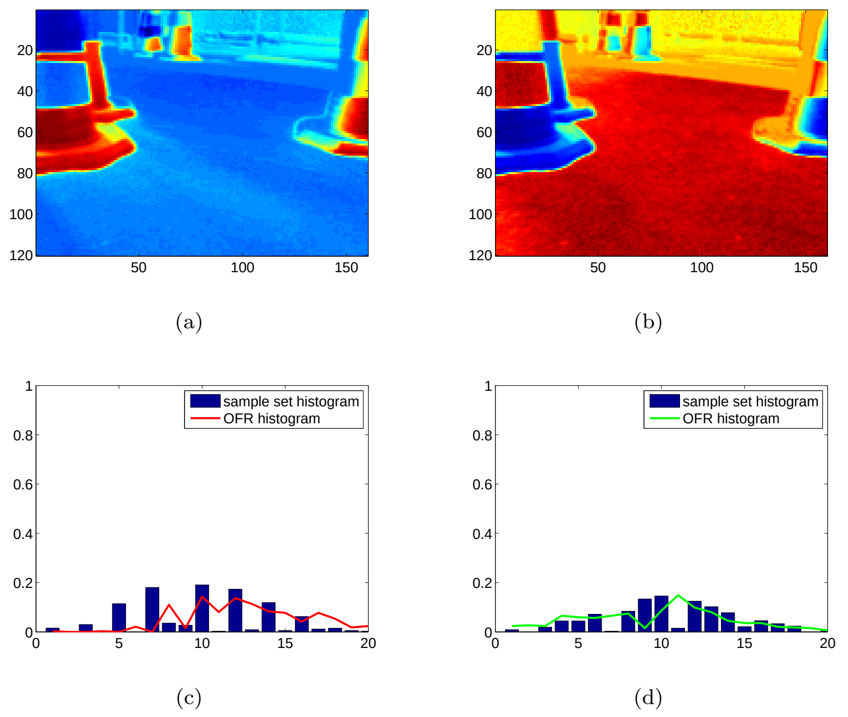
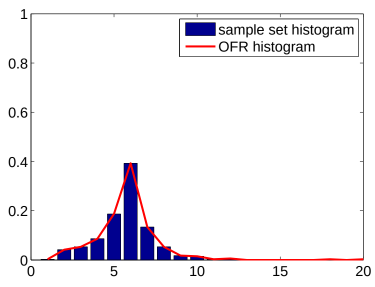


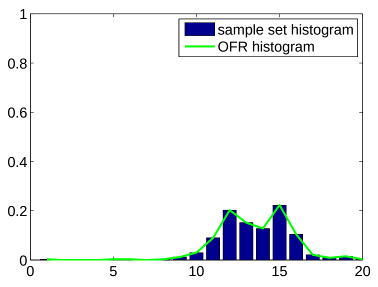
Figure 4.14: Normalized images and histograms: (a) Normalized red image (b) normalized green image (c) normalized red histogram (d) normalized green histogram

This continuous update rule provides a dynamic adaptation to the environmental changes in illumination and reflections. However, to adapt the sudden significant changes in the ground plane model, (e.g. from carpet to tiles) we needed another mechanism. We keep histograms of both OFR and sample pixels $\mathbf{S}(x, y)$. We re-initialize $\mathbf{S}(x, y)$ if these histograms differ significantly and if the ultrasonic sensor indicates that OFR is free of obstacles. Significant difference of one of the histograms (normalized red or normalized green) is sufficient for the model re-initialization. The significant change is detected by a percentage threshold (Figure 4.14(c)-(d)). Figure 4.15 shows an example situation where both histograms are similar therefore there is no need to re-initialize the sample sets.

In this approach, the update mechanism of $\mathbf{S}$ is also dependent on the ultrasonic distance sensor data (UD). At each frame, we read ultrasonic distance sensor (UD). The reading gives us the distance to the closest obstacle in front of the robot in terms of centimeters in range (2,300). We let the update of $\mathbf{S}$ if the



(a)



(b)

Figure 4.15: Sample set and OFR histograms: (a) Normalized red histogram (b) normalized green histogram. Both histogram pairs (sample set and OFR) are similar.

mobile robot can safely move one step forward afterwards. In other words, we control the update using a "safe-to-move-threshold" value (t).

Thus, this pre-update rule mechanism that is based on the safe distance check prevents our robot to update the model unselectively. If the ultrasonic distance sensor is not used, the ground plane model continues updating from OFR whether there is an obstacle or not in the front. Moreover, if the appearance of the ground plane changes suddenly (e.g. carpet to tile), the robot cannot move towards to the different appearing ground plane because the robot treats this new appearance as an obstacle. However, when the ultrasonic distance sensor gives a safe-to-move-threshold for the new ground plane, the robot updates its ground plane model effectively with respect to the subsampling factor. If UD is above t and update decision is confirmed by the subsampling factor, we replace two samples from $\mathbf{S}(x, y)$ with values one is randomly chosen from OFR and the value of q at the location (x, y). Also one of the adjacent pixel's sample set is updated in the same way. Note that, during the update process, only the sample sets of pixel locations that are classified as ground plane are updated. Figure 4.16 shows the distribution of $\mathbf{S}$ at two different time lines. Only two samples for each $\mathbf{S}(x, y)$ are drawn on the scatters.

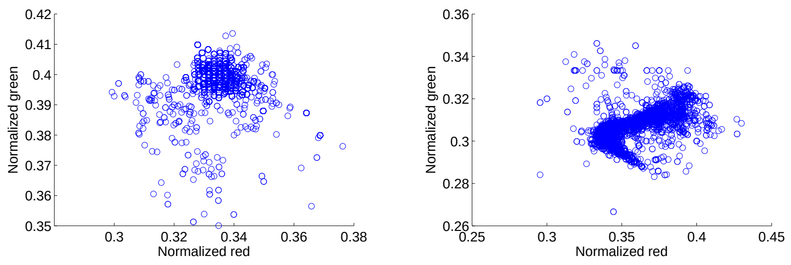


Figure 4.16: Distribution of sample sets at different time lines

4.2.2.4 Experiments

During the tests, we run the algorithm in the normalized RGB color space. Normalizing red and green channels eliminate problems that occur due to differently illuminated ground plane areas. However, we also performed some preliminary experiments using HLS (Hue-Lightness-Saturation) color space but we observed that using Hue value creates more problems than normalized red-green does in dark or saturated bright regions. We recorded five different datasets to test our algorithm. The videos were recorded in our laboratory testing pitch which has illumination reflections, shadows, different colors of floor and it contains many obstacles that have different colors and shapes. Also, at each frame ultrasonic sensor's reading is also recorded.

Each RGB frame in the recorded dataset has an image size of 160x120 pixels where the square that we collect samples from (OFR) contains only 20x20 pixels. During the experiments, we set the size of the sample sets (N) as 20 and UD threshold (t) as 100cm. In addition, we set the threshold value for the number of close samples (M) as 2, and we set distance threshold value (R) as 0.05. Note that, the pixel values are mapped into the 0-1 interval after normalization. Figure 4.17 shows the average accuracies (ACC) that is calculated from the outputs of our algorithm for varying R values and the ground truth videos.

