CHAPTER 24

Open-Pit Mine Autonomous Bot

Apoorva Parashar
Maharshi Dayanand University
Rohtak, India

Anubha Parashar and Vidyadhar Aski
Manipal University Jaipur
Jaipur, India

CONTENTS

24.1 INTRODUCTION

Open-pit mining is mining of an open-cut or a cast that is open. This method is used to extract minerals by removing them from open burrows of the surface of the earth. The project was focused on developing an unmanned ground vehicle for working in the environment of open-pit mines [1–3]. These techniques are distinct from the ones that

DOI: 10.1201/9781003048381-24

require extracting by tunneling the earth. Mines that are open are more useful when ore deposits are closer to the surface of earth [4]. Extracting is done here because the surface structure is thin and material is not suitable for tunneling, for instance, gravel or sand. Underground mining can be useful for searching for the ores that are hard to retrieve because the rock is hard.

Mining in an open-pit is very risky considering its economic viewpoint. One of the starting point of risk is the stability risk. Stabilizing slope comes with many risks in multiple ways, for instance, when we do not have full understanding of geological environments, the harm that mining can cause, and work procedures not able to be executed till completion.

There are two classes of geotechnical perils: moments of slope and falling rock that were not planned [5]. Major consequences of all the geotechnical perils include possible casualties. The probability of all the outcomes must be evaluated very carefully. The risk of falling rocks can lead to casualties and loss in many ways. Production might even halt because of safety issues [6].

The proposed vehicle would easily maneuver along the path of the pit autonomously (Figure 24.1) transporting ore or disposing waste rocks to the dumping ground. Due to the robust localization technique, the vehicle could operate in remote areas without any external help. The objective of the project is to minimize worker hazards and ensure human safety in dangerous places like mining.

Two of the most treacherous postings in mining are transporting goods and their haulage as most of the mishaps and loss happen while carrying out this work. Lately,

FIGURE 24.1 Autonomous driverless car.

due to developments in the automation sector, mining is also on the verge of change; most of the work will be carried out by driverless vehicles, which will reduce the risk factor [7–15].

Vehicles that are driver-less can keep running for a very long time. Such vehicles can run non-stop for a year saving up to 500 hours of work every year. Not only can these vehicles work non-stop, but they also help in solving problems like reduction of fatality and error. Using these vehicles will hugely influence mining in jobs and make them easier (Figure 24.2) [8, 9].

There are few limitations to wall followers, i.e., it is not an intelligent system as the bot is not operational in conditions like landslides and when there are intersections on the path. The bot can easily be misguided into following the wrong path, which could lead it to fall off the edge of the road. The bot is not able to correct itself successfully if it comes too close to the wall. Any attempt to correct its position resulted in the bot moving in a zigzag pattern and thus it failed to follow a straight path as intended. The bot moved with constant speed, which resulted in near collision with the wall in front at turnings, as the ultrasonic sensor in the front of the bot perceived those walls as obstacles so the bot movement was not fluent. Bot movement was inconsistent at turnings in the pit, as distance from the walls at turning is not constant, and the bot tried to maintain a constant distance with the wall.

FIGURE 24.2 Autonomous vehicle architecture.

The following features are incorporated in our final iteration of the bot:

1. The bot can be trained to follow any irregular path, thus making it expendable for any rogue mining environment. Training can be done for as many times as the user wants.
2. Localization of the setup is done using AprilTags, which can identify the live position of the bot in the pit (*X*, *Y*, *Z* coordinates along with its yaw, roll, and pitch).
3. AprilTags provide us with an ingenious solution to localization, which can be implemented in any environment and is cost-effective compared with conventional GPS techniques.

   Accuracy of GPS depends on blockage of signal, conditions related to atmosphere, blockage of any signal, and designing or attributes of the receiver. Moreover, precision decreases near trees and bridges. Thus using AprilTags was more logical in this scenario.
4. The training and autonomous drive initiation are done through a user-friendly graphical user interface (GUI) based on Python.

24.2 PROPOSED METHODOLOGY

In this chapter, we have enhanced the working principle according to best articles in recently published research papers. Before reaching to the final build for the bot, several iterations were attempted to drive the vehicle autonomously in the pit. In the following sections we will walk through the same (Figure 24.3).

The pit consists of seven segments of slope stacked together to form a spiral platform of increasing height (Figure 24.4).

A sample dimensional unit of a slope is represented in Figure 24.4. The labeled sides (a–f) are the repeating units of dimensions on each slope and are tabulated for the seven slopes in Table 24.1.

The initial idea was to build a simple wall follower to trace the path of the pit by maintaining a certain distance from the walls on the left side of the pit. To perform it, an ultrasonic sensor is necessary that would require a minimum level of complexity as the bot would have to find the total distance from the wall. We can use an ultrasonic sensor for observing barriers when the robot is in motion as these sensors have a high-level range and a low price [16–24].

The detection mechanism of an ultrasonic sensor is that it discharges bursts of wavelengths in order to listen the echo produced. A short pulse of 40 kHz is released by the sensor when it is controlled by a microcontroller host. The pulse creates advances in the air until it is stopped by any object. After hitting the object surface, the pulse retraces back to the origin point sensor. After the echo is perceived by the sensor, an output signal is generated to request the host to terminate. This is the full timeline for calculating the object distance via returned pulse width [25].

