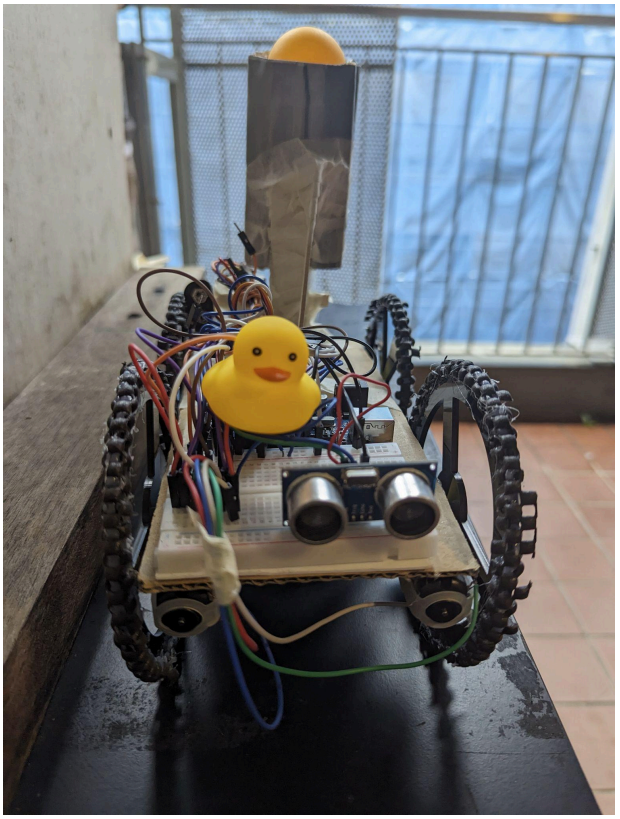
xThe design of a robot car that navigates an obstacle course



Abstract. We describe the design and implementation of an Arduino-based robot car that is able to navigate a simple obstacle course involving a bump and a ramp, as well as to launch a ping-pong ball over a short wall. We discuss our initial considerations when designing the car, our failed attempts, and the final design that was able to smoothly navigate the obstacle course. Details on how to replicate our final car design are provided in an appendix.

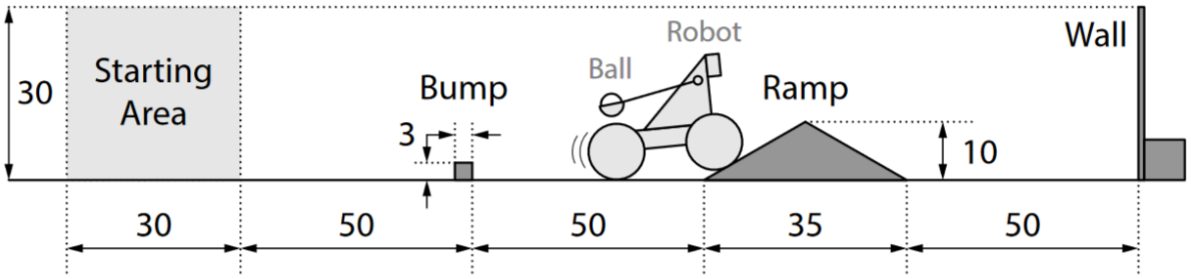
AY 2023/24, Semester 1

EG1311 B04 Team 6

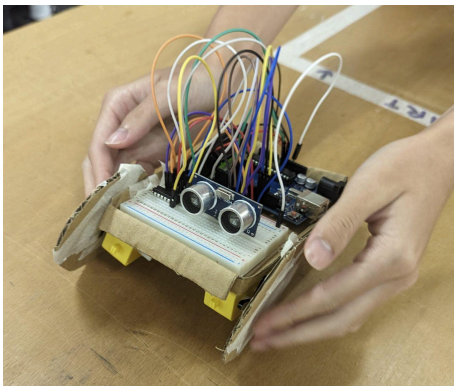
Kieran Ho
Claire Lee
Chang Jia Jun
Samantha Lau
Ho Boon Suan

Section 1. Overview of the design

First ideas. We were given a number of constraints when tasked to design this robot car. For example, the final car must fit in a cube of side length 30 cm, and the car must be constructed from the electronic components (an Arduino Uno, standard hobby gearmotors, a breadboard, etc.) and materials (cardboard, and a rather limited amount of acrylic we were allowed to laser cut) provided.



Our starting point was the template robot. We were given a simple design that was able to use the ultrasonic sensor to stop before hitting a wall, but was too simple to overcome any obstacles. It was from reflecting on why this template robot failed that we were able to work towards our final, successful design.



The template robot could not overcome the bump, and was stuck at the base of the ramp, unable to move up. It also had no ball launcher, so we would have to design that from scratch. Finally, the template car could not reverse, so we would have to figure that out too.

The bump and the ramp. We quickly determined that the problem here was a lack of friction / traction — the wheels were simply too small and flimsy! We decided to make bigger wheels, and to make them from acrylic instead of cardboard. Eventually, we also glued strips of non-slip mat along the wheel rims to improve traction. See Section 2 for details on the design process for our wheels.

Launching a ball over a wall. The next problem was designing a mechanism to launch a ping-pong ball over a 30 cm wall, controlled by a servomotor. We initially went with a "cannon" design, which was a cardboard cylinder making use of the energy stored in stretched rubber bands held by a stick attached to the servomotor coupler. This ended up being hard to control consistently, and so we settled on something like a catapult. See Section 3 for details on the design process for our ball launcher.

The base. In designing our first prototype, we decided to keep things simple, opting to use a 10 cm × 23 cm cardboard rectangle as the base / frame of our car. We made sure to orient the length of the board along the grain to increase the rigidity and strength of the base, and consequently the car. We chose cardboard over acrylic largely for practical reasons — we were given a very limited amount of acrylic, and cardboard was good enough to work with, and would also give us more freedom to make rapid changes and small fixes along the way.

Things fall apart. Of course, nothing ever goes right the first try. This subsection provides a selection of some of the hurdles we had to overcome in one way or another to make the car work.

