ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/325475976

Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm

Chapter *in* Advances in Intelligent Systems and Computing · January 2019

DOI: 10.1007/978-3-319-78452-6_9

CITATIONS
5

READS
3,044

5 authors, including:

Aswath Suresh
New York University
17 PUBLICATIONS **124** CITATIONS
SEE PROFILE

Dhruv Gaba
New York University
8 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Debrup Laha
Polytechnic Institute of New York University
7 PUBLICATIONS **44** CITATIONS
SEE PROFILE

Siddhant Bhambri
Bharati Vidyapeeth College of Engineering, Delhi
7 PUBLICATIONS **44** CITATIONS
SEE PROFILE

All content following this page was uploaded by Aswath Suresh on 19 May 2019.

The user has requested enhancement of the downloaded file.

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

Design and Development of Intelligent Self-Driving Car using ROS and Machine Vision Algorithm

Aswath Suresh[1], Dhruv Gaba[1], Debrup Laha[1], and Siddhant Bhambri[2]

[1] Department of Mechanical and Aerospace Engineering, New York University, USA
[2]Department of Electronics and Communication Engineering, Bharati Vidyapeeth's College of Engineering, India

as10616@nyu.edu, dg3035@nyu.edu, dl3515@nyu.edu, siddhantbhambri@gmail.com

Abstract. The technology of self-driving cars is quite acclaimed these days. In this paper we are describing the design and development of an Intelligent Self Driving Car system. The system is capable of autonomously driving in places where there is a color difference between the road and the footpath/roadside, especially in gardens/parks. On the basis of the digital image-processing algorithm, which resulted into optimal operation of the self-driving car is based on a unique filtering and noise removal techniques implemented on the video feedback via the processing unit. We have made use of two control units, one master and other is the slave control unit in the control system. The master control unit does the video processing and filtering processes, whereas the slave control unit controls the locomotion of the car. The slave control unit is commanded by the master control unit based on the processing done on consecutive frames via Serial Peripheral Communication (SPI). Thus, via distributing operations we can achieve higher performance in comparison to having a single operational unit. The software framework management of the whole system is controlled using Robot Operating System (ROS). It is developed using ROS catkin workspace with necessary packages and nodes. The ROS was loaded on to Raspberry Pi 3 with Ubuntu Mate. The self-driving car could distinguish between the grass and the road and could maneuver on the road with high accuracy. It was able to detect frames having false sectors like shadows, and could still traverse the roads easily. Thus, self- driving cars have numerous advantages like autonomous surveillance, car- parking, accidents avoidance, less traffic congestion, efficient fuel consumption, and many more. For this purpose, our paper describes a cost-effective way for implementing self- driving cars.

Keywords: Autonomous, Drive, Image Processing, ROS, SPI

1 Introduction

An autonomous car is a vehicle that is capable of sensing its environment and navigating without human input. Many such vehicles are being developed, but as of May 2017 automated cars permitted on public roads are not yet fully autonomous.

All self-driving cars still require a human driver at the wheel who is ready to take control of the vehicle at any moment. Autonomous cars use a variety of techniques to detect their surroundings, for path planning and efficient decision-making they uses radar, laser light, GPS, odometer, and computer vision techniques and many more for correct behavior of the car. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant road signs. Path planning and obstacle avoidance utilizing the information obtained by the above-mentioned sensors and techniques to drive to the goal without intervention of a driver [1].

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

Most of the driver assistance systems for the self-driving cars are developed to assist the drivers and to improve the driver's' safety such as in lane keeping [2], autonomous parking [3], and collision avoidance [4]. Such systems are used heavily to reduce traffic and accident rates. And currently rigorous Research is going on in this field to improve the accuracy of the self-driving cars.

The major impetus which leads to the development of the self-driving car technology was the Grand Challenge and Urban Challenge organized by DARPA (Defense Advanced and Research Projects Agency) [5][6]. These were first organized in 2004, then in 2005 (Grand Challenge), and then followed in 2007 (Urban Challenge). This has led to numerous studies regarding accurate path planning, which are categorized into three major areas namely, Potential-field approach [7], Roadmap based approach [8] [9] [10] [11], and Cell decomposition based approach [12].

In paper [13], Voronoi cell based approach is used for path representation, collision detection and the path modification rather than waypoint navigation. But the inaccuracy in predicting the size of these cells in a dynamic environment makes the approach ineffective. On the other hand, our color segregation based approach (explained in Section 4) that does not require cells and path planning done in real time environment, has a higher performance.

