AuE 835
Automotive Electronics Integration

PROJECT 2: ADAPTIVE CRUISE CONTROL AND AUTONOMOUS LANE KEEPING

Project Schedule

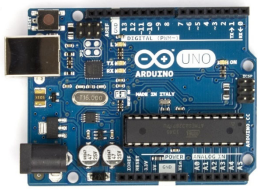
- Nov 2 – Arduino and programming
- Nov 4 – Ultrasonic sensing & Project 1 Announcement
- Nov 9 – Signal processing review and hands-on teaching
- Nov 11 - Project 1 debugging, Q&As, Test details
- Nov 16 - Project 1 Presentation and Test

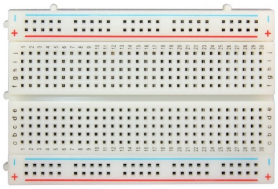
- **Nov 18 – Vehicle Control & Project 2 Announcement**
- **Nov 23 – Control review and hands-on teaching**
- **Nov 30 – Project 2 debugging and Q&As, Test details**
- **Dec 2 – Project 2 Presentation and Test**

- **Dec 10 - Project reports**

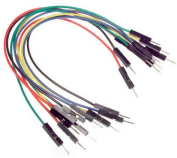
RC Car for the Project 2

- You need the following

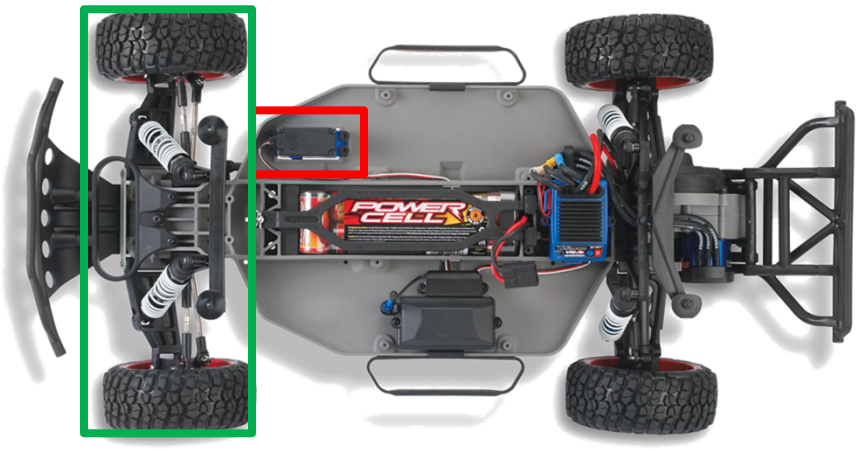






The Vehicle

Steering Servo Motor



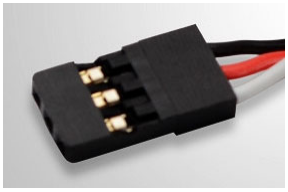
Steering servo: This controls the steering of the front wheels of the vehicle. It requires 5V input.

Electronic Speed Controller (ESC)

ESC: This controls the speed (throttle) of the motor. It also produces 5V output for servo.

ESC and Servo Connections

- The ends of ESC and servo cables look like this:
  - We have 1 of these for each actuator



Remember:

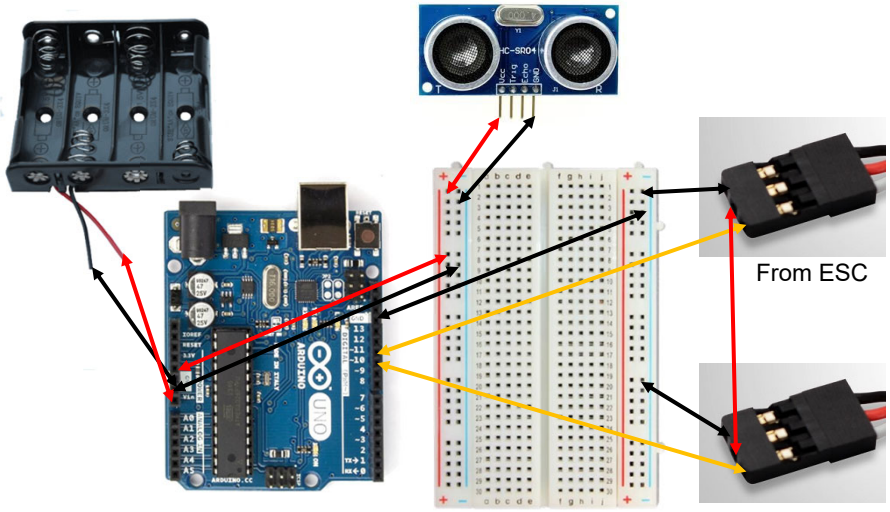
White – Signal
Red – Voltage (5V)
Black – Ground

- We want to make connections that allow Arduino to control steering servo (steering) and ESC (throttle) of the vehicle

Arduino's Power Supply

- To stabilize the readings from the sensors, you'll need a separate power system for sensors and Arduino.
- Use battery pack (4XAA batteries, 6V output) as the independent power source.
- Create circuit so that:
  - ESC supplies power to servo motor
  - Arduino, ESC, Servo, and battery pack have common ground
  - Connect the battery pack to the Vin port and GND port of Arduino
  - Connect the 5V output port of Arduino to the VCC ports of sensors

Wiring



Laptop safety

- **Do Not** connect laptop to Arduino while Arduino is supplied with battery pack.
- To observe the vehicle input on serial monitor, you need to connect your PC to the Arduino. So if you plug Arduino into your PC and connect Arduino to battery pack's power with VIN, it may cause damage to your laptop.
- So if you want to upload the codes or observe serial output while the vehicle is powered on, you have to remove the *Vin* line from your Arduino. Failing to do so may end you up with a dead USB port and maybe even a dead laptop.

Servo control with Arduino

- Arduino has a dedicated library for servo motors, called *Servo.h*. Usually, servos can be controlled between 0 and 180 here, with default being 90.
- For example, for the RC vehicle control, 90 would be the default where it does not steer and or move. Setting ESC at 100, it will move forward and at 80 it will move backward. Similarly, setting steering to 45 will turn left and 135 will turn right.

Vehicle control code – Initial part

Download basecode.ino from Canvas. Below is the code:

```
#include <Servo.h> //define the servo library

Servo ssm; //create an instance of the servo object called ssm
Servo esc; //create an instance of the servo object called esc

int steering=90, throttle=90; //defining global variables to use
later
```

Vehicle control code – Setup part

```
void setup() {
  //Serial.begin(9600); //start serial connection. Uncomment for PC

  ssm.attach(10);       //define that ssm is connected at pin 10

  esc.attach(11);       //define that esc is connected at pin 11
}
```


Vehicle control code – Loop code

```
void loop() {

// Add control logic here
steering=90;
throttle=90;

// Call the setVehicle function to set the vehicle steering and
throttle values
setVehicle(steering, throttle);
}
```

Vehicle control code – setVehicle function

```
//********************** Vehicle Control **********************//
//***************** Do not change below part *****************//
void setVehicle(int s, int v)
{
  s=min(max(0,s),180);  //saturate steering command
  v=min(max(70,v),110); //saturate velocity command
  ssm.write(s); //write steering value to steering servo
  esc.write(v); //write velocity value to the ESC unit
}
//***************** Do not change above part *****************//
```


Start and Stop the Vehicle

- After the code and wires are connected properly, you can start the vehicle.
- To start the vehicle, press the EZ-SET button on ESC.
- Do not hold for too long as it will enter calibration mode after holding the button for 2 seconds
- When the vehicle is properly started, the led light will first blink green, and then turn red.

Start and Stop the Vehicle

- To stop the vehicle, press and hold the EZ-SET button on ESC for 5 second. The ESC will be turned off when you release the button. Remember to detach Arduino Vin Pin.

- The *setVehicle function* is used to send throttle and steering signals to the vehicle, and limit the vehicle's throttle in a safe range. *Do not modify this part*.

Practice: Throttle Control

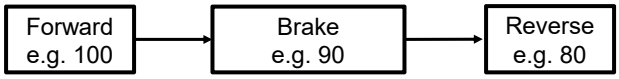
- **Task:** Download the base code from Canvas titled "basecode.ino". Create logic for the below challenges.

Challenge 1: Control the throttle

- Make sure the vehicle's rear wheels do not touch the ground while uploading the code.
- Drive forward by setting throttle to 100, keep 2 seconds.
- Brake the vehicle by setting throttle to 90, keep 2 seconds.
- Drive back by setting throttle to 80, keep 2 seconds.
- Brake the vehicle by setting throttle to 90, keep 2 seconds.
- Repeat the steps above.
- *Notice: Without braking, some vehicles may not be able to reverse, and sometimes even the motor will be stalled.*

Switch between driving Forward and Backward

- Change from forward to reverse



Example: Throttle Control

- Add this code to basecode.ino

```
void loop() {
  setVehicle(90, 100);
  delay(2000);
  setVehicle(90, 90);
  delay(2000);
  setVehicle(90, 80);
  delay(2000);
  setVehicle(90, 90);
  delay(2000);
}
```

Practice: Steering Control

- **Task:** Download the base code from Canvas titled "basecode.ino". Create logic for the below challenges.

Challenge 2: Control the steering

- Make sure the vehicle's rear wheels do not touch the ground.
- Steer to left by setting steering to 45, keep 2 seconds.
- Steer back to neutral by setting steering to 90, keep 2 seconds.
- Steer to right by setting steering to 135, keep 2 seconds.
- Steer back to neutral by setting steering to 90, keep 2 seconds.
- Repeat the steps above.

