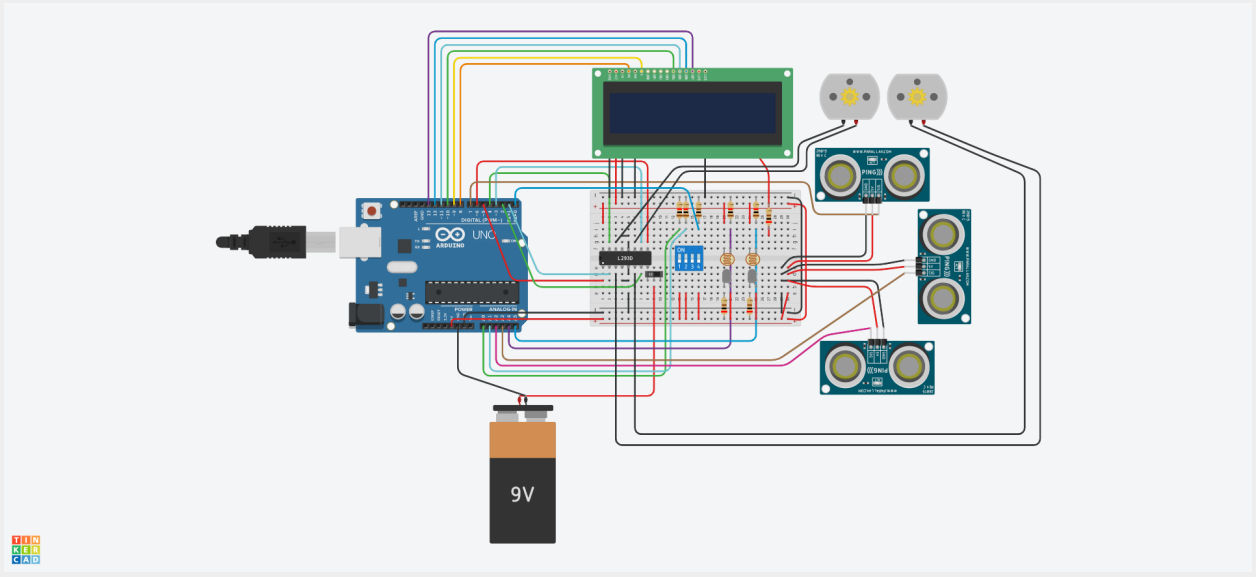
SumoBot TinkerCAD Submission

YouTube link to testing video (backup for file submission): https://youtu.be/RRb-iBDXoa8

Link to circuit: https://www.tinkercad.com/things/3VDqxxp8A2C



Design Strategies

Weight distribution:

- Longitudinal axis: weight should be placed as much as practicable on the rear wheels, but with the center of mass a few cm in front of the axle. This provides more downforce on the wheels (gravity), increasing maximum traction. Putting the center of mass in front of the axle retains rotational stability by increasing rotational inertia (so the car doesn't flip over around on the rear axle). Keeping weight towards the rear wheels also reduces inertia when trying to turn the car, since the car rotates at the back wheels and has to push the front.
- Vertical axis: weight should be distributed as low as practicable to lower the center of mass. This increases rotational stability so the car doesn't flip over on impact with the opponent sumobot.
- Lateral axis: weight should be distributed symmetrically, with center of mass close to the centerline. This just makes sure that lateral weight distribution has minimal impact on the robot's handling.

Wheel position: rear wheel drive (caster at front). Rear-wheel drive encourages oversteer as opposed to understeer, which makes the turning of the robot more responsive at some cost to stability.
Center of gravity: to calculate the center of gravity (CoG) of a group of i objects in one axis, use the following formula: $\bar{x} = \frac{\sum_{i=1}^{n} m_i x_i}{\sum_{i=1}^{n} m_i}$. That is, the position of the CoG is equal to the sum of the mass times position of each object, divided by the sum of masses of all objects. To find the CoG in 3 dimensions (x, y, z), either calculate the position of the CoG in the x, y, and z axes separately (split into components), or treat the positions x_i as algebraic vectors.
If an object itself takes up space, its position should be its own CoG, which can be approximated as the center of the object in most cases.