"Surely this is a hardware issue, right?" We faced frustrating setbacks where the car would just stop moving. Some debugging revealed that one of the issues was caused by the ultrasonic sensor giving sporadic and erroneous readings. The code that went into this debugging process is discussed in the Appendix.

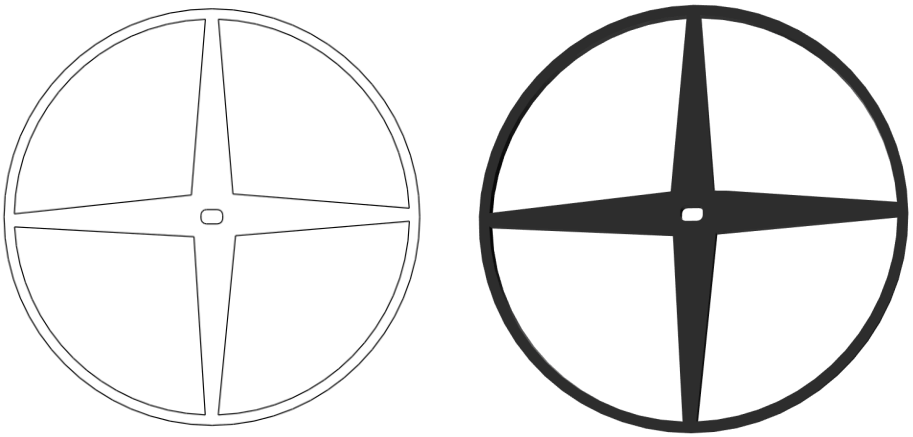
"Okay, the car is moving, but it isn't going straight." Sometimes the wires would get caught by the wheels, since they were all jumbled up and we did not do much cable management. We went with the cheap fix of taping the wires together to make sure they didn't move much.

"Wait, did that wheel just break?" There were a few instances of mild stress when we broke an acrylic wheel, either because it was too flimsy of a design, or we did not get the measurements right. The story of how we resolved all these wheel issues is told in Section 2.

"This code doesn't do what I want it to do…" This was also a regular source of frustration. Since we only have five pages in the report body, the reader is referred to the Appendix for some discussions of programming bugs we had to overcome.

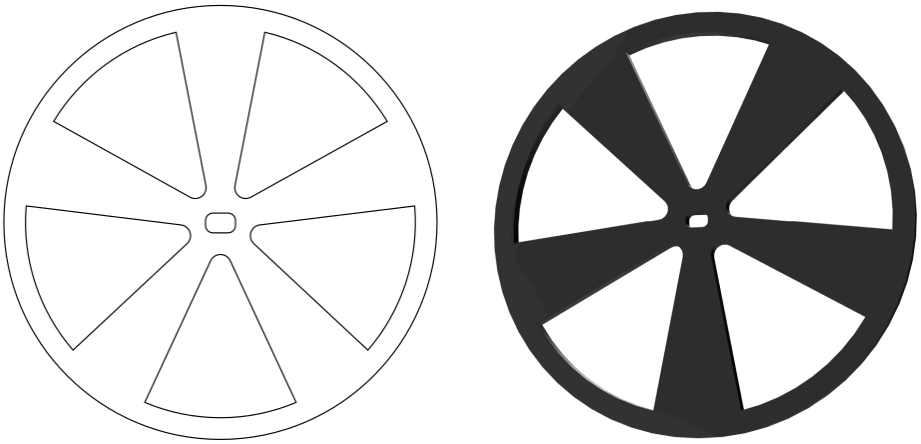
Section 2. Making a wheel

The task was to design a wheel that would help our robot car go over a short bump and a ramp. Naturally the task was to increase the traction of the wheel, and the easiest start to doing so is by making the wheels bigger. And so we made a simple wheel with four spokes in Fusion 360, of diameter 10 cm (in part because that would be the largest size such that we could cut a set of four wheels from the 20 cm × 20 cm acrylic boards we were given), with a hole in the center to connect to the hobby motor:

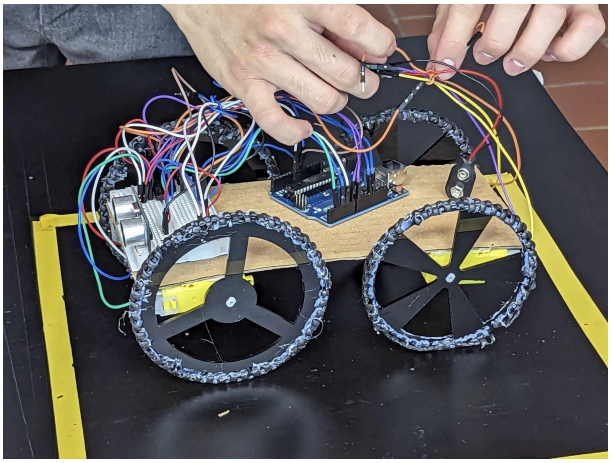


This design got the car moving, but we needed even more traction to overcome the bump. For this, we ended up carefully hot-gluing thin strips of anti-slip mat to the rims of the wheels, by analogy to how real cars have rubber tires that surround their rims. However, after several rounds of testing, even though this anti-slip wheel was able to overcome the obstacles most of the time, it simply proved to be too thin and flimsy, and we ended up breaking a few wheels. The rims were too thin, which led to poor stability and traction; and the spokes were also too thin and few to provide adequate support for the load.

The answer was naturally to use wheels with thicker rims, thicker and more spokes, and just more material:



This was not the end of our troubles, however. We did encounter some issues with fitting the newly-printed wheel designs to the hobby motors, which seemed to be due to wear on the motor causing the dimensions to be inexact. Since we also broke a wheel, we had no choice but to print another batch of wheels, which gave us an opportunity to make minor changes to the design. So the last design was printed with slightly larger holes for the motor to fit. Thankfully, it all worked out in the end.



The photo on the right features the old wheel design on the left and the new design on the right, both wheels including their hot-glued non-slip mat tires.

