MINI PROJECT 1 (19EI5PWMP1)
ON
"ML Based Traffic Light Detection and IR Sensor Based Proximity Sensing for Autonomous Cars"

A MINI PROJECT 1 REPORT
Submitted by

SUDARSHANA.S.RAO (1BM18EI053)
SUDAMSHU.S.RAO (1BM18EI052)
VIGNESH.V (1BM18EI062)

Submitted towards the fulfillment of requirements for completion of
Mini Project 1 (19EI5PWMP1)
IN
ELECTRONICS AND INSTRUMENTATION ENGINEERING

Under the Guidance of

Prof. Vani A
Assistant Professor
Department of Electronics and Instrumentation Engineering

DEPARTMENT OF ELECTRONICS AND INSTRUMENTATION ENGINEERING
B. M. S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

2020-2021

"ML Based Traffic Light Detection and IR Sensor Based Proximity Sensing for Autonomous Cars"

MINI PROJECT 1 REPORT

Submitted by,
SUDARSHANA.S.RAO (1BM18EI053)
SUDAMSHU.S.RAO (1BM18EI052)
VIGNESH.V (1BM18EI062)

Submitted towards the fulfillment of requirements for completion of
Mini Project 1 (19EI5PWMP1)
in
ELECTRONICS AND INSTRUMENTATION ENGINEERING
B.M.S. COLLEGE OF ENGINEERING

Under the guidance of
Prof. VANI A
Assistant Professor
B.M.S. College
of Engineering
Bangalore
560019

Department of Electronics and Instrumentation Engineering
B.M.S. COLLEGE OF ENGINEERING
Bengaluru - 19
Bull temple road,
Bangalore -560019

2020-2021

B.M.S. COLLEGE OF ENGINEERING
(Autonomous College Under VTU)

Department of Electronics and Instrumentation Engineering

BONAFIDE CERTIFICATE

This is to certify that the mini project 1 report titled, "**ML Based Traffic Light Detection and IR Sensor Based Proximity Sensing for Autonomous Cars"** is a bonafide work carried out by **Sudarshana.S.Rao (1BM18EI053), Sudamshu.S.Rao (1BM18EI052) and Vignesh.V (1BM18EI062),** in Submitted towards the fulfillment of requirements for completion of Mini Project 1 (19EI5PWMP1***)*** during the academic year 2020-2021.

Prof. Vani A	**Dr. Veena N Hegde**	**Dr. B V Ravishankar**
Project Guide	Professor & H.O.D	Principal
Assistant Professor	BMSCE	
Department of EIE, BMSCE		

Internal Examiner **External Examiner**

B.M.S. COLLEGE OF ENGINEERING
BANGALORE -560019
BONAFIDE CERTIFICATE FROM THE GUIDES

Department: Electronic and Instrumentation Engineering
Candidates Degree Registered For: B.E
Name of the Guide: Prof. Vani A

CANDIDATE DETAILS:

SL. NO	Student Name	USN	Student Signature
1	SUDARSHANA.S.RAO	1BM18EI053	
2	SUDAMSHU.S.RAO	1BM18EI052	
3	VIGNESH.V	1BM18EI062	

Certified that this mini project 1 titled "ML Based Traffic Light Detection and IR Sensor Based Proximity Sensing for Autonomous Cars" the candidates have carried out the project to my satisfaction. This 5th semester academic year 2020-21 dissertation report was thoroughly scrutinized and corrected by me.

All corrections are incorporated by the students. The work is original and the mini project 1 report is the final one and of high standard. I duly certify the same.

SL NO	Guide Name	Dept/Organization/Designation	Signature
1	Prof. Vani A	Assistant Professor Department of Electronics and Instrumentation Engineering **B.M.S. COLLEGE OF ENGINEERING** Bull Temple Road, Bangalore 560019	

Date:

Seal:

Guide/s Approval Certificate for Incorporating all corrections in the mini project 1 report

We **Sudarshana.S.Rao, Sudamshu.S.Rao** and **Vignesh.V** students of 5th semester B.E in Electronics and Instrumentation, BMSCE, Bangalore, hereby declare that the mini project 1 work entitled **"ML Based Traffic Light Detection and IR Sensor Based Proximity Sensing for Autonomous Cars"** has been carried out under supervision of **Prof. Vani A** of Electronics and Instrumentation Department towards the fulfillment of requirements for the completion of mini project 1 (19EI5PWMP1).

SUDARSHANA.S.RAO

SUDAMSHU.S.RAO

VIGNESH.V

Guide Signature:

Prof. VANI A

HOD Signature:

Dr. VEENA N. HEGDE

ACKNOWLEDGEMENT

The completion of this undertaking could not have been possible without the assistance and participation of so many people. Their contributions are sincerely appreciated and greatly acknowledged. The team would like to express their deep appreciation particularly to the following people.

We extend our heartfelt thanks to the principal Dr. Ravishankar B. V. and the management and staff of BMSCE for providing us with necessary infrastructure that enables us to implement anything our engineering minds desire.

Words cannot describe our gratitude to Dr. Veena N Hegde, HOD, Department of EIE, BMSCE for her constant support throughout this journey.

We wish to express our sincere acknowledgement to our guide Prof. Vani A, Assistant Professor, Department of EIE, BMSCE for her valuable suggestions and guidance throughout the course of our project. We are grateful for her belief in us for the implementation of a project of this magnitude. We are truly grateful for her constant support and for allowing us to work flexibly which enabled us to set our own deadlines and objectives.

A large amount of material has been obtained from online journals, textbooks and numerous other scholarly and voluminous sources. We thank all those unseen faces for their invaluable contribution to our venture.

We thank our beloved parents on whose blessings we live and thrive. It is their prayers that have helped us translate our efforts into fruitful achievements.

Abstract

In the modern world where everything is being automated, smart cities are being built and autonomous cars become norm in such cities. The most critical feature of any autonomous vehicle is traffic light detection and recognition. The outdoor perception problem is a major challenge for driver-assisted and autonomous vehicle systems. Autonomous cars must perceive traffic lights status to share the streets with human drivers. Machine learning techniques have shown great performance in the field of traffic related problems. Another feature is IR proximity sensing, for sensing any obstacles near the car to avoid a collision when an obstacle is sensed.