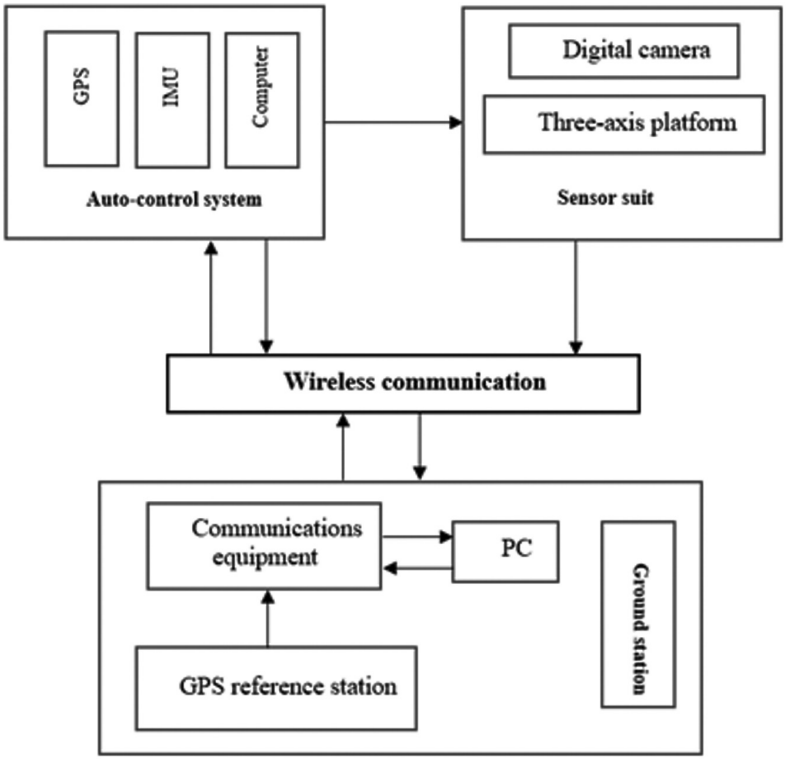

FIGURE 24.3 Proposed methodology.

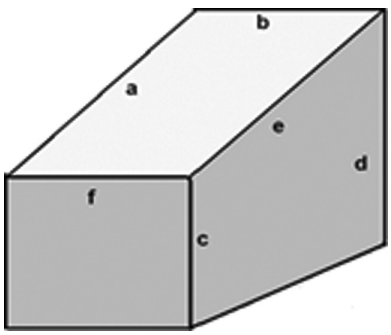

FIGURE 24.4 Slope of the pit.

TABLE 24.1 Pit Dimensions (in Centimeters)

Slope	a	b	c	d	e	f
1	50	27	0	16	40	34
2	40	25	16	23	24	27
3	47	27	23	28	27	25
4	46	26	28	32	25	27
5	47	27	32	37	28	26
6	46	25	37	41	29	27
7	47	31	41	46	57	25

The sensor keeps transmitting waves as the robot walks on the path selected. The distance is computed each time there is an obstacle on the path [26]. If the distance is greater than a threshold the Arduino commands the bot to change the steering angle toward the wall to avoid falling over the edge. The distance between the object and SONAR sensor can be computed by taking the time spent from generation of a wave to the wave retracing back after hitting any object (Figure 24.5).

Echo, Vdd, Trigger, and GND are the four pins found in the HC-SR04 sensor. An ultrasound wave is discharged as a pulse is bid to the Trigger pin. The function of the Echo pin is to transmit the waves detected by the receiver to the controller. The pulse at the Echo pin is directly proportional to the distance between obstacle and the sensor [27].

The following formula is used to calculate distance (units: centimeters):

$$\text{Distance} = 0.5 \times (344 \times \text{reflection time})$$

To reach the closest to the left wall, the robot is instructed to turn left and move forward as soon as the power is turned on. The robot is also instructed to read values from the sensor and to keep moving forward. It rotates its direct current (DC) motor to the right with full speed to take a left turn and does so till it achieves a minimum value. The bot stops as soon as an object approaches its front side.

To overcome the limitations of a wall follower, another solution was to build a line follower that could follow a prescribed pair of lines to keep itself in the middle of the path. A line follower has a direct relation to the light when it comes to its working mechanism. Functions of light on a black and white surface are used here. The light gets consumed when the surface is black and is outright reflected when falling on a white surface. Ground rules for a line-following robot are established on this very principle.

The line-following bot sticks to and detects lines drawn in the area and is autonomous. The lines are drawn black or white depending on the surface color. The robot senses the line (here, we have used a black line) using two proximity sensors mounted vertically parallel at the front of the bot. The servo motor is used to direct the bot along the path [28]. Its rotation angle is 0–180° and mounted at the front side, whereas rear wheels of the bot are

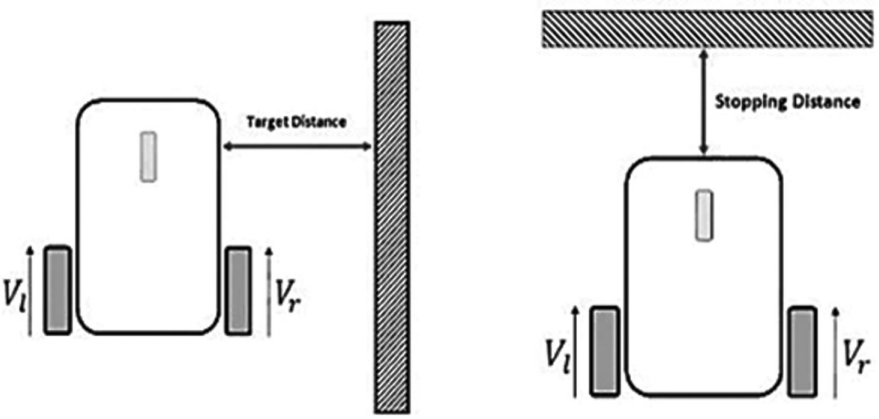

FIGURE 24.5 Proposed operation.