The complete system is subdivided into the drive mechanism and the control system of the car. The drive mechanism is responsible for the locomotion of the car as it has the motors, tires, mount for RaspiCam, chassis etc. for the operation of the car. But, the control system is the most important part of the system because the entire intelligence of the system is residing in this part. The control system is subdivided into a master control unit and a slave control unit, which are explained in detail in the Section 3. The reason for choosing distributed control systems for our self-driving car is to improve the overall performance of the system.

In this paper we are explaining the algorithm that was created for the autonomous car and its implementation on the self-driving car prototype to verify its capabilities. The key aspect of our algorithm behind the autonomous behavior of the car is the detection of color difference between the road and the green grass to determine the direction in which the car must turn based on the run-time camera input and removal of noise for error free decision making is also done and explained in the Section 4. The programming of the prototype was done in “OpenCV (Python)”.

The upcoming sections explains the following information:

Section 2: Mechanical Structure and Design of the system

Section 3: Electrical connections and architecture of data flow

Section 4: Algorithm used for decision making

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

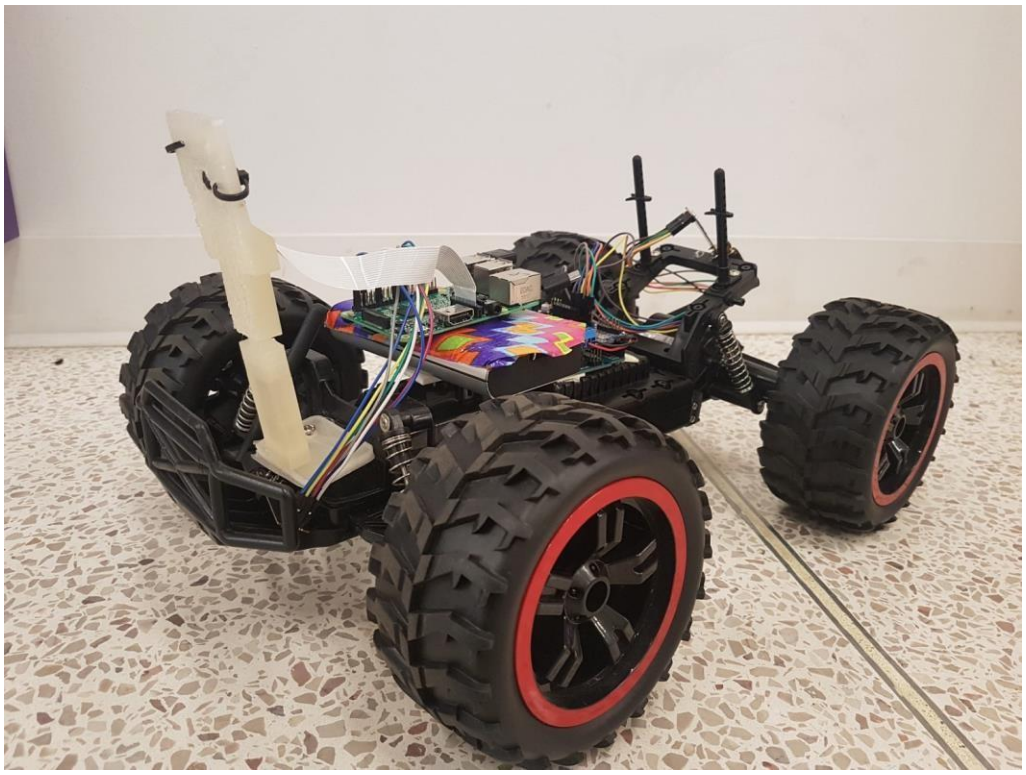


Fig 1: Assembled Autonomous System

2 Mechanical Design

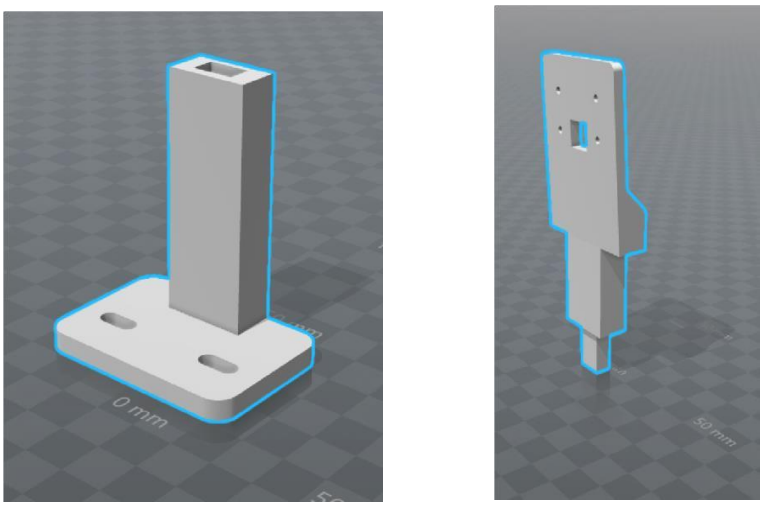
The mechanical design of the car is shown in the Fig. 1. The car in the image is a prototype for representing our algorithm for effective determination of the direction of the car's movement.

