

# Car Autopilot System with Cruise Control

By Ye Song

## Description of the product:

The cars' auto driving system has always been a popular and interesting topic in modern times where's industry is highly developed. A system with constant speeding cruise control and automatic driving system can not only bring considerable convenience to people's life, but also accelerate the development of human technology to the greatest extent. The assignment 5 introduced the method of tuning motor with PI controller to show some behaviors of rejecting environmental disturbances, which extremely simulated my interests about wokring on a real car's cruise controll system and added extra autopilot features by including frontal and back object sensors. And besides that, since our microkits also contained one load cell, which made the entire ideation much more interesting. Because I could possibly use the load cell as the representation of car's throttle with applied different level of stress-strain forces onto the load cell. The much harder I pressed the load cell, the more power would be applied onto the motor(engine). All in all, the entire system included : esp32, load cell, motor, motor driver, encoder and connector, 5V power supply, switch, object sensors, another DC motor as representation of step motor (did not buy), and finally the HW-29 load cell connector. Above introduced equipments would result a plant model which could be able to let the DC motor reject environmental disturbances, use the load cell as throttle, sensed approaching front and back objects then making warning, turning on the mock step motor to steer the axis with universal joints assembly to the left or right. Also if I pressed the load cell when cruise control turned on mode, the motor would slowly decrease to zero when released the load cell, just like a real car.

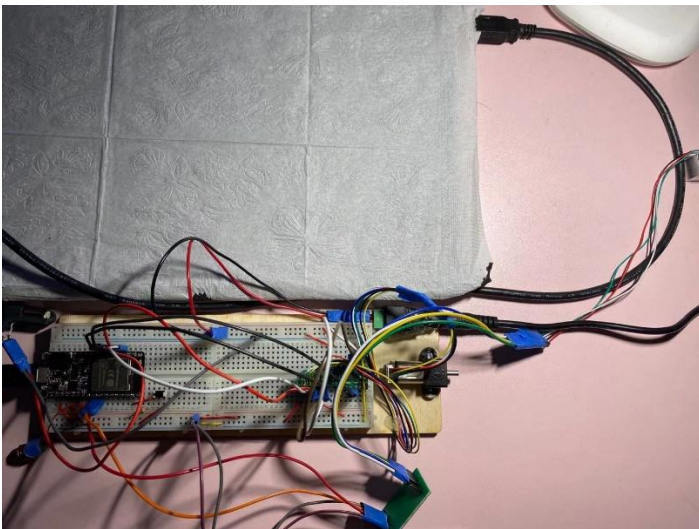


Figure 1: General set up of the entire system

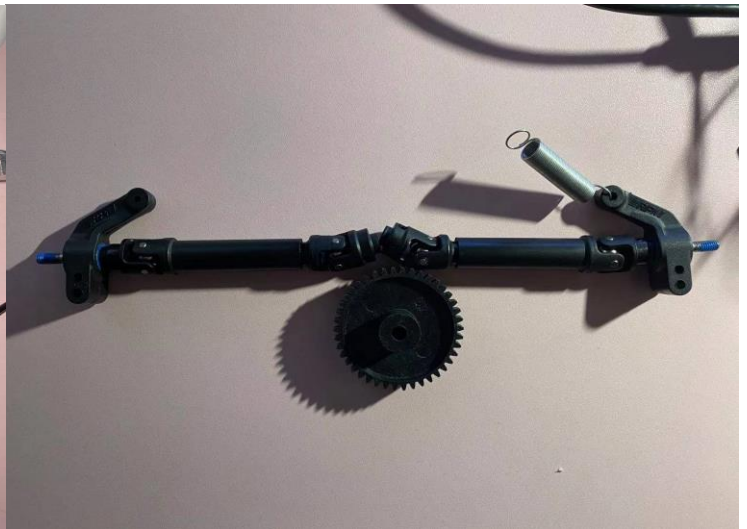


Figure 2: Mock steering system with universal joints assembly

For a video example of the device's operation and introduction, visist: <https://www.youtube.com/watch?v=aXhMwYXHlBl&feature=youtu.be>.

## Electromechanical details:

### Controller plant:



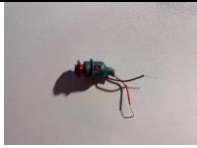







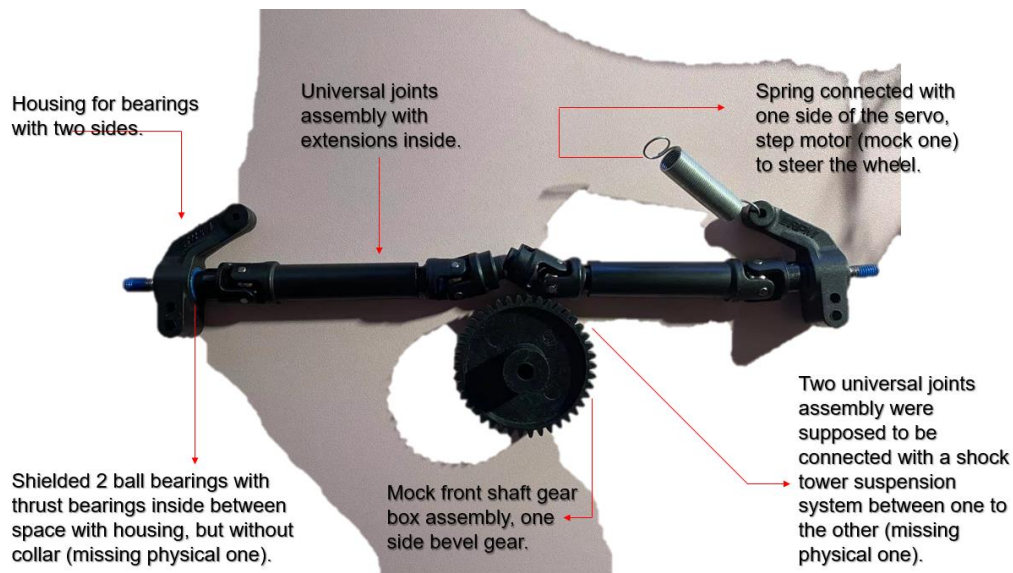
<u>Picture</u>	<u>Name</u>	<u>Funtionality</u>	<u>Quantity</u>
	Unknown Manufacturer's Object Sensor	2 used corresponded sensors represent the frontal and back one, which would sense any objects who are approaching the sensor then sending the analog signal back to the micro controller then resulting next set up behaviors of the plant.	2
	75:1 Micro Metal Gearmotor extended Motor Shaft with connector	The corresponded motor and connector set up was a mock representation of car's engine. And the encoder would transferring counts into calculations inside of the arduino sketch, resulted a calculated real-time output angular velocity in rpm.	1
	DRV8833 Dual Motor Driver Carrier	Giving the written car's cruise control codes from arduino skectch onto the board then to adjust the real time behavior the motor to a satisfy level. Also taking encoder's signals into considerations.	1
	Load Cell	Transferring the applied stress-strain forces to analog input signal. The applied stress-strain forces were mock representations of pushing back and forth onto the car's throttle then powering the engine(motor).	1
	ESP-32 Micro Controller	The core of this entire plant model. Controlling all sensors' and motors' behavior, and receiving all the digital, analog signals, then manipulating them as we wished.	1
	Unknown Manufacturer's DC Motor	This is just a mock model of the step motor (because I don't have one), which would control steering the frontal shaft left and right.	1
	Pushbutton Switch	Taking control of turning on and off the motor plant.	1
	HX711 Load Cell Amplifier	Transferring the analog input signal received by the load cell onto the micro controller for manipulating them later on.	1

Table 1: Table concluded all parts in the plant model

Frontal shaft assmebly:



## Circuit & Coding:

Interesting but also challenging part of this plant model was the calibrating processes of load cell and object sensors. Because I did not purchase some real fancy motion or object sensors, those two poor ones I disassembled from my older drone were really crappy and contained a large amount of noises, shown on the below figure 5. Those two sensors also show up some random unexpected errors during tuning, but I had a chance to make it work with a standable level. On the other hand, the system was supposed to contain at least one step motor to move the its output shafts onto a specific angle so that it would be able to steer the axis left or right. But the motor I used on it was also a relative cheap one disassembled from the same drone. It's a DC motor, so it is just the mock representation of the real step motor.