equipped with a 12-V DC motor to move it forward along the path. The bot is also installed with an inertial measurement unit (IMU) sensor. The IMU device has a 6050 MPU plus a digital motion processor (DMP) along with a three-axis accelerometer and gyroscope on a disc of silicon to operate algorithms of the six-axis motion fusion. The IMU values are mapped with the speed of the motor such that the bot's speed increases during inclination and reduces during the downhill climb [29].

The main disadvantage of a line follower is that to follow a path, its color should contrast with the surrounding environment. There are two chief segments of infrared (IR) sensor, the IR transmitter and receiver [30]. The main jobs of this transmitter and receiver are transmitting and receiving the IR waves, respectively. Data are sent to the Vout pin constantly in binary form (0 or 1) via receiver.

The waves from the front of an object are received by the receiver. The sensor gives a binary 0 as an output in this case. When there is no object and the receiver does not get any wave, the output is 1.

Hard objects like doors or a wall, sunlight, fog, smoke, smog, and dust affect the frequencies of an IR signal; therefore, the IR frequencies have difficulty operating efficiently in outdoor conditions, which makes it impossible to use them in open-pit mines. The bot was controlled using keys through the keyboard library of Python. The module can be downloaded using the pip install command on keyboard. The module is used to get full control of the keyboard. The small Python library can hook global events, register hot keys, simulate key presses, and much more. The inbuilt function keyboard.is pressed was used to detect key presses from the user.

Keyboard libraries helps to enter keys, record the keyboard activities, and block the keys until a specified key is entered and simulate the keys. It captures all keys; even onscreen keyboard events are captured. The keyboard module supports complex hot keys. Using this module, we can listen to and send keyboard events. It works on both windows and the Linux operating system [31].

To establish serial communication over a certain port-serial, the serial ('/dev/ttyACM0', 9600) command is used to connect to the ACM0 port of Pi over a 9600-baud rate and the command "readline ()" was used to read a line via serial communication. The limitation of the keyboard library was that it could not be incorporated into Python code for the GUI (explained later) as the keyboard requires the "sudo" command, which is not supported by GUI.

To overcome the shortcomings of the line follower, further attempts were made to apply lane detection through the pi camera. Lane detection is a technique of identifying both the lanes in front of the vehicle and maintaining an average center distance from the lanes.

24.3 LANE DETECTION PROCEDURE

24.3.1 Convert Original Image to Gray Scale

Using the following command any image can be converted to a gray-scaled version.

```
textitcv2.cvtColor(img, cv2.COLOR BGR2GRAY)
```

24.3.2 Apply Slight Gaussian Blur

A Gaussian filter is used to convolute an image in the Gaussian Blur operation rather than the box filter. The image can be smoothed with the help of Gaussian filter as it helps in reducing parts that are of high frequency and is a low-pass filter.

```
cv2.GaussianBlur(img, (5,5), 0)
```

24.3.3 Apply Canny Edge Detector

One of the most admired algorithms for detecting edges is Canny edge detection. John F. Canny generated this algorithm in 1986. This algorithm works well in varied environments. The parameters of this algorithm can be molded according to the edges with various attributes.

The process of this edge detection algorithm can be broken down into five different steps:

1. To remove noise, we smooth the surface with the help of Gaussian filters.
2. Gradients of image intensity are searched.
3. Non-maximum extinction is used to steer clear of fake feedback from edge detection.
4. Threshold of double power is used for regulating the possible edges.
5. Irrationally track the edges: conclude the edge-based detection by subduing weak edges; then there is no connection with the strong ones.

24.3.4 Define Region of Interest

Defining the region of interest helps to weed out unwanted edges detected by the Canny edge detector.

24.3.5 Retrieve Hough Lines

A well-known method for shape detection (of shapes that can be portrayed in a mathematical arrangement) is called the Hough transform. It is a powerful technique that can represent distorted or fragmented shapes. Detecting shapes like ellipses, lines, and circles sometimes becomes problematic with automatic analysis. To acquire pixels or points of an image in a particular curve, preprocessing is done with an edge detector.

At times there are pixels or points missing on the curve or divergence among points on the edge and shapes because of defects in the edge detector. Therefore, grouping of the attributes extracted to their defined shapes (circle, lines, and ellipses) is significant.

The Hough transform helps to make groups of points on edge into candidates of an object. The transform does so by carrying out a direct voting system in a framework of objects.

```
cv2.HoughLinesP(a, 3, np. pi/180, 100, np. array ([]),
          minLine- Length=30, maxLineGap=5)
```

An array of theta and rho values is given in which theta and rho are calculated in radians and pixels, respectively. Before using the Hough transform, an algorithm for detecting edges (Canny edge) or a threshold is implemented so that images are in a binary format. Other criteria for precision are theta and rho. Last variable is threshold; it requires minimal votes to be deemed a line. Consolidate and extrapolate the Hough lines and draw them on an original image extrapolation, which means creating a straight line that is tangential for the data known. Deducting the graph linearly will yield good products when used on a function of linear nature that is not extended tangentially.