Section 3. On cannons and catapults

We were tasked to have the car launch a ping-pong ball when it reached the wall at the end of the obstacle course. The two main goals of such a launcher are (1) to be able to shoot far and high (at the very least to clear the 30 cm wall that is about 5–10 cm away from the car at time of launch); and (2) to do so consistently; meaning it fires when we want it to and doesn't fire when we don't want it to.

The cannon. Our first idea was the following design, inspired by cannons:

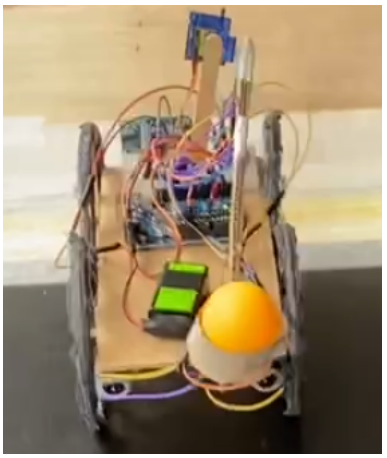



Since we were not provided with gunpowder, we decided that our cannon design would instead harness the elastic potential energy that is stored in stretched rubber bands. The rubber bands are held in tension by the small ledge and when released would propel the ball out of the cannon. The angle of the cannon would be adjusted as needed by changing where the supporting stick was attached to the body of the cannon (see the photos above) — perhaps slightly clunky, but it worked since we had a particularly narrow use case. As the servo is not strong enough to hold the rubber bands in tension, we made use of a "hair-trigger" mechanism for launching the ball, where the servomotor is able to flick an ice cream stick attached to the stretched rubber bands to set them off and consequently launch the ping-pong ball.

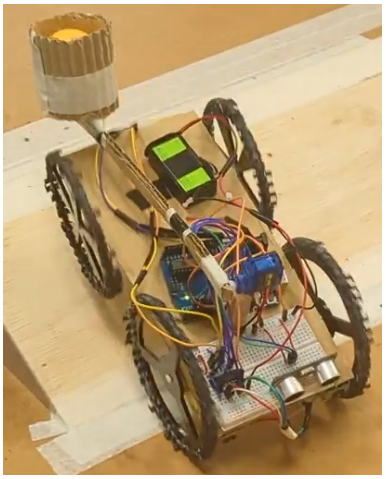
We also tested the design with various cannon lengths, and ended up shortening the cannon barrel slightly and increasing the number of rubber bands from one to two so as to increase the strength of the launch. Finally, we used a good amount of tape and sticks to make our cannon structure as rigid as possible.

However, the design did not work as well as we hoped, and many issues began to arise. Upon attaching the cannon to the base of the car, we found that it was simply too large, and it moved the center of gravity of the entire car upwards, to a point where the robot car became rather unstable and fell over easily when moving. The trigger was held in place too strongly, and was simply too unwieldy to be moved consistently by the servomotor, so the ball would just fail to launch now and then. Conversely, the sensitivity of the launch mechanism led to accidental triggering at times, where for example the ping-pong ball would suddenly be launched when the car was going over the ramp, presumably due to all the shaking and bumping going on. Finally, there was really no easy way to attach and angle the servo motor without complicating things.

The catapult. At this point, we had a car that could navigate decently, but no reliable ball launcher, and we were getting closer and closer to the deadline. Nearing the end, we decided that we just had to abandon our old design and work out something new. We settled on the classic and conceptually simple idea of a catapult, where the servomotor coupler would be attached to an ice-cream stick on the end closer to the base, and to a 'cup' holding the ball on the other end. The rotation of the servomotor arm would then directly launch the ball, and we wouldn't have to deal with any intermediate conversions (such as harnessing the elastic potential energy of stretched rubber bands).



While this design definitely worked better, the first iteration had its problems. The servomotor was placed some height above the base so as to give the catapult arm some space to pivot, but this attachment was rather ad hoc and was not particularly stable, using an ice-cream stick taped to the breadboard at the base. Also, the ball holder was rather flimsy, formed by a thin strip of cardboard formed into a cylinder (pictured top right). As such, the ball would fall out abruptly at times as the robot car moved over the ramp. Luckily this was a relatively minor issue, and we were able to fix it and make the ball launch consistently on command by building a sturdier holder for the ball, and attaching the servomotor-on-a-stick more rigidly to the base of the car (pictured bottom right).



Section 4. Wires and code

So far, we have discussed the physical design of the car, from the production of sturdy wheels that generate traction to the design of a reliable ball launcher. We have built a body, and it is time for us now to give it a mind.

Apart from just getting the system to work, there was the additional challenge of making the robot car reverse after reaching the end and launching the ping-pong ball over the wall, which would give our group an extra point in our final score.

To make the robot move forward and thereafter reverse, the polarity of potential difference across the motors had to be able to reverse. As the arduino is only able to supply positive voltages, we could not simply supply 5V to move it forward and change it to -5V to move it backward. Thus we used the H-bridge. In the template, one of the terminals is connected to ground, so it is always at 0V. As such, the potential difference across the motor to always be positive allowing the robot to only move in one direction.

In our design, we connected each terminal of the motor to a separate output pin of the H-bridge, so we can control the voltage of both terminals. The 2 output pins would work as a pair to power the motors. We supply 5V to input 1 and 0V to input 2 resulting in a potential difference of 5V allowing the robot to move forward, and we supply 0V to input 1 and 5V to input 2, resulting in a potential difference of -5V to make the robot reverse.

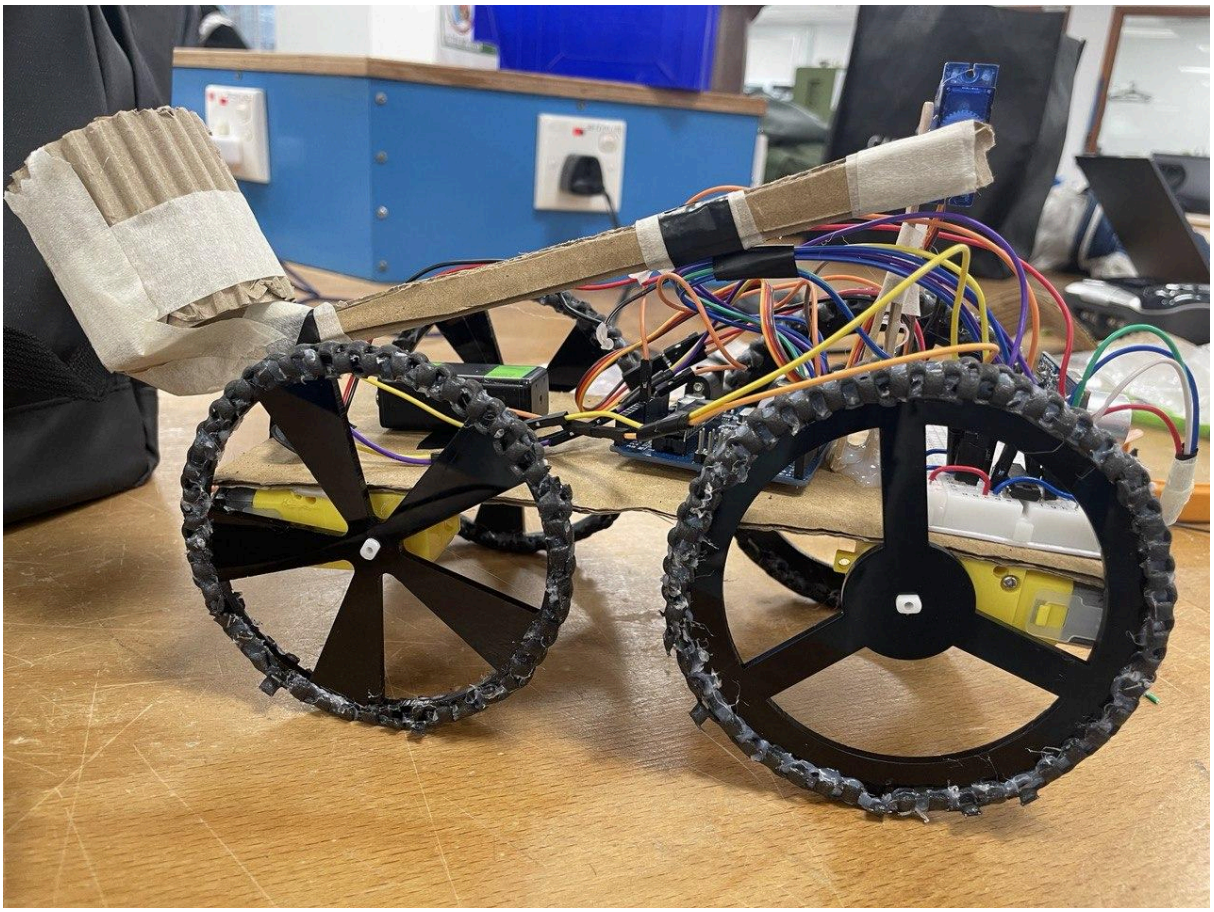
Moreover, our ultrasonic sensor presented issues too. The ultrasonic sensor outputs several erroneous values, causing the robot to detect that it is close to the wall when there is nothing in front of it. We solved this by tweaking the range of values our program considers, and implementing a double check to rule out random errors. (See the Appendix.)

Another problem we faced was that after the robot reversed away from the wall, the ultrasonic sensor would detect that it is no longer close to the wall and move forward once again. We considered using a very long delay to force the robot to reverse back to the start, however if the obstacle course was longer or our robot took a longer time to reach the start, it would start moving forward again before it reached the start. Thus we decided to use a boolean value to indicate whether it is moving forward or backward so that our robot will keep reversing after reaching the wall, no matter how long it takes.

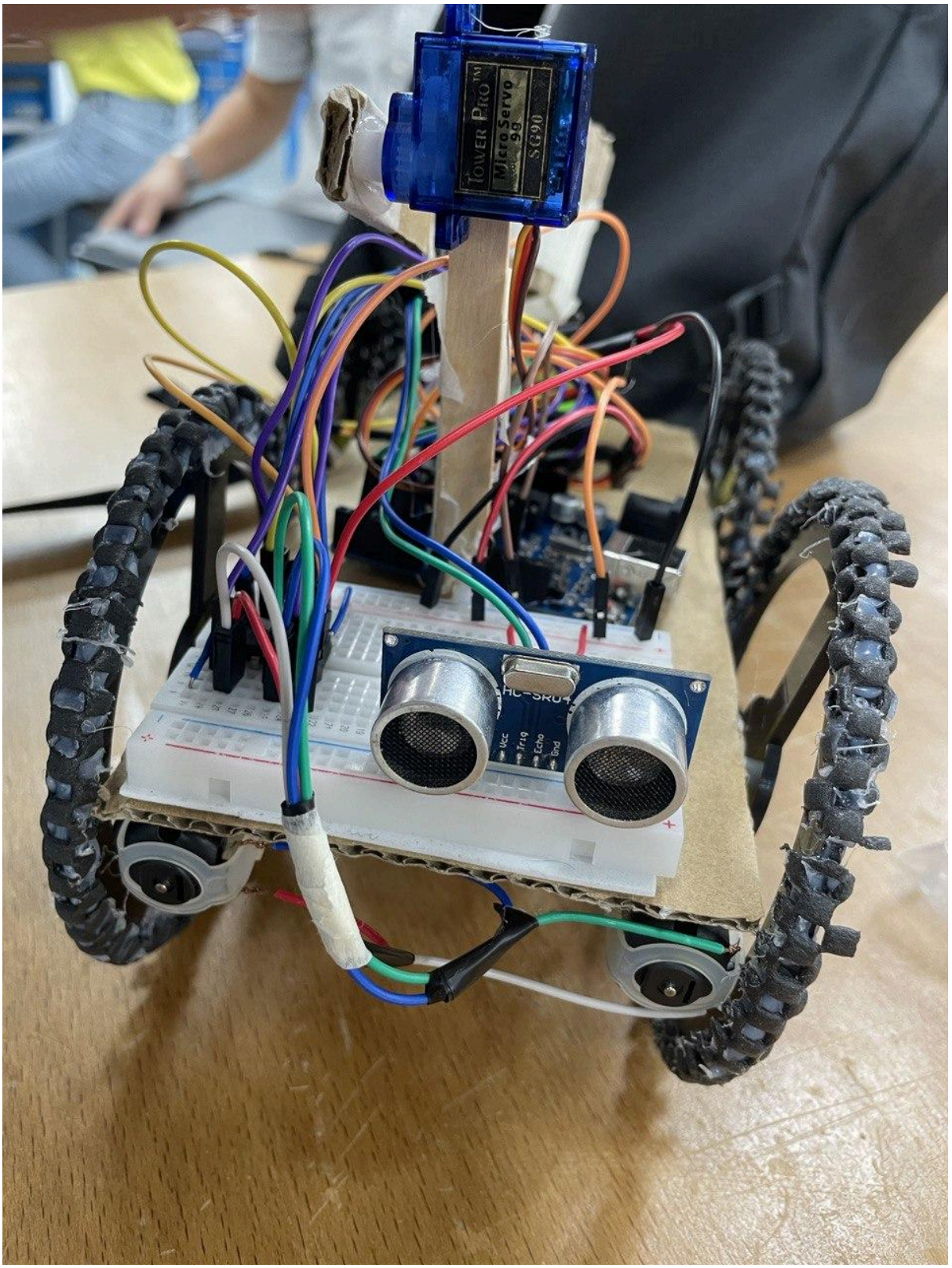
Lastly, as our front wheels are bigger than the back, when reversing, the robot would flip over. To tackle this, we reduced the speed of the robot reversing. We wired one of the input pins of the H-bridge to the analog pins of the arduino and used PWM to control the speed.