Example: Steering Control

- Add this code to basecode.ino

```
void loop() {
  setVehicle(45, 90);
  delay(2000);
  setVehicle(90, 90);
  delay(2000);
  setVehicle(135, 90);
  delay(2000);
  setVehicle(90, 90);
  delay(2000);
}
```

Option for High-Resolution Controls

- Normal control is in the range of 0 and 180. e.g., in steering, turn left for $0 \leq angle < 90$, turn right for $90 < angle \leq 180$, and keep neutral at 90
- $write()$ – accepts value in [0, 180] as argument
  - e.g, in steering, argument converted to time in μs in order to control duty cycle of PWM
- $writeMicroseconds()$ – directly accepts time in μs as argument
  - e.g., in steering, $1000 \leq time < 1500$ to turn left
  - e.g., in steering, $1500 < time \leq 2000$ to turn right
  - e.g., in steering, 1500 is for neutral
- Advantages of using $writeMicroseconds()$
  - higher resolution: ~ 1000 different speed inputs versus just 181
  - May provide smoother controls in some cases
- Possible to use both functions interchangeably
  - use $writeMicroseconds()$ or $write()$ based on your test needs
  - "*basecode_HRControl.ino*" in canvas is a demo for high-resolution controls

Calibration Code

- Download *Calibration.ino* from Canvas, use this code to calibrate your vehicle if your vehicle's motor doesn't move accordingly to your control code.
- **Steps for Calibrating the ESC:**

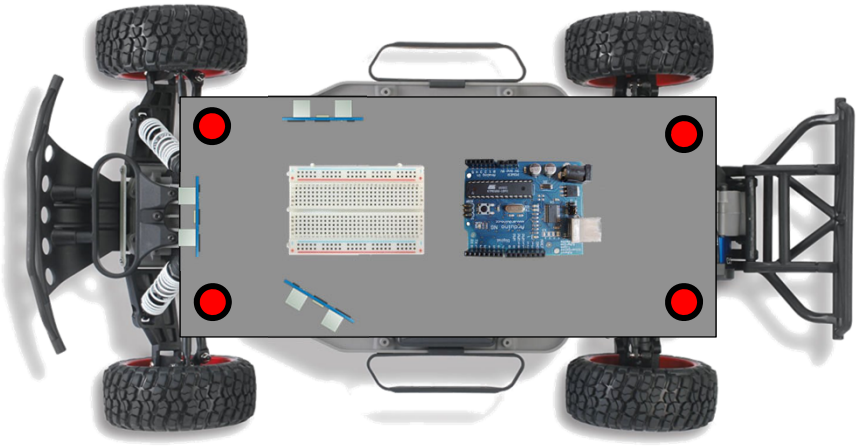
1. Upload this code to Arduino and connect Arduino to vehicle.
2. Immediately after code is uploaded, press and hold the button on ESC, the LED will first turn solid green and then solid red. Release the button after it turns solid red.
3. In calibration mode, the LED will blink red once, and after a while blink twice. Wait until the LED fast blink green and finally turns into solid red.
4. Turn vehicle off.
5. Calibration completed.

```
#include <Servo.h> //define the servo library
Servo esc; //define that esc is a servo

void setup() {
esc.attach(11);
pinMode(13,OUTPUT);
//******Vehicle Calibration******//
esc.write(90);  delay(10000); // Start during neutral
esc.write(180); delay(5000);  // Set the high limit
esc.write(0);   delay(5000);  // Set the low limit
esc.write(90);  delay(5000);  // Set neutral value
}

void loop() {
  digitalWrite(13,HIGH); // Calibration Over
  delay(1000);
  }
```

Building the fixture

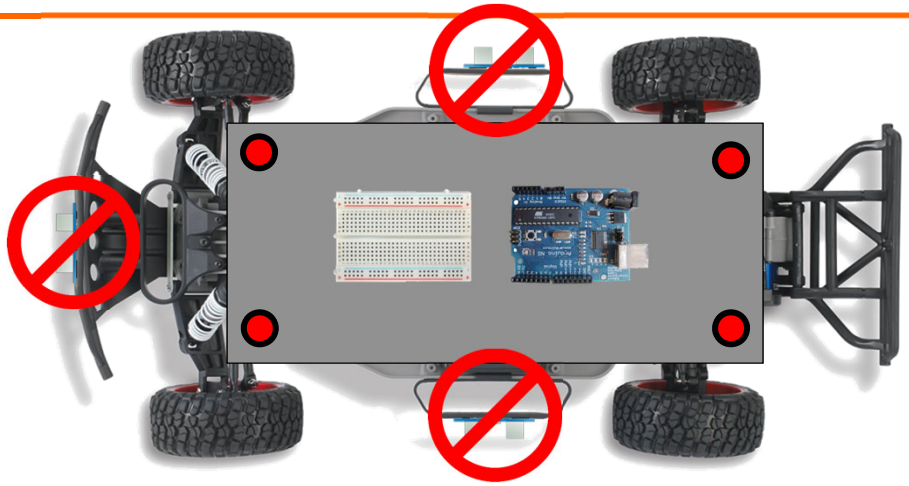
- Design your configuration of ultrasonic sensors to achieve the adaptive cruise control and lane keeping functions
- Design your controllers for throttle control and steering control based on ultrasonic sensing (you can use any feedback controllers)
- To setup your own design, you'll have to build a fixture that you can use to mount your Arduino, breadboard and the ultrasonic sensors.
- You need to make sure your design is safe and won't damage any equipment.

Setting up the RC car



One example of setup

Setting up the RC car



Do not place the sensors at the edge of the RC car. You may damage your sensors while collisions occur during testing.