In addition to first Figures 4.13(a), 4.13(c), Figure 4.19 shows more input-output-ground truth triplets of our algorithm. We observed that the method detects obstacles generally successfully. However, false negatives occur when the obstacle

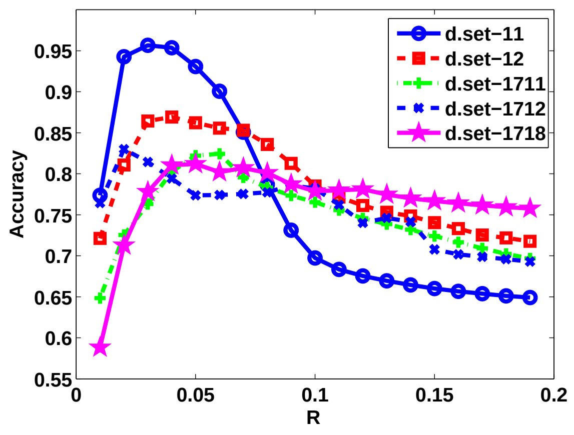


Figure 4.17: Accuracy vs. R

is in the same color with the ground plane (see Figures 4.13(a)-4.13(c)). We also observed that the algorithm performs not well specifically for gray colored ground plane and gray colored obstacles (e.g walls, doors) with the above parameter settings.

The distance threshold R value was set to 0.05 for all datasets to provide consistency through processing all the datasets. However, we observed that changing this value could improve performance of different datasets because lower R values perform better for gray colored ground plane and grayish obstacles. This is possibly due to the fact that a constant threshold value is not optimal through normalized red-green color space.

Furthermore, to observe the effect of parameters N and M, we plot a ROC (Receiver Operating Characteristics) curve [22] for their different values for one the datasets (1718). To plot ROC curves we fixed obstacle pixels as positive class and ground pixels as negative class. Horizontal axis of the ROC curve shows false positive rate (FPR) whereas the vertical axis shows true positive rate (TPR).

- True positive (TP): Ground truth is obstacle and our algorithm classifies as obstacle
- False negative (FN): Ground truth is obstacle but our algorithm classifies as ground

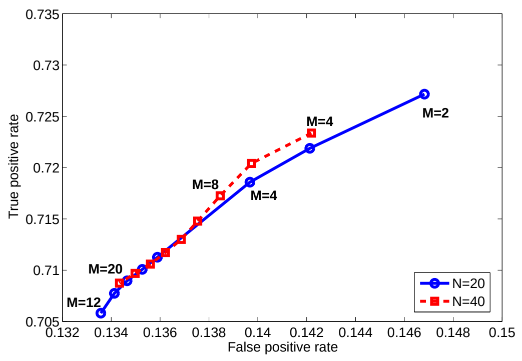


Figure 4.18: ROC curve for varying M values

- False positive (FP): Ground truth is ground but our algorithm classifies as obstacle

$$\begin{aligned} ACC &= (TP+TN)/(TP+TN+FP+FN) \\ TPR &= TP/(TP+FN) \\ FPR &= FP/(TN+FP) \end{aligned} \tag{4.9}$$

It can be seen in Figure 4.18 that the larger values of M lowers both false and true positive rates whereas lower values produce the opposite. From different ROC curves obtained from different N values (20,40), we observed that the algorithm is not so sensitive to the sample set size N, so we set it as 20.

In overall, we observed that our combined obstacle detection algorithm performed acceptable. Furthermore, most false positive detections were either isolated pixels or very small regions. Hence, they can be removed using morphological operations like erosion.

4.2.3 Comparison of the Obstacle Detection Algorithms

In order to compare the performance of the obstacle detection algorithms, we prepared manually segmented ground truth videos. There are 5 different videos

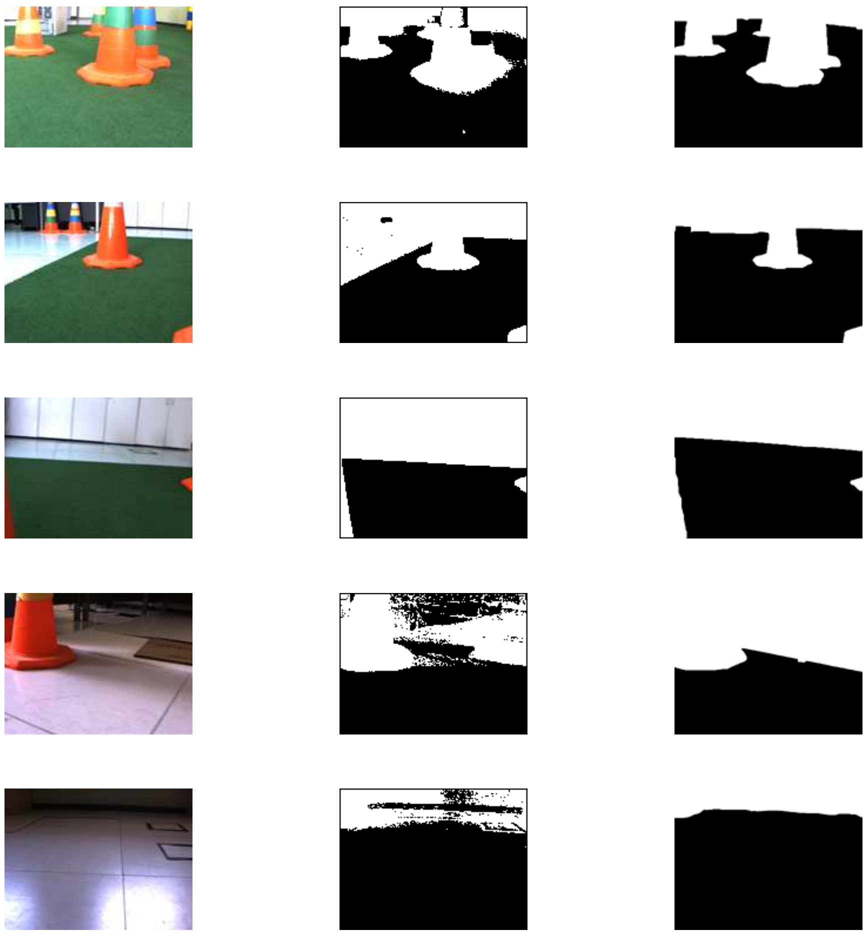


Figure 4.19: Input-output-ground truth: Input image, obstacle map and ground truth triplets are shown in left, center and rights columns respectively.

Table 4.1: Dataset specifications

Dataset Name	♯ of frames	fps
11	300	20
12	300	20
1711	100	4
1712	100	4
1718	100	4

whose specifications can be seen in Table 4.1.

- **Dataset-11:** This dataset starts on the green carpet where the robot's front is free of obstacles while there are traffic cones on the right side of the robot. This dataset recorded under white fluorescent light. During the recording, the robot does not come across to any obstacles, the obstacles are at the left or right sides of the robot.

- **Dataset-12:** This dataset starts on the green carpet where there are many obstacles (e.g. traffic cones, box, lockers, table) across it. The recording is done under white fluorescent light and the illumination level increases at approximately 100th frame of the dataset. Then the robot moves to the grayish tile from the green carpet. There are desktop computers on the floor which are almost in the same color with tiles. During the recording, the robot does not come across to any obstacles.

- **Dataset-1711:** This dataset starts with a white computer case in the front. The initial ground plane comprise of green carpet where there are many traffic cones on the ground. The orange traffic cones are partially covered with green, yellow, blue patches. There are shadows on the ground plane that are caused by the sunlight that comes from the window. During the recording, the robot gets very close to the obstacles. Moreover, there's a transition between different colored ground planes (green carpet to grayish tile). The sunlight generates white regions on the grayish floor.

- **Dataset-1712:** This dataset starts on the green carpet where there are traffic cones across the robot but the reference area is free of obstacles. The robot gets closer to the traffic cone which is covered with a green colored paper that is almost in the same color with the ground plane. In the end, the half of the floor is covered by the green carpet where the other half is covered by gray tiles. In addition, the lockers across the robot is almost in the same color with the gray tiles.

- **Dataset-1718:** This dataset starts on the green carpet where the sunlight generates almost white regions on the carpet. There's a transition from one ground plane (green carpet) to another (gray tiles). There are many different colored obstacles in the scene during the recording.

Table 4.2: Comparison of the obstacle detection algorithms

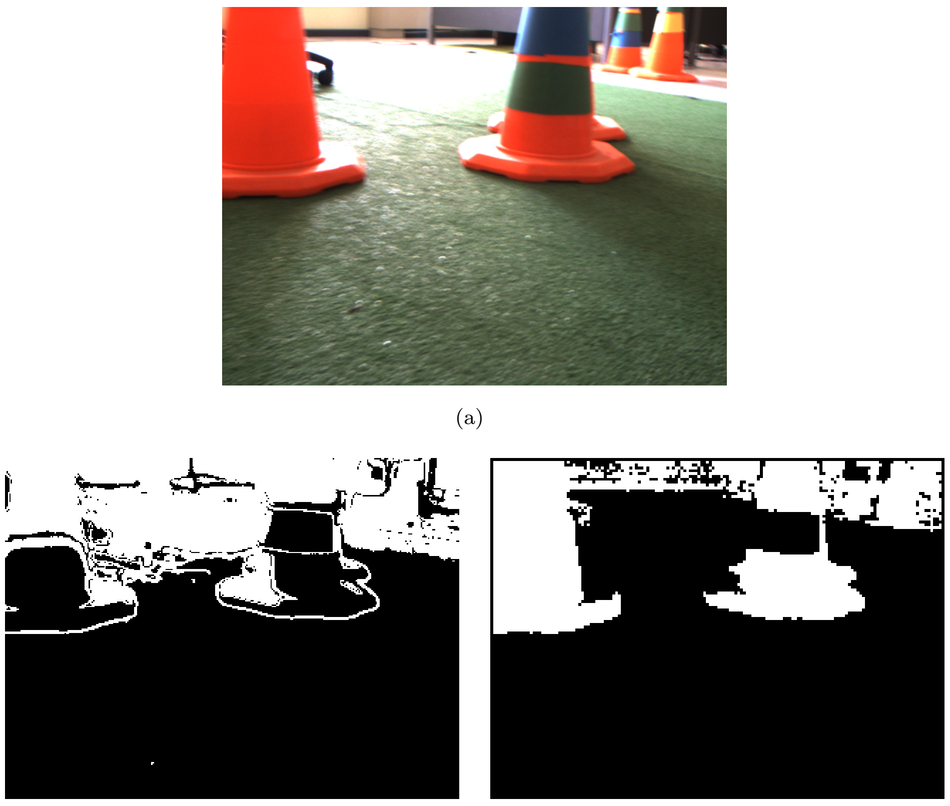
	Algorithm	True Positive	False Positive	Accuracy
Dataset 11	Histogram Method	0.8679 ±0.1150	0.0461 ±0.0221	0.9131 ±0.0503
	Camera and Sensor Method	0.8578 ±0.0833	0.0213 ±0.0107	0.9307 ±0.0327
Dataset 12	Histogram Method	0.8479 ±0.2017	0.0557 ±0.0243	0.8976 ±0.0901
	Camera and Sensor Method	0.7687 ±0.2042	0.0712 ±0.1227	0.8621 ±0.0956
Dataset 1711	Histogram Method	0.5069 ±0.1624	0.2167 ±0.1021	0.6203 ±0.1624
	Camera and Sensor Method	0.7006 ±0.3001	0.1159 ±0.0981	0.8221 ±0.1026
Dataset 1712	Histogram Method	0.8007 ±0.2451	0.1226 ±0.1207	0.7890 ±0.2125
	Camera and Sensor Method	0.8171 ±0.2555	0.1527 ±0.1582	0.7737 ±0.2268
Dataset 1718	Histogram Method	0.6871 ±0.2575	0.2447 ±0.1441	0.7161 ±0.1200
	Camera and Sensor Method	0.7240 ±0.2232	0.1486 ±0.1961	0.8123 ±0.1533

Table 4.2 shows the true positive rates, false positive rates, accuracies and standard deviations of these data. For each dataset, every frame is individually processed with its ground truth and the algorithms' outputs to calculate these rates. Then, the rates which are calculated for each frame are averaged to construct the table.

Our proposed algorithm is robust against illumination differences, shadows and the changes in the appearance of the ground plane. Table 4.3 shows the performances of the obstacle detection algorithms for Figures 4.20-4.21-4.22.

As it can be seen in Figure 4.20, the ground plane has varying illumination. Our algorithm could successfully detect the ground plane while achieving 0.7848 true positive rate for this frame where the other method's true positive rate is 0.6168.

In Figure 4.21, there are several obstacles in front of the robot. Our algorithm detects these obstacles and does not include them in the ground plane model. For that frame, our algorithm's true positive rate is 0.6976 where the histogram method's true positive rate is 0.3947.


(a)

(b) (c)

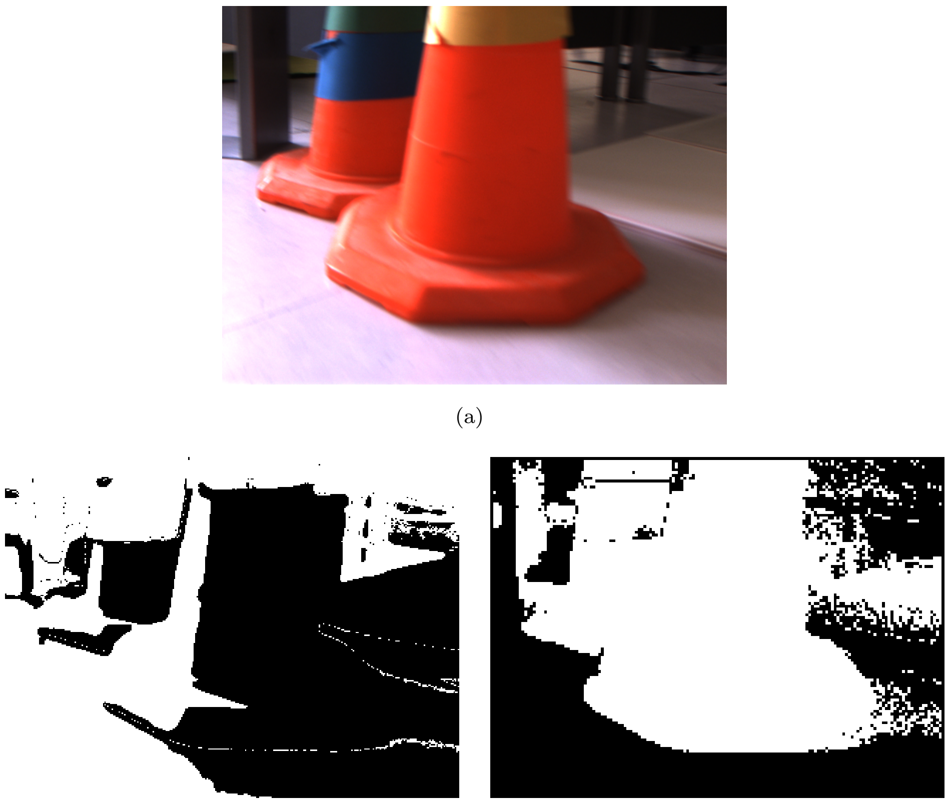
Figure 4.20: Sample frame that shows differently illuminated regions: (a) Input image (b) output of histogram method (c) output of our proposed method

Table 4.3: Performances of the algorithms on individual frames

	Algorithm	TP	FP	Accuracy
Figure 4.20	Histogram Method [2]	0.6168	0.1559	0.7752
	Camera and Sensor Method	0.7848	0.0063	0.9292
Figure 4.21	Histogram Method [2]	0.3947	0.4114	0.4579
	Camera and Sensor Method	0.6976	0.0541	0.7777
Figure 4.22	Histogram Method [2]	0.3876	0.3117	0.6264
	Camera and Sensor Method	0.2440	0.1035	0.7596

Figure 4.22 contains a shiny area in the front. Our algorithm classifies that region as ground plane and achieve 0.8965 true negative rate while the other one fails with 0.6883 true negative rate.

Furthermore, we draw box plots of the true positive rates for each frame in the datasets. As it can be seen from the box plots, our proposed method's median is greater than the histogram method [2] in 4 of the datasets. Also, there's a considerable difference between the medians of the two methods in the challenging


(a)

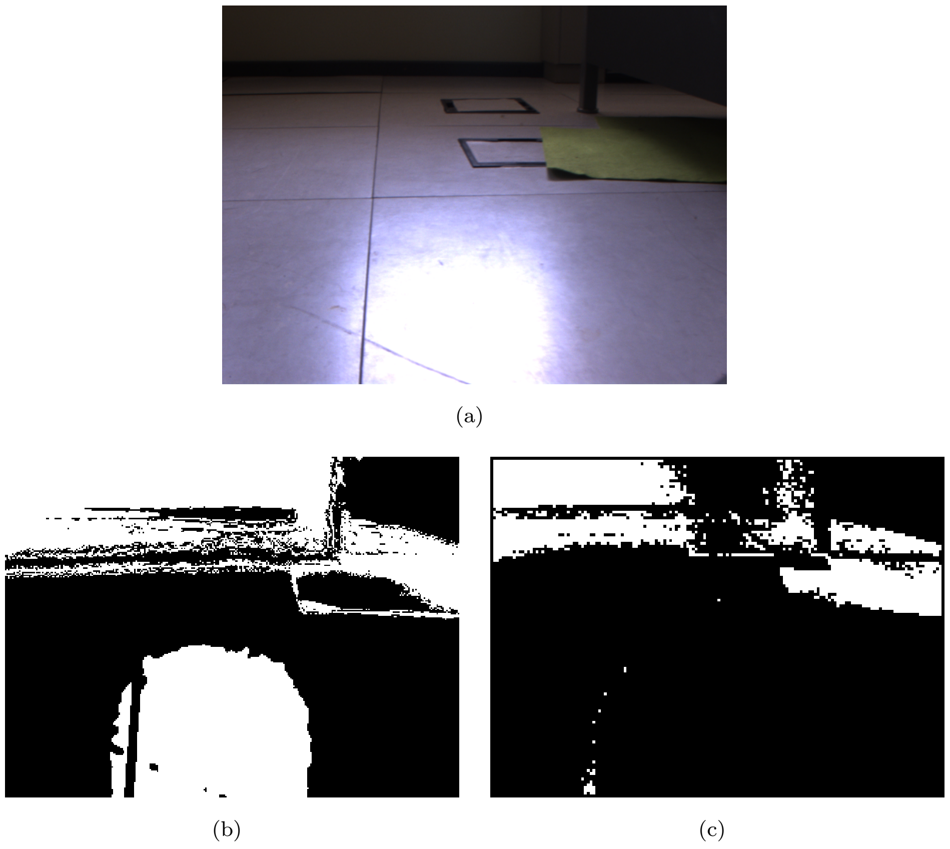
(b)

(c)

Figure 4.21: Sample frame that contains obstacle in the front: (a) Input image (b) output of histogram method (c) output of our proposed method

datasets such as dataset-1711. Our proposed method performs better than the other one for challenging datasets.

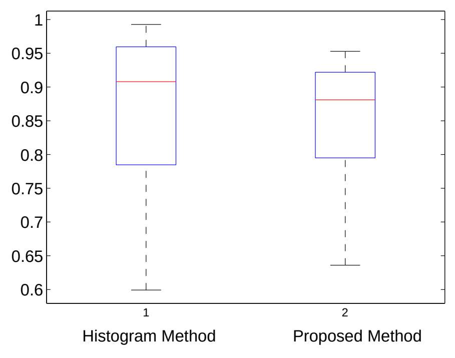
In addition, the true positive rates are spread into a large interval for datasets 1711 and 1712. There are very successfully classified frames as well as poorly classified ones. The true positive rates of some frames are very low due to the transition from one ground plane to another. Since these frames comprise small amount of the dataset, they are considered as outliers as in dataset 1712.


(a)

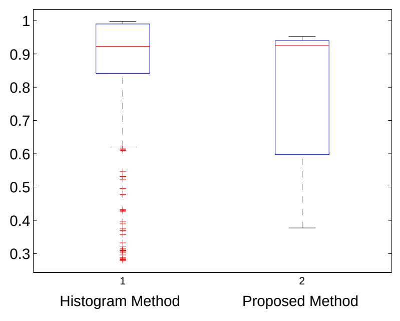
(b)

(c)

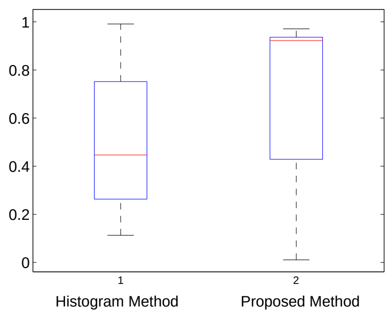
Figure 4.22: Sample frame that has shiny areas: (a) Input image (b) output of histogram method (c) output of our proposed method



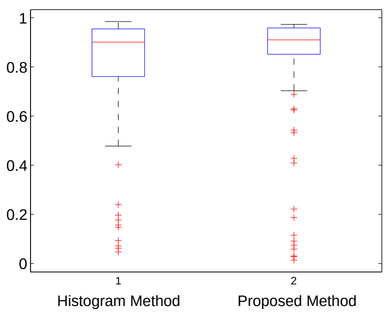
(a)



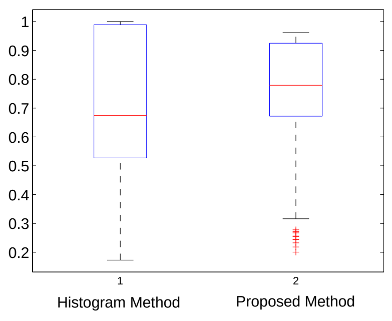
(b)



(c)



(d)



(e)

Figure 4.23: Box plots of true positive rates of the algorithms: (a) Dataset-11 (b) Dataset-12 (c) Dataset-1711 (d) Dataset-1712 (e) Dataset-1718

Chapter 5

Navigation

Obstacle detection process can be seen as the first step of this study. The second step is navigating the robot autonomously in indoor environment. The aim is to navigate the robot safely without crashing obstacles that might be present in its path. The robot can navigate in a controlled environment such as in a region that is surrounded by traffic cones, or it can navigate in the laboratory without any limitations.

The main goal of the navigation is to move from one place to another. In order to achieve that goal, the robot needs information about its position in the space and environment. The location information can be easily obtained from a GPS sensor in terms of world coordinates, however the GPS sensors that we have can only be used in outdoor because of the signal transmission with GPS satellites. Since, it is not possible to activate GPS sensor in indoor environments our discussions do not include GPS based location information.

In order to navigate the robot, we used two main different systems. The first one which will be explained in Section 5.1 uses only ultrasonic sensors, the second one which will be discussed in Section 5.2 uses visual information for navigation with obstacle avoidance.

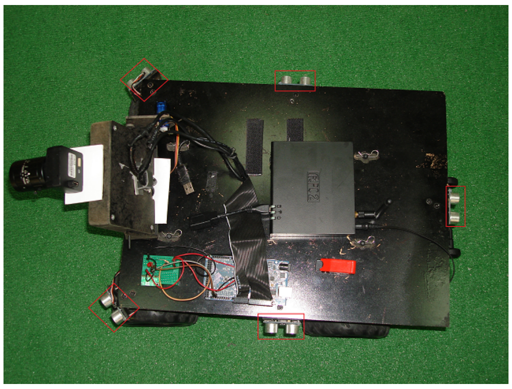


Figure 5.1: Robot with sensors installed

5.1 Navigation with Ultrasonic Range Finders

Our robot is equipped with 6 ultrasonic sensors. Three of them placed in front of the robot, one of them is on the rear, two of them are placed on the left and right sides of the robot (Figure 5.1). Our observations show that the range of the ultrasonic sensors is 2cm to 3meters.

The sensors on the left and right sides of the robot were not used in our navigation method, but they might be useful when navigating through a corridor or to observe if an obstacle region has finished when moving around a big obstacle such as a rock (e.g. Bug Algorithm) [29].

5.1.1 Basic Obstacle Avoidance

In this part, we aimed to move the robot (move forward, turn right, turn left, move backwards) by avoiding obstacles from an initial position until it reaches a spot that is no further movement is possible. That situation is aroused when the robot is surrounded by obstacles. Figure 5.2 illustrates the obstacle avoidance procedure that is followed by the robot using ultrasonic sensors. Our robot can safely (without hitting obstacles) move in a bounded environment like in Figure 5.3 as well as in the laboratory without any limitations.

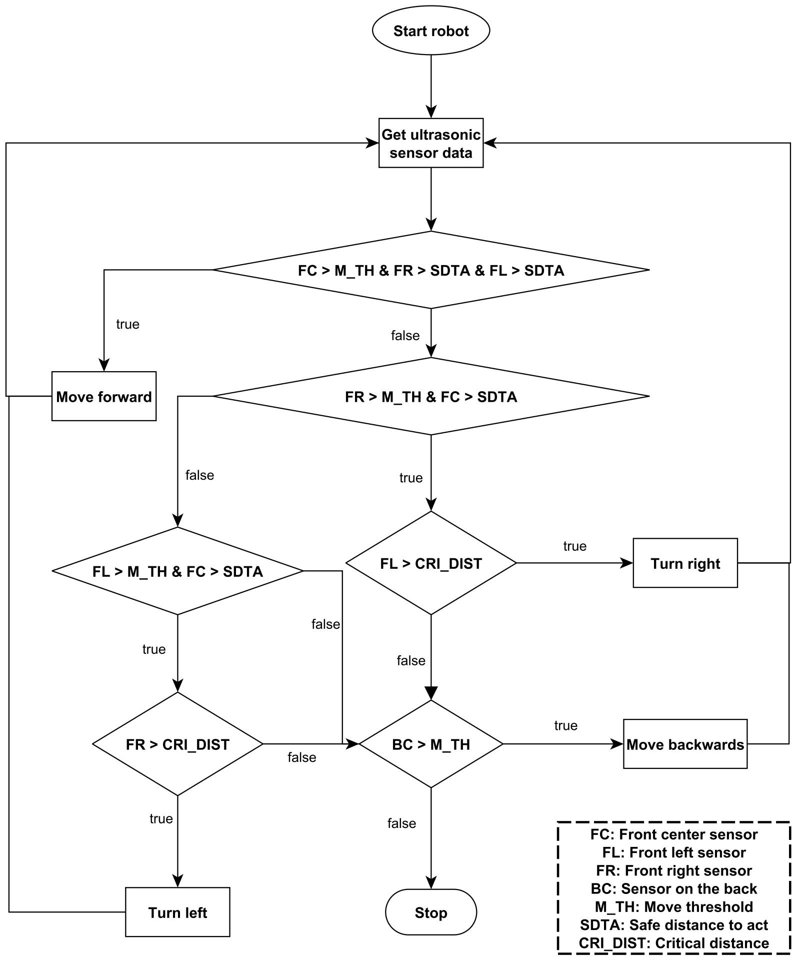


Figure 5.2: Block diagram of basic obstacle avoidance

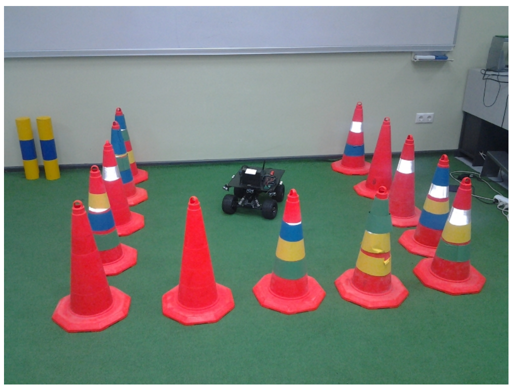


Figure 5.3: Robot in bounded environment

As it's seen in the block diagram in Figure 5.2, after starting the robot, ultrasonic sensor data are acquired and most passable path is determined by evaluating these data to move the robot. The robot first checks the condition to move forward if the condition does not satisfy it checks turning right, turning left and moving back conditions in order until it can find a free direction. Our robot moves in "steps". It acquires the sensor data, determines the free path and moves one step. One step duration is 500ms with given direction and constant speed. After each step, there's a delay (3000ms) to gather sensor readings and proceed to next step. We only evaluate the next step as we do not plan for a long path. Hence, there are three different threshold values that are used in obstacle avoidance routine. These are "move threshold", "safe distance to act" and "critical distance threshold". The movement or stopping decision of the robot is determined by examining sensor data and considering these threshold values. Also, these thresholds could be adjusted according to the robot's speed, but we assumed that our robot always moves with a constant speed.