For the prototype of the RC car, at first the circuit of the RC car is removed, and the control of the system is given to the Arduino development board using the Adafruit shield. In this mechanical structure, Ping Sensors (Ultrasonic Sensor) are attached for the autonomous braking to avoid collision.

The RC car used for the prototype is based on differential drive mechanism. Differential drive robots are based on differential drive mechanism in which the motion of the system is divided into two sections which are right and left parts. The wheels of both segments are attached to common axis and have independent motion and rotation capabilities. If the system consists of two wheels, both the wheels can independently be driven either forward or backwards. In differential drive, we can vary the velocity of each wheel independently and change the direction of motion of the whole unit.

Further, the camera mount is designed for mounting the RaspiCam on the front end of the car to get the required field of vision for the efficient image processing. The Camera angle from the car must be 20° at the height of 20 cm. For this purpose, we have designed and 3D printed the camera mount, as shown in the Fig. 2 and Fig 3. Also, prototype of car driving autonomously in park is as shown in Fig. 4.

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham



(a) Isometric View of Lower part (b) Isometric View of Upper Part

Fig 2: Camera Mount Design

Fig 3: 3D Printed Camera Mount

The camera mount is fixed on the front end of the car as shown in the fig.3.

Fig 4: Car driving autonomously in park

3 Electrical Design

The control system of the autonomous car is subdivided into master and slave control unit. The master control unit is the Raspberry Pi B3 microprocessor and the slave control unit is based on Arduino development platform. We separated the two control units and distributed the task allocated to them because by this manner we can achieve higher responsiveness and better performance of the system.

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

Architecture of Control system: Control and Data flow

The architecture and flow of control as well as data in our system is explained in the following subsection as shown in Fig.5.:

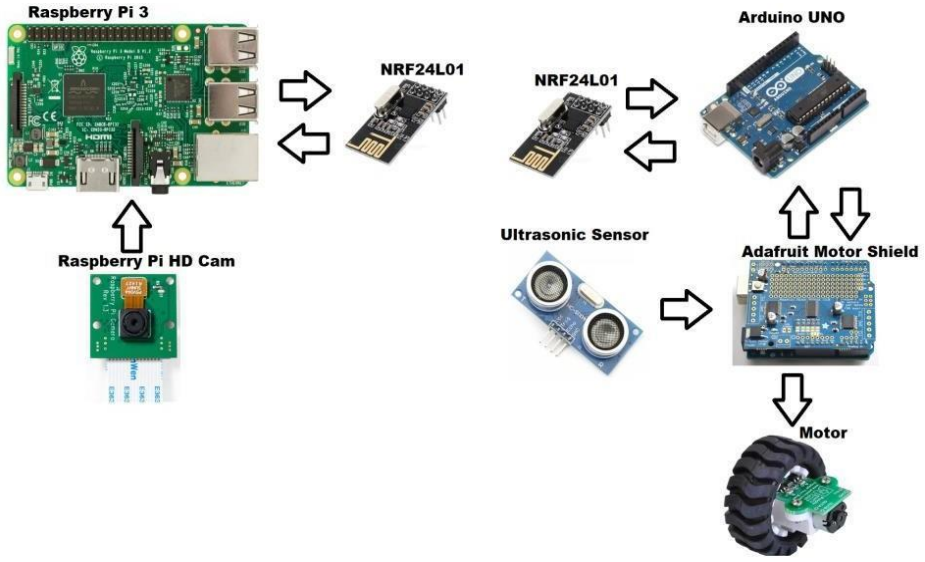


Fig 5: Architecture of Data flow in the system.

3.1 RaspiCam Video Camera

The video is captured using the RaspiCam, which is a 5MP camera attached with the Raspberry Pi B3 such that the video (i.e. image frames) are recorded and then sent to the Raspberry Pi microprocessor for processing of data and decision making.

3.2 Master Control Unit: Raspberry Pi B3 Microprocessor

The master control unit is used for video processing in the system. The video recorded by RaspiCam is converted into image frames and processed by Raspberry Pi B3 microprocessor because of its higher processing power and additional capabilities which makes it the most suitable choice for this application. The algorithm used for the processing of the image frames is explained in detail in section 4. At the end of the algorithm a message array of 8 bits is created which is transmitted wirelessly to the Arduino platform for further executions.