Table of contents

Table of images

Chapter 1-

Introduction

Traffic light detection is a vital feature for an autonomous car and so is the proximity sensing for obstacles around it. Recognition of traffic lights (TLs) is an integral part of Driver Assistance Systems (DAS) in the transitional period between manually controlled cars and a fully autonomous network of cars. Autonomous driving is an essential topic of research in the development of intelligent transportation systems [1]. The great ambition with autonomous vehicles is fully replacing the human driver by a computer system, without compromising and eventually improving safety and efficiency [2]. The human ability of "seeing" the environment (For example- The roads, pedestrians, road signs and other vehicles) and behaving accordingly, need to be carefully reproduced by the computer system. Autonomous terrestrial vehicles must be capable of perceiving traffic lights and recognizing their current states (red, amber, green). When some form of computer-controlled automation is involved with dangerous objects such as cars, safety and reliability is of utmost importance since machines aren't 100% accurate it is good to have redundancies in case the machine fails. The worst-case scenario would be a false positive from a red coloured road sign resulting in the assistance system determining that a red light is imminent when it is not the case and unnecessarily distracting the driver, or worse affecting the driver to perform an emergency braking operation. Most current research is focused on detection and recognition during day time with plenty of light, which makes it much easier to reject false positives, for example- tail lights, street lights, red coloured road signs and various other reflections [3]. Any machine learning program will not be fully reliable since 100% accuracy isn't possible but Tesla and Google have come up with efficient software. Google makes use of a 3-D camera for traffic light detection, the code automatically labels the traffic light image, then the light is classified as red, amber or green, it estimates the position of the traffic light using image-to-image association, least squares triangulation and camera extrinsic calibration and finally the traffic light semantics [4]. Tesla's feature is enabled if Autopilot's Autosteer or Traffic-Aware Cruise Control is active on Model 3 and Model Y Teslas. The vehicle will slow down to a stop when it identifies a stop sign or traffic lights. It uses a combination of cameras and GPS data to detect traffic lights, stop signs, and road markings [5].

Proximity sensing is crucial as the autonomous car should come to a stop as soon as the distance between it and the obstacle is close. IR sensor is much reliable than any other sensor mainly because of its flexible range, that's why an IR sensor is used for proximity sensing and not an ultrasonic sensor because simply light travels much faster than sound. One more reason for IR sensor to be widely used for this purpose is that it can easily be integrated to any surface and can be interfaced with any of the micro-controllers. Rigging up of the circuit for IR sensor isn't complex and the usage of the IR sensor is versatile. Tesla's autopilot feature uses Wide, Main and Narrow Forward Cameras. These three cameras mounted behind the windshield provide broad visibility in front of the car, and focused, long-range detection of distant objects.

- Forward Looking Side Cameras
- Rearward Looking Side Cameras
- Rear View Camera
- Radar send out radio waves that detect objects and gauge their distance and speed in relation to the vehicle in real time
- Ultrasonic Sensors for redundancy [6]

BMW makes use of LiDAR for proximity sensing and a highly sensitive, vehicle-mounted laser scanner shoots high-frequency pulses of laser light. These are reflected by objects and returned to the sensor, which can measure the distance from each individual point [7]. Google also uses rotating roof-top LiDAR – a camera that uses an array of 32 or 64 lasers to measure the distance to objects to build up a 3D map at a range of 200m, for detecting any obstacles.

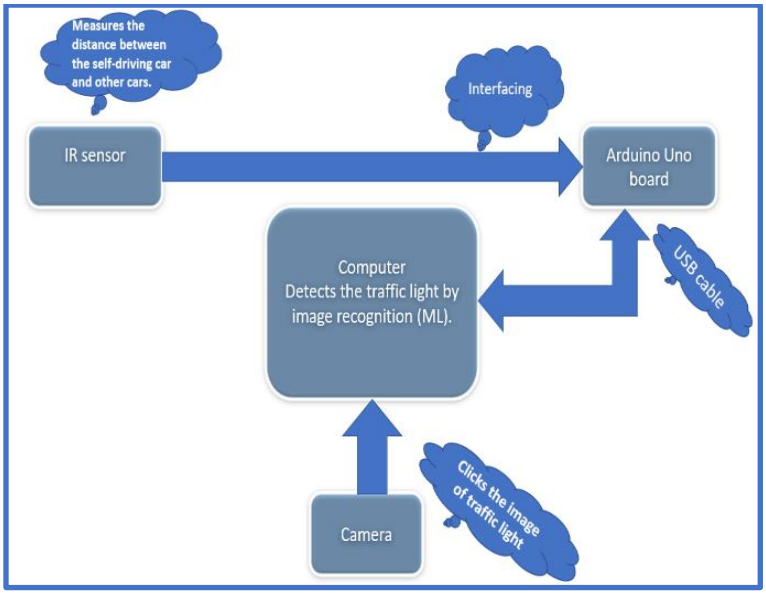


Figure 1.1: Overall block diagram

1.1 Problem Definition

For the autonomous vehicles to be as efficient as the human driven vehicles, the autonomous vehicles need to be cognizant of its surroundings. The autonomous vehicles have to obey the traffic signals and avoid impediments-stationary and moving.
The capabilities of the detection devices should match the sensing and capabilities of the humans.

Chapter 2 –

Literature Survey

1) Traffic Light Detection A Learning Algorithm and Evaluations on Challenging Dataset. By - Philipsen, Mark Philip; Jensen, Morten Bornø; Møgelmose, Andreas; Moeslund, Thomas B.; Trivedi, Mohan M. Published in: 2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC 2015), DOI (link to publication from Publisher): 10.1109/ITSC.2015.378, Publication date: 2015 Document Version. Accepted author manuscript, peer reviewed version (Page numbers: 5-6).

ML is used for training the computer to detect and recognise the traffic lights was referenced from the above paper.

2) Machine Learning for Next-Generation Intelligent Transportation Systems: A Survey Tingting Yuan, Wilson Borba da Rocha Neto, Christian Rothenberg, Katia Obraczka, Chadi Barakat, Thierry Turletti HAL Id: hal-02284820 https://hal.inria.fr/hal-02284820v2 Preprint submitted on 28 Nov 2020 (Page numbers: 9-11).

The above paper gave the input that open cv2 is an in-built python module which is primarily used for colour detection.

3) Traffic Light Recognition Using Deep Learning and Prior Maps for Autonomous Cars. By- Lucas C. Possatti_z, Rânik Guidolini_, Vinicius B. Cardoso_, Rodrigo F. Berriel_, Thiago M. Paixão_y, Claudine Badue_, Alberto F. De Souza, Senior Member, IEEE_ and Thiago Oliveira-Santos__Universidade Federal do Espírito Santo, Brazil yInstituto Federal do Espírito Santo, Brazil, zEmail: lucas.possatti@lcad.inf.ufes.br (Page numbers: 3-7).

The knowledge around Hough circles and Hough gradient method was obtained from the above paper and Hough circles can be used to draw circles around the detected colours and Hough gradient method is used to reduce the loss function.

4) Traffic Light Mapping and Detection Nathaniel Fairfield Chris Urmson {nfairfield, curmson}@google.com Google, Inc. Proceedings of ICRA 2011 (Page numbers: 4-6).

The above paper describes that Google created a working model of the traffic light detection system. It makes use of a 3-D camera for traffic light detection, the code automatically labels the traffic light image, then the light is classified as red, amber or green, it estimates the position of the traffic light using image-to-image association, least squares triangulation and camera extrinsic calibration and finally the traffic light semantics.

5) Driver-initiated Tesla Autopilot Disengagements in Naturalistic Driving Alberto Morando MIT AgeLab, Massachusetts Institute of Technology morando@mit.edu Pnina Gershon MIT AgeLab, Massachusetts Institute of Technology pgershon@mit.edu Bruce Mehler MIT AgeLab, Massachusetts Institute of Technology bmehler@mit.edu Bryan Reimer MIT AgeLab, Massachusetts Institute of Technology reimer@mit.edu (Page numbers: 2-5).