Height of center of gravity:
List of items of significant weight: estimated mass and estimated y-positions (of their center) from the floor, according to the SketchUp model:

- Main wooden frame: 1100 g, 5.75 cm
- Arduino Uno: 25 g, 2.30 cm
- Breadboard: 50 g, 1.95 cm
- 9V battery: 45 g, 2.75 cm
- x2 Motors: 100g, 3.00 cm
- x3 Ultrasonic distance sensors: 20 g, 13.1 cm
- Raised mounting for ultrasonic distance sensors: 130 g, 11.5 cm
- x2 Wheels: 10 g, 3.00 cm

Total mass: 1630 g
Sum of all objects mass*position: 9544.75 g*cm
Center of gravity = 9544.75 g*cm / 1630 g = 5.85 cm

Instruction Manual / Description of Testing Video

Before testing, make sure that the **LDRs are set to high light** to prevent the immediate activation of “near edge” mode. Also make sure that the DIP switches are all set to off, and that the motor slide switch is set to the left (on).

The order in which the features are listed is roughly the order in which they are tested in the demonstration video.

Functionality	Goal	Testing Method
LCD	The LCD should continuously show the distances from the 3 distance sensors, the analog reading from the LDRs, the current system mode, and the initial turning angle.	Ensure that the sensor readings (distance sensors and LDRs) are updated when performing tests. The updates may take up to half a second to appear.

	The LCD should be updated around once every 100 ms. The readings from the 3 distance sensors in cm should be shown in a triangle on the left side. The position of each reading should correspond to the position of the distance sensor (left, forwards, right). The analog readings from the pins reading from the 2 LDRs should be shown in the middle. The left LDR reading should be on top, and the right LDR reading should be on the bottom. The current system mode should be displayed in the top-right as a 3-letter code (OFF for off, SRC for search, TRK for facing/track, and EDG for near edge). The initial turning angle should be displayed in the bottom-right once the robot is switched on using switch 4.	
Motor Control	The L293D half-bridge motor control chip should be used to control the speed and direction of the left and right motors independently.	Ensure that the motor rpms are consistent with the stated direction and speed required for the tests below. Full forwards is roughly 16000 rpm.
DIP switch and Initial turning	The DIP switch should be used to control the initial turning angle, then switch on (and off) the robot. Switch 4 is the on/off switch; flipping it to on should execute the initial turning and activate the motors. The binary representations of switches 1 and 2 **at the time switch 4 is flipped** should correspond to initial turning angles as follows: (00, 0°), (01, 90° CW), (10, 180°), (11, 90° CCW). The initial turning angle should be displayed at the bottom right of the LCD, with 90° CCW being represented as 270° instead.	Before starting the test, ensure that both LDRs are set to high light level and that all DIP switches are off. Set switches 1 and 2 to (OFF, ON), then set switch 4 to ON. The LCD should display the system mode INI in the top-right and display an initial turning angle of 090 in the bottom-right. The left motor rpm should be positive and the right motor rpm should be negative for around 1 second (turn 90° CW). After 1 second, the displayed system mode should change to either TRK or SCH, as should the motor speeds. Set switch 4 to OFF. The system mode should return to OFF and the motors should stop spinning. Repeat this test, but set switches 1 and

		2 to (ON, OFF) instead. The initial turning angle displayed should be 180. The left motor rpm should be positive and the right motor rpm should be negative for around 2 seconds (turn 180° CW). Repeat this test, but set switches 1 and 2 to (ON, ON) instead. The initial turning angle displayed should be 270. The left motor rpm should be negative and the right motor rpm should be positive for around 1 second (turn 90° CCW). Repeat this test, but set switches 1 and 2 to (OFF, OFF) instead. The initial turning angle displayed should be 000. The system mode should immediately change to SRC or TRK, with no initial turning (turn 0°).
Mode: **Search** (SEARCH, SCH)	If no distance sensor detects an object, the robot should rotate in place and scan for the opponent.	For all 3 distance sensors, move all 3 objects out of range. On the LCD, the three distances should display as 335. The system mode should be SCH. The left motor rpm should be positive (~5000 rpm) and the right motor rpm should be negative (~-5000 rpm) (continuous turn CW).
Mode: **Facing** (TRACK, TRK) Functionality of ultrasonic distance sensors.	The LCD should constantly update with the correct distance from each ultrasonic distance sensor (UDS). For this circuit, the LCD should display 335 if no object is detected. (This is likely a TinkerCAD-specific behaviour: if pulseIn times out normally, then the distance returned is 0. I hypothesize that TinkerCAD forces a pulse return duration consistent with 335 cm. My program code would be different for real-life testing.) In the program logic, a distance sensor is considered to have an object in range if the measured distance is less than 300 cm. If the opponent is detected in at least one ultrasonic distance sensor, the robot should turn to face it and continue moving	When doing this test, the LCD should update with the correct distance from each distance sensor, or 335 if no object is detected. Start with all 3 distance sensors being out of range of their object.The system mode should be SCH. Move an object into range (2 cm < distance < 300 cm) of the left distance sensor. The system mode should switch to TRK. Left motor speed should be ~6000 rpm and right motor speed should be ~16000 rpm. Move an object into range of the left and forwards distance sensors. Left motor speed should be ~11000 rpm and right motor speed should be ~16000 rpm. Move an object into range of the