3.3 Slave Control Unit: Arduino Development Platform

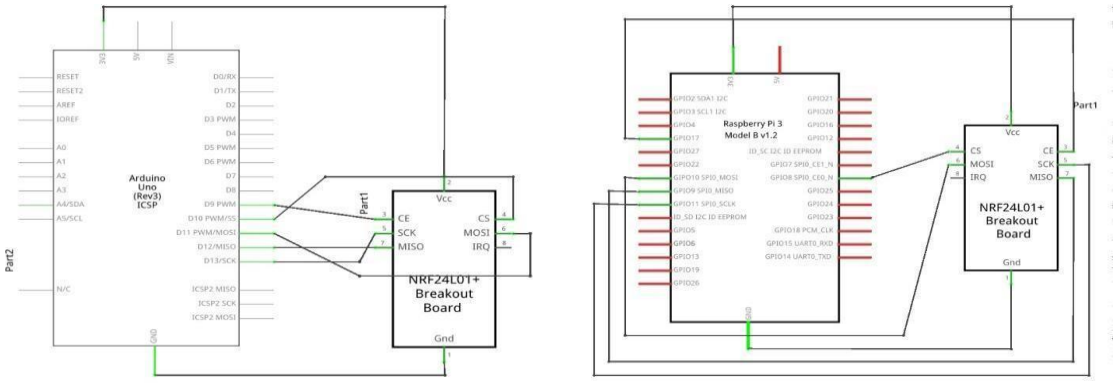
The slave micro-controller is used for controlling the drive system of the car. The Arduino is connected to the Adafruit motor shield which drives power from the Lithium Polymer battery. Arduino commands the motor shied and then it controls the operation of the motors for the drive of the car.

3.4 Wireless Serial Peripheral Interface Communication: nRF24L01 Module

The results generated after the processing of data by implementing the algorithm in the master control unit, are to be transmitted to the slave control unit (Arduino Development Platform) for maneuvering the car. The communication between the master and the slave control unit is achieved using the Serial Peripheral Interface (SPI) communication. SPI wireless communication is achieved using the nRF24L01 (2.4 GHz) transceiver modules and the data is transmitted in the form of 8-bit signal. The signal bits are interpreted in the following manner at the slave control unit:

1. Bits 1-3: Driving motor speed
2. Bits 4-6: Steering motor speed
3. Bit 7 : Driving motor direction
4. Bit 8 : Steering motor direction

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham



(a) Arduino and NRF module (b) Raspberry pi and NRF module

Fig 6: Circuit Diagram of NRF module

3.5 Ping Sensors (Ultrasonic Sensor)

For emergency breaking system of the autonomous self-driving car, we have used Ping sensors. The slave control unit monitors the readings from the Ping Sensors, such that if by any means something or someone comes in the way of the car the breaks are applied immediately to avoid collision.

3.6 Adafruit Motor Drive Shield

The Adafruit Motor Shied is attached to the Arduino development platform. Arduino commands the motor shied which further controls the motors of the car. We have used motor shield to segregate the high current battery supply for the motor from the low current supply for the electronic systems.

3.7 Power Source: Lithium Polymer Battery

We are making use of Lithium Polymer Battery (11.1 V-3s) for providing power supply to the entire system. Direct supply from the battery is given to the motors as they have high current rating and regulated voltage (5V from 7805IC) for the electronics controllers as they operate on 5 Volts. This is the common source of power for the system.

3.8 ROS: Robot Operating System

All the software frame work of the system is well developed using ROS catkin workspace with necessary packages and nodes. The ROS was loaded on to Raspberry Pi 3 with Ubuntu Mate. Python scripts had been used for programming in ROS.

3 Algorithm Explanation

We extract the video feedback from the RaspiCam, which is recorded by the Raspberry Pi 3, and dynamically store in it for processing. Raspberry Pi processes the input frame by frame, on which we sequentially implement the following filters for decision making.

3.1 Resizing of Image Frame

The RaspiCam provides us with highly detailed images. It has 5MP camera for HD recording. For rapid response of our system, we have chosen our image to have 600 Pixel of width because processing on large matrices (derived from the images) will take larger amount of time to process it. So, to avoid lag in our system we are

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

reducing the size of our image by using “imutils.resize()” command of “OpenCV” library as shown in Fig.7. This way all our frames are resized to 600 pixels width, which makes our filtering procedure more convenient and efficient.

```
image = imutils.resize(frame, width=600)
```

Fig 7: Function for resizing the frame width.