End of Report Body

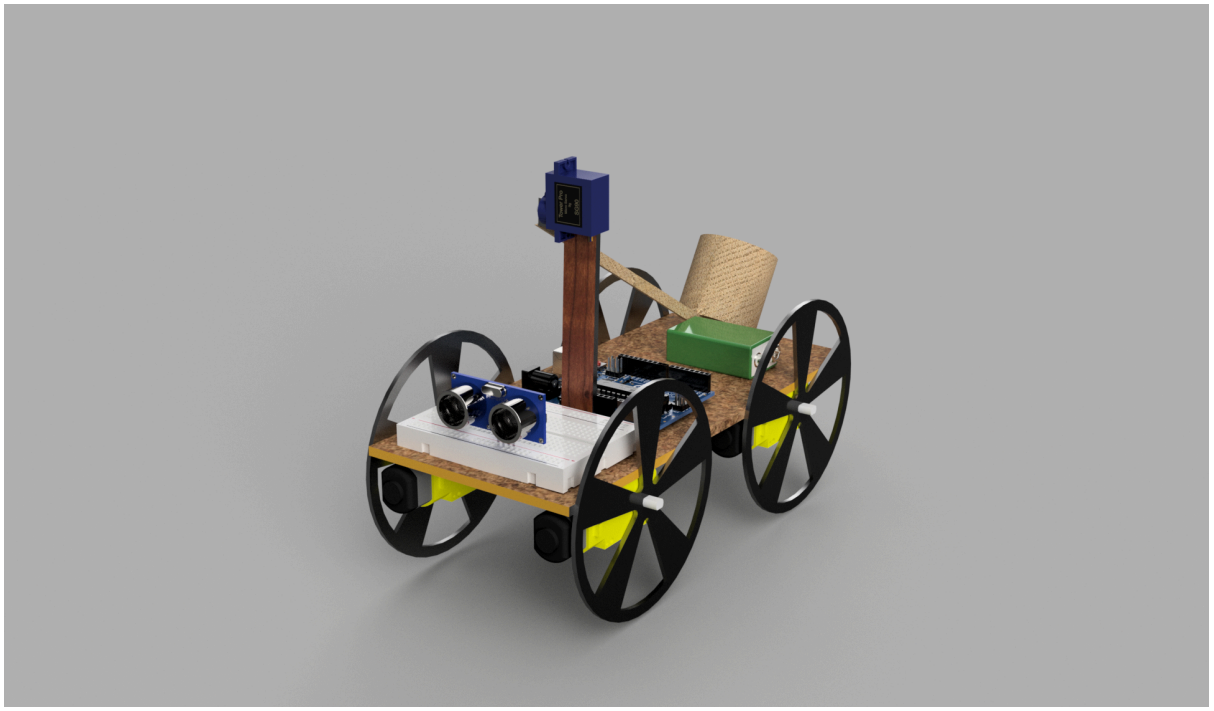
Appendix. CAD models, Arduino code, and everything else