	towards it. The three ultrasonic distance sensors (UDSs) will be referred to as L (left), F (forwards), and R (right). The logic should be as follows:
	
	If (all three UDSs detect OR only F detects): full speed forwards
	If (only L and F detect): slight left turn
	If (only R and F detect): slight right turn
	If (only L detect): hard left turn
	If (only R detect): hard right turn
		forwards distance sensor. Left and right motor speed should be ~16000 rpm.
		
		Move an object into range of the forwards and right distance sensors. Left motor speed should be ~16000 rpm and right motor speed should be ~11000 rpm.
		
		Move an object into range of the left, forwards, and right distance sensors. Left and right motor speed should be ~16000 rpm.
		
		Move an object into range of the right distance sensor. Left motor speed should be ~16000 rpm and right motor speed should be ~6000 rpm.
		
		Move the objects out of range of all 3 distance sensors again. The system mode should switch back to SCH.

Mode: **Near edge** (NEAR_EDGE, EDG) Functionality of LDRs.	The LCD should constantly update with the analog reading from each LDR. For this circuit, the reading should be between 6 at minimum light and 679 at maximum light. In program code, a reading of less than 550 (around half light) should be considered as detecting the edge of the ring. If at least one LDR detects that the robot has touched the edge, it should stop, turn away from the edge, go forwards a bit, then return to search mode. If only the left LDR detects (approaching edge from left), the turn should be 135 degrees right. If only the right LDR detects (approaching edge from right), the turn should be 135 degrees left. If both LDRs detect (approaching edge nearly perpendicular), a turn in either direction is acceptable.	Start with the system mode in SCH (all 3 distance sensors out of range). As tested before, the left motor rpm should be positive (~5000 rpm) and the right motor rpm should be positive (~-5000 rpm). Set the left LDR to low light level (beyond halfway point). The system mode should switch to EDG and the LDR reading will update; at this point return the LDR to high light level (else near edge mode will activate repeatedly). The following sequence of events should take place: -Both motors will go to 0 rpm (full stop) -For 1.5 seconds, left motor speed is ~6000 rpm, right motor speed is ~-6000 rpm (corrective turn 135° CW/right) -For 1.5 seconds, motor speed is ~12000 rpm (forwards) -System mode reverts to SCH and motor speeds update accordingly. On the LCD, LDR and UDS readings will start updating again. Set the right LDR to low light level

		(beyond halfway point). The system mode should switch to EDG and the LDR reading will update; at this point return the LDR to high light level (else near edge mode will activate repeatedly). The following sequence of events should take place: -Both motors will go to 0 rpm (full stop) -For 1.5 seconds, left motor speed is ~-6000 rpm, right motor speed is ~6000 rpm (corrective turn 135° CCW/left) -For 1.5 seconds, motor speed is ~12000 rpm (forwards) -System mode reverts to SCH and motor speeds update accordingly. On the LCD, LDR and UDS readings will start updating again. If both LDRs are set to low level, whether the robot will make its corrective turn to the left or right depends on which LDR goes low first. If both go low at the same time, the corrective turn will be to the right, as the left LDR is checked first in program logic.
Motor power slide switch	The motor slide switch should be able to cut off and restore power to the motor at any time via the motor control chip. If the slide switch is set to off, the motors should shut off. If the slide switch is set to on, the motors should resume functioning normally	Flip the slide switch to the off (right) position. The motors should shut off and go to 0 rpm. Flip the slide switch back to the on (left) position. The motors should start spinning again, with their rpm determined by the current system mode.