To avoid intensive processing on Raspberry Pi, all the processing took place on a PC. To implement that the following changes were made:

1. Establishing camera streaming such that it could be accessed on a PC over the same network was done by creating an mjpg stream from Raspberry Pi.

   Duplicating JPEG frames into output plug-ins is done by an application based on command line streamer-mjpg. This software can be used in converting any JPEG stream on a network based on IP addressing to distinct candidate software that can receive streams in the MJPG format like Cambozola, Chrome, mplayer, VLC, and Firefox.

2. To be able to send commands to Raspberry Pi using a server.

Making a connection for communication with the help of two nodes can be done with the use of socket-based programming. To build an IP connection one vertex takes the input, whereas the other connects to a server. Client extends to the server for instigating a connection.

The Raspberry Pi was used as a server while the client program was run from another computer. The server sends the information regarding controls of the bot to the Arduino through serial communication, while the said data come enters the server.

```
s = socket. Socket (socket.AF INET, socket. SOCK STREAM)
```

An illustration of the node is built and is given two variables (Figure 24.9): a transmission control protocol (TCP) (SOCK STREAM) and ipv4 AF NET. Sockets can either be user data protocol (UDP), i.e., does not make connection, or TCP, i.e., form a connection. For this purpose TCP was preferred.

With the visibility of the lanes the bot tries to drive through the path taking turns as and when required. Whenever the right lane gets out of visibility the bot turns left and vice versa. To take a left turn the PC would send a left command to the server, which would further be relayed to the Arduino using serial communication. The Arduino receives the command and processes it using our previously made program (refer here). The same procedure is followed when taking a right turn.

1. Lane detection using Hough transform is difficult for curves.
2. At times, the lane may not be visible to the camera due to different inclinations of the path.

3. Lane detection using edges of walls and cliffs is difficult due to the uneven surface.
4. At times when there is a car approaching from the opposite direction the lane will not be visible.
5. The paths would be required to be cleared for any obstruction such that edges are clear enough to be distinguished as a lane.

If the lane is not visible to the camera due to a difference in inclination of path, a wide-angle camera can solve the issue. The vehicle can be equipped with actuators to control the facing angle of the camera such that it turns as per the inclination of the path. Through this the path ahead will be constantly visible and, hence, any obstruction can be detected using it.

24.4 IMPLEMENTATION DETAILS

The keypress control of the bot provided a base for the idea to use coordinates of the said path as references to autonomously drive the vehicle once all the points of the path are registered. The coordinates were proposed to be stored in CSV format, which was later accessed to compare with live coordinates detected by using AprilTags placed directly on the top side of the bot. A camera above the pit reads the AprilTags.

The AprilTags detect *X*-, **Y**-, and *Z*-coordinates of the bot along with the roll, yaw, and pitch of the position. While training the bot these coordinates are stored to CSV format on each keypress event along with the steering angle of the bot. AprilTag is a trustworthy system for calibrating the camera and reality based on augmentation and robotics. Exact three-dimensional (3D) location, identity, and direction of camera-related tags can be created with the help of this system.

Theoretically, AprilTags and QR codes are alike as both of them have a bar code in a two-dimensional (2D) format. Nevertheless, they can be detected from a long range as they were built for cyphering payloads of small lengths ranging from 4 to 12 bits. For this research, Tag36h11 is used (Figure 24.6).

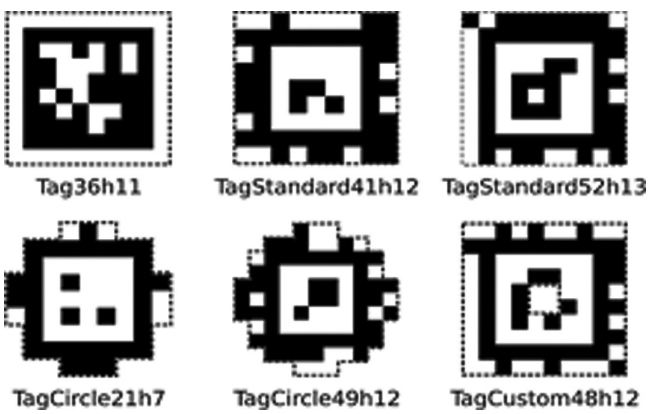

FIGURE 24.6 AprilTag family.

To calculate the parameters of AprilTags, the following parameters are used:

1. *HAMMING distance:* It is the amount of locations where two distinct symbols are present; thus, it is better to have a high ratio of such symbols. Recognition of distinct tags is a result of a high hamming distance. 36h11 is a family of tags containing a high hamming distance, hence, it is used more.
2. *Family size:* The 36h11 tag family has 586 tag members, thus incorporation of many bots with different IDs is possible in future implementations.

To capture and extract information from AprilTags, the camera needs to be calibrated to account for various distortions.

$$x_{corrected} = x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right)$$
$$y_{corrected} = y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right)$$

Spiral formation makes lines seem as though they are curved. Distortion increases with the increase in distance from the center point of an image. Radial distortion is when a lens that captures the images is not parallel to the plane of imaging, and a tangential formation occurs. It makes a few places appear nearer than presumed. Total distortion via tangents is as follows.

$$x_{corrected} = x + \left[2p_1 xy + p_2\left(r^2 + 2x^2\right)\right]$$
$$y_{corrected} = y + \left[p_1\left(r^2 + 2y^2\right) + 2p_2 xy\right]$$

Five coefficients of distortion are computed. Coefficients of distortion = (k3, p1, k1, p2, k2) and other information like variables of camera are needed. Framework specified to a camera are called intrinsic. Optical centers (*cy*,*cx*) and focal length (*fy*, *fx*) are intrinsic coefficients. Both of these parameters are used to make a matrix of the camera. The matrix of a camera is particular to each camera; hence, once deduced, it can be used on images captured with a specific camera. Matrix is denoted by a 3×3 grid.