The corresponding sensor reading must be always greater than "move threshold" which the robot will move on through. "Move threshold" is the greatest threshold value which is set as 50cm. The reason why this threshold is set as 50cm is, our robot is able to perform a safe stop in 50cm if it encounters an obstacle. Likewise, our robot does not displace more than 50cm in one step.

When the robot aims to move through a direction (e.g. forward), adjacent sensors' (e.g. front left, front right) readings must be greater than "safe distance to act" threshold. In other words, that threshold guarantees that the robot won't move even if the aiming direction's sensor reading is above "move threshold" if there's an obstacle nearby the adjacent direction. As it's seen in Figure 5.4(a), the front center sensor reading is greater than move threshold but the adjacent sensor reading (front right) is below safe distance to act threshold. Although the front center sensor reading is clear, the robot shouldn't move considering the obstacle which is detected by the front right sensor. Similarly, in Figure 5.4(b), the robot is prevented by a traffic cone on the left side. In Figure 5.4(c), front left and front

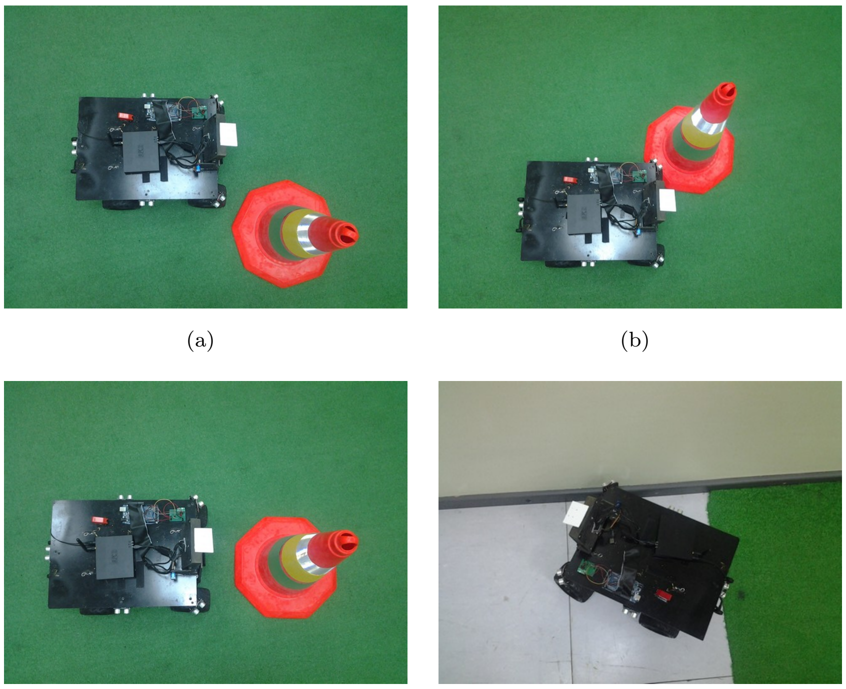

(a) (b) (c) (d)

Figure 5.4: Various cases for determining thresholds for ultrasonic sensor readings

right sensor readings are clear however the front center sensor reading is below safe distance to act threshold, so the robot cannot maneuver through left or right directions. “Safe distance to act” is set as 20cm.

Critical distance threshold is checked when turning left or turning right. The sensor reading which mounted on the opposite direction of turning must be greater than “critical distance” threshold in order to perform a safe turn and immediate move afterwards. “Critical distance” is the lowest threshold value which is set as 5cm. As it’s seen in Figure 5.4(d), front left sensor reading is clear, but the robot cannot turn to left because it’s very close to the wall on the opposite side (front right sensor reading below critical distance threshold).

5.1.2 Navigation Through Given Heading Value

In this section, we aimed the robot to navigate in indoor environment with guidance of the electronic compass. Our robot aims to navigate through the given heading value and tries to keep its orientation close to the heading while avoiding the obstacles. As mentioned in Section 2.1.4, the electronic compass gives us the heading which is the angle between the robot's position and magnetic north. The heading value that will be navigated through is given by the user at start-up. The heading value must be a valid integer that fits in 0-360 interval.

Algorithm 2 and 3 shows the calculation steps of the steering angle by evaluating compass and ultrasonic sensor readings. Heading is an angular value for which 0 and 360 degrees point to the same direction. For that reason, at every iteration we get the current heading value from the compass and calculate the angular difference to the user's desired heading (Algorithm 2).

We determined three main directions for our robot to move along (in the desired heading) which are left, forward and right directions. We set 90 as our steering angle to move forward, 60 and 120 are set for steering to right and left directions respectively.

After calculating the required steering angle to maneuver through the given heading, a check is necessary if the desired direction is free or not. At every iteration, the best possible steering angle is calculated using ultrasonic sensor readings if the required direction is not free of obstacles. If the calculated steering direction (left, center, right) using Algorithm 3 is closer than "tolerance limit", initially calculated steering angle is used. Otherwise, the robot chooses to move along the best possible free direction by the constraint of the heading it moves on. If all of the three front directions (left, center, right) are prevented by obstacles, the robot moves backwards after checking the ultrasonic sensor reading on the back.

The reason why we have used "tolerance limit" is, we assumed that if the ultrasonic reading determines direction x is free of obstacles, we can say that $direction_x \pm TOL\_LIM$ is also free of obstacles.

Algorithm 2 Navigation through given heading

```
$FORWARD\_ANGLE = 90$
$TOL\_LIM = 10$
robotDirectionAngles[3]={60,90,120}
difference=currentHeading-desiredHeading
angularDif=difference
**if** difference<-180 **then**
    angularDif=difference+360
**end if**
**if** difference>180 **then**
    angularDif=difference-360
**end if**
steerAngle=FORWARD_ANGLE+(angularDif/6)
index=calculateTheBestSteeringAngle(steerAngle)
**if** index is not valid **then**
    **if** $BC > M\_TH$ **then**
        move backwards
    **end if**
**else**
    **if** $abs(steerAngle - robotDirectionAngles[index]) > TOL\_LIM$ **then**
        steerAngle=robotDirectionAngles[index]
    **end if**
**end if**
steer using steerAngle and move
```

Algorithm 3 Calculate the best steering angle

```
$ultraArray[3] = \{FR, FC, FL\}$
absAngleDiff[3]={0,0,0}
**for** $i = 1$ **to** 3 **do**
    absAngleDiff[i]=abs(steerAngle-robotDirectionAngles[i])
**end for**
sortedIndices=sortArrayAsIndices(absAngleDiff)
**for** $i = 1$ **to** 3 **do**
    **if** $ultraArray[sortedIndices[i]] > M\_TH$ **then**
        **return** sortedIndices[i]
    **end if**
**end for**
**return** -1
```

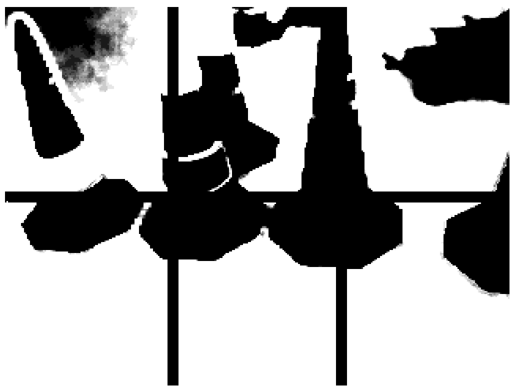


Figure 5.5: Average obstacle map

5.2 Navigation using Vision Information

In this section, navigation of the robot using visual information will be discussed. Similar to Section 5.1.2, the aim of the robot is to navigate through the given heading value, but this time using only the visual input. We used the algorithm which is investigated in Section 4.2.1 in order to obtain the binary obstacle map.

While the robot is moving in the environment, it's not possible to give a rational decision to control the robot's movement by checking only a single frame (obstacle map) at a certain time (t). Thus, we developed a mechanism that gathers 10 frames, and determines the average of these ten frames. In order to direct the robot along the free spaces, we used that averaged frame that we have calculated using 10 consecutive binary outputs. As it can be seen in Figure 5.5, we inverted the colors on the binary obstacle map for easy viewing. In Figure 5.5, white areas denote ground plane while the other regions denote obstacles.

In our vision based navigation technique, the robot could move through one of three front directions (left, forward, right). For that reason, the obstacle map is split into three columns (Figure 5.5). Each column's average is calculated using pixel values in that region to determine obstacle existences in each region. In the obstacle map, the pixels that have value of 255 are ground plane. Therefore, the region that has the highest mean value is assumed to be the freest path. If the desired direction that provides the closest orientation to the given heading value is not free of obstacles, second and third directions are considered for maneuvering. If the robot is absolutely prevented by obstacles in the front side, it moves backwards after checking the ultrasonic sensor on the back. Steering angle calculation process is as same as in Algorithm 3.

5.3 Comparison of the Navigation Algorithms

We discussed two main navigation techniques above. We test these algorithms in RAVLAB's testing pitch. Furthermore, we recorded the test scenes and upload the videos to Youtube. The videos can be viewed from the website: `http://www.youtube.com/user/towakina` also they're available on RAVLAB's website: `http://ravlab.isikun.edu.tr/multimedia`.

We run the robot for several hours in indoor environment. Our robot could operate on the carpet as well as on tiles or marbles. We placed several obstacles (e.g. traffic cones, boxes, chair) randomly in the testing area during these tests, and our robot successfully avoided these obstacles throughout the navigation routine.

However, besides the successful navigation, there are some minor problems. During these tests we observed a few problems related to the ultrasonic distance sensors. The most important issue is the placement of the sensors on the robot. In some cases, blind spots may occur in front of the robot that neither of the sensors could detect the obstacles in that field (Figure 5.6). Hence, the robot crashes the obstacles that it couldn't detect.

In vision based navigation algorithm, we split the obstacle map into three regions and calculate the average of each region. Therefore, we can estimate the obstacle density in each region by comparing the average values and navigate the robot according to that result. However, in some situations this assumption becomes invalid. A region may contain less obstacles than others, but the positions of the obstacles in that region prevent the robot navigate through that region. In Figure

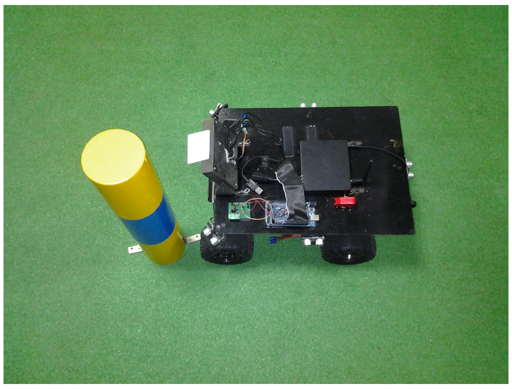


Figure 5.6: Blind spot in front of the robot

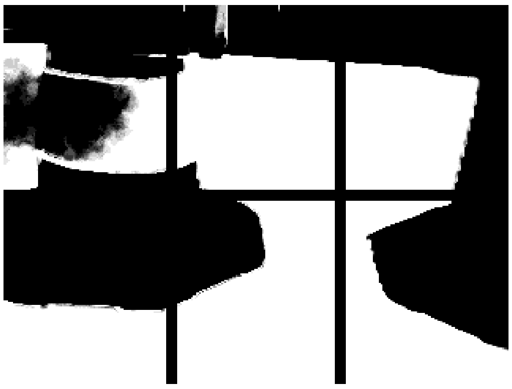


Figure 5.7: Limitation of the visual navigation

5.7, the center region is the freest region but the robot cannot pass between the traffic cones considering its width.

Conclusion

The aim of this thesis is to built an autonomous robot which can navigate in a previously unseen environment. Such a robot must be mobile and be able to gather information about its surroundings which require special equipment. In this study, we used a small-scale model car as a base mobile platform for the robot and we installed several components to facilitate sensing. These components were ultrasonic sensors, an electronic compass, a camera and a microcontroller board. The microcontroller board (i.e. Arduino) collects data from all of the sensors (except the camera) and can transmit moving/steering commands to the robot. The information collected from the sensors and the camera must be analyzed to coordinate the moving/steering actions of the robot. This analysis is performed by an onboard installed computer (i.e. FitPC). The onboard computer also requires programs to communicate with microcontroller board to retrieve the collected sensory data, and to control the robot with respect to the outcome of the analyzed data.

After building the mobile platform and integrating the necessary equipment, we developed two different programs. The first (main) program contains multiple threads which include communicating with the Arduino board over serial port, sending instructions to the robot over Arduino, performing data analysis, producing the navigational movement decisions. This main program works on the onboard PC. The second program that we have developed runs on the Arduino board, and it is responsible for reading/serving the sensor data, receiving instructions that are sent from the main program, and transmitting these instructions to the robot's servos.

Our robot senses the environment through ultrasonic sensors and cameras. The main issue in this study is to detect the obstacles in the environment by using the camera input. Therefore, we have developed different algorithms to detect obstacles in the surrounding environment. We have grouped the obstacle detection algorithms under two titles which are detection by obstacle modeling and detection by ground plane modeling. In the first approach, we aimed to detect orange colored traffic cones (as obstacles) by using the pixel color models of the traffic cones. We achieved good results on that approach however, we know that these results require homogenous color distribution of obstacles which is also different than the ground plane. In practice, the robot must operate in much more complex environments.

Following that approach, we have focused on the modeling the ground plane since it is easier to generalize. We developed an adaptive obstacle detection method based on ground plane modeling [30]. The method integrates monocular color vision and ultrasonic distance sensor data. Our approach assumes an obstacle free ground plane region in front of the robot in the initial frame. The algorithm uses individual pixel sets to model the ground plane and was inspired from a powerful background subtraction algorithm [1]. However, we showed that the method dynamically adapts to its environment in the succeeding frames guided by ultrasonic distance sensor reading. We showed that the method detects obstacles generally successfully. However, false negatives occur when the obstacle is in the same color with the ground plane. We also observed that it performs not well specifically for gray colored ground plane and gray colored obstacles (e.g. walls, doors) if they exist at the same environment. Furthermore, we implemented a simplified version of Ulrich and Nourbakhsh's algorithm [2] which uses histograms in order to model the ground plane. In order to make such a comparison, we have prepared 5 different videos that are recorded under different conditions (e.g. lighting etc.), and manually segmented these videos to create the ground truth map. We compared these methods in Section 4.2.3, and observed that our method performs better than its counterpart in most datasets.