### (1) Load cell and amplifier:

Load cell and its amplifier were representation of the throttle of real cars. Any small forces we applied onto the rod's surface would result changing its resistance and then transferring that amount of changes to analog input signal so that the ESP-32 could read it and manipulate that signal (**calibration process see figure 7 at appendix A, page 1**). After calibrations of it, it could be taken as the duty cycle output value of the motor by `ledcWrite` function. So finally, it would work just like a throttle of the motor, how hard you pressed it would give how much power output of the engine (motor).

### (2) Object sensors:

The object sensors were taking a really important role of the autopilot plant model. They're sensing the front or back safeness. Any approaching objects would result a change of analog input signal to the corresponded ESP-32 pin. Then the written sketch would give decisions on what action the plant would do for the next step, either warning or steering the front shaft to the nearest safe lane.

### (3) Mock step motor:

Mock step motor was simulating the steering system of the frontal shaft. It would be triggered when any objects are approaching the rear side of the plant. Then if there isn't any object sensed in the front, the mock step motor would steer the plant to the nearest safe lane (left or right).

### (4) PI controller:

PI controller has no physical phenomenon presenting except rejecting the environmental disturbances and system noises. It is a written script inside of arduino sketch. General open loop control would not show any behaviors of rejecting the environmental disturbances or noises of the system. But PI controller would take those two factors into calculations and considerations. Just acting like a regular car cruise control which would apply more torque when facing rocks and hills on the road.

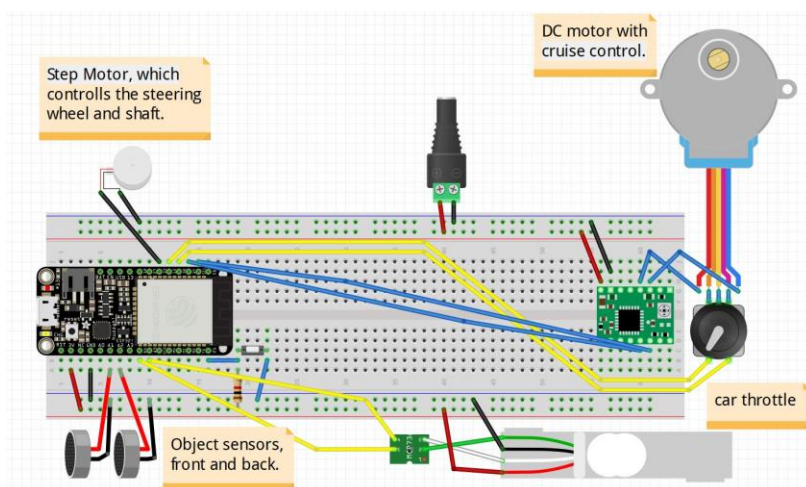


Figure 4: Circuit diagram of the plant model without shaft assembly

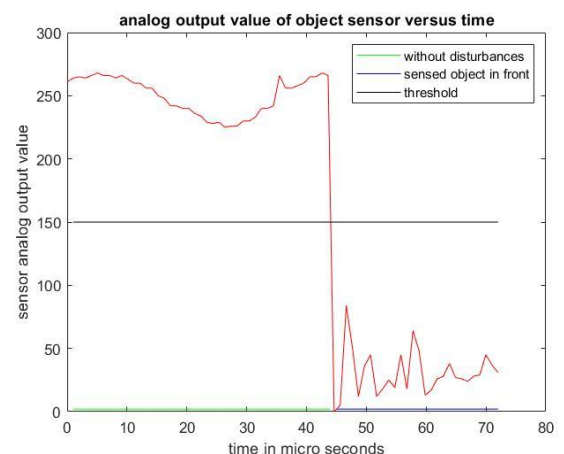


Figure 5: Calibration plot of the object sensor

## Finite State machine:

The entire plant's behavior was mainly controlled by the ESP-32 and correlated various sensors. Turning on the switch back and forth would give a signal of cruise control and motor operation, ON and OFF. When the switch was turned on, the plant would act pretty normal until any additional forces were applied onto the load cell. It's just like you stepped on the gas pedal of your car when it was running at cruise mode on (see figure 8 at appendix A, page 1). The car would accelerate or decelerate, changing along with how hard you pressed the load cell (gas pedal). But the interesting point of it is when you release the load cell, the motor would slowly decrease to zero power, just like a regular car. Another point, when the normal cruise mode was on, if you move some object really close to the front object sensor, the frontal state would turn into HIGH. At this condition, if the back object sensor also sensed some objects were approaching to the model, the monitor would show up "Front and back collision detected, please switch lane or protecting yourself with a proper action". On the contrary, if the rear object sensor did not sense anything at all, the entire plant would just make the motor slowly decrease to zero power output, for preventing hitting the frontal sensed object, which either could be a car or a rock. Now trace all back to the normal cruise control mode on state. If the back sensor got activated and front object did not sense anything at all, the screen would also show up a warning that telling you which your car detected something is approaching to your rear (see figure 5 in the previous page). As a certain time going by, the system would make the choice for you, which including turning on the step motor, dragging the spring loaded on the frontal shaft, then twisting the universal joint (front axis) to nearest safe lane (left or right). Since we had already concluded the situation of both sensor activated mode at the other if statement we setup, so it won't be necessary to do that again here. Those should work just fine, like a real autopilot car's system.

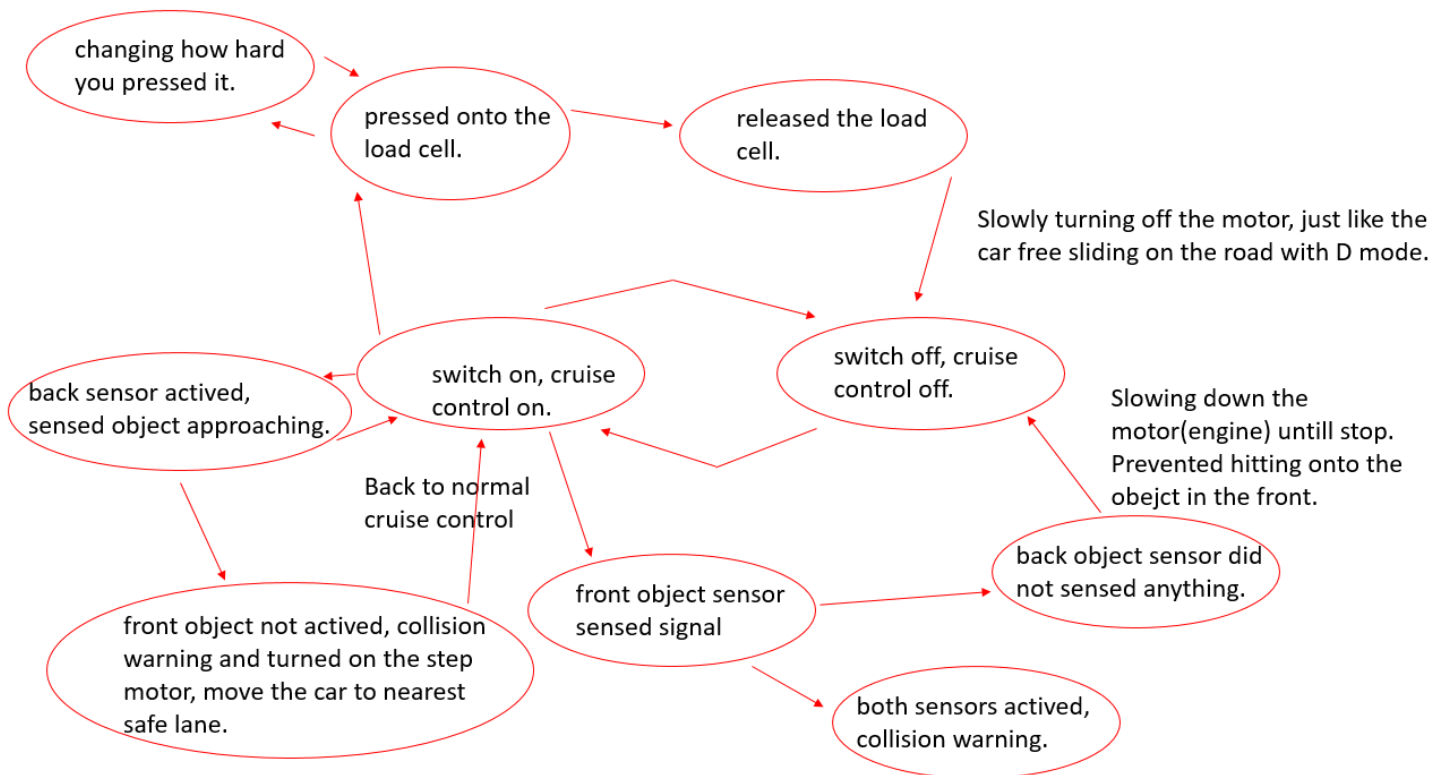


Figure 6: Sample FSM diagram of the entire autopilot plant model

## **Appendix A: Figures**

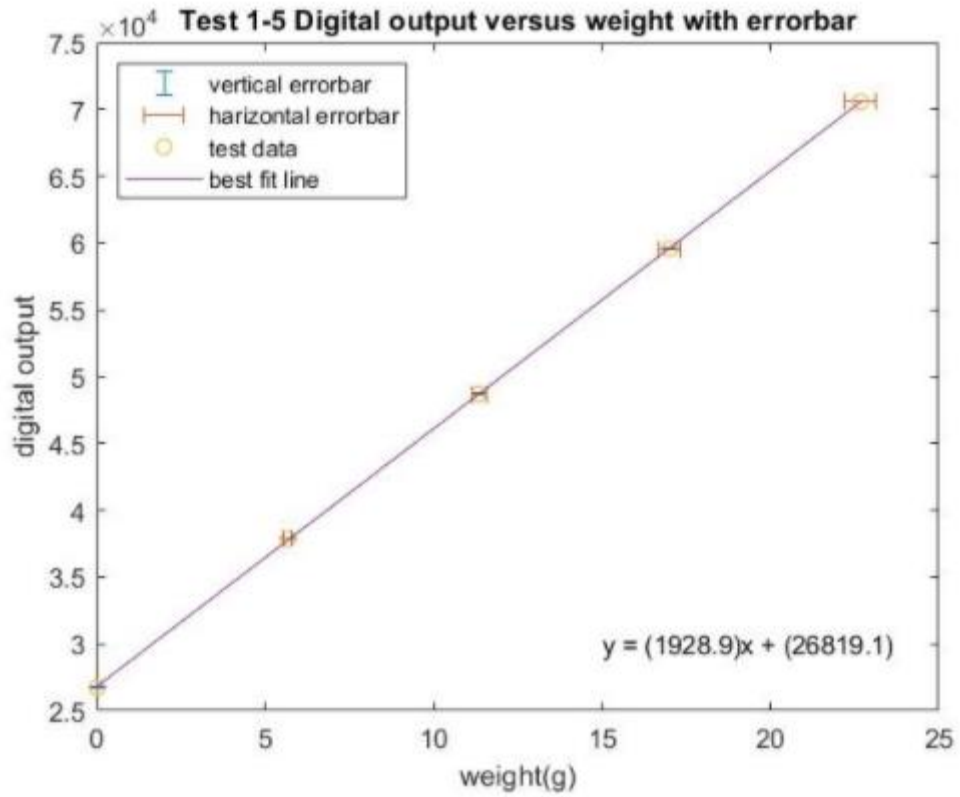


Figure 7: Plot of digital output versus weight with errorbar for test 1-5 on load cell.

Equation of the best fit line is shown at the bottom right

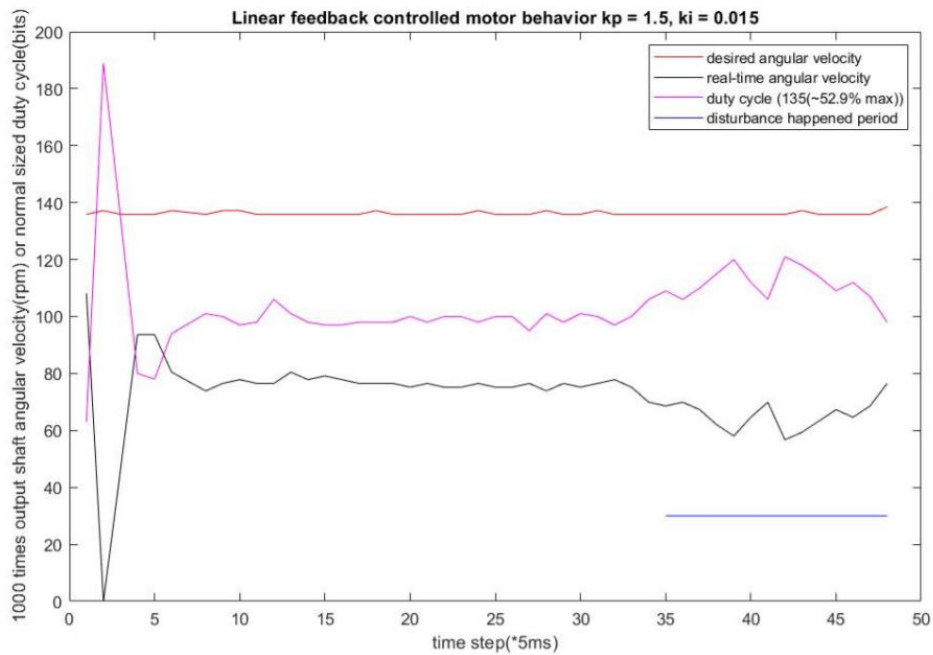


Figure 8: Plot of digital output versus time for both, facing disturbances, and without disturbances rising and steady states

## **Appendix B: Arduino Sketch**

```

#include <ESP32Encoder.h>
#include "HX711.h"
ESP32Encoder encoder2;
unsigned long encoder2lastToggled;
#define BUTTON_PIN 21
#define ENCODE_PINA 32
#define ENCODE_PINB 14

#define MOTOR_PIN0 27
#define MOTOR_PIN1 33
#define motorsm 15

#define FREQ 12000
#define RESOLUTION 8
#define LED_CHANNEL0 0
#define LED_CHANNEL1 1

#define FEEDBACK_PERIOD 50000

#define DEBOUNCE 120
hw_timer_t * timer = NULL;
volatile SemaphoreHandle_t timerSemaphore;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

int memc;
int dutyCycle_state1 = 0;
int reading = 0;
int front = 200;
int back = 200;
boolean loadcellstate = 0;
boolean state = 0;
const int LOADCELL_DOUT_PIN = A4;
const int LOADCELL_SCK_PIN = A5;
const int objectsensor_front = A1 ;
const int objectsensor_back = A2;
volatile int errorSum = 0;
volatile int velocity = 0;
volatile int error = 0;
volatile int velGoal = 0;
volatile int counts = 0;
volatile int buttonTimer = 0;
volatile int buttonTimer_last = 0;
volatile int memt = 0;
volatile int meme = 0;
volatile int memr = 0;
volatile int memd = 0;
volatile int memb = 0;
volatile float kp = 1.5;
volatile float ki = 0.015;
volatile boolean buttonPressEvent = false;
HX711 scale;

void IRAM_ATTR onTimer() {
    portENTER_CRITICAL_ISR(&timerMux);
    counts = encoder2.getCount();
    velocity = -(((memc - counts)*60000)/45.486);
    error = abs(velGoal - velocity);
    portEXIT_CRITICAL_ISR(&timerMux);+
    xSemaphoreGiveFromISR(timerSemaphore, NULL);
}

```



```

    memc = counts;
    memt = memt + 1;
    meme = meme + (error * 0.00008333);
    memr = reading;
    memd = dutyCycle_state1;
}

void setup() {
  Serial.begin(115200);
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
  pinMode(MOTOR_PIN0, OUTPUT);
  pinMode(MOTOR_PIN1, OUTPUT);
  pinMode(ENCODE_PINA, INPUT_PULLUP);
  pinMode(ENCODE_PINB, INPUT_PULLUP);
  pinMode(objectsensor_front, INPUT);
  pinMode(objectsensor_back, INPUT);
  pinMode(motorsm, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonIsPressed, RISING);
  ledcSetup(LED_CHANNEL0, FREQ, RESOLUTION);
  ledcSetup(LED_CHANNEL1, FREQ, RESOLUTION);
  ledcAttachPin(MOTOR_PIN0, LED_CHANNEL0);
  ledcAttachPin(MOTOR_PIN1, LED_CHANNEL1);

  ledcWrite(LED_CHANNEL0, 0);
  ledcWrite(LED_CHANNEL1, 0);
  digitalWrite(LED_BUILTIN, LOW);

  timerSemaphore = xSemaphoreCreateBinary();
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &onTimer, true);
  timerAlarmWrite(timer, FEEDBACK_PERIOD, true);
  encoder2.attachFullQuad(ENCODE_PINA, ENCODE_PINB);
}

void loop() {
  switch (state) {
    case 0:
      Service0();
      if (memd > 0) {
        memd = memd - 5;
        delay(500);
      }
      else {
        memd = 0;
      }
      //front = analogRead(objectsensor_front);
      //back = analogRead(objectsensor_back);
      //if (back > 30 && front >= 200) {
        //Serial.println("Warning: Back and front collisions detected, please take
cover.");
      //}
      if (buttonPressEvent == true) {
        state = 1;
        Service1 ();
        buttonPressEvent = false;
      }
    }
}

```

```

break;
case 1:
front = analogRead(objectsensor_front);
back = analogRead(objectsensor_back);
if (scale.is_ready()) {
reading = scale.read();
reading = reading/10000 + 120;
if (reading > 205) {
reading = 205;
}
velGoal = ((reading - 29.485)/(0.7766))*1000;
}
Service1();
Serial.print(memt);
Serial.print(",");
Serial.print(velocity);
Serial.print(",");
Serial.print(velGoal);
Serial.print(",");
Serial.println(dutyCycle_state1);
if (((memr - reading) < -5) && memr!=0 && memr!=117 ) {
loadcellstate = 1;
}
if (loadcellstate == 1 && reading == 117) {
loadcellstate = 0;
state = 0;
Service0();
reading = 0;
buttonPressEvent = false;
}
if (front >= 300) {
if (back == 0) {
loadcellstate = 0;
state = 0;
Service0();
reading = 0;
buttonPressEvent = false;
}
}
if (back > 100) {
if (front < 200) {
Serial.println("Warning: Back collision dectected, try to switch lane.");
digitalWrite(motorsm,HIGH);
delay(500);
digitalWrite(motorsm,LOW);
}
}
if (buttonPressEvent == true) {
state = 0;
Service0();
reading = 0;
buttonPressEvent = false;
}
break;
default:
timerAlarmDisable(timer);
ledcWrite(LED_CHANNEL0, 0);
ledcWrite(LED_CHANNEL1, 0);
digitalWrite(LED_BUILTIN, LOW);

```

```

        delay(500);
        digitalWrite(LED_BUILTIN, HIGH);
        delay(500);
        break;
    }
}

void Service0() {
    digitalWrite(LED_BUILTIN, LOW);
    timerAlarmDisable(timer);
    ledcWrite(LED_CHANNEL0, memd);
    ledcWrite(LED_CHANNEL1, 0);
    memt = 0;
}

void Service1() {
    digitalWrite(LED_BUILTIN, HIGH);
    timerAlarmEnable(timer);
    dutyCycle_state1 = reading;
    if (velocity < velGoal) {
        dutyCycle_state1 = kp * error + (kp * ki * meme);
        dutyCycle_state1 = ((0.7766 * (dutyCycle_state1))/1000) + 29.485;
        if (dutyCycle_state1 > 205) {
            dutyCycle_state1 = 205;
        }
        ledcWrite(LED_CHANNEL0, dutyCycle_state1);
        ledcWrite(LED_CHANNEL1, 0);
    }
    else if (velocity >= velGoal) {
        if (dutyCycle_state1 > 205) {
            dutyCycle_state1 = 205;
        }
        ledcWrite(LED_CHANNEL0, dutyCycle_state1);
        ledcWrite(LED_CHANNEL1, 0);
    }
}

void buttonIsPressed() {
    buttonTimer = millis();
    if ((buttonTimer - buttonTimer_last) > DEBOUNCE) {
        buttonPressEvent = true;
        buttonTimer_last = buttonTimer;
    }
}

```