3.2 Gaussian Blur Algorithm

Every image, which we get from the RaspiCam is having some or the other form of high frequency noise. So, to remove this noise we are making use of low pass filters. One of the most common forms of algorithm which will not only remove the sharp noises, but also blur the image is the Gaussian Blur filters.

For operation of this filter we need to specify the width and height of the kernel to the function, which should be positive and odd. Here we have used a kernel of 11 sizes. Further, we will have to specify the standard deviation in X and Y direction, i.e. σ_X and σ_Y; here we are using both as zeros, thus they are calculated from the kernel size. Therefore, via Gaussian blur we are able to remove Gaussian noise very effectively as shown in Fig.8.

```
image = cv2.GaussianBlur(image,(11,11),0)
```

Fig 8: Illustrating implementation of Gaussian blur and its function code.

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

ALGORITHM FLOWCHART:

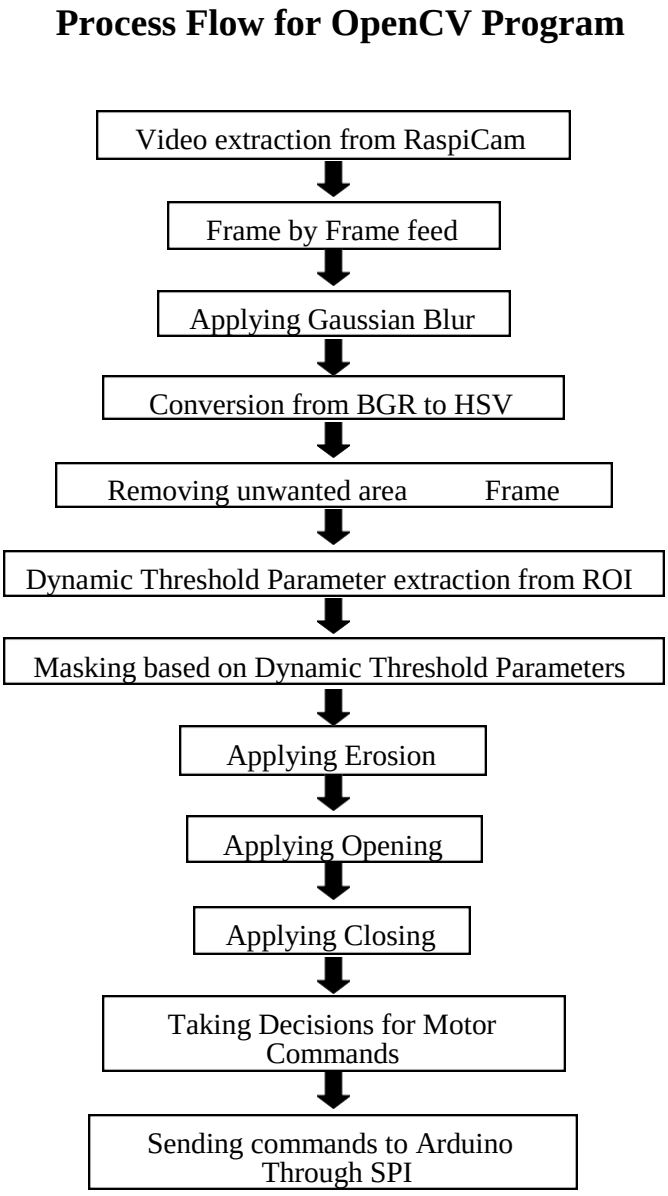
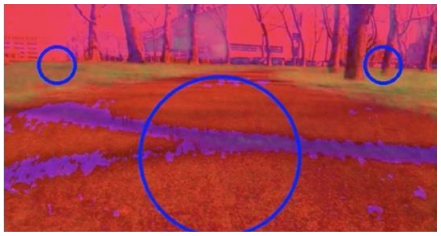


Fig 9: Flowchart for the decision making algorithm

The decision making algorithm of the self-driving car is as shown in flow chart in Fig.9.