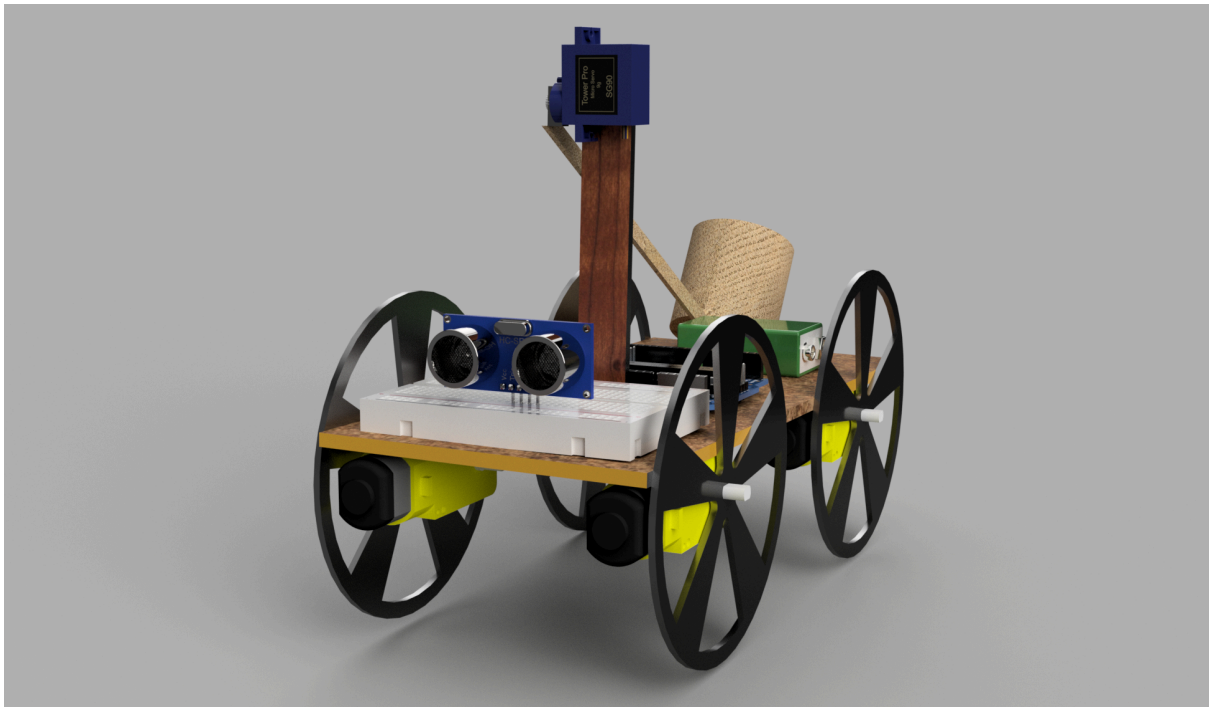


Photograph of the final physical robot.

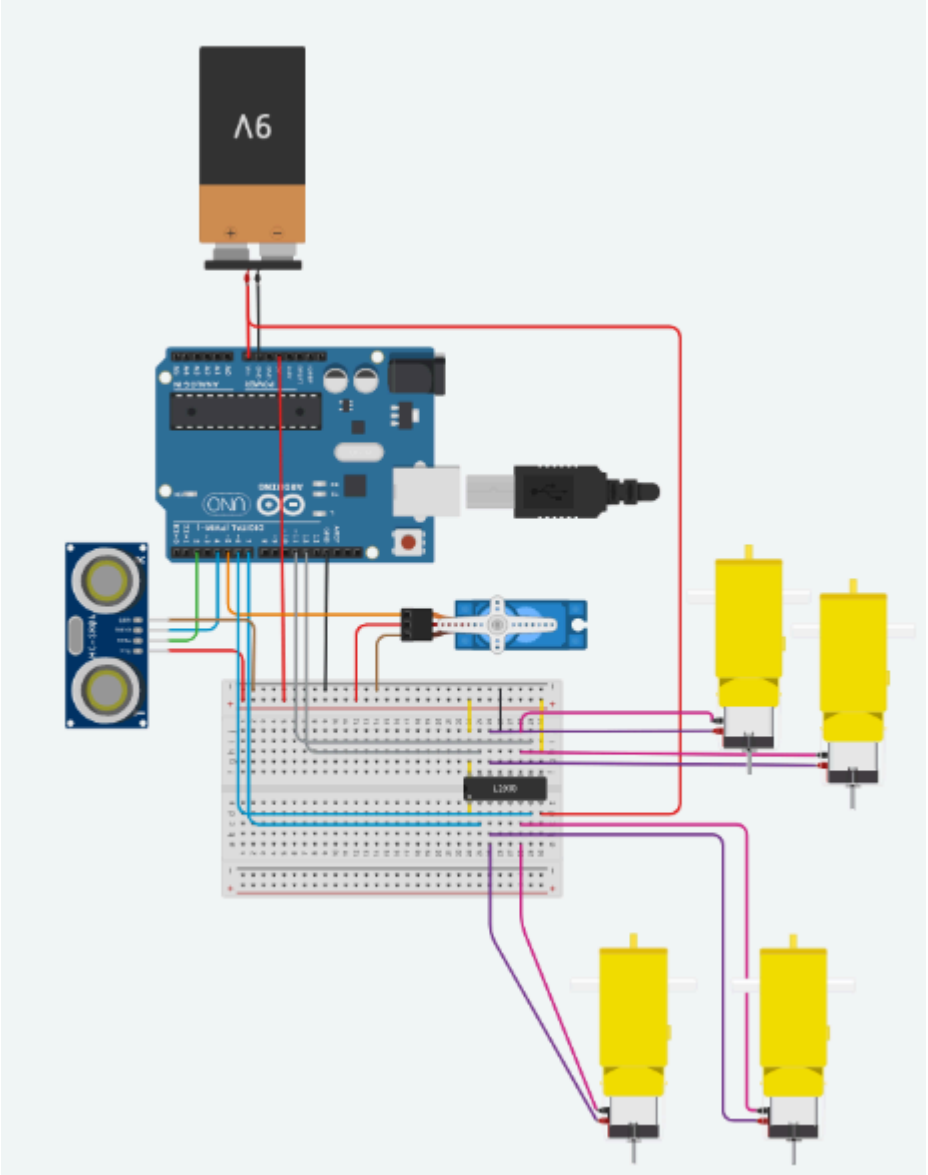


Another photograph of the final physical robot.



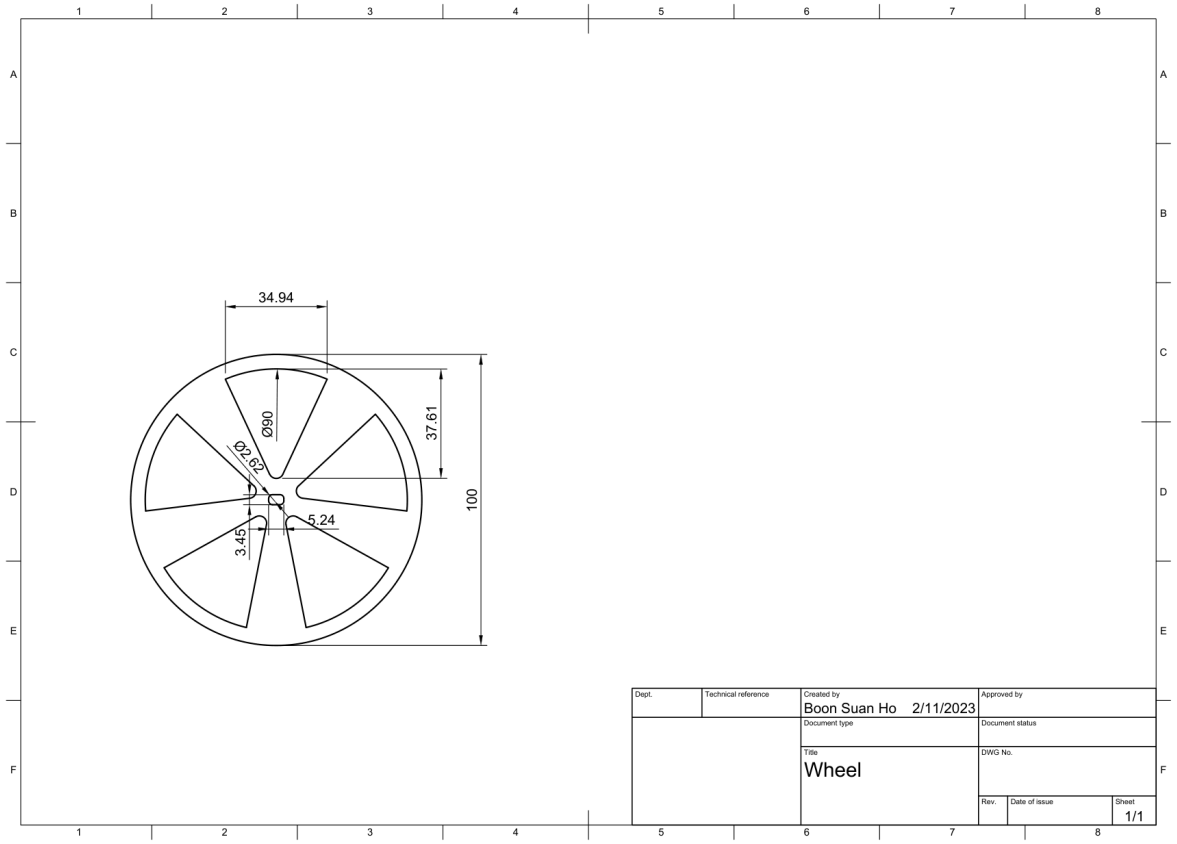


CAD renders in Fusion 360, of a model for the complete final robot.

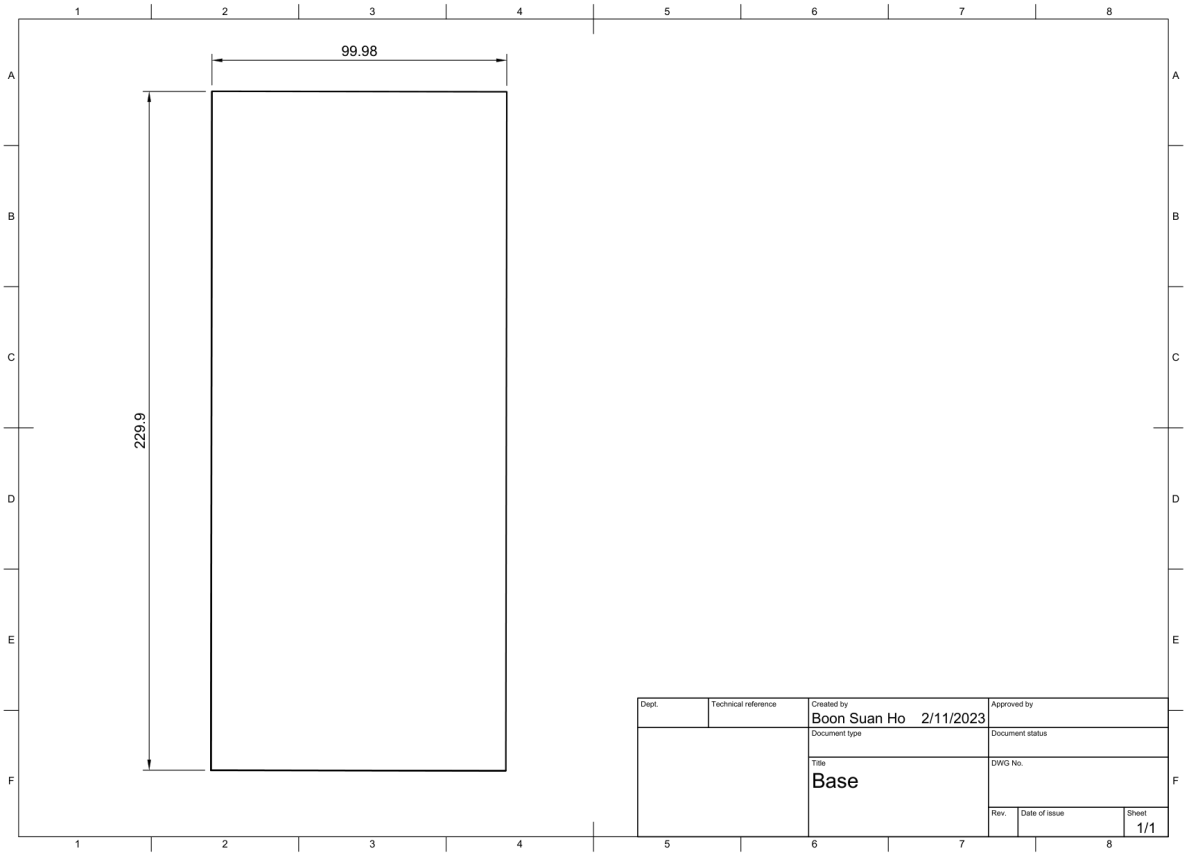


TinkerCAD diagram corresponding to the circuitry of the final design.

Acrylic wheel:



Cardboard base:



Fully-dimensioned 2D CAD drawings for each structural component made from sheet material used in the final design, depicting how it was cut.