The above paper describes that Tesla created a working model of the traffic light detection system. Tesla's feature is enabled if Autopilot's Autosteer or Traffic-Aware Cruise Control is active on Model 3 and Model Y Tesla. The vehicle will slow down to a stop when it identifies a stop sign or traffic lights. It uses a combination of cameras and GPS data to detect traffic lights, stop signs, and road markings.

6) Revisiting the role of modular innovation in technological radicalness and architectural change of products: The Case of Tesla X and Roomba Tufail Habib University of Engineering and Technology, Peshawar, Pakistan Jimmi Normann Kristiansen Aalborg University Business School, Aalborg University, Denmark Mohammad B. Rana Aalborg University Business School, Aalborg University, Denmark Paavo Ritala Lappeenranta University of Technology, Finland (Page numbers: 10-11).

Ideas regarding the IR sensor distance measurement was understood from the above paper.

7) 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) October 25-29, 2020, Las Vegas, NV, USA (Virtual). A Scalable Framework for Robust Vehicle State Estimation with a Fusion of a Low-Cost IMU, the GNSS, Radar, a Camera and Lidar Yuran Liang1,2, Steffen Muller ¨ 1,

Daniel Schwendner2, Daniel Rolle2, Dieter Ganesch2, Immanuel Schaffer2 (Page numbers: 5-9).

Usage and interfacing of Arduino Uno board with the IR sensor was understood from the above paper.

Chapter 3 -

Design and Implementation

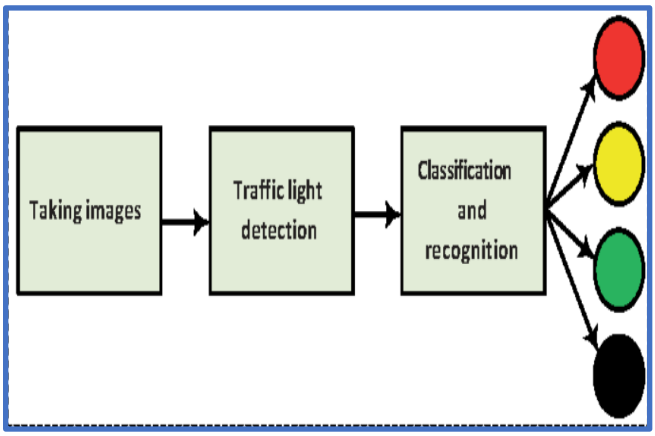


Figure 3.1: Block diagram of traffic light detection and recognition

- The ML code was programmed in python 3 language on Jupiter notebook (anaconda navigator) compiler.
- OS module is used to access the datasets folder stored on this computer.
- OpenCV2 or Open-Source Computer Vision Library is a library designed for colour detection.
- NumPy is a python library used for working with arrays and mathematical functions.
- Detect is a function which has two arguments- filepath and file.
- Font is a variable which stores the font: cv2.FONT_HERSHEY_SIMPLEX.
- cv2.imread() method loads an image from the specified file where filepath + path is a string representing the path of the image to be read.
    - **Syntax**: cv2.imread(path, flag)
    - **Parameters: Path-** A string representing the path of the image to be read. (filepath+path)
    - **Flag-** It specifies the way in which image should be read. Its default value is cv2.IMREAD_COLOUR
    - **Return Value-** This method returns an image that is loaded from the specified file.
- cimg is a variable which stores the loaded image from the specific file.

- Cv2.cvtColor converts the image from BGR to HSV format because opencv2 uses HSV format and stores it in hsv variable.
- Here img is the image whose colour space is to be changed and cv2.COLOUR_BGR2HSV is the colour space conversion code.
    - **Syntax:** cv2.cvtColour(src, code[, dst[, dstCn]]).
    - **Parameters: src-** It is the image whose colour space is to be changed. (img)
    - **Code-** It is the colour space conversion code. (cv2.COLOUR_BGR2HSV) BGR to HSV (Hue-Saturation-Value or brightness)
    - **dst-** It is the output image of the same size and depth as src image. It is an optional parameter.
    - **dstCn-** It is the number of channels in the destination image. If the parameter is 0 then the number of the channels is derived automatically from src and code. It is an optional parameter.
    - **Return Value-** It returns an image.
- Red, Green and Blue in RGB are all co-related to the colour luminance or intensity, i.e., colour information can't be separated from luminance. HSV or Hue Saturation Value is used to separate image luminance from colour information. This is why the colour conversion is done.
- The array values are the colour ranges for the red, yellow and green respectively.
- Red has four matrices because of the different intensities and to increase the accuracy of the red colour detection.
- The cv2.inRange function has three arguments: the first is the image where the colour detection is performed, the second is the lower limit of the colour to be detected, and the third argument is the upper limit of the colour. After this a binary mask is returned, where white pixels which have values from 1 to 255 also known as foreground colour represent pixels that fall into the upper and lower limit range and black pixels (0) or background colour do not.
- Cv2.inRange(type_colour, lower bound, upper bound).
- .shape function returns the height and width of the image.
- Cv2.HoughCircles() are used to detect circles on the image where maskr, maskg and masky are the images, cv2.HOUGH_GRADIENT is the detection method, 1 is the minimum distance between the centres of the detected circles, param1 is the higher threshold of the two passed to the Canny edge detector, param2 is the accumulator

threshold for the circle centres at the detection stage where the accumulator is a two-dimensional array which contains the x and y co-ordinates of the circle, minRadius is the minimum circle radius (in pixels) and maxRadius is the maximum circle radius.

- The circle Hough Transform (CHT) is used for detecting circles in imperfect images. The circle parameters are produced by "voting" in the Hough parameter space and then selecting local maxima in an accumulator matrix by using the radius and central co-ordinates of the circle. The circle will be drawn according to the equation

$$((x - x_{center})^2 + (y - y_{center})^2 = r^2) \quad \text{.....} \quad (3.1)$$

image 8-bit, single-channel, grayscale input image.

circles output vector of found circles (cv.CV_32FC3 type). Each vector is encoded as a 3-element floating-point vector (x,y,radius) .

method detection method. Currently, the only implemented method is HOUGH_GRADIENT

minDist minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbour circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.

param1 first method-specific parameter. In case of HOUGH_GRADIENT, it is the higher threshold of the two passed to the Canny edge detector (the lower one is twice smaller).

param2 second method-specific parameter. In case of HOUGH_GRADIENT, it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the falser circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.

- Hough gradient method is used to reduce the loss by drawing a line from the centre of the circle to the edge of the image, then a ratio of centre of circle to the number of times the iteration has gone outside the bounds of the image is taken; it is compared to the optimal epoch (the number of iterations) value, if the condition is satisfied the loss is significantly reduced. The edge of the image is detected by Canny edge detector which works by detecting discontinuities in brightness in the image.
- r is the optimum value of the radius of Hough circle in pixels.

- Bound is a constant value used in Hough gradient which is multiplied to the height of the image to ensure that the colour is in the detection frame.
- The if condition checks if r_circles is empty or not.
- uint16 creates an unsigned integer whose range is from 0 to 65535 and it converts the dimensions of the image to pixels and np.around() function rounds off the numeric value to the nearest whole number.
- The subsequent conditions omit the part of the image which are outside the detected edge.
- h and s are the loss function variables.
- The for statements ensure that the Hough circles are drawn around the detected colour. The if condition ensures that no two Hough circles will overlap each other. H is the centre of the circles and s is the number of times the iteration has gone beyond the edge of the detection frame.
- The if condition checks if the ratio is greater than the epoch value, since s is in the denominator of the ratio it should be of lower value for accurate detection.
- cv2.circle() method is used to draw a circle on any image where img is the image on which the circle is to be drawn, i[0] and i[1] are the x, y co-ordinates of the circle, i[2]+10 is the radius of the circle, the tuple (255,0,0) is the colour of the circle it is of the order blue-green-red and 2 is the thickness of the circle.
    - **Syntax:** cv2.circle(image, center_coordinates, radius, colour, thickness)
    - **Parameters: image-** It is the image on which circle is to be drawn. center_coordinates- It is the center coordinates of circle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).
    - **Radius-** It is the radius of circle.
    - **Colour-** It is the colour of border line of circle to be drawn.
    - **Thickness**- It is the thickness of the circle border line in px. Thickness of -1 px will fill the circle shape by the specified colour.
    - **Return Value:** It returns an image.

- The first cv2.circle statement will draw a circle on the image with width+param2. The second statement will use the radius of the circle to accurately draw the circle. cv2.putText() method is used to draw a text string on any image where cimg is the image on which text is to be displayed, RED STOP is the string, i[0] and i[1] are the co-ordinates where the text needs to be displayed, font is the FONT_HERSHEY_SIMPLEX type, 1 is the size of the font, the tuple is the colour of the font, 2 is the thickness and cv2.LINE_AA is the font style.
    - **Syntax:** cv2.putText(image, text, org, font, fontScale, colour[, thickness[, lineType[, bottomLeftOrigin]]])
    - **Parameters: image-** It is the image on which text is to be drawn.
    - **Text-** Text string to be drawn.
    - **Org-** It is the coordinates of the bottom-left corner of the text string in the image. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).
    - **Font-** It denotes the font type. Some of font types are FONT_HERSHEY_SIMPLEX, FONT_HERSHEY_PLAIN, etc.
    - **FontScale-** Font scale factor that is multiplied by the font-specific base size.
    - **Colour-** It is the colour of text string to be drawn. For BGR, we pass a tuple. example: (255, 0, 0) for blue colour.
    - **Thickness-** It is the thickness of the line in px.
    - **LineType-** This is an optional parameter. It gives the type of the line to be used.
    - **BottomLeftOrigin-** This is an optional parameter. When it is true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.
    - **Return Value-** It returns an image.
- The if condition (r_circles) and subsequent steps for the red colour are same for the detection of green and amber colours.
- cv2.imshow() method is used to display an image in a window where 'Detected results' is the window name and cimg is the image to be displayed.
    - **Syntax:** cv2.imshow(window_name, image)

  - **Parameters: window_name-** A string representing the name of the window in which image to be displayed.
  - **Image-** It is the image that is to be displayed.
  - **Return Value-** It doesn't return anything.
- cv2.imwrite() method is used to save an image to any storage device. This will save the image according to the specified format in current working directory. It takes two arguments namely: filename and image.
  - **Syntax:** cv2.imwrite(filename, image)
  - **Parameters: filename-** A string representing the file name. The filename must include image format like .jpg, .png, etc.
  - **Image-** It is the image that is to be saved.
  - **Return Value-** It returns true if image is saved successfully.
- cv2.waitKey(0) command doesn't delay in displaying the next image and cv2.destroyAllWindows() command will destroy all created windows.
- The next two statements calculate and displays the loss respectively.
- if __name__ == "main": is used to execute some code only if the file was run directly, and not imported.
- Os.path.abspath command is used to access the folder in which the images are stored. In this program '…' are used because the folder is in the same working directory of the python interpreter.
  - **Syntax:** os.path.abspath(path)
  - **Parameter: Path-** A path-like object representing a file system path.
  - **Return Type-** This method returns a normalized version of the pathname path.
- os.listdir() method in python is used to get the list of all files and directories in the specified directory.
  - **Syntax:** os.listdir(path)
  - **Parameters: path (optional)-** path of the directory
  - **Return Type-** This method returns the list of all files and directories in the specified path. The return type of this method is list.
- Prints the name of every image, the if condition checks if every file stored on the folder is an image or not and the last line is the function call.
- This ML model has been trained and tested with 25 data sets (input images).

3.1. ML Code

```
import os
import cv2
import numpy as np # np is an arbitrary variable which is used as an abbreviation instead of typing numpy every time.

def detect(filepath, file) : # detect is the function which recognises the traffic light.
    font = cv2.FONT_HERSHEY_SIMPLEX # The appropriate font is chosen.
    img = cv2.imread(filepath + file) # This command loads the images from a specific folder.
    cimg = img # cimg is a variable which stores the datasets.
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # This command converts the image in RGB to HSV.
    # hsv is a variable which stores the HSV value.
    lower_red1 = np.array([0, 100, 100])
    upper_red1 = np.array([10, 255, 255])
    lower_red2 = np.array([160, 100, 100])
    upper_red2 = np.array([180, 255, 255])
    # The above four arrays determine the red colour intensity ranges.
    lower_green = np.array([40, 50, 50])
    upper_green = np.array([90, 255, 255])
    lower_yellow = np.array([15, 150, 150])
    upper_yellow = np.array([35, 255, 255])
    # The above eight lines of code defines the array values for the colour ranges in HSV.

    mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
    mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
    maskg = cv2.inRange(hsv, lower_green, upper_green)
    masky = cv2.inRange(hsv, lower_yellow, upper_yellow)
    maskr = cv2.add(mask1, mask2)
    # The above five lines of code sets the threshold values.
    size = img.shape # This command determines the dimensions of the image.
```

```
    r_circles = cv2.HoughCircles(maskr, cv2.HOUGH_GRADIENT, 1, 80, param1 = 50,
param2 = 10, minRadius = 0, maxRadius = 30)
    g_circles = cv2.HoughCircles(maskg, cv2.HOUGH_GRADIENT, 1, 60, param1 = 50,
param2 = 10, minRadius = 0, maxRadius = 30)
    y_circles = cv2.HoughCircles(masky, cv2.HOUGH_GRADIENT, 1, 30, param1 = 50,
param2 = 5, minRadius = 0, maxRadius = 30)
    # Hough circle command is used to draw circles around the detected colour.


    r = 5 # r is the optimum value of radius for the Hough circle.
    bound = 0.4 # bound is the multiplying constant to define the boundaries of the image.
    # Constant values for defining the image boundary.
    if r_circles is not None :
        r_circles = np.uint16(np.around(r_circles)) # This command returns a 16-bit integer
number.
        for i in r_circles[0, :] : # Edge detection.
            if i[0] > size[1] or i[1] > size[0] or i[1] > size[0]*bound :
                continue
            # It skips if the iteration goes outside the bounds of the image.
            h, s = 0.0, 0.0 # Loss function variables.
            for m in range(-r, r) : # Edge detection.
                for n in range(-r, r) :
                    if (i[1] + m) >= size[0] or (i[0] + n) >= size[1] :
                        continue
                    h += maskr[i[1] + m, i[0] + n] # Central co-ordinates of the circle.
                    s += 1 # The number of times (count) the iteration has gone outside the
bounds of the image.
            if h/s > 50 : # Condition for reducing the loss (epoch value).
                cv2.circle(cimg, (i[0], i[1]), i[2] + 10, (255, 0, 0), 2) # The values in the tuple is
of the order -BGR.
                cv2.circle(maskr, (i[0], i[1]), i[2] + 30, (255, 255, 255), 2)
                # Prints the circle around the detected colour.
                cv2.putText(cimg, 'RED STOP', (i[0], i[1]), font, 1, (0, 69, 255), 2,
cv2.LINE_AA)
                # Prints the text on the output image.
    # Detection of green light.
```

```
    if g_circles is not None :
        g_circles = np.uint16(np.around(g_circles)) # This command returns a 16-bit integer number.
        for i in g_circles[0, :] : # Edge detection.
            if i[0] > size[1] or i[1] > size[0] or i[1] > size[0]*bound :
                continue
            # It skips if the iteration goes outside the bounds of the image.
            h, s = 0.0, 0.0 # Loss function variables.
            for m in range(-r, r) : # Edge detection.
                for n in range(-r, r) :
                    if (i[1] + m) >= size[0] or (i[0] + n) >= size[1] :
                        continue
                    h += maskg[i[1] + m, i[0] + n] # Central co-ordinates of the circle.
                    s += 1 # The number of times (count) the iteration has gone outside the bounds of the image.
            if h/s > 100 : # Condition for reducing the loss (epoch value).
                cv2.circle(cimg, (i[0], i[1]), i[2] + 10, (255, 0, 0), 2) # The values in the tuple is of the order -BGR.
                cv2.circle(maskg, (i[0], i[1]), i[2] + 30, (255, 255, 255), 2)
                # Prints the circle around the detected colour.
                cv2.putText(cimg, 'GREEN GO',(i[0], i[1]), font, 1, (0, 69, 255), 2, cv2.LINE_AA)
                # Prints the text on the output image.
    # Detection of amber light.
    if y_circles is not None :
        y_circles = np.uint16(np.around(y_circles)) # This command returns a 16-bit integer number.
        for i in y_circles[0, :] : # Edge detection.
            if i[0] > size[1] or i[1] > size[0] or i[1] > size[0]*bound :
                continue
            # It skips if the iteration goes outside the bounds of the image.
            h, s = 0.0, 0.0  # Loss function variables.
            for m in range(-r, r) : # Edge detection.
                for n in range(-r, r) :
                    if (i[1] + m) >= size[0] or (i[0] + n) >= size[1] :
```

```
                continue
            h += masky[i[1] + m, i[0] + n] # Central co-ordinates of the circle.
            s += 1 # The number of times (count) the iteration has gone outside the
bounds of the image.
        if h/s > 50 : # Condition for reducing the loss (epoch value).
            cv2.circle(cimg, (i[0], i[1]), i[2] + 10, (255, 0, 0), 2) # The values in the tuple is
of the order -BGR.
            cv2.circle(masky, (i[0], i[1]), i[2] + 30, (255, 255, 255), 2)
            # Prints the circle around the detected colour.
            cv2.putText(cimg, 'AMBER GET READY', (i[0], i[1]), font, 1, (0, 69, 255), 2,
cv2.LINE_AA)
            # Prints the text on the output image.
    cv2.imshow('Detected results', cimg) # Prints the output images.
    cv2.imwrite(path + '//result//' + file, cimg) # It saves the images in the specified path.
    cv2.waitKey(0) # This command doesn't delay in displaying the next image.
    cv2.destroyAllWindows() # This command will destroy all created windows.
    Loss = abs(100 - (h/s)) # Calculates the loss.
    print('Loss =', Loss/100, '%') # Prints the loss.


if __name__ == '__main__' : # Main function.
    path = os.path.abspath('..') + '//light//' # The address of folder which has the datasets.
    for f in os.listdir(path) : # f, the iteration variable iterates through every single file
stored in the particular folder.
        print(f)
        if f.endswith('.jpg') or f.endswith('.JPG') : # This condition will check for images in
the folder.
            detect(path, f) # Function call.
```

3.2 ML Program Outcomes

i. The ML code results into the detection of traffic lights.

ii. The detected traffic lights are recognised as red, amber or green.

iii. The recognised light types are printed as:

a) Red- Stop

b) Amber- Get ready

c) Green- Go

3.3 IR Sensor Description

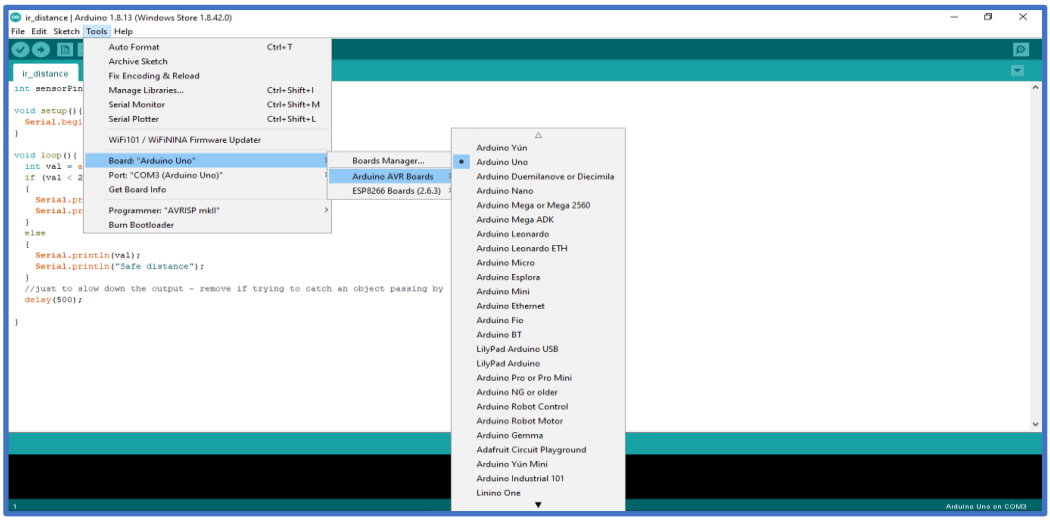


Figure 3.2: Arduino Uno board configuration

The above diagram is elaborated as follows-

1. The Arduino Uno board is connected to the computer via an USB cable and Arduino IDE is opened.
2. In IDE, select the Tools menu, configure the COM3 serial port and download the required libraries for Arduino Uno board.

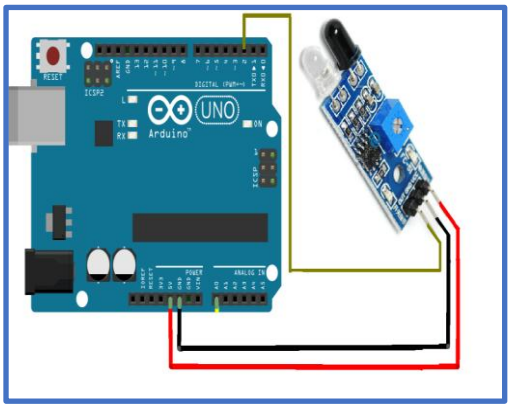


Figure 3.3: Schematic representation of distance measurement using IR sensor

- Serial. begin(9600): passes the value 9600 to the speed parameter. This tells the Arduino to get ready to exchange messages with the Serial Monitor at a data rate

of 9600 bits per second. That's 9600 binary ones or zeros per second, and is commonly called a baud rate.

- The integer reading reads the value of the ir sensor, it reads the analog input of the ir sensor then gives the output between 0 to 1023 (1024 bits(2^10 of adc)).
- The distance has been calibrated such that if the obstacle is detected within 5 cms / 20 of analog output it displays a "warning" else "safe distance" is displayed. A delay to display the outputs slowly on the serial monitor.
- IR sensors mounted on the outer body of autonomous car will emit an infrared beam of light via its transmitter.
- If an object is present in front of the sensor the IR will be bounced back and it'll be detected by the IR sensor's receiver.
- The distance can be calculated using the formula: -

  Distance = $\frac{(\text{Speed}*\text{Time})}{2}$ ..... (4.1)
- The speed is $3*10^8$ ms^{-1} (speed of light) and time is the interval between the transmission of IR beam and the detection.
- The time is divided by 2 because only the time interval of the detection of IR beam is considered as the IR bounces from the obstacle to the IR receiver.
- If the distance is in the range of say 100 cm to 150 cm then "Safe distance" is displayed on the output screen and if the distance is less than say 100 cm then "Warning" will be displayed.

3.4. IR Sensor Code

```
#define IR A0 // IR connection to A0.

int reading; // Declaration of valriables.

void setup()
{
  Serial.begin(9600);
}

void loop()
{
```

```
 reading = analogRead(IR); // Reading value.

 if(reading < 100) // Safe or not condition.
  {
   Serial.println("WARNING"); // Printing warning.
   Serial.println(reading); // Printing value.
  }
 else
 {
   Serial.println("SAFE"); // Printing safe if safe.
   Serial.println(reading); // Printing analog value.
 }

 delay(500); // Delay.

}
```

3.5 IR Sensor Program Outcomes

A. The IR sensor will measurement the distance between the autonomous car and the obstacles.
B. If the distance is less than the threshold value (5cm) then it will display a text “Warning”.
C. If the distance is above the threshold value (5cm) then it’ll display a text “Safe distance”.

Table No.1- Cost analysis

Components	Cost
Arduino Uno with USB cable	₹425
IR Sensor	₹35
Jumper cables	₹30
Total:	₹500

Chapter 4 -

Results & Conclusions

Using Python3 for coding the ML, the autonomous vehicles can detect and recognise the traffic lights without any human interaction. The ML has been programmed to eliminate any noise signals (other lights during day and night).

Figure 4.1: Green light detection

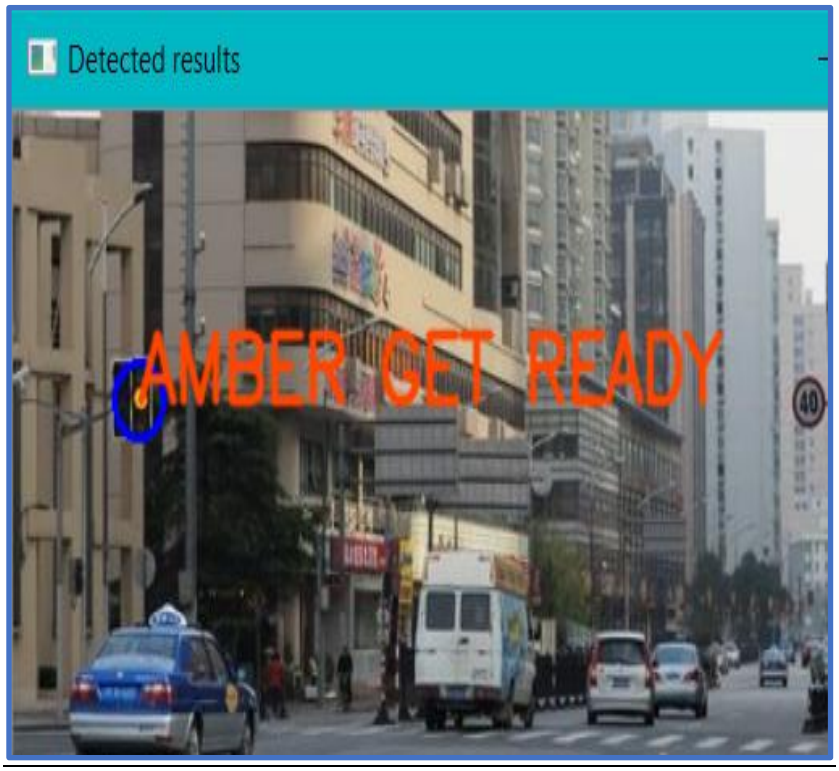


Figure 4.2: Amber light detection

Figure 4.3: Red light detection

Output loss reduction of the traffic light detection and recognition-

12BGTRAFFICJAM.jpg
Loss = 1.55 %
images.jpg
Loss = 0.8725 %
ntuser.dat
ntuser.dat.LOG1
ntuser.dat.LOG2
ntuser.dat{00ea0362-2a6c-11eb-ab85-c03eba38570c}.TM.blf
ntuser.dat{00ea0362-2a6c-11eb-ab85-c03eba38570c}.TMContainer00000000000000000001.regtrans-ms
ntuser.dat{00ea0362-2a6c-11eb-ab85-c03eba38570c}.TMContainer00000000000000000002.regtrans-ms
TRAFFIC_LIGHT (1).jpg
Loss = 1.4480000000000002 %
TRAFFIC_LIGHT (10).jpg
Loss = 1.55 %

TRAFFIC_LIGHT (11).jpg

Loss = 0.32599999999999996 %

TRAFFIC_LIGHT (12).jpg

Loss = 0.3769999999999999 %

TRAFFIC_LIGHT (13).jpg

Loss = 1.5245 %

TRAFFIC_LIGHT (14).JPG

Loss = 1.4735 %

TRAFFIC_LIGHT (15).JPG

Loss = 1.55 %

TRAFFIC_LIGHT (16).JPG

Loss = 1.55 %

TRAFFIC_LIGHT (17).JPG

Loss = 1.499 %

TRAFFIC_LIGHT (18).JPG

Loss = 0.3625 %

TRAFFIC_LIGHT (19).JPG

Loss = 1.4735 %

TRAFFIC_LIGHT (2).jpg

Loss = 0.4535714285714285 %

TRAFFIC_LIGHT (20).JPG

Loss = 1.0655000000000001 %

TRAFFIC_LIGHT (21).JPG

Loss = 0.6685 %

TRAFFIC_LIGHT (22).jpg

Loss = 0.9490000000000001 %

TRAFFIC_LIGHT (23).jpg

Loss = 0.8215 %

TRAFFIC_LIGHT (3).jpg

Loss = 1.244 %

TRAFFIC_LIGHT (4).jpg

Loss = 1.0 %

TRAFFIC_LIGHT (5).jpg

Loss = 0.7959999999999999 %

TRAFFIC_LIGHT (6).jpg

Loss = 1.0 %

TRAFFIC_LIGHT (7).jpg

Loss = 1.55 %

TRAFFIC_LIGHT (8).jpg

Loss = 1.0 %

TRAFFIC_LIGHT (9).jpg

Loss = 0.31150000000000005 %

Inference: Even though the loss was initially high, as the code iterated through images the loss was reduced significantly. The ML model encounters the RED light initially and if the RED-light repeats, then the machine learns and the loss is low. In the event, the ML model hits another colour after RED (say AMBER or GREEN), then the loss spikes until the model learns about those colours.

Using FC-51 IR sensors interfaced with an Arduino Uno board (R3 CH340G ATmega328p), for proximity sensing, the autonomous vehicle is made aware of obstacles in its near proximity.

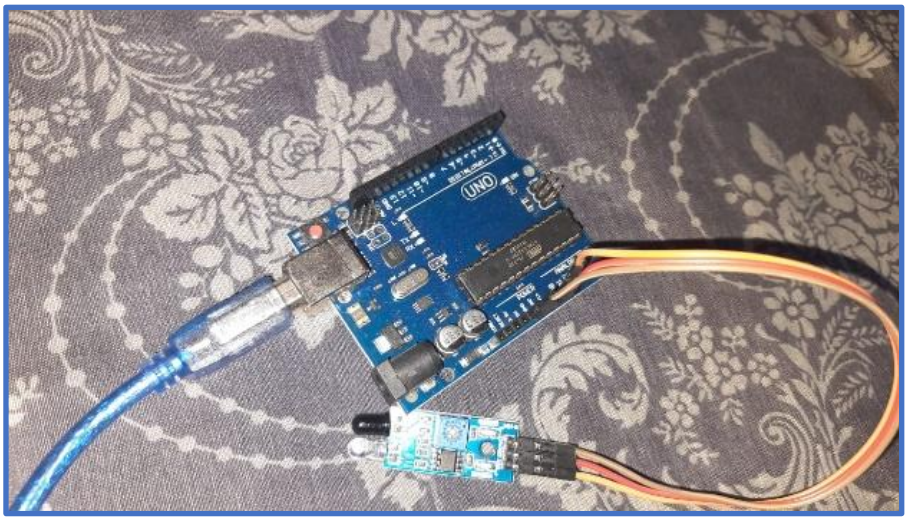


Figure 4.4: Circuit diagram of the IR sensor

The circuit diagram consists of Arduino Uno board (R3 CH340G ATmega328p) and FC-51 IR sensor. The IR sensor is interfaced with the Arduino Uno board with jumper cables where:

- Vcc pin of IR sensor is connected to the 5V pin of the Arduino board.
- OUT pin of IR sensor is connected to the A0 (Analog pin) of the Arduino board.
- GND pin of IR sensor is connected to the ground pin of the Arduino board.

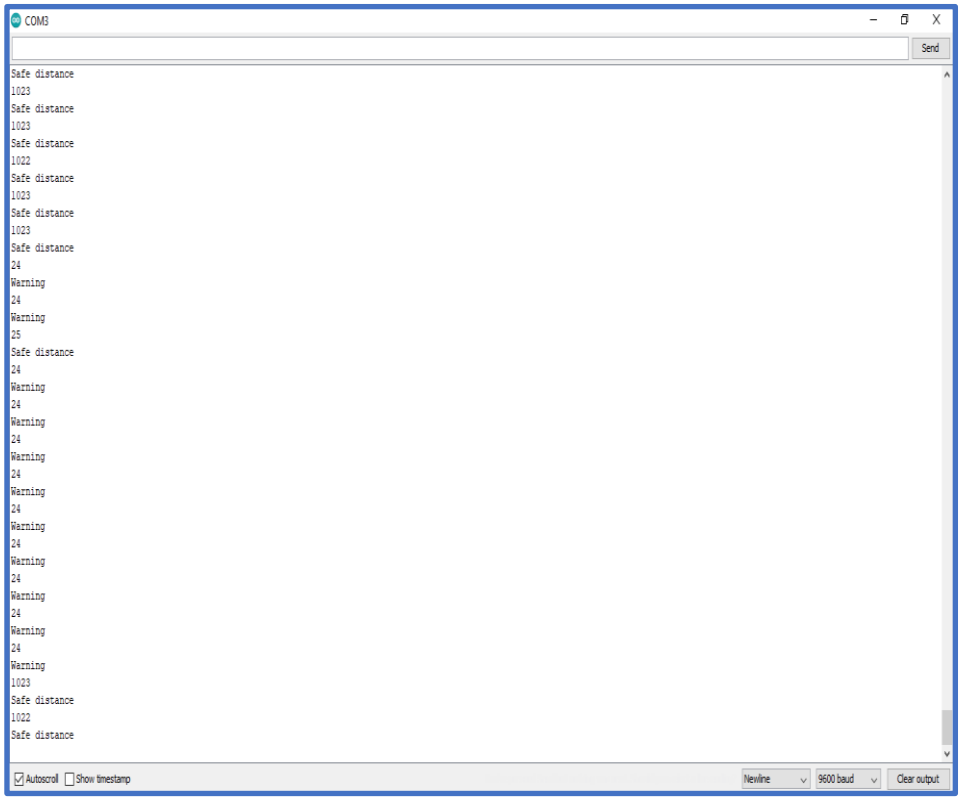


Figure 4.5: Output of the IR sensor

Table No.2 - Output tabular column

Arduino Digital Output Value Range	Scaled-down Value
17-25	Lesser than 5 cm (Warning)
959-1023	Greater than 5 cm (Safe)

The threshold value set for the above table is 5 cm.

4.1. Further scope

- ✓ A node MCU which is nothing but an ESP-8266 Wi-Fi module can be interfaced with the Arduino Uno board. The output from IR sensor can be sent to a centralised server via the internet (IoT) for further analysis and data processing.
- ✓ The data sent by the Wi-Fi module to the centralised server in the testing stations can be used in the simulation of the autonomous car in real time environment and a decision can be taken whether the vehicle is safe for usage on the roads.