3.3 Conversation from BGR (Blue, Green, Red) to HSV (Hue, Saturation, Value)

We are changing the color space from BGR to HSV such that we can effectively segregate the road from the green grass. In this step we are simply converting the entire BGR image/frame (width = 600) to HSV color space so that just by simply making a range mask, we are able to segregate the desired color range which is grey road from the green grass in the case of a park as shown in Fig 10.



```
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

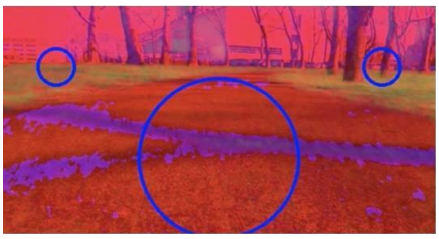
Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

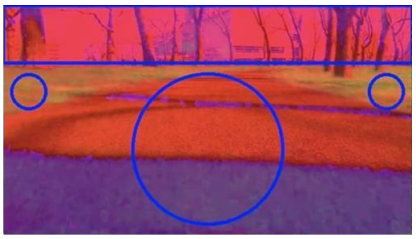
Fig 10: Illustrating the conversion of BGR to HSV color space.

3.4 Removing unwanted area from Image Frame

After the image is converted from BGR color space to HSV color space, we find the area having the most information present and remove the unwanted area from the frame. For our system to perform with utmost accuracy in any unstructured environment, we need to remove all unnecessary information from the frame. For example, we will have to remove those regions which do not contain the road and grass information. These unwanted areas like the sky, the trees, twigs, sticks, etc. which do not have road and grass, are to be removed. This parameter of new height of the frame to be used is dependent on the height of the camera and the vertical inclination of the camera. Therefore, after this step we get the updated frame for higher performance.

Here we are making use of np.size() function to get the height and width dimension of the HSV color space. Using this, we adjusted the new frame for our processing as shown in Fig.11.





```
height = np.size(hsv,0)
width = np.size(hsv,1)
hsv = hsv[5*height/16:height, 0:width]
```

Fig 11: Illustrating removal of unwanted areas.

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

3.5 Dynamic Threshold Parameter

In this step we are adjusting our frame size such that we could determine the required Region of Interest (ROI). As shown in the Fig. 12, we have used the following command to get it.

```
path = hsv[1*height/8:height,width/2-(2*width/8):width/2+(2*width/8)]
```

Fig 12: Dynamic thresholding of the Parameter ROI

3.6 Dynamic Threshold Setting

In this code we could take values from the path [] matrix (created from hsv [] matrix) and assign its corresponding values to h [], s [], and v [] arrays. Then we simply find the values which are the maximum and minimum of the h [], s [], and v [] arrays and save them in hmax, hmin, smax, smin, vmax, and vmin. And then correspondingly using them to create a two "numpi" arrays ("rangemin[] and rangemax[]"), one having the maximum ranges and the other having the minimum ranges as shown in Fig.13.

```
path = hsv[1*height/8:height,width/2-(2*width/8):width/2+(2*width/8)]

h = []
s = []
v = []

for i in path:
    for j in i:
        h.append(j[0])
        s.append(j[1])
        v.append(j[2])

hmin = min(h)
hmax = max(h)
smin = min(s)
smax = max(s)
vmin = min(v)
vmax = max(v)

rangeMin = np.array([hmin, smin, vmin])
rangeMax = np.array([hmax, smax, vmax])
```

Fig 13: Code for finding the minimum and maximum ranges of hue, saturation, and value in HSV color space.

3.7 Masking based on Dynamic Threshold Parameters

The following code creates the mask for transforming the image frame in accordance to the min and max range provided from the previous steps as shown in Fig.14. We have made us of cv2.inRange () function to create this mask.

```
mask = cv2.inRange(hsv, rangeMin, rangeMax)
```

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

Fig 14: Illustrating mask creation from the max and min ranges of HSV.

3.8 Applying Erosion, Opening, and Closing

In this step we are performing Erosion, Opening and Closing functions one by one to remove various kinds of noises from our image frames, which is the basis of our decision making.

- Erosion will remove the boundaries of the foreground object (displayed in white) as the kernel (25x25) slides over the image (convolution), and the pixel in the original image is considered 1 only if all the pixels in the kernel are 1, otherwise it is eroded 0.
- Opening is another function to remove noise and it functions in a manner of erosion followed by dilation, thus removing small particle noise from the image frame.
- Closing is reverse of Opening, i.e. Dilation followed by Erosion; it is useful for closing small holes, or small black points on the object.

Therefore, implementation of these filters one by one helps the robot to render the clean and noise free image frame as shown in the Fig. 15.

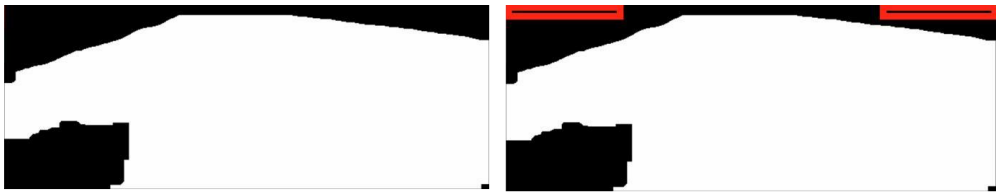
```
kernel = np.ones((25,25),np.uint8)
erosion = cv2.erode(mask,kernel,iterations = 1)
cv2.imshow("Erosion",erosion)
opening = cv2.morphologyEx(erosion, cv2.MORPH_OPEN, kernel)
cv2.imshow("opening",opening)
kernel = np.ones((15,15),np.uint8)
closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)
cv2.imshow("closing",closing)
```

Fig 15: Implementation of Erosion, Opening, and Closing filters in code and frames.

3.9 New Regions of Interest for Turning Decisions/ Taking Decisions for Motor Commands

The following code represents the 'if-else' conditions which are used to determine the direction of the drive motor and the steering motor. These 'if-else' loops check for the white and black regions on the top left and right side of the image frame. This way we can decide in which direction the car has to move, and then transmit the data to the slave controller (Arduino).

Also, the second image Fig 16 shows the formatting of the 8-bit data encoding which is transmitted to the slave controller. In Fig. 16 the first three bits represent the Drive Wheel Velocity, the next three consecutive bits represent the Steer Wheel Velocity, the second last bit represents the direction of the Drive Wheel, and the last bit represents the direction of the Steer Wheel.