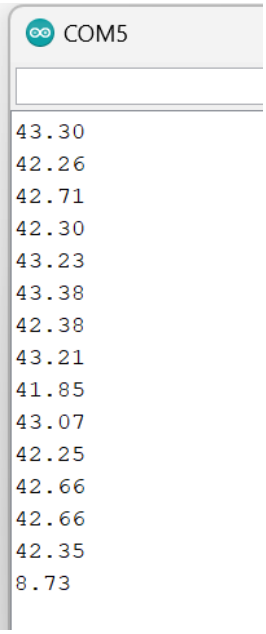
The following is a listing of the Arduino source code deployed on the final design of our robot car:

```
// C++ code
//
#include <Servo.h>

Servo servo;

int TRIG_PIN = 2;
int ECHO_PIN = 4;
int MOTOR_PIN1A = 12;
int MOTOR_PIN1B = 11;
int MOTOR_PIN2A = 7;
int MOTOR_PIN2B = 6;
int SERVO_PIN = 5;

bool forward = true;

void setup()
{
  // Motors
  pinMode(MOTOR_PIN1A, OUTPUT);
  pinMode(MOTOR_PIN2A, OUTPUT);
  pinMode(MOTOR_PIN1B, OUTPUT);
  pinMode(MOTOR_PIN2B, OUTPUT);

  // Sensor
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  Serial.begin(9600);

  // Servo
  servo.attach(SERVO_PIN, 660, 2400);
  servo.write(110);

  delay(5000);
}

float detect_distance(){
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  int microsecs = pulseIn(ECHO_PIN, HIGH);
  float cms = microsecs/58;
  Serial.println(cms);
  return cms;
}


// continued on the next page...
```

```
void loop()
{
  float cms = detect_distance();
  delay(60); // 60ms measurement cycle
  if ((cms > 2) && (cms < 10)){
    //check twice to make sure its not js some random error
    cms = detect_distance();

    if ((cms > 2) && (cms < 10)){
      // stop moving
      digitalWrite(MOTOR_PIN1A, LOW);
      digitalWrite(MOTOR_PIN2A, LOW);

      //release ball
      servo.write(20);
      delay(500);
      servo.write(110);
      // reverse
      forward = false;
    }
  }

  if (forward == true) {
    digitalWrite(MOTOR_PIN1A, HIGH);
    digitalWrite(MOTOR_PIN1B, LOW);
    digitalWrite(MOTOR_PIN2A, HIGH);
    digitalWrite(MOTOR_PIN2B, LOW);
  } else { // reverse
    analogWrite(MOTOR_PIN1B, 200);
    analogWrite(MOTOR_PIN2B, 200);
    delay(100000);
  }
}
```

Appendix discussion: Dealing with certain bugs

In this subsection of the Appendix, we discuss two bugs that really bugged us. The first is a programming bug where the ultrasonic sensor behaved in a rather strange way that we didn't really understand; the second is an issue with the motors that gave us a headache. No conclusions are reached.



Problem 1. Somewhere along the line, the ultrasonic sensor started to detect random distances sporadically. This really confused us.

For example, here was the serial output when there was nothing in front of the sensor. The ultrasonic sensor detects the correct distance at 42 and suddenly detects a distance of 8.73.

We wrote some code to debug the issue, by repeatedly querying the ultrasonic sensor and using serial communication to output the results.

```
// ultrasonic sensor code
float detect_distance(){
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
    int microsecs = pulseIn(ECHO_PIN, HIGH);
    float cms = microsecs / 58;
    Serial.println(cms);
    return cms;
}

void loop()
{
    float cms = detect_distance();
    delay (60);
// check twice to make sure it's not just some random error
    if ((cms < 2) && (cms < 10)){
        delay (1000);

        cms = detect_distance();
        if ((cms < 2) && (cms < 10)){

            // reverse
            forward = false;
        }
    }
    // the rest of the code for loop() has been omitted
}
```

The ultrasonic sensor tends to give a reading of 0.00 right at the start, presumably due to needing some time to initialize. As such, we test if `cms > 1` to ignore this case. Finally, after detecting that the distance from the sensor is less than 10 cm, we check again (using a second if statement) after a delay of 1 second to determine whether it really is a legitimate reading.

This debugging code was not perfect however… Running it, we kept getting problematic readings with value 2.00. From the data sheet for the ultrasonic sensor (HC-SR04), we found that the working range of the ultrasonic sensor was between 2cm and 4m. Thus the 2.00 reading was likely due to the object not being in the working range of the ultrasonic sensor. As such, our program needs to ignore values that are 2.00 and below. Thus we increased the minimum value from 1cm to 2cm by changing `cms > 1` to `cms > 2` (highlighted in yellow). Also, we learned from the data sheet that the measurement cycle for the sensor is 60 ms, so we added the line `delay(60);` (highlighted in red).

These changes allowed our ultrasonic sensor to work much more reliably, allowing our robot to detect the wall much more accurately.

Problem 2. The motors were not spinning quickly enough. This puzzled us to no end. It turns out that there were three issues going on at the same time: Firstly, the electrical connection between the motors and the Arduino was done poorly — we did not connect the wires well, and over time the connection weakened because of this. So we had to remove our existing wiring, strip the wire again, and properly attach the end of the wire to the motor. Similarly in cases where we had to extend a wire by connecting it to another wire, we went back to redo all the connections. Secondly, the 6-volt battery pack was simply not enough power for our car setup, which used four motors attached to one wheel each. So the solution in this case was straightforward — switch over to using a 9-volt battery. There was the worry that our 9V battery would run out of power quickly (in 15 min according to the lecture video). However, after testing, the 9-volt battery had sufficient charge for multiple runs so we used that instead. Lastly, during test runs, our robot's front left wheel did not have enough power to push the robot over the ramp. As this was a mechanical issue, we replaced the motor with a new one.

The End

The final frontier — the sixth group member, being the only one of us small enough to ride the robot car, navigates the vehicle to its final launch, and ushers glory.