The internal coefficient list:

- *Focal length:* This is collected in a 2×1 vector of pixels.
- *Principal point:* 2×1 vector *cc* is used to depict the coordinates of a main point.
- *Skew coefficient: c* is a scalar alpha, which contains an angle between the axis of the pixel in *x*,*y*.
- *Distortions:* 5×1 vector depicts coefficients of image framework.

$$\begin{bmatrix} \mathrm{fc}(1) & \mathrm{alpha\_c} * \mathrm{fc}(1) & \mathrm{cc}(1) \\ 0 & \mathrm{fc}(2) & \mathrm{cc}(2) \\ 0 & 0 & 1 \end{bmatrix}$$

Finding the location of a mobile robot is known as localization. To decide future actions, one of the important things for a bot is to have an idea of its location. To search the location, localization comes into play. Given an AprilTag inside a known (global) location, is it possible to use the relative location and orientation data to extract the bot's absolute global position and orientation.

Upon running the program, the following steps are replicated:

1. *Find tags:* Each tag containing AprilTags in the view of the field is recognized by the camera. To get localization data, it is important to have one tag in the robot's field of view.
2. *Store data:* Pitch, yaw, and roll for all tags that are recognizable are stored inside a structured CSV format when training is done, and the same data are used to drive the bot autonomously when run on test mode.
3. *Obtaining global-tag data for pose:* To acknowledge the localization cycle, the lookup table contains θ, *z*, *y*, and *x*, which are read via segment (Figure 24.7).

There is no requirement for libraries and tools in Flask, and it is a substructure in Python. It is void of a layer of abstraction or other libraries. A server was used to create a host server in which all data from the asset (bot) are collected and displayed for reference. The server can be accessed over the network; hence, its data can be used by any other application. We are using Flask to view the live asset data at our end. It has two components: a debugger and a server of development. Flask shows a unified foundation for testing units. Remits relaxed demands. For sessions for the client, it provides cookies.

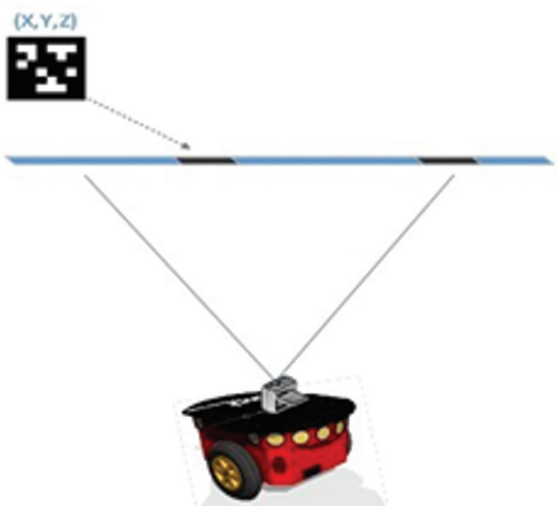

FIGURE 24.7 Tag detection.

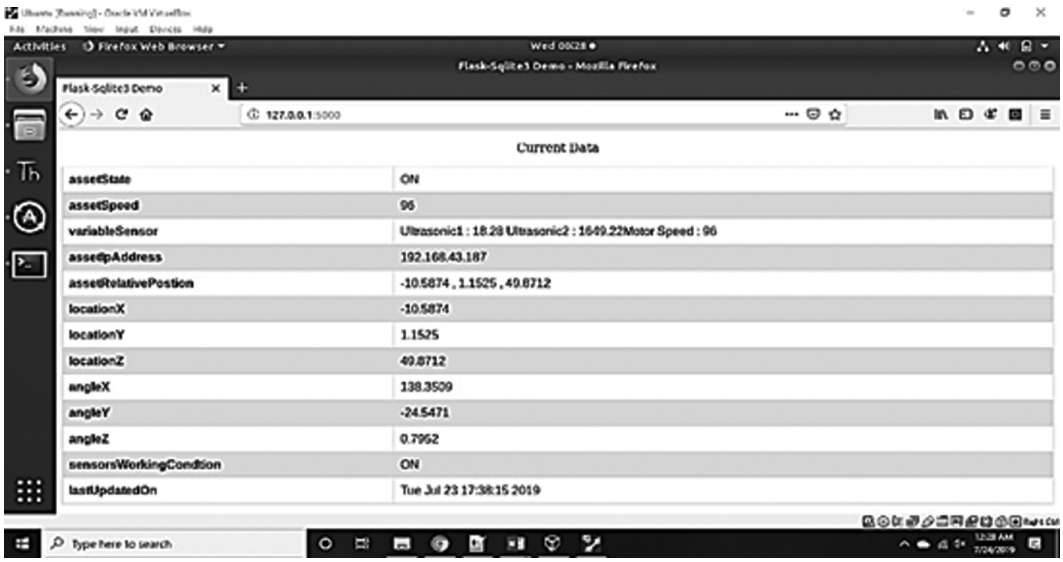

FIGURE 24.8 Sample Flask server interface.