In the last chapter, we discussed the obstacle avoidance and navigation of our robot in the indoor environment. We tried two different approaches for our robot's obstacle avoidance which include usage of ultrasonic sensor data and visual input. We recorded several videos during the robot's navigation which can be reached on RAVLAB's website.

Our obstacle detection algorithm performs with 77% average accuracy in the worst case with the recorded datasets. However, there are some improvements that could be made in order to increase the accuracy. As mentioned before, the algorithm does not perform well if the ground plane is grayish and the obstacles in the scene are in the shades of gray. In such cases, an intensity only check could be made for classification by keeping intensity sample sets. In addition, it may be possible to extend this work by adding a computationally feasible textural feature (e.g. edge).

Different equipments could be used due to the decrease of the prices of items (e.g. PC, camera). For example, the onboard PC (FitPC) on the robot has a single processor, single core. A multi-core onboard PC is preferable to assign each thread to work on a different core to cancel the delays between threads.

Finally, the obstacle avoidance algorithms that we developed could be improved. We split the region in front of the robot into three parts. In order to make the robot's turns more smooth, the region could be split into more parts while considering the obstacle density in each region. More sophisticated avoidance algorithms can be used.

References

[1] O. Barnich and M. Van Droogenbroeck. Vibe: A universal background subtraction algorithm for video sequences. *IEEE Transactions on Image Processing*, 20(6):1709 –1724, June 2011.

[2] Iwan Ulrich and Illah R. Nourbakhsh. Appearance-based obstacle detection with monocular color vision. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 866–871. AAAI Press, 2000.

[3] G.N. Desouza and A.C. Kak. Vision for mobile robot navigation: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):237 –267, feb 2002.

[4] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems*, 53(3):263–296, November 2008.

[5] M.S. Guzel and Bicker R. Vision based obstacle avoidance techniques. *Recent Advances in Mobile Robotics*, 2011.

[6] H.P. Moravec. The stanford cart and the cmu rover. *Proceedings of the IEEE*, 71(7):872 – 884, july 1983.

[7] Ian Horswill. Polly: A vision-based artificial agent. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 824–829. Press, 1993.

[8] I. Horswill. Visual collision avoidance by segmentation. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, volume 2, pages 902 –909 vol.2, sep 1994.

[9] L.M. Lorigo, R.A. Brooks, and W.E.L. Grimsou. Visually-guided obstacle avoidance in unstructured environments. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 373 –379 vol.1, sep 1997.

[10] J. Borenstein and Y. Koren. Obstacle avoidance with ultrasonic sensors. *IEEE Journal of Robotics and Automation*, 4(2):213 –218, apr 1988.

[11] A. Elfes. A sonar-based mapping and navigation system. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 1151 – 1156, apr 1986.

[12] A. Discant, A. Rogozan, C. Rusu, and A. Bensrhair. Sensors for obstacle detection - a survey. In *30th International Spring Seminar on Electronics Technology*, pages 100 –105, May 2007.

[13] Miao Yu and Li Shu-qin. A method of robot navigation based on the multi-sensor fusion. In *2nd International Workshop on Intelligent Systems and Applications (ISA)*, pages 1 –4, may 2010.

[14] I. Ohya, A. Kosaka, and A. Kak. Vision-based navigation by a mobile robot with obstacle avoidance using single-camera vision and ultrasonic sensing. *IEEE Transactions on Robotics and Automation*, 14(6):969 –978, dec 1998.

[15] M. Margolis. *Arduino Cookbook, 2nd Edition*. O'Reilly Media, 2011.

[16] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[17] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[18] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt. Performance of optical flow techniques. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 236 –242, jun 1992.

[19] H.M. Deitel and P.J. Deitel. *C How to Program, 3rd Edition.* Prentice Hall, NJ, 2001.

[20] Simon Haykin. *Communication Systems, 4th Edition.* Wiley, 2001.

[21] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing, 3rd Edition.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[22] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification, 2nd Edition.* Wiley-Interscience, 2000.

[23] Arthur R. Weeks. *Fundamentals of electronic image processing / Arthur R. Weeks, Jr.* SPIE Optical Engineering Press ; IEEE Press, Bellingham, Wash., USA : New York :, 1996.

[24] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning opencv, 1st Edition.* O'Reilly Media, Inc., first edition, 2008.

[25] Pierre Soille. *Morphological Image Analysis: Principles and Applications, 2nd Edition.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[26] Edward R. Dougherty and Roberto A. Lotufo. *Hands-on Morphological Image Processing.* SPIE Press, Bellingham, WA, 2003.

[27] O. Barnich and M. Van Droogenbroeck. Vibe: A powerful random technique to estimate the background in video sequences. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2009)*, pages 945–948, April 2009. PDF available on the University site or at the IEEE.

[28] P.-M. Jodoin, M. Mignotte, and J. Konrad. Statistical background subtraction using spatial cues. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(12):1758 –1763, dec. 2007.

[29] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.

[30] İ.K. İyidir, F.B. Tek, and D. Kırcalı. Adaptive visual obstacle detection for mobile robots using monocular camera and ultrasonic sensor. In *Proceedings of ECCV2012 Ws/Demos, Part II, LNCS 7584*, pages 526–535, October 2012.

Curriculum Vitae

Publications

[1] İ.K.İyidir, F.B.Tek, D.Kırcalı, Adaptive Visual Obstacle Detection Using Monocular Camera and Ultrasonic Sensor. *In 12th European Conference on Computer Vision* (ECCV2012),- *3rd Computer Vision in Vehicle Technologies Workshop* (CVVT2012), Florence, Italy, 2012.

Appendix A

Color Space Conversion

The cameras that we have used in the system produce frames in RGB (red, green, blue) color space. In the method in Section 4.2.1 as a pre-processing step, we first converted RGB input irames into HSV (hue, saturation, value) color space. The main advantage of HSV color space is that we can separate color (HS) and intensity (V) components. Hue is a color attribute that describes the pure color, lightness refers to the intensity, and saturation denotes the strength of the color. There are other color spaces that comprise hue and saturation components such as HSI (hue, saturation, intensity), and HLS (hue, lightness, saturation) [21][23]. The determination of V, L and I components are different among these color spaces. Therefore, this difference affects the calculation of the saturation component. Calculation of the value component is shown in 5.1 where intensity in HSI model is calculated by averaging the red, green and blue pixel values. In addition, OpenCV[1] [24] library has RGB to HSV and HLS transformation functions. In OpenCV library, RGB to HSV color space conversion is performed as in 5.1.

Note that, hue is an angular value, and OpenCV [24] modifies those $H-S-V$ components according to the image data type that is used. For example, V component is mapped between 0-255 where H component is mapped into 0-180 interval if the data type is an 8-bit image.

[1] http://opencv.willowgarage.com/documentation/cpp/

$$
\begin{aligned}
V &\leftarrow \max(R, G, B) \\
S &\leftarrow \begin{cases} \frac{V - min(R,G,B)}{V} & \text{if } V \neq 0, \\ 0 & \text{otherwise} \end{cases} \\
H &\leftarrow \begin{cases} 60(G-B)/S & \text{if } V = R, \\ 120 + 60(B-R)/S & \text{if } V = G, \\ 240 + 60(R-G)/S & \text{if } V = B \end{cases}
\end{aligned}
\tag{5.1}
$$

if $H < 0$ then $H \leftarrow H + 360$.