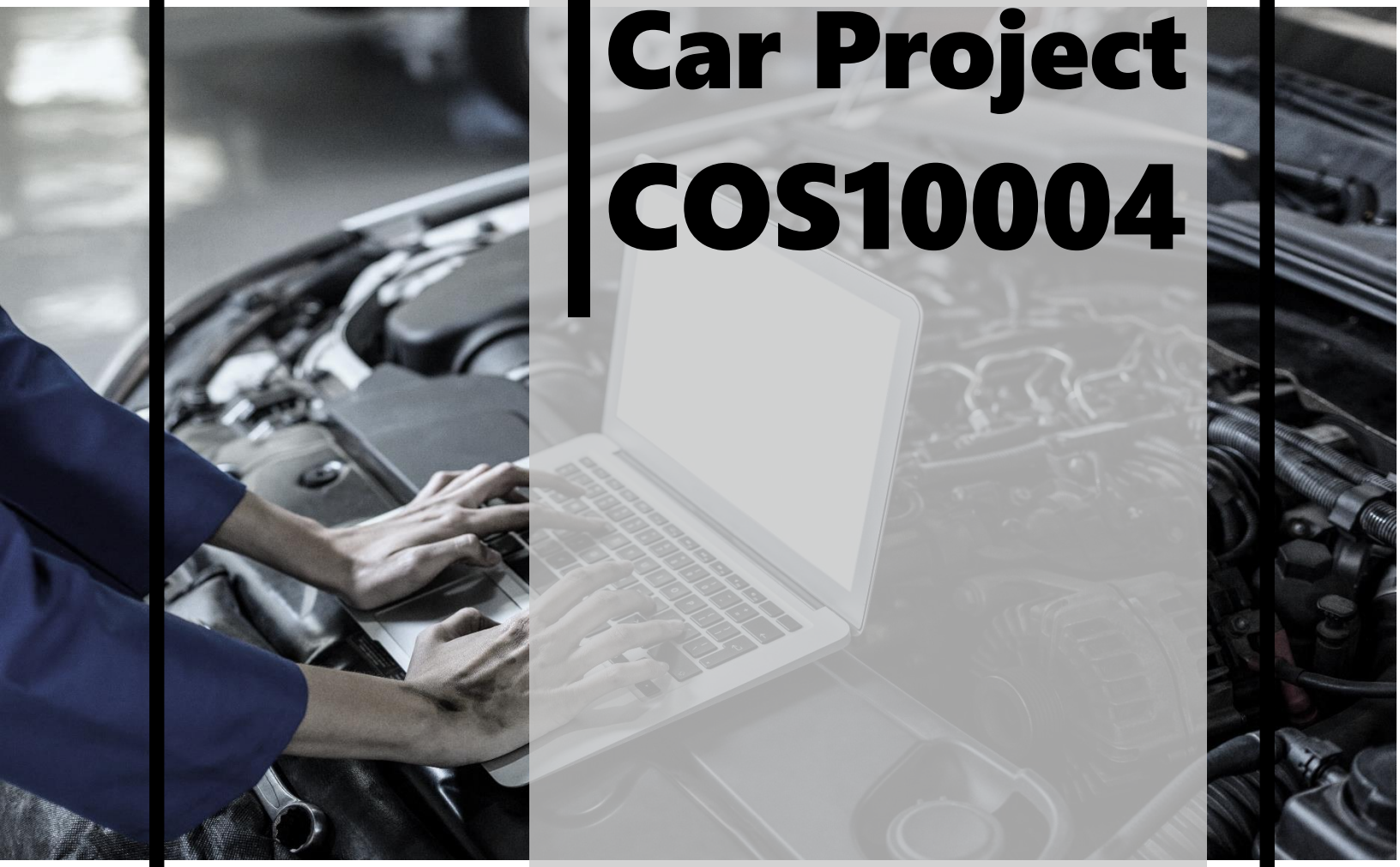
Autonomy Car Project COS10004



Faculty of Science, Engineering and Technology

2020-HE Semester (Sep)

Introduction

The aim of the report is to summarize the process and functions of the autonomy car in COS10004 subject (additional project).

The autonomy car project is to help student with the IOT applications in reality thus programming design in the real project. Students are expected to design the sketch of the products within the requirements from the scenario provided (however we encouraged creativity). The student will be designed the options as well as the circuit for a car platform, and control programming to become the intelligent system.

The students will interact with materials and programming languages, along with essential skills for the job prospects in the future.

This is the report from

Car Assembly

Hardware Components

In this project, we were provided with a remote-control car kit and we were asked to develop an autonomous car. In the kit we have used:

- 4WD Car Kit (motors, wheels, frames, screws)
- Arduino UNO R3 + Sensor Shield
- Raspberry Pi 3B+
- HC-05 Bluetooth Module
- HC-SR04 Ultrasonic Sensor
- 16x2 LCD Display
- Micro Servo 9G
- L298N Motor Driver
- Jumper Wires
- Breadboard
- Potentiometer
- 2x 18650 Battery + Holder
- LED
- Buzzer

In addition, we have bought some supporting components to help us add more functions to the car:

- Buck converter
- DC converter for Arduino
- 5th Grade Car Assembly Kit

Coding Process

After assembly, the functions of the car depend heavily on the coding of the Arduino so we needed to find the most effective way of building this code from scratch.

CarV2

```
#include <SoftwareSerial.h>
#include <Servo.h>  //servo library
Servo myservo;      // create servo object to control servo
```

Fig 1. Code Libraries

After tinkering with the=code we have decided to use ENA, ENB as the motor defaults (no motor library) for controlling the wheels.
Servo motor library to control the servo that spins the ultrasonic sensor.
Finally, the software serial library to extend the Arduino Uno's limited serial ports, which helps provide a digital serial port that can be used by the Bluetooth aside from the Arduino Uno's original singular serial port. (Shown in Fig 1.)

```
void forward(){
  analogWrite(ENA,carSpeed);
  analogWrite(ENB,carSpeed);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
  Serial.println("Left");
  delay(175);
  getstr = 's';
}
void back(){
  analogWrite(ENA,carSpeed);
  analogWrite(ENB,carSpeed);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
  Serial.println("Right");
  delay(175);
  getstr = 's';
}
void right(){
  digitalWrite(ENA,carSpeed);
  digitalWrite(ENB,carSpeed);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
  Serial.println("Forward");
  delay(175);
  getstr = 's';
}
void left(){
  digitalWrite(ENA,carSpeed);
  digitalWrite(ENB,carSpeed);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
  Serial.println("Back");
  delay(175);
  getstr = 's';
}
```

Fig 2. Directional Functions

```
void stop(){
  digitalWrite(ENA,LOW);
  digitalWrite(ENB,LOW);
  Serial.println("Stop!");
}
```

Fig 3. Stop Function

We created 4 core functions that control the motors to move a certain way: Forward(), Left(), Right() and Back(). This controls the direction for the car to go, we have also added a delay to Stop() function at the end of each directional function for ease of control. (Shown in Fig 2,3.)

However, during the development of the Autonomous phase of the car, we ran into some trouble with the code as the car needs to go continuously without stopping until it reaches an obstacle. We came up with the solution to create 4 more identical functions but this time removing the delay to Stop() functions that we have added above for ease of Bluetooth control. The four core functions for the Autonomous mode of the car is Forwardauto(), Leftauto(), Rightauto() and Backauto().

```
int Distance_test() {
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(20);
  digitalWrite(Trig, LOW);
  float Fdistance = pulseIn(Echo, HIGH);
  Fdistance= Fdistance / 58;
  return (int)Fdistance;
}
```

Fig 4. Distance Test Function

We also created a DistanceTest() function for the Autonomous mode, for the car to keep track of when it is nearing an obstacle to stop and scan the area.

```
void loop() {
  if (mySerial.available()> 0){
  getstr = mySerial.read();
  Serial.println(getstr);
  }
  switch(getstr){
    case 'f': forward(); break;
    case 'b': back();   break;
    case 'l': left();   break;
    case 'r': right();  break;
    case 's': stop();   break;
    case 'a':
```

Fig 5. Bluetooth Control Function

We implemented the Software Serial library to create an additional serial port for Bluetooth connections. We then put it in a loop for it to be checked constantly for data and broke down the cases in a switch, each case calls upon its corresponding function. (Shown in Fig 5.)

```
case 'a':
myservo.write(110);  //setservo position according to scaled value
delay(500);
middleDistance = Distance_test();

if(middleDistance <= 50) {
  stop();
  delay(500);
  myservo.write(180);
  delay(1000);
  rightDistance = Distance_test();
  delay(500);
  myservo.write(110);
  delay(1000);
  myservo.write(10);
  delay(1000);
  leftDistance = Distance_test();
  delay(500);
  myservo.write(110);
  delay(1000);
  if(rightDistance > leftDistance) {
    rightauto();
    delay(80);
  }
  else if(rightDistance < leftDistance) {
    leftauto();
    delay(80);
  }
  else if((rightDistance <= 50) || (leftDistance <= 50)) {
    backauto();
    delay(100);
  }
  else {
    forwardauto();
  }
```

Fig 6. Autonomous Function

The code in Autonomous mode will test the distance on the car's front, left, and right sides to determine the most efficient pathway. In the code, you can see that if it detects the front has little distance as well as the right, then the car will choose to go to the left side, same as the other way. If all three directions have no space to move, then the car will spin 180 degrees to go back to where it came. (Shown in Fig 6.)

Car Features

Remote Bluetooth Controls

The car can be operated manually via Bluetooth connections with any device. The car once it is turned on, will constantly check for data being sent via Bluetooth with ‘f’ = Forward, ‘l’ = Left, ‘r’ = Right and ‘b’ = Back. Each time, the data is received the car will go in that direction for 0.175 seconds before stopping. The stop and delay are put in for ease of control which after doing some tests with fellow students within the Swinburne VN campus were highly approved.

Autonomous Mode

Aside from manual controls, the car can be set to automatically roam the terrain. Using servo motors and ultrasonic sensors, we have created a simple algorithm for obstacle avoidance so that the car will find the best way out when blocked by obstacles on the front, left, or right.

As the car moves, the sensors will constantly be checking for too small distances so that the car can avoid frontal obstacles. Once triggered, the car will stop and the sensors will scan in both the left and right direction for each of their respective distances as well as rechecking the front distance as well. After running the algorithm, the car will then pick the route that has the most amount of space for it to travel, rinse and repeat.

This mode also merges seamlessly with our Manual control mode as this mode will only be activated if the car gets data ‘a’ via Bluetooth connections.

Shock Absorber

In front of the car are 5 rubber tires that mitigate the shock the car will receive in situations that it crashes into a wall. This will greatly increase the car’s lifespan, as well as the user’s personal comfort when driving the car knowing that it is a very sturdy car that is very difficult to break.