It is based on one code, requires a great deal of authentication, and is compatible with Google apps (Figure 24.8).

The data from the asset (bot) are stored in a database that can be accessed through the Flask server. Postgre was generated in at the University of California Department of Computer Science. Many ideas were developed by Postgres, which is a database built on enterprise edition of class and can be installed and set up easily. Support for NoSQL and SQL is provided by Postgres. Any problems related to PostgreSQL can be solved with the help of a database system community:

1. A community that is developing fast.
2. Other software for SQL server, DB2, and Oracle.
3. Can work on any operating system.
4. Simultaneously helps users.
5. Provides great performance, indexing is increased.
6. Assists applications like JSON and XML.
7. Contribution of code and skill is given by ANSI SQL.
8. Stores data in a well-organized format, foreign keys are used.
9. Retrieves data, views and Table joins are used.
10. Backup of data is made for replicating and increasing scalability to read.

The bot comes with a few fallback conditions to avoid any major accidents at the pit. Some of the conditions are as follows.

1. The client and server communication are continuously monitored and if this communication breaks for 1 second, a stop signal is passed to the bot.
2. The bot is installed with an ultrasonic sensor on the front, and when it detects any object in front of it (at a range of 10 cm) the bot stops traveling and the user is informed of the object.
3. An ultrasonic sensor is also installed on the left side of the bot that continuously checks if the bot is too near the wall. If it comes too close to the wall, its front wheels turn 170° to the right side. The live coordinates of the bot then correct its path by comparing with the trained model.

The training and testing part is integrated on a GUI using the PyQT library. The PyQT is a library for a cross platform and contains toolkit for widgets and an interface for QT. It is a mix of libraries of QT and Python. This application, which is built on a GUI, is driven by an object rather than executed in sequence like other console-based applications. Events are executed when a user performs any task like mouse selection or clicking. To make an event successful, widgets are used in a building interface for users. To provide a signal response for events, a Qobject class widget is used. The signal does not carry out the operation alone, but a slot connection provides the output.

A GUI was made on PyQT and a brief walkthrough with its elements are mentioned next. The first window a user would get after opening the application is the boot screen. In the background, the app tries to connect with the server (Figure 24.9).

When a successful relationship between a server and a client is built, another screen pops up that tells the user to choose an AprilTag from a set of 36 tags. These tags represent bots, as tags are placed on them (Figure 24.10).

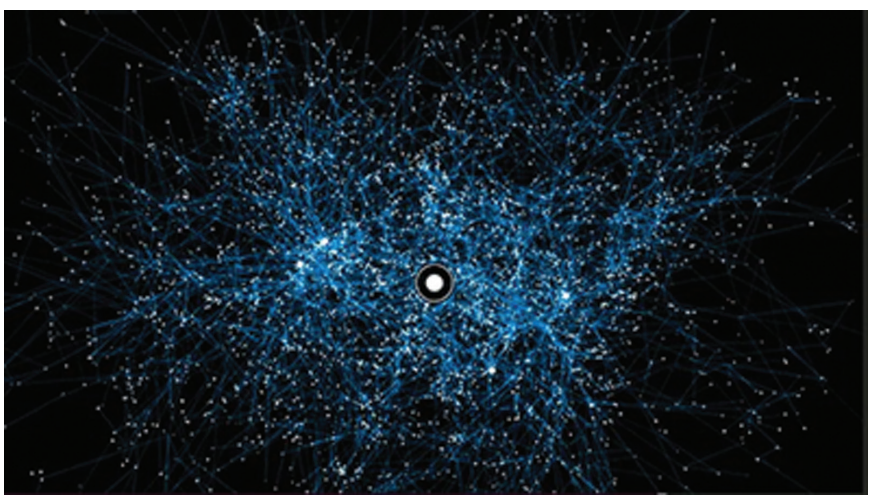

FIGURE 24.9 Boot animation.

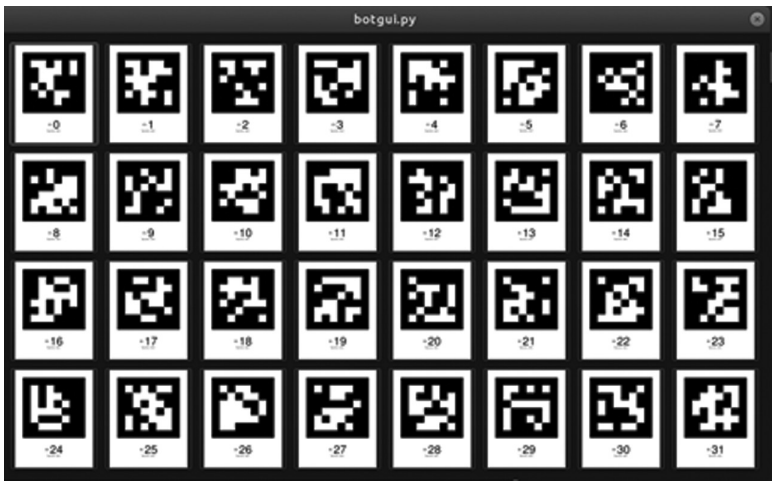

FIGURE 24.10 Menu.

After choosing the tag, the user will be directed to another window where a live feed from the top camera of the pit will be available along with a small picture-in-picture view from the bot's perspective. The window also has buttons to train or test the bot (Figure 24.11a).

Also, if the tag is not in view of the camera, a warning on the screen will be displayed regarding the same.

24.5 RESULTS

24.5.1 Discussion

There are multiple areas in which this model can be used to give better outputs and add more features:

1. The algorithm behind the steer angle prediction can be improved. Right now, it tries to imitate the closest path present on the trained data set. It compares the current coordinate with the stored coordinates in the data set and sets the steer angle accordingly. This can be improved by calculating the angle between the current coordinate and the next coordinate and set the steer angle such that it directs the car to that coordinate.
2. We can implement a better object detection algorithm using the Raspberry Pi camera present on the model.

(a) (b) (c)

FIGURE 24.11 (a) Test or train mode, (b) train mode, and (c) autonomous mode.

3. The vehicle can be equipped with actuators to control the facing angle of the camera such that it turns as per the inclination of the path. Through this the path ahead will be constantly visible; hence, any obstruction can be detected using it.
4. Machine-to-machine communication can be implemented such that multiple vehicles can be moving in the same path using the same program. If a vehicle is approaching another vehicle it can communicate and send a command to stop such that another vehicle can pass by and later it continues its path.
5. More asset data are to be monitored and stored in a database, such as:
    a. Engine oil pressure:
        i. Unit: PSI
        ii. Low: <15–20 PSI
        iii. High: >80 PSI
    b. Low engine oil:
        i. Unit: L
        ii. Low: <25%
    c. Tire Pressure:
        i. Unit: PSI
        ii. ii. Avg: 55–80 PSI
    d. Thermal Sensor data
    e. Speed monitoring:
        i. Unit: km/hr
        ii. Limit: <25 km/hr
    f. Low battery level indication
    g. Engine vibration
    h. Lubricant oil level sensor
    i. Temperature sensor

24.5.2 Training

24.5.2.1 Train Mode

Train mode consists of left, right, up, and down buttons that change colors if corresponding keys are pressed on the keyboard. It also consists of a save button to register a successful run of the bot and append the data collected to the CSV format. The start button

initiates training, whereas the stop button puts the bot in halt. The following are used to train the bot:

1. After installation, run the GUI.
2. The application will now connect to the server. From the tag selection menu, select the tag that is installed on the bot.
3. A new window appears; select the option "train" to navigate to the training window.
4. Make sure the bot is in position and then click on the start button.
5. The up arrow key moves the bot forward and the left arrow key turns the bot to left side by 20° each time the button is pressed. Similarly, the right button turns the bot to right side by 20° each time it is pressed.
6. After a successful run click the save button to store the pit coordinate data into the CSV file format. (This file will be utilized during the autonomous run.) The GUI has the option to record more than one "training Run."
7. When finished with the training, click "Reset" or the "r" button on keyboard to turn the wheels 90° and end the training session.
8. The user can return to another bot for training by navigating back to tag selection or end the session by clicking the close button at the top right corner of the window (Figure 24.11b).

24.5.2.2 Autonomous Mode

In this mode, the user can request the autonomous run of the bot. All other bot controls are disabled (highlighted in white; Figure 24.11c).

24.5.2.2.1 Training Model and Parameters

Place the camera above the pit so that the entire path of the pit is visible. Connect the camera to the user's PC and place the bot at the foot of the path, but make sure the tag of the bot is visible at all times. Run the server program and start the GUI (Figure 24.12).

Run the bot through the path of the pit as many times as is needed until it is trained. Run the GUI file and proceed with the manual for training as discussed earlier or, if already trained, proceed with the autonomous run.

FIGURE 24.12 Training process.

24.6 CONCLUSION

After a few iterations of models and diverse techniques, we were able to create a bot that was able to drive itself autonomously in an open pit. Hence, through this project we learned about many technologies included in a compact self-driving vehicle. We implemented wall follower and line follower initially and went for lane detection. As the challenges faced in those techniques outweighed the success of the objective, we implemented the localization technique through AprilTags and attempted to drive the bot autonomously based on the coordinates received from the position of the bot and training it to drive on those coordinates. Also incorporated were fallback conditions in the program for object detection and wall detection along with a fail-safe condition to stop the bot if the server or client did not respond for a certain period. A Python-based GUI was also created to make the entire process of training and testing the bot easier and user friendly. The sensor data and controls from Raspberry Pi were serially transmitted, and coordinates of the bot were successfully uploaded to a database and rendered using PostgreSQL and the Flask server.

REFERENCES

1. Zhang, J., 2019. End-to-End Learning for Autonomous Driving. PhD dissertation, New York University.
2. Udacity, Inc. 2019. Self-Driving Car Simulator. https://github.com/udacity/self-driving-car-sim. Accessed 5 Feb 2019.
3. Wang, Z., 2018. Self Driving RC Car. https://zhengludwig.wordpress.com/projects/self-driving-rc-car/. Accessed 22 Jan 2019.
4. Desai, S. and Desai, S., 2017. Smart vehicle automation. International Journal of Computer Science and Mobile Computing, 6(9).
5. Hussain, R. and Zeadally, S., 2018. Autonomous cars: research results, issues, and future challenges. IEEE Communications Surveys & Tutorials, 21(2), pp.1275–1313.
6. Rajasekhar, M.V. and Jaswal, A.K., 2015. Autonomous vehicles: the future of automobiles. In 2015 IEEE International Transportation Electrification Conference (ITEC) (pp. 1–6).
7. Rosenzweig, J. and Bartl, M., 2015. A review and analysis of literature on autonomous driving. E-Journal Making-of Innovation, pp. 1–57.
8. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J. and Zhang, X., 2016. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.
9. Jackel, L.D., Krotkov, E., Perschbacher, M., Pippine, J. and Sullivan, C., 2006. The DARPA LAGR program: Goals, challenges, methodology, and phase I results. Journal of Field Robotics, 23(11–12), pp. 945–973.
10. Lecun, Y., Cosatto, E., Ben, J., Muller, U. and Flepp, B., 2004. Dave: autonomous off-road vehicle control using end-to-end learning. Courant Institute/CBLL, DARPA-IPTO Final Report. http://www.cs.nyu.edu/yann/research/dave/index.html
11. Podpora, M., Korbas, G.P. and Kawala-Janik, A., 2014. YUV vs RGB-choosing a color space for human-machine interaction. FedCSIS Position Papers, 18, pp. 29–34.
12. Pan, Y., Cheng, C.A., Saigol, K., Lee, K., Yan, X., Theodorou, E. and Boots, B., 2017. Agile autonomous driving using end-to-end deep imitation learning. arXiv preprint arXiv:1709.07174.
13. Wang, Z., Lim, E.G., Wang, W., Leach, M. and Man, K.L., 2014. Design of an Arduino-based smart car. In 2014 International SoC Design Conference (ISOCC) (pp. 175–176).

14. Parashar, A., 2019. IoT based automated weather report generation and prediction using machine learning. In Proceedings of the 2019 2nd IEEE International Conference on Intelligent Communication and Computational Techniques (ICCT), Jaipur, India.
15. Parashar A., Parashar A. and Goyal S., 2018 Classifying Gait Data Using Different Machine Learning Techniques and Finding the Optimum Technique of Classification. In: Mishra D., Nayak M., and Joshi A. (eds) Information and Communication Technology for Sustainable Development. Lecture Notes in Networks and Systems, vol 10. Springer, Singapore. https://doi.org/10.1007/978-981-10-3920-1_31
16. National Institute for Occupational Safety and Health (NIOSH), 2018. NIOSH mine and mine worker charts. https://wwwn.cdc.gov/niosh-mining/MMWC#disasters. Accessed 17 July 2018.
17. Bartnitzki, T., 2017. Mining 4.0 – importance of industry 4.0 for the raw materials sector. Mining Report, 153(1), pp.25–31.
18. Yarkan, S., Guzelgoz, S., Arslan, H. and Murphy, R., 2009. Underground Mine Communications: A Survey. IEEE Communications Surveys & Tutorials, 11(3), pp.125–142.
19. Chung, T., 2018. DARPA subterranean challenge aims to revolutionize underground capabilities. https://www.darpa.mil/news-events/2017-12-21.
20. Ferguson, D., Morris, A., Hähnel, D., Baker, C., Omohundro, Z., Reverte, C., Thayer, S., Whittaker, C., Whittaker, W., Burgard, W. and Thrun, S., 2004. An autonomous robotic system for mapping abandoned mines. In proceedings in Advances in Neural Information Processing Systems (vol 16, pp. 587–594).
21. Zlot, R. and Bosse, M., 2014. Efficient Large-Scale 3d Mobile Mapping and Surface Reconstruction of an Underground Mine. In K. Yoshida and S. Tadokoro (eds) Field and Service Robotics, ser. Springer Tracts in Advanced Robotics, vol. 92, pp. 479–493. Springer, Berlin Heidelberg.
22. Grehl, S., Donner, M., Ferber, M., Dietze, A., Mischo, H. and Jung, B., 2015. Mining-rox – mobile robots in underground mining. In Third International Future Mining Conference AUSIMM, (pp. 57–64).
23. Murphy, R., Kravitz, J., Stover, S. and Shoureshi, R., 2009. Mobile robots in mine rescue and recovery. IEEE Robotics & Automation Magazine, 16(2), pp. 91–103.
24. Güth, F., Wolf, F., Grehl, S., Lösch, R., Varga, S., Rezaei, N., Mischo, H., Jung, B., Benndorf, J., Rehkopf, A. and Joseph, Y., 2018. Autonomous robots and the Internet of Things in underground mining. In Conference on Smart Systems Integration, Dresden, Germany (pp. 215–222).
25. Innok Robotics GmbH, 2015. Heros 444 FG – Handbuch. Innok Robotics GmbH, Tech. Rep., Münchsried, Germany.
26. Grehl, S., Sastuba, M., Donner, M., Ferber, M., Schreiter, F., Mischo, H. and Jung, B., Towards virtualization of underground mines using mobile robots – from 3D scans to virtual mines. In International Conference in Mine Planning & Equipment Selection.
27. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. and Mg, A., 2009. ROS: an open-source robot operating system. In Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics.
28. Koenig, N. and Howard, A., 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. 2004 IEEE/RSJ IROS, vol. 3, IEEE, pp. 2149–2154.
29. Labbe, M. and Michaud, F., 2014. Online global loop closure detection for large-scale multi-session graph-based SLAM. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 2661–2666).
30. Labbe, M. and Michaud, F., 2013. Appearance-based loop closure detection for online large-scale and long-term operation. IEEE Transactions on Robotics, 29(3), pp. 734–745.
31. Rösmann, C., Hoffmann, F. and Bertram, T., 2015. Planning of multiple robot trajectories in distinctive topologies. In European Conference on Mobile Robots (ECMR) (pp. 1–6).