```
leftTop  = closing[0][2*wMask/16]
rightTop = closing[0][14*wMask/16-1]
if leftTop == 0 and rightTop == 0:
    message = "10000011"
elif leftTop == 255 and rightTop == 255:
    message = "10000011"
elif leftTop == 255 and rightTop == 0:
    message = "10012011"
elif leftTop == 0 and rightTop == 255:
    message = "10012010"
print message
```

```
8 Bit data encoding

1 2 3 - Drive Wheel Velocity

4 5 6 - Steer Wheel Velocity

7 - Drive Direction

8 - Steer Direction
```

Fig 16: Determination of the direction of turning.

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

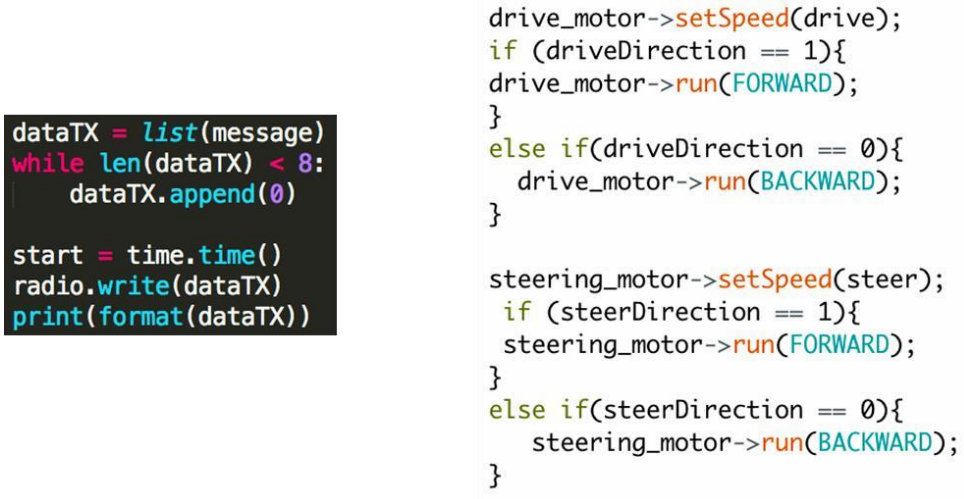
3.10 Sending commands to Arduino through SPI

Once the algorithm has determined the direction in which the car has to move, the master controller (Raspberry Pi) will transmit the data via wireless serial communication to the slave controller (Arduino).

The dataTX takes the message generated from the previous step and it will save it in the dataTX which is transmitted serially to Arduino. Also, in this step we check if the length of the dataTX is less than 8. If it is less than 8, we will have to append a 0 to make it 8-bit to avoid any errors which could happen during wireless communication.

Now, serial communication begins and we record time using time.time() function into the start variable. After that we write this data to the radio transmitter which will transmit it wirelessly to Arduino for movement and simultaneously print it onto the console as well.

The transmitted code is then received by the Arduino, which is further decoded by it for implementation of the locomotion commands. For example, if the drive or steering direction bit is 1 then move the respective motor in forward direction, whereas if the drive or steering direction bit is 0 then move the respective motor is moved in the backward direction as shown in Fig.17.



```
dataTX = list(message)
while len(dataTX) < 8:
    dataTX.append(0)

start = time.time()
radio.write(dataTX)
print(format(dataTX))
```

```
drive_motor->setSpeed(drive);
if (driveDirection == 1){
drive_motor->run(FORWARD);
}
else if(driveDirection == 0){
  drive_motor->run(BACKWARD);
}

steering_motor->setSpeed(steer);
 if (steerDirection == 1){
 steering_motor->run(FORWARD);
}
else if(steerDirection == 0){
   steering_motor->run(BACKWARD);
}
```

Fig 17: Illustration of message signal transmission from Raspberry Pi and decoding of the message signal at Arduino.

Conclusion

Thus, for reducing the cost of self-driving cars we use normal cameras and sensors instead of complicated sensors, which tend to be quite expensive. So, our system resulted into a more cost-effective system as compared to the current ones. Additionally, the digital image processing techniques used in the algorithm of the self-driving car had satisfactory results. We implemented and tested the self-driving car in Cadman Plaza Park.

Future Scope

As for future development, we could use machine learning and deep learning techniques for even higher accuracy of the system and dynamic upgradation of the systems parameters according to its surroundings could also be implemented onto the system.

Also, for those roads which do not have significant color difference between road

Cite this paper as:
Suresh A., Sridhar C.P., Gaba D., Laha D., Bhambri S. (2019) Design and Development of Intelligent Self-driving Car Using ROS and Machine Vision Algorithm. In: Kim JH. et al. (eds) Robot Intelligence Technology and Applications 5. RiTA 2017. Advances in Intelligent Systems and Computing, vol 751. Springer, Cham

and the side roads, we could use the road dividers as markers to stay on the road as the object recognition algorithms will track these road dividers for driving the car on the road.

Acknowledgement

The authors would like to thank Makerspace, New York University to provide support and resources to carry out our research and experiments.

References

[1] J. Wit, C. D. Crane, and D. Armstrong, "Autonomous Ground Vehicle Path Tracking," Journal of Robotic Systems, vol. 21, no. 8, pp. 439-449, 2004.
[2] J. Wang, S. Schroedl, K. Mezger, R. Ortloff, A. Joos, and T. Passegger, "Lane Keeping Based on Location Technology," IEEE Transactions on Intelligent Transportation Systems, vol. 6, no. 3, pp. 351-356, September 2005.
[3] Li, TS and Yeh, Ying-Chieh and Wu, Jyun-Da and Hsiao, Ming-Ying and Chen, Chih-Yang, "Multifunctional intelligent autonomous parking controllers for carlike mobile robots", IEEE Transactions on Industrial Electronics, Vol. 57, No. 5, pp. 1687-1700, 2010.
[4] Kwanghyun Cho, Seibum B. Choi, "Design of an Airbag Deployment Algorithm Based on Pre-Crash Information", IEEE Transaction on Vehicular Technology, Vol. 60, No. 4, pp. 1438-1452, 2011.
[5] Ü. Özgüner, C. Stiller, and K. Redmill, "Systems for safety and autonomous behavior in cars: The DARPA Grand Challenge experience," Proc. IEEE, vol. 95, no. 2, pp. 397-412, Feb. 2007.
[6] Thrun, S., et al., "Stanley: The robot that won the DARPA Grand Challenge", Journal of Field Robotics, 23(9), 661-692, 2006.
[7] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," The International Journal of Robotics Research, vol. 5, no. 1, pp. 90-98, March 1986.
[8] S. Koenig and M. Likhachev, "Fast Replanning for Navigation in Unknown Terrain," IEEE Transactions on Robotics, vol. 21, no. 3, pp. 354-363, June 2005.
[9] N. S. V. Rao, N. Stoltzfus, and S. S. Iyengar, "A Retraction Method forLearned Navigation in Unknown Terrains for a Circular Robot," IEEE Transactions on Robotics and Automation, Vol. 7, No. 5, pp. 699-707, October 1991.
[10] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," IEEE Transactions on Robotics and Automation, Vol. 12, No. 4, pp. 566-580, August 1996.
[11] H. M. Choset, Principles of Robot Motion: Theory, Algorithms, and Implementation. Cambridge, MA: MIT Press, 2005.
[12] L. Zhang, Y. J. Kim, and D. Manocha, "A Hybrid Approach for Complete Motion Planning," in IROS, October 2007, pp. 7-14.
[13] Unghui Lee, Sangyol Yoon, HyunChul Shim, Pascal Vasseur, Cedric Demonceaux, “Local path planning in a complex environment for self-driving car,” The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems June 4-7,2014, Hong Kong, China.

View publication stats