- ✓ The results obtained by the IR sensor and TLs can be used to set a threshold for the autonomous car to take the appropriate action and can be used for calibration of the various sensors used in those cars.
- ✓ With the output from the IR sensor the autonomous car can take routes which have less traffic congestion with the help of the road mapping and GPS as the IR sensor will detect the traffic ahead of the autonomous car.

References

[1] Traffic Light Detection A Learning Algorithm and Evaluations on Challenging Dataset. By- Philipsen, Mark Philip; Jensen, Morten Bornø; Møgelmose, Andreas; Moeslund, Thomas B.; Trivedi, Mohan M. Published in: 2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC 2015), DOI: 10.1109/ITSC.2015.378. Accepted author manuscript, peer reviewed version.

[2] ML for Next-Generation Intelligent Transportation Systems: A Survey Tingting Yuan, Wilson Borba da Rocha Neto, Christian Rothenberg, Katia Obraczka, Chadi Barakat, Thierry Turletti HAL Id: hal-02284820 https://hal.inria.fr/hal-02284820v2 Preprint submitted on 28 Nov 2020.

[3] Traffic Light Recognition Using Deep Learning and Prior Maps for Autonomous Cars. By- Lucas C. Possatti.z, Rânik Guidolini, Vinicius B.Cardoso, Rodrigo F. Berriel, Thiago M.Paixão.y, Claudine Badue, Alberto F.De Souza, Senior Member, IEEE and Thiago Oliveira-Santos Universidade Federal do Espírito Santo, Brazil yInstituto Federal do Espírito Santo, Brazil, zEmail: lucas.possatti@lcad.inf.ufes.br

[4] Traffic Light Mapping and Detection Nathaniel Fairfield Chris Urmson {nfairfield, curmson}@google.com Google, Inc. Proceedings of ICRA 2011.

[5] Driver-initiated Tesla Autopilot Disengagements in Naturalistic Driving Alberto Morando MIT AgeLab, Massachusetts Institute of Technology morando@mit.edu Pnina Gershon MIT AgeLab, pgershon@mit.edu Bruce Mehler MIT AgeLab, bmehler@mit.edu Bryan Reimer MIT AgeLab, reimer@mit.edu.

[6] Revisiting the role of modular innovation in technological radicalness and architectural change of products: The Case of Tesla X and Roomba Tufail Habib University of Engineering and Technology, Peshawar, Pakistan Jimmi Normann Kristiansen Aalborg University Business School, Aalborg University, Denmark Mohammad B. Rana Aalborg University Business School, Aalborg University, Denmark Paavo Ritala Lappeenranta University of Technology, Finland.

[7] 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) October 25-29, 2020, Las Vegas, NV, USA (Virtual). A Scalable Framework for Robust Vehicle State Estimation with a Fusion of a Low-Cost IMU, the GNSS, Radar, a Camera and Lidar Yuran Liang1,2, Steffen Muller ¨ 1, Daniel Schwendner2, Daniel Rolle2, Dieter Ganesch2, Immanuel Schaffer2.

Appendix

Arduino Uno datasheet-

HK Shan Hai Group Limited
Room 620 , Yuhan building , songling road , Futian district , Shenzhen

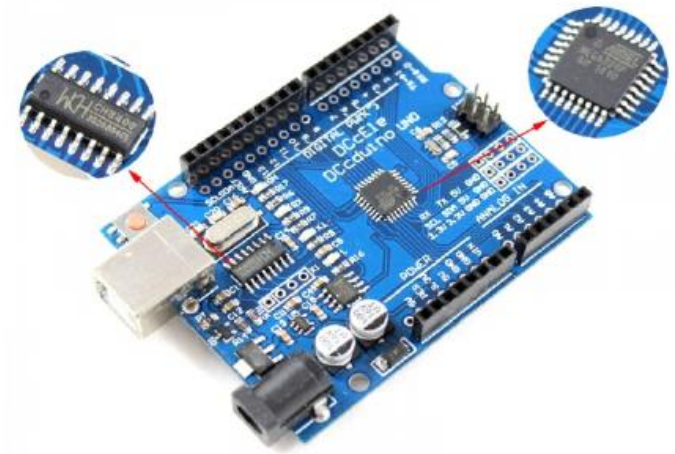
Technical Specification

EAGLE files: arduino-duemilanove-uno-design.zip Schematic: arduino-uno-schematic.pdf

Summary

Microcontroller	ATmega328P-AU
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

the board



HK Shan Hai Group Limited
Room 620 , Yuhan building , songling road , Futian district , Shenzhen

Power

The Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The Atmega328 has 32 KB of flash memory for storing code (of which 0,5 KB is used for the bootloader); It has also 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. TThese pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.

- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

HK Shan Hai Group Limited
Room 620 , Yitian building , zongling road , Futian district , Shenzhen

The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **I²C: 4 (SDA) and 5 (SCL).** Support I²C (TWI) communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

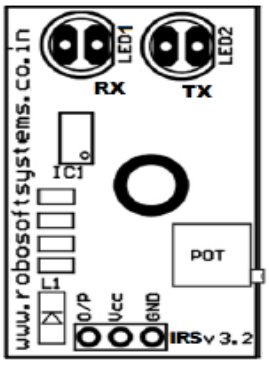
See also the mapping between Arduino pins and Atmega328 ports.

FC – 51 IR sensor datasheet-

Robosoft Systems

IR Sensor Based obstacle detection sensor module (Single)

IR Sensor - Single



General Description

The IR Sensor-Single is a general purpose proximity sensor. Here we use it for collision detection. The module consist of a IR emitter and IR receiver pair. The high precision IR receiver always detects a IR signal.

The module consists of 358 comparator IC. The output of sensor is high whenever it IR frequency and low otherwise. The on-board LED indicator helps user to check status of the sensor without using any additional hardware.

The power consumption of this module is low. It gives a digital output.

Pin Configuration

The figure to the right is a top view of the IR Sensor module. The following table gives its pin description.

Pin No.	Connection	Description
1	Output	Digital Output (High or Low)
2	VCC	Connected to circuit supply
3	Ground	Connected to circuit ground

Application Ideas

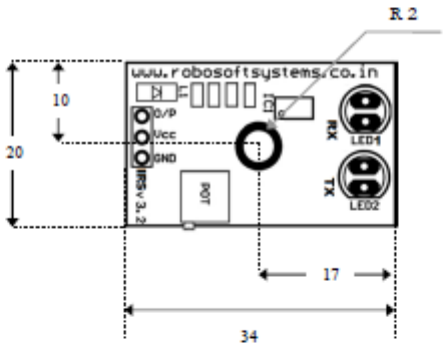
- Obstacle detection
- Shaft encoder
- Fixed frequency detection

IR Sensor - Single

Maximum Ratings

Symbol	Quantity	Minimum	Typical	Maximum	Unit
o/p	Output Voltage	0	-	5	V
V_{CC}	Operating Voltage	4.5	5	5.5	V
GND	Ground Reference voltage	-	0	-	V

Pin Out Dimensions



Note : All dimension in mm
Error of ±5% is subjected because of component soldering