**Self-Driving Car Report**

Name: Mamert Vonn G. Santelices

Student ID: 90026174

Date: 26 April 2023

Word Count:

Table of Contents

[Introduction 2](#_Toc131187649)

[Background 2](#_Toc131187650)

[Current Research 2](#_Toc131187651)

[Methodology 3](#_Toc131187652)

[Materials 3](#_Toc131187653)

[Procedure 3](#_Toc131187654)

[Schematics 4](#_Toc131187655)

[Blueprint 5](#_Toc131187656)

[Code 6](#_Toc131187657)

[Documentation 13](#_Toc131187658)

[Conclusion 15](#_Toc131187659)

[Results 15](#_Toc131187660)

[Discussion 15](#_Toc131187661)

[Recommendations 15](#_Toc131187662)

[Bibliography 16](#_Toc131187663)

# Introduction

## Background

The Unit Outline of CAT01S1 requires the enrolled student to build a self-driving robot car using an Arduino and the associated parts provided by the tutor. The included associated parts are brought from the Geekcreit DIY L298N 2WD Ultrasonic Smart Tracking Moteur Robot Car Kit**.**The Robot Car Kit includes 2 Wheels powered by DC Motors; a pivot wheel placed in the middle-back; and a distance-measuring sensor intended to assist the Robot Car to navigate through its surroundings. Another goal set for this project is to implement a ‘dance’ feature in the robot car. The robot car will be programmed to dance to the rhythm of a chosen song along with the ability to drive itself.

## Current Research

A quick overview of the project presents itself as constructing a simple Proto-type Roomba with a few gimmicks. Tribelhorn and Dodds (2007) assert the value of Roombas for research and educational purposes because it is a low-cost resource. Tribelhorn and Dodds (2007) list the benefits of a Roomba such as its commercially available hardware interface; released serial API; its sensors and actuator’s accuracy along with the extensibility of its computing and sensory abilities; and its current spatial reasoning algorithms. The ideal result for this project is to produce something similar to the listed capabilities of the iRobot Roomba.

# Methodology

## Materials

|  |  |  |
| --- | --- | --- |
| Unit  Code | Item | Number of Pieces |
| U1 | Arduino Uno R3 | 1 |
| U1.1 | Sensor Expansion Shield V5.0 | 1 |
| U2 | L298N Motor Driver Module | 1 |
| U3 | HC-SR04 Ultrasonic Module Distance Measuring Sensor | 1 |
| BAT2 | 9V Battery Case Holder (w/ six 1.5V Battery Slots) | 1 |
| MS | Micro Servo Motor SG90 | 1 |
| M1,M2 | DC 3V-6V Single Axis Gear Reducer Motor | 2 |
| LW,RW | Rubber Wheels | 2 |
| PW | Pivot Wheel | 1 |
| S2 | ZF SPST, On-None-Off Rocker Switch Panel Mount | 1 |
| J | DC Barrel Jack Plug (Male) | 1 |
| Miscellaneous Materials | | |
| Electrical Cables | | 8 |
| Female-to-female Jumper Wires | | 4 |
| Nut Pole Connecting Screw Rod | | 10 |
| Flat Screws | | 2 |
| Nut & Bolt | | **many** |
| 3D-Printed Placeholder Chassis | | 1 |
| 3D-Printed T-Piece | | 2 |

## Procedure

Firstly, assemble the modules at the top of the chassis. At the tail end of the car, screw four Nut Poles at the appropriate holes for the Arduino Uno. Thereafter, connect the Sensor Shield on top of the Arduino Uno. Next to the Arduino Uno and adjacent to two thin rectangular holes are two rounded holes for two Nut Poles. On top of that Nut Pole would be the Motor Driver Module. Lastly, at the bottom of the Motor Driver Module is a wide rectangular hole for the Panel Mount Switch. Moving on to the bottom of the chassis, screw four Nut Poles at the tail end of the car, then place the Pivot Wheel at the Nut Poles. Secondly, using a flat screw place the Battery Case next to the pivot wheel. Next to the Battery Case, at the left and the right are two thin rectangular holes to insert two T-Pieces. Then, screw in the two DC Motors at the left and right T-Pieces. Look at Figure 2.\* as a reference.

The Battery Case is the source of energy supply for the system. Thus, the DC Barrel Jack Plug is wired from the Battery Case to the Arduino Uno. The Batteries then distribute the power to the Motor Driver by connecting the positive and negative wires to the Voltage and GND pins, respectively. In between that connection is a Panel Mount Switch to regulate the on-and-off function of the system. The DC Motor runs using the output pins of the Motor Driver, wherein two output pins are each connected to one of the two DC Motors. The system can control the motors by using the Motor Driver’s four input pins connected to the Arduino Uno’s digital or analog pins. Further insight can be seen in Figure 1.1

## Schematics

*Figure 1.1: Setting Up Two DC Motors and The Motor Driver*

## Blueprint

|  |  |
| --- | --- |
| *Figure 2.1: On the Top of the Chassis* | *Figure 2.2: On the Bottom of the Chassis* |

## 

## Code

|  |
| --- |
| Car.h |
| #pragma once  #include <Arduino.h>  template<  uint8\_t forwardLeftWheel, uint8\_t backwardLeftWheel,  uint8\_t forwardRightWheel, uint8\_t backwardRightWheel  > class DigitalCar2W {  /\*\*  \* @brief 8-bits of toggles  \* 1st & 2nd bit: stir\_left(10) or keep\_center(11) or stir\_right(01)  \* 3rd bit: isReverse(1) or isNotReverse(0)  \* 4th bit: (1) should update on run or not (0)  \* 5th bit: go(0) or stop(1)  \*/  uint8\_t data = 0b00000011;  inline const bool isGo();  inline const bool isReverse();  inline const bool isLeft();  inline const bool isRight();  public:  DigitalCar2W() {}  void begin() const;  void run();  void reset();  void setGo(const bool p\_go);  void setReverse(const bool p\_reverse);  enum class STIR {  RIGHT = 1,  LEFT,  NEUTRAL  };  void setStir(const STIR p\_stir);  };    template<  uint8\_t forwardLeftWheel, uint8\_t backwardLeftWheel,  uint8\_t forwardRightWheel, uint8\_t backwardRightWheel  > class AnalogCar2W {  float speed;  float angle;  bool isUpdate = false;  void update() const;  public:  AnalogCar2W() {}  void begin() const;  void run() const;  void reset();  void setSpeed(const float p\_speed);  void setAngle(const float p\_rad);  }; |

|  |
| --- |
| Car.cpp |
| #include "Car.h"  #define templateCar2W template<uint8\_t forwardLeftWheel, uint8\_t backwardLeftWheel, uint8\_t forwardRightWheel, uint8\_t backwardRightWheel>  #define DigitalCar2W DigitalCar2W<forwardLeftWheel, backwardLeftWheel, forwardRightWheel, backwardRightWheel>  #define AnalogCar2W AnalogCar2W<forwardLeftWheel, backwardLeftWheel, forwardRightWheel, backwardRightWheel>  templateCar2W  void DigitalCar2W::begin() const {  pinMode(forwardLeftWheel, OUTPUT);  pinMode(backwardLeftWheel, OUTPUT);  pinMode(forwardRightWheel, OUTPUT);  pinMode(backwardRightWheel, OUTPUT);  }  templateCar2W  void DigitalCar2W::run(){  if(!(data & 0b00001000))  return;  digitalWrite(forwardLeftWheel, (isGo() & !isReverse() & isRight()? HIGH: LOW));  digitalWrite(backwardLeftWheel, (isGo() & isReverse() & isLeft()? HIGH: LOW));  digitalWrite(forwardRightWheel, (isGo() & !isReverse() & isLeft()? HIGH: LOW));  digitalWrite(backwardRightWheel, (isGo() & isReverse() & isRight()? HIGH: LOW));  data &= 0b11110111;  }  templateCar2W  void DigitalCar2W::reset() {  data = 0b00001011;  }  templateCar2W  void DigitalCar2W::setGo(const bool p\_go) {  data &= 0b00001111;  data |= 0b00001000 | (p\_go? 0b00010000 : 0);  }  templateCar2W  inline const bool DigitalCar2W::isGo() {  return (data >> 4);  }  templateCar2W  void DigitalCar2W::setReverse(const bool p\_reverse) {  data &= 0b00011011;  data |= 0b00001000 | (p\_reverse? 0b00000100 : 0);  }  templateCar2W  inline const bool DigitalCar2W::isReverse() {  return (data & 0b00000100);  }  templateCar2W  void DigitalCar2W::setStir(const DigitalCar2W::STIR p\_stir) {  data &= 0b00011100;  data |= 0b00001000 | (uint8\_t)(p\_stir);  }  templateCar2W  inline const bool DigitalCar2W::isLeft() {  return (data & 0b00000010);  }  templateCar2W  inline const bool DigitalCar2W::isRight() {  return (data & 0b00000001);  }  templateCar2W  void AnalogCar2W::begin() const {  pinMode(forwardLeftWheel, OUTPUT);  pinMode(backwardLeftWheel, OUTPUT);  pinMode(forwardRightWheel, OUTPUT);  pinMode(backwardRightWheel, OUTPUT);  }  templateCar2W  void AnalogCar2W::run() const {  if(!isUpdate)  return;  isUpdate = false;  if(speed == 0.0) {  digitalWrite(forwardLeftWheel, LOW);  digitalWrite(backwardLeftWheel, LOW);  digitalWrite(forwardRightWheel, LOW);  digitalWrite(backwardRightWheel, LOW);  return;  }  update();  }  templateCar2W  void AnalogCar2W::reset() {  speed = 0;  angle = 0;  isUpdate = true;  }  templateCar2W  void AnalogCar2W::update() const {  const uint8\_t s = speed > 0? speed : -speed;  float a = cos(angle);  const uint8\_t sl = 255 \* (speed > 0? (angle < 0? (a<0?-a:a): 1.0) : (angle > 0? (a<0?-a:a): 1.0)) \* s;  const uint8\_t sr = 255 \* (speed > 0? (angle > 0? (a<0?-a:a): 1.0) : (angle < 0? (a<0?-a:a): 1.0)) \* s;  if(speed > 0? (a < 0 && angle > 0) : (angle >= 0 || a >= 0)) {  analogWrite(forwardLeftWheel, sl);  digitalWrite(backwardLeftWheel, LOW);  } else { // speed < 0 && (angle <= 0 || a >= 0)  digitalWrite(forwardLeftWheel, LOW);  analogWrite(backwardLeftWheel, sl);  }  if(speed > 0? (a < 0 && angle < 0) : (angle <= 0 || a >= 0)) {  analogWrite(forwardRightWheel, sl);  digitalWrite(backwardRightWheel, LOW);  } else {  digitalWrite(forwardRightWheel, LOW);  analogWrite(backwardRightWheel, sl);  }  }  templateCar2W  void AnalogCar2W::setSpeed(const float p\_speed) {  speed = p\_speed;  isUpdate = true;  }  templateCar2W  void AnalogCar2W::setAngle(const float p\_angle) {  angle = p\_angle;  isUpdate = true;  } |

|  |
| --- |
| routine.h |
| #include <Car.h>  using DC2W = DigitalCar2W<6,9,10,11>;  void my\_routine(DC2W& car) {  //example:  //forward  car.setGo(true);  car.run();  delay(1000);  //left forward  car.setStir(DC2W::STIR::LEFT);  car.run();  delay(1000);  //right forward  car.setStir(DC2W::STIR::RIGHT);  car.run();  delay(1000);  //right backward  car.setReverse(true);  car.run();  delay(1000);  //left backward  car.setStir(DC2W::STIR::LEFT);  car.run();  delay(1000);  //backward  car.setStir(DC2W::STIR::NEUTRAL);  car.run();  delay(1000);  car.setGo(false);  car.setReverse(false);  delay(1000);  }  void dance(DC2W& car) {  //137 bpm per minute song  car.setGo(true);  car.setStir(DC2W::STIR::RIGHT);  car.run();  delay(448);    car.setStir(DC2W::STIR::LEFT);  car.run();  delay(448);  car.setStir(DC2W::STIR::NEUTRAL);  car.setReverse(true);  car.run();  delay(448);  car.setReverse(false);  car.setStir(DC2W::STIR::LEFT);  car.run();  delay(448);  car.setReverse(true);  car.setStir(DC2W::STIR::RIGHT);  car.run();  delay(446);    car.setReverse(false);  car.run();  delay(448);  car.setReverse(true);  car.setStir(DC2W::STIR::LEFT);  car.run();  delay(448);  car.setReverse(false);  car.run();  delay(448);  car.setStir(DC2W::STIR::NEUTRAL);  car.setReverse(true);  car.run();  delay(448);  car.setReverse(false);  car.run();  delay(446);  } |

|  |
| --- |
| main.ino |
| #include <Arduino.h>  #include "routine.h"  DC2W car;  void setup() {  car.begin();    car.reset();  car.run();  delay(5000);  }  void loop() {  //my\_routine(car);  dance(car);  } |

## Documentation

**Getting Started with the API:**

Install VS Code as your IDE. Then, add the “PlatformIO” extension in the VS Code editor. Download the framework from the Github repository: https://github.com/marthvon/RobotCar.git

Then, add the lib files to the same directory as the codebase.

|  |
| --- |
| #include “Car.h”  DigitalCar2W<6,9,10,11> car; //or  //AnalogCar2W<6,9,10,11>  void setup() {  car.begin();  } |

The car serializes by calling begin() in the void setup() function. The template arguments are placed in between the angular brackets [<. >], which refers to the:

<forwardLeftWheel, backwardLeftWheel, forwardRightWheel, backwardRightWheel>

Replace the values in between the angular brackets with the pin numbers used by the Arduino to control the wheels of the Robot Car. If the Robot Car where to use Analog pins, it is valid to enter macros, such as A1, A2, and so on, at the template arguments. The order of the pin numbers entered for the template arguments of forward(Left | Right)Wheel and backward(Left | Right)Wheel is important. The forward pins mean that if the pin is switch HIGH and the backward pin is switched LOW, then it goes forward. While, backward pins mean that if the pin switch HIGH and the forward pins is switched LOW, then it goes backward.

There are two classes that may be used for the Robot Car. The DigitalCar2W class is used for a two-wheeled car controlled by digital pins. While, the AnalogCar2W class is also used for a two wheeled car, but instead controlled by analog pins. Thus, allowing you to control the speed of the motor with more precision.

**Member Function** of DigitalCar2W

|  |  |  |
| --- | --- | --- |
| **Function** | **Parameters** | **Description** |
| **setGo** | (bool) | The car goes if true else it stops |
| **setReverse** | (bool) | The car goes in reverse if true else not |
| **setStir** | (STIR::LEFT, STIR::NEUTRAL, STIR::RIGHT) | Set the stir of the car to the left, neutral, or forward |

**Member Function** of AnalogCar2W

|  |  |  |
| --- | --- | --- |
| **Function** | **Parameters** | **Description** |
| **setSpeed** | (float) | Enter values in between -1.0 to 1.0, wherein 0 means to stop and negative means to go reverse |
| **setAngle** | (float) | Enter values in between to controlling the car’s stir |

Note that all changes will not affect the robot car unless run() is called. For example:

|  |
| --- |
| car.setGo(true);      car.run(); |

Wherein, car is declared as a variable of type DigitalCar2W.

Another function is reset(); , that will reset all the configurations of the car. Using the same variable as the last example, reset can be called by doing: car.reset();

# Conclusion

## Results

The codebase successfully implements a serial API used in re-programming the Robot Car’s behavior. The designed framework is intuitive and easy to understand for other users. The design takes into consideration principles of Object-Oriented Programming, such as abstraction and encapsulation. It was built in a way that code can easily be built on top of it.

## Discussion

The Robot Car was successfully constructed with the provided parts and the designed API made it incredibly easy to configure the cars movements. Thus, a one-minute video of a dancing robot was produced. Although, the wheels were hitting the edges of the chassis when going in reverse. This causes the motor to move slower than when it was going forward. A slight problem of the design was the placement of the battery case was colliding with the motors. Further, redesigns of the chassis is suggested. Often times, the Nuts and bolts were prone to coming loose and falling off. Based on the requirements listed by Tribelhorn and Dodds, the Robot Car failed to implement any sensory abilities or spatial algorithm to truly be a self-driving car, but remains a low-cost resource purchasable at only AUD 40. Overall, the robot car was doable for the assessment requirements but could be greater improved upon.

## Recommendations

The Robot Car’s sensory capabilities can further be improved by using a temperature sensor and a relative humidity sensor to enhance the accuracy of the Ultrasonic Distance Measuring Sensor. If the Robot Car were to install a Radar sensor, the temperature, and relative humidity sensors can also be used to improve the data collected from the Radar. The suggested temperature and relative humidity sensor used be a BME280 Module and the researcher recommends using the Adafruit BME280 Library for the codebase. There are several modules to install, so to improve the Robot Car’s spatial sensory extensibilities, such as Lidar and a Camera. The Lidar, also known as light detection and ranging or laser imaging, detection and ranging, is a detection system that works on the principle of radar but uses light from a laser. The Lidar and the Camera would require a parallel processing unit to properly implement image processing and possibly Machine Learning models on the Robot Car. There are two drawbacks to installing a large number of modules. The more parts rely upon or intertwined with each other, then it is more likely to be damaged with maintenance and repair being laborious and difficult. And, the Robot Car fails one of its objectives to be a low-cost resource when installing more expensive modules. It can also be recommended to use a different microprocessor from the Arduino Uno. The researcher suggests using an ESP32, as not only does it run at a higher frequency than an Arduino Uno, but it also has a built-in WiFi and Bluetooth module. The WiFi and Bluetooth module can be used to log and monitor the Robot Car’s status while running, so to easily analyze the collected data and find bugs. This may also be used to implement an interface to remotely control to Robot Car through Bluetooth or WiFi.

# Bibliography

Tribelhorn, Ben and Zachary Dodds. 2007. “Evaluating the Roomba: A Low-cost, Ubiquitous Platform for Robotics Research and Education” Proceedings 2007 *IEEE International Conference on Robotics and Automation*. https://doi.org/10.1109/ROBOT.2007.363179.