Code

```
//Alexander Li
//AL TER4M SumoBot
//Version: 2021-05-13
//Course: TEJ4M1
//Teacher: Mr. Wong
//Function: sumo-wrestling bot that aims to push an opponent bot out of a ring

#include <LiquidCrystal.h>
```

```
//Pin variables

//Pin variables
int on_off_pin = 0;
int input1_pin = 1;
int input2_pin = 2;
int input3_pin = 3;
int input4_pin = 4;
int enable12_pin = 5;     //RIGHT motor
int enable34_pin = 6;     //LEFT motor
int uds_left_pin = 7;
int uds_right_pin = A2;
int uds_center_pin = A3;
LiquidCrystal lcd(8, 9, 10, 11, 12, 13);
int ldrLeft_pin = A4;
int ldrRight_pin = A5;
int dip1_pin = A0;
int dip2_pin = A1;

//Voltage reading for LDRs (drop across second resistor)
int ldrLeft_state = 0, ldrRight_state = 0;

//Distance readings from UDSs
long udsLeft_cm = 0, udsRight_cm = 0, udsCenter_cm = 0;
const int MAX_DIST = 300;

//LCD
long updateLCDTime = 0;

//System mode
const int OFF = 0, INITIAL = 1, SEARCH = 2, TRACK = 3, NEAR_EDGE = 4;
int systemMode = OFF; //0 for idle, 1 for search, 2 for tracking, 3 for facing, 4 for
near edge
int previousSystemMode = OFF;

//Motor variables
int defaultSpeed = 0;

//Adjust these for turning
int motorLeftSpeed = 0, motorRightSpeed = 0;  //Speeds from -255 to +255
long turnFinishTime = -1; //-1 if not in use, positive time otherwise. In
milliseconds.


void setup() {
      lcd.begin(16, 2);

      //Pin modes
```

```
	pinMode(on_off_pin, INPUT);
	pinMode(input1_pin, OUTPUT);
	pinMode(input2_pin, OUTPUT);
	pinMode(input3_pin, OUTPUT);
	pinMode(input4_pin, OUTPUT);
	pinMode(enable12_pin, OUTPUT);
	pinMode(enable34_pin, OUTPUT);
	pinMode(ldrLeft_pin, INPUT);
	pinMode(ldrRight_pin, INPUT);
	pinMode(dip1_pin, INPUT);
	pinMode(dip2_pin, INPUT);

	//Write enable pins to L293D to high. (Power controlled by duty cycle)
	digitalWrite(enable12_pin, HIGH);
	digitalWrite(enable34_pin, HIGH);

	Serial.end();
}

void loop() {
	//On/Off switch
	if (digitalRead(on_off_pin) == LOW){
		systemMode = OFF;
	previousSystemMode = systemMode;
		defaultSpeed = 0;
		motorLeftSpeed = 0;
		motorRightSpeed = 0;
		updateMotorStates();
		return;	//skips to next run of loop()
	}
	if (previousSystemMode == OFF && digitalRead(on_off_pin) == HIGH){
//Robot turned on by switch 4
	systemMode = INITIAL;
		initialTurning();
	}

	//Update LDRs
	ldrLeft_state = analogRead(ldrLeft_pin);
	ldrRight_state = analogRead(ldrRight_pin);
	//Update LCDs
	udsLeft_cm = updateUDS(uds_left_pin)/58.2377;
	udsRight_cm = updateUDS(uds_right_pin)/58.2377;
	udsCenter_cm = updateUDS(uds_center_pin)/58.2377;

	//MODE: NEAR EDGE
	//NEAR_EDGE will not be deactivated until escape maneuver has been performed.
Revert to search.
	if (ldrLeft_state < 550 || ldrRight_state < 550){
```

```
            systemMode = NEAR_EDGE;
            //Hard stop: high impedence by setting enable12_pin and enable34_pin to
0 duty cycle
            motorLeftSpeed = 0;
            motorRightSpeed = 0;
            updateMotorStates();
            delay(200);

            //Corrective turn
            if (ldrLeft_state < 550){ //Turn CW (right) 135 degrees
                  motorLeftSpeed = 100;
                  motorRightSpeed = -100;
                  turnFinishTime = millis() + 1500;
            }
            else { //Turn CCW (left) 135 degrees
                  motorLeftSpeed = -100;
                  motorRightSpeed = 100;
                  turnFinishTime = millis() + 1500;
            }

            updateMotorStates();
            delay(turnFinishTime - millis()); //Maintain current motor speeds for
duration of turn
            //Travel forwards for 1.5 seconds to get away from edge
            motorLeftSpeed = 200;
            motorRightSpeed = 200;
            turnