CSE520 - PiCar

Nicole Wang, Lin Li, Xueke Yin
{l.wang12,lilin1,xueke.y}@wustl.edu

ABSTRACT

This project builds a smart robotic car based on Raspberry Pi with the functionalities of obstacle avoidance and object recognization, and also implements real time data visualization on Android application with help of AWS IoT, AWS DynamoDB and AWS Cognito services. The PiCar is able to stop automatically before knocking onto any obstacle detected within a specific distance, and it also can approach an object with the recognized target color. The data collected from the PiCar will be passed to AWS DynamoDB, and with the access permission from AWS Cognito then is passed to the Android application.

1 INTRODUCTION

The PiCar is set up using a Adeept Mars Rover PiCar-B car kit as shown below.

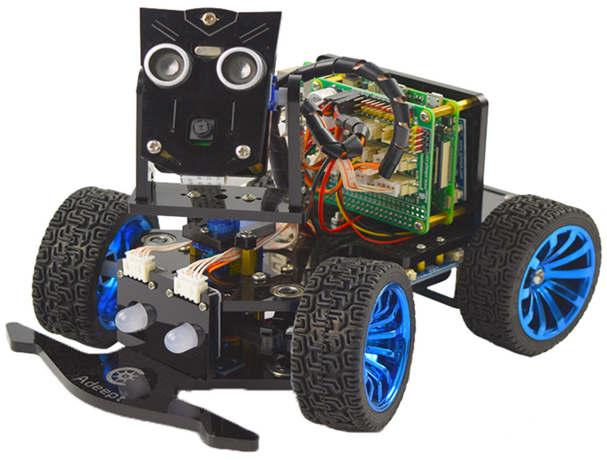


Figure 1: Adeept Mars Rover PiCar-B Hardware

It is equipped with a camera on the front of the car body and an ultrasonic radar at the head. The camera takes pictures of the current view while it is moving, which will play a role in the object recognition. The ultrasonic module is used to scan the semicircle range in front of the car and display the obtained obstacle information on the ultrasonic radar, which will help in the obstacle avoidance.

The PiCar is controlled by Raspberry Pi which uses Raspbian as its operating system. To control the Raspberry Pi remotely, we enable the SSH service and set Wi-Fi for Raspberry Pi. Once the Raspberry Pi boots up, we will receive an email which contains the IP address of Raspberry Pi. Using an SSH client to connect to the IP, we can remotely control Raspberry Pi easily.

When the PiCar is running, the data relating to obstacle avoidance and object recognition will be published to AWS IoT using MQTT client. To visualize data on Android App, AWS DynamoDB will get the real-time data from IoT publisher. With the help of Cognito, the data will be passed from DynamoDB to Android App and displayed on screen. The Android App also has control over the PiCar by connecting with Raspberry Pi via socket. The detail implementation is explained in section 4.3.

2 GOALS

In this project, the PiCar aims to have two features. The first one is obstacle avoidance. The PiCar uses the ultrasonic sensor to detect the obstacles ahead and avoids hitting into them. We set a threshold value in the program and if the distance between PiCar and obstacles decreases to this value, the car motor will stop automatically. Taking the inertia effect of PiCar into consideration, this threshold value should be a little larger. Detailed implementation explained in section 4.1.

The other feature is object recognition. Specifically, there are target object in front of the PiCar, and the PiCar will head to the object of target color directly and ignore objects of other colors. To implement this, we use the camera on PiCar to take pictures of the current view and use OpenCV to mask out the target color in the pictures. Detailed implementation explained in section 4.2.

Other than develop above two software features for the PiCar, this project also aims to display the real-time data (i.e. distances to obstacles and angles to target objects) on AWS IoT, and further visualized on Android App. The user interface designed on Android APP would let user not only visualize the real-time data, but also manipulate the two features mentioned above.

3 RELATED WORKS

The physical PiCar model was built from Adeept Mars Rover PiCar-B Smart Robot Car Kit. All pieces built up and pins connections are followed by instructions documented in [1]. We have used [8] in order to send software command from Python to hardware of the physical PiCar, receive image from car's camera and distance data from car's ultrasonic sensor.

4 DESIGN AND IMPLEMENTATION

4.1 Obstacle Avoidance

The obstacle avoidance feature was implemented with the help from the ultrasonic sensor on the PiCar. The goal for this feature is the car is able to get as close to any objects as possible and stops before hitting them.

When the PiCar got command to read distance from ultrasonic sensor, the sensor that located at the front side of the car would emits sound waves at a high frequency so that humans would not able to hear. It then wait for the sound to be reflected back, calculating distance based on the time required and send the distance data back to us as shown in figure 2.

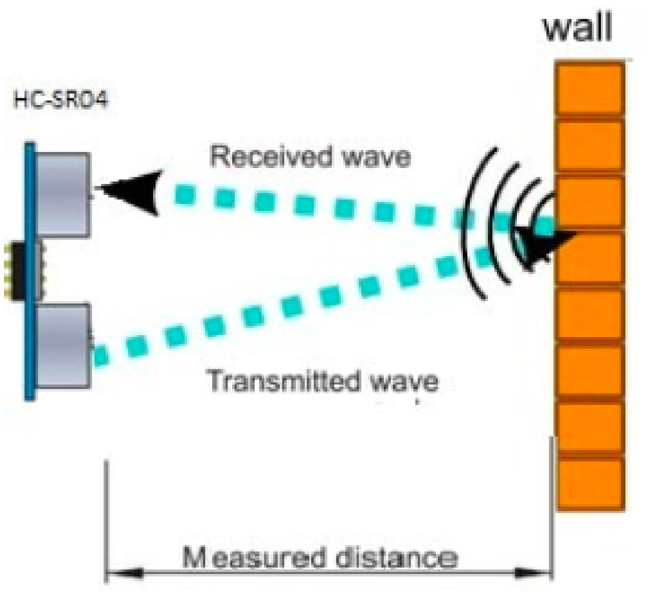


Figure 2: How Ultrasonic Sensor Read Distance

The car would be set with some speed: *initialSpeed* when the program starts. The ultrasonic sensor would then measure the distance between the car and the avoidance it's facing every $freq$ seconds. The car will keep moving until the distance measured by ultrasonic sensor is smaller than the *minDistance* threshold. The car's speed would then be set to 0 and stop.
The entire process in pseudocode is shown below:

Algorithm 1 Obstacle Avoidance

```
1: car.setMotor(initialSpeed)
2: distance = car.readDistanceFromUltrasonicSensor()
3: while distance >= minDistance do
4:     sleep(freq)
5:     distance = car.readDistanceFromUltrasonicSensor()
6: end while
7: car.setMotor(0)
```

The *initialSpeed* should be chosen so that the car is at least moving. Therefore, the car's motor ability and the moving surface should be considered during the decision. For example, a floor covered with carpet may require larger *initialSpeed* than a smooth floor. Also every motor is different, so it is common to have different cars moves at similar speed on same surface but with different value of *initialSpeed*.

The *minDistance* should be chosen so that the car is able to stop as close to the obstacle as possible without hit it. Notice the inertia counts heavily in this part. It is reasonable to set this value related to the speed of the car (in this case is the *initialSpeed* since the speed is constant in this algorithm).

The $freq$ can be chosen in two ways. A simple way is to set it to a constant number. In this case $freq$ can only related to the speed of the car (i.e. larger speed results less $freq$ in order to make sure the car would not hit any obstacle). Another more compatible but more optimized way is to set it related to the distance measured by ultrasonic sensor and the speed of the car, so that the distance will be measured more and more frequently when the car gets closer to any obstacle.

4.2 Object Recognition

The object recognition feature was implemented with the help from the camera on the PiCar and openCV package [3] in Python. Every *pictureFreq*, the car would take a picture from its current view and recognize the object by finding its corresponding color in the picture using openCV. Next, the car would find the center of the target color in the picture and then calculate the angle between the center and itself. Finally, the car would adjust its steering angle in order to approach the target object. The entire process is designed as below:

Algorithm 2 Object Recognition

```
    car.setMotor(initialSpeed)
2:  for every pictureFreq do
        distance = car.readDistanceFromUltrasonicSensor()
4:      if distance < minDistance then
            car.setMotor(0)
6:          car.closeCamera()
            break the for loop
8:      else
            picture=car.takePictureFromCamera()
10:         mask=picture.extractColor(targetColorRange)
            center=mask.findCenter()
12:         if center exists (targetColor is found in picture) then
                angle=findAngle(center)
14:             car.setSteerServo(angle)
            end if
16:     end if
    end for
```

4.2.1 *Color Mask.* .

This section will explain how *extractColor*() method works with given *picture* and *targetColorRange*.

The *picture* taken by camera is in hsv form. The *targetColorRange* is a pair of tuples, indicates the lower and upper bound of hue, saturation and value, as [(hLowerBound, sLowerBound, vLowerBound), (hUpperBound, sUpperBound, vUpperBound)]. The bound value of hue is able to chosen based on user's choice of target color and the color map of hue as shown in 3:

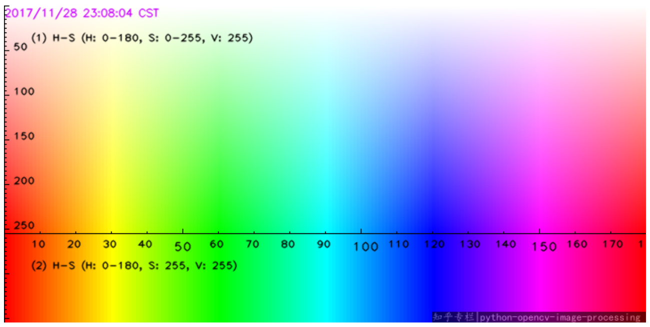


Figure 3: HSV Color Map: Hue

For example, if the target object is in yellow color, it is reasonable to set $hLowerBound = 25$ and $hLowerBound = 35$.
The Saturation and Value variables describe the brightness and darkness of the color respectfully. They both ranged from 0 to 255. As shown in figure , $Saturation = 0$ results in completely white and $Value = 0$ gives completely black.

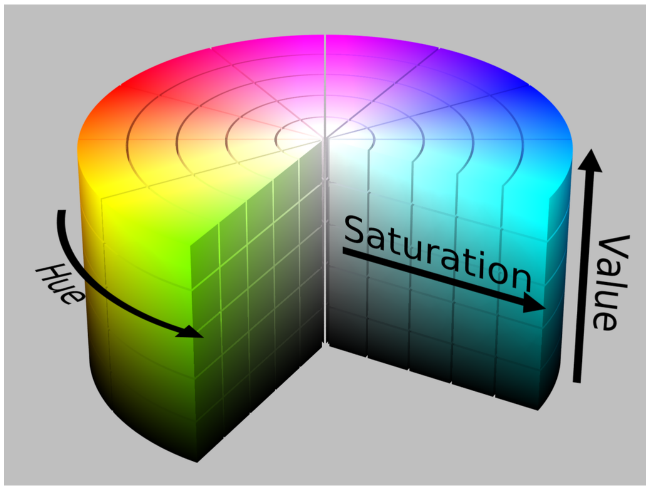


Figure 4: HSV Color Map: Saturation and Value

The range of saturation and value is able to chosen based on the brightness or intensity of the target's color as well as the lighting environment.

4.2.2 *Find Angle from Picture.* .

This section will explain how $findCenter()$ and $findAngle()$ work from given masked picture *mask*.

After extract target color from given color, the mask picture would put white pixels on each pixel that falls in the *targetColorRange* in original picture, and put black pixels everywhere else. $findCenter()$ first get image moments from masked picture using openCV:

$$moments = openCV.moments(maskedPicture)$$

Then center position (cx, cy) is able to calculated in following method:

$$cx = int(moments["m10"]/moments["m00"])$$

$$cy = int(moments["m01"]/moments["m00"])$$

Notice $moments["m00"] = 0$ means the target color is not find in the given picture, which means target does not exist in front of the car.

The angle between the center of the target object and the car is related to the image size and whether the target is in the left half or the right half of the picture.

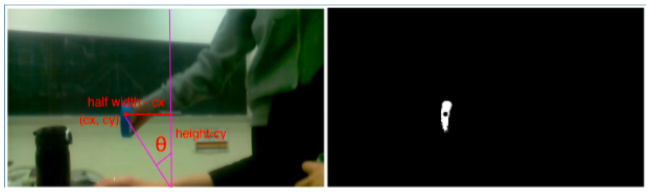


Figure 5: Center in Left half of the picture

As shown in figure , if the center falls in left half of the picture, the angle is able to be calculated as

$$angle = tan^{-1}(\frac{imageHeight/2 - cy}{imageWidth - cx})$$

Similarly, if the center falls in right half of the picture, the angle would be calculated as

$$angle = tan^{-1}(\frac{cx - imageHeight/2}{imageHeight - cy})$$

Finally, the value range taken by car's steer motor is between -10 and 10 while the angle results in range -90 and 90. So it is needed to divide the value by 9 before pass it to set the car's steer motor.

4.3 Data Visualization on Android APP

From Figure 6, this is the whole design architecture of this project. Raspberry Pi which consists the python programs of the Picar features makes connection with AWS Iot to publish data via MQTT protocol. It also connects with Android Application via a socket, to perform specific action that is triggered in the app. With three AWS serices we used, DynamoDB stores the data from Raspberry Pi program through an AWS IoT topic; Cognito gives access permission of DynamoDB data to the users of the Android application, so that the user can see the table data in the application.

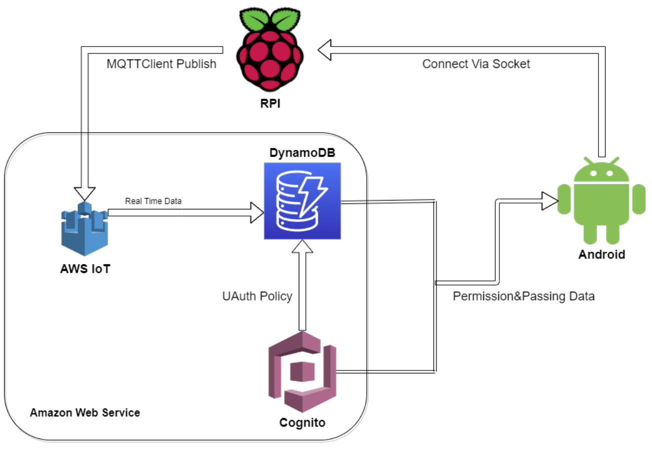


Figure 6: Design Diagram

4.3.1 *Raspberry Pi and AWS IoT Connection.* [6]

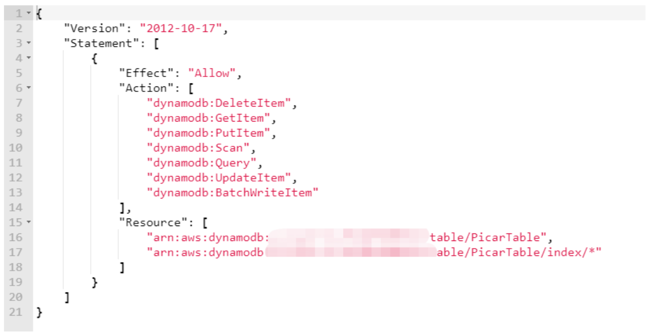
- `AWS IoT Policy`: Created a Policy called `picar_policy`; Action: iot:*; Resource ARN: *; Clicked on Creat.
- `AWS IoT thing`: Created a Thing called `picar_project`; Downloaded Cetificates (private key, device certificate, and a root certificate authority); Activated it; Attach the policy `picar_policy` to the Thing.
- `Install AWS IoT SDK`: In RPI terminal and under project folder, git clone https://github.com/aws/aws-iot-device-sdk-python, open the folder, run python setup.pu install.
- `Add certificates to project`: Put all downloaded certificates and keys in our project folder.
- `Add MQTT protocol to project`: Open up python execution files, import AWSIoTMQTTClient library, update AWS IoT certificate based connection with Shadow RESTful API, certificates, and publishing topic.
- `Publish message`: Finally, publish to the topic using a payload variable.

4.3.2 *AWS DynamoDB Action.* [5]

- `Create DynamoDB rule`: In AWS IoT console, create a rule called `picar_dynamodb`. For query statement, put SELECT * FROM '`picar_project/test`', since our publish topic is `'picar_project/test'`. For action, we chose DynamoDBv2.
- Create DynamoDB table: By choosing create a new resource, on Amazon DynamoDB console, click on Create table. Making a table called PicarTable, partition key as String message, and add sort key as String timestamp.
- Configure action: Back to DynamoDB rule creation, choose PicarTable, put our respective partition key and sort key. For "Write message data to this column", put Distance.
- Create role: create role called picar520, and add it to the rule.

4.3.3 *AWS Cognito Setup.*

- Create identity pool: Open AWS Cognito console, choose "Manage Identity Pools" and "Create new identity pool".
- Enable access and allow default roles: Enable unauthenticated identities.Hit "Create Pool", and choose "Allow" for the two default roles, one for authenticated identity and one for unauthenticated identity.



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:[redacted]table/PicarTable",
                "arn:aws:dynamodb[redacted]able/PicarTable/index/*"
            ]
        }
    ]
}
```

Figure 7: Setup unauth role

4.3.4 *AWS DynamoDB, Cognito and Android.*

- AWS IAM Setup: Open AWS IAM console, choose the unauth role called `'Cognito_Picar520Unauth_Role'`, attach a policy and modify the JSON summary with dynamodb actions, DeleteItem, GetItem, PutItem, Scan, Query, UpdateItem, BatchWriteItem as shown in Figure 7. Although we only used the GetItem and Scan for this project, we did all actions for future usage if needed. And update resource as PicarTable ARN and its index.
- AWS Configuration in Android: Open Android App, add a new json file to `"\app\src\main\res\raw"` called awsconfiguration.json as default. Modify the pool ID, region for Cognito identity, and the region for DynamoDB object mapper.
- Android Configuration: Add AWS Mobile SDK in project dependencies namely Cognito and DynamoDB, add internet permission to manifest
- AmazonDynamoDBClient Setup [4]: Using AWSMobileClient to enable user credentials to access DynamoDB table. Instantiate AmazonDynamoDBClient using credential provider context for Cognito and setup DynamoDB mapper.
- Data Model Setup: From imported dynamodbmapper library, we can access the table name of our database, the hash key (partition key = message), the range key (sort key = timestamp), and the additional attribute (distance). Create model for each of the attributes (Get() and Set() methods).
- Read Data: Start a runnable thread, scan the model class with an expression but without a filter due to reading all the data
- Display Data: Create a table view for the data with column ID, timestamp, distance and message.

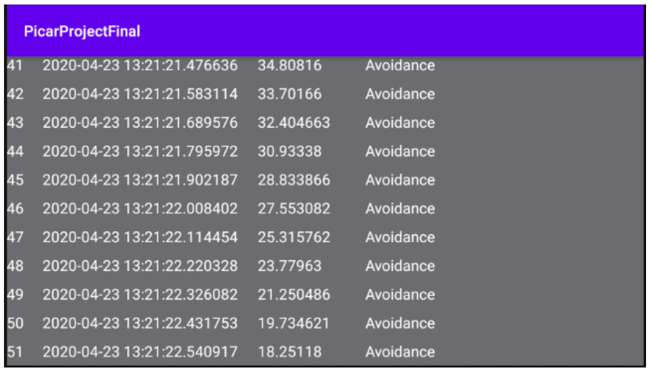
4.3.5 *Raspberry Pi and Android Connection.* .

Set up a socket connection using port 8080 to connect Raspberry Pi and Android application. For each of the car features, namely avoidance and recognization, make a button onClick event to trigger the car to run its respective task.

5 EXPERIMENTS

We setup the Android application using compileSdkVersion 29, minSdkVersion 21 and build gradle 3.6.3 on the Android Studio. For AWS mobile package, we use Amazon Cognito dependency aws-android-sdk-mobile-client:2.7.+@aar, and Amazon DynamoDB dependency aws-android-sdk-ddb-mapper:2.7.+.

5.1 Obstacle Avoidance



```
PicarProjectFinal
41  2020-04-23 13:21:21.476636  34.80816   Avoidance
42  2020-04-23 13:21:21.583114  33.70166   Avoidance
43  2020-04-23 13:21:21.689576  32.404663  Avoidance
44  2020-04-23 13:21:21.795972  30.93338   Avoidance
45  2020-04-23 13:21:21.902187  28.833866  Avoidance
46  2020-04-23 13:21:22.008402  27.553082  Avoidance
47  2020-04-23 13:21:22.114454  25.315762  Avoidance
48  2020-04-23 13:21:22.220328  23.77963   Avoidance
49  2020-04-23 13:21:22.326082  21.250486  Avoidance
50  2020-04-23 13:21:22.431753  19.734621  Avoidance
51  2020-04-23 13:21:22.540917  18.25118   Avoidance
```

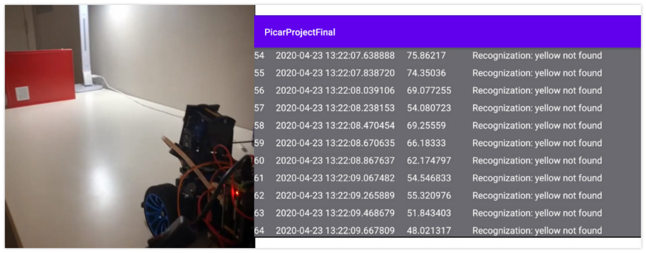
Figure 8: Obstacle Avoidance Data

By putting a box in front of the car, if the car is within the minDistance, the car should stop, and publish its information onto AWS IoT. On Figure 8, It shows the passed data from the raspberry pi to AWS IoT and stored in DynamoDB and is displayed in our Android application. Since we have tested and set minDistance to 20, so when the distance is less than 20, the car will eventually stop. We can see that the distance on the third column on Figure 5 is decreasing gradually and eventually stops.

5.2 Object Recognization

5.2.1 *Data collected without yellow object.* .

From Figure 9, on the left, it is the view that Picar's camera can



```
PicarProjectFinal
54  2020-04-23 13:22:07.638888  75.86217   Recognization: yellow not found
55  2020-04-23 13:22:07.838720  74.35036   Recognization: yellow not found
56  2020-04-23 13:22:08.039106  69.077255  Recognization: yellow not found
57  2020-04-23 13:22:08.238153  54.080723  Recognization: yellow not found
58  2020-04-23 13:22:08.470454  69.25559   Recognization: yellow not found
59  2020-04-23 13:22:08.670635  66.18333   Recognization: yellow not found
60  2020-04-23 13:22:08.867637  62.174797  Recognization: yellow not found
61  2020-04-23 13:22:09.067482  54.546833  Recognization: yellow not found
62  2020-04-23 13:22:09.265889  55.320976  Recognization: yellow not found
63  2020-04-23 13:22:09.468679  51.843403  Recognization: yellow not found
64  2020-04-23 13:22:09.667809  48.021317  Recognization: yellow not found
```

Figure 9: Data without yellow object

capture, there is no yellow object inside the camera view, so all the message we got from DynamoDB are "yellow not found". With the decrease of the distance between the Picar and the object, the Picar will eventually stop before hitting the object.

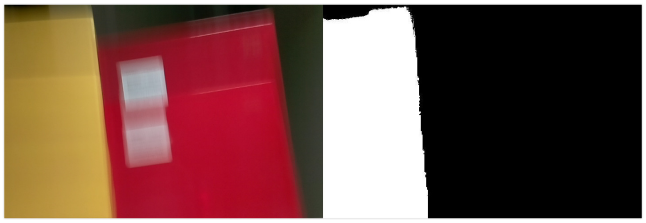
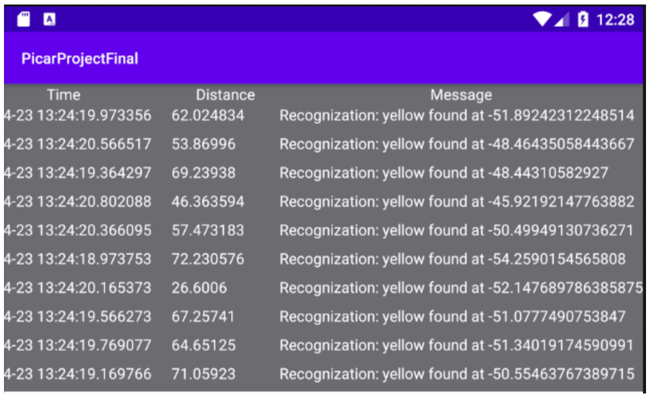


Figure 10: Original and mask image with yellow object

5.2.2 *Data collected with yellow object.* .

On Figure 10, it shows the image that was taken by the Picar on the left and its mask image converted on the right. the Picar has captured the yellow color and created a mask for the yellow part (described in Section 3.2.1).



```
12:28
PicarProjectFinal
      Time            Distance    Message
4-23 13:24:19.973356  62.024834   Recognization: yellow found at -51.89242312248514
4-23 13:24:20.566517  53.86996    Recognization: yellow found at -48.46435058443667
4-23 13:24:19.364297  69.23938    Recognization: yellow found at -48.44310582927
4-23 13:24:20.802088  46.363594   Recognization: yellow found at -45.92192147763882
4-23 13:24:20.366095  57.473183   Recognization: yellow found at -50.49949130736271
4-23 13:24:18.973753  72.230576   Recognization: yellow found at -54.2590154565808
4-23 13:24:20.165373  26.6006     Recognization: yellow found at -52.147689786385875
4-23 13:24:19.566273  67.25741    Recognization: yellow found at -51.0777490753847
4-23 13:24:19.769077  64.65125    Recognization: yellow found at -51.34019174590991
4-23 13:24:19.169766  71.05923    Recognization: yellow found at -50.55463767389715
```

Figure 11: Data with yellow object

After it captured and calculated the angle between itself and yellow object, it will turn to a specific angle and locate the center of the yellow object, and finally eventually move to the yellow object. From Figure 11, we can see that the message shows that "yellow found at <a specific angle>". In the table, we can see from time = 13:24:18.973753 to time = 13:24:20.802088, the angle is in decreasing trend gradually from -54 Degree to -45 Degree. It means the car did turn to the yellow object while it is approaching the object.

6 CONCLUSION

This project built up a PiCar with the features of obstacle avoidance and object recognition and further implemented the data visualization and feature manipulation on Android App. It successfully met all goals mentioned in section 2 above.

According to the tests we made, all the functions work properly as we expected. In the obstacle avoidance test, the PiCar stops the motor at the distance of 20 cm and begins sliding. When the PiCar becomes still, the final distances to the obstacles are around 18 cm. In the object recognition test, if there is no target color in the current view, the car will keep its original moving direction and just perform obstacle avoidance. If there is an object of target color in view along with a non-target object, the PiCar will recognize the target color object, turning to the target and approaching it.

With the successful connection among Raspberry Pi, AWS Iot, DynamoDB, Cognito, and Android Application, during the tests, the distance data and angles data successfully show on the Android App by clicking "show table" button. Clicking the "obstacle avoidance" or "object recognition" buttons on Android App will lead the PiCar perform the corresponding functions that are activated in the python program in Raspberry Pi that is attached to the Picar hardware.

7 FUTURE WORK

There are two optional features, which we mentioned in proposal description, that are able to added to the PiCar for future work.

The first one is line tracking, which the PiCar is able to follow the given line based on infrared reflection on the bottom of the PiCar. In order to implement this feature, we will use black line and a white floor (cardboard), since IR light will only reflect white color surface and absorb black color. We will use two IR sensors to make sure the car follows the line by placing them on each side of the car. If possible, we will use one more IR light in the middle to keep track of black line. If the two IR lights on the sides do not detect the black line, the car will go straight. If the left IR light detects the black line, we will ask the car to turn left to allow the black line get back to the middle, and same idea for the right IR light. If all sensors detect black line, the car will stop. [2]

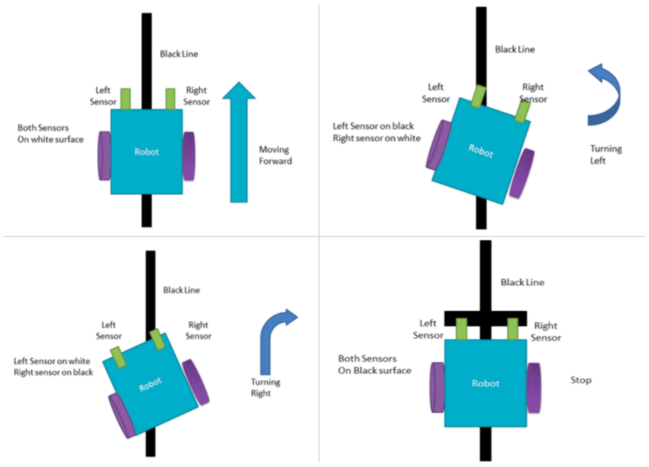


Figure 12: Line Tracking
Source: https://circuitdigest.com/microcontroller-projects/raspberry-pi-line-follower-robot

The other feature is Speech Recognition, which the car is able to understand words given by the user and execute corresponding commands such as "turn left", "turn right", "go straight" and so on. In order to implement this feature, we will first capture the incoming sound from the microphone and preprocess the audio waves, resampling and dealing with the noise. Then, we will convert the speech to text using the API in python and find the corresponding command. Lastly, the PiCar will make its move to follow the command.

In addition, there also can be more works on data visualization. For instance, we can stream the real-time video data from Raspberry Pi camera to the cloud and use other APIs to play the video. As a result, we can experience the real-time view of the PiCar and monitor its movement.

One most important feature we are thinking to add in the future and is inspired by TA, Ruixuan Dai, is that using AWS service to make communication between Raspberry Pi and Android application, instead of using a socket connection between Raspberry Pi and the Android application. To do that, Android application should connect to AWS Iot Service and subscribe to a topic. When a button is triggered to do an action, the application will send a message to Raspberry Pi via the device shadow in AWS service. With the activated application in Raspberry Pi, it should notice a message from AWS and make the Picar to perform. Everything left is same as what we have done in data visualization, specifically real time distance and timestamp data sent back to AWS Iot and displayed in Android application via AWS DynamoDB.

8 ACKNOWLEDGMENTS

We would like to thank Professor Chenyang Lu and two TAs, Ruixuan Dai and Jiangnan Liu of this course (CSE520S Real-Time Systems [7]) for their help and guidance throughout this semester for this project. We also appreciate the hardware support of PiCar from Professor Jim Feher, the Electrical and System Engineering department of Washington University in Saint Louis.

REFERENCES

[1] 20. Adeept Smart Car Robot Kit for Raspberry Pi PiCar-B. https://www.adeept.com/video/static1/itemsfile/Manual-PiCar-B-V2.0.pdf.
[2] 2017. Raspberry Pi Based Line Tracking Robot Example. https://circuitdigest.com/microcontroller-projects/raspberry-pi-line-follower-robot.
[3] 2019. OpenCV API Reference. https://docs.opencv.org/2.4/modules/refman.html.
[4] 2020. Android application, AWS DynamoDB, AWS Cognito Connection Official Documentation. https://docs.aws.amazon.com/aws-mobile/latest/developerguide/mobile-hub-add-aws-mobile-nosql-database.html.
[5] 2020. AWS DynamoDB Setup Official Documentation. https://docs.aws.amazon.com/iot/latest/developerguide/iot-ddb-rule.html.
[6] 2020. AWS Iot and Raspberry Pi Connection Example Official Documentation. https://docs.aws.amazon.com/aws-mobile/latest/developerguide/mobile-hub-add-aws-mobile-nosql-database.html.
[7] 2020. CSE 520S Course Website. https://www.cse.wustl.edu/~lu/cse520s/.
[8] Ethan Shry. 2019. PiCar. https://github.com/ESE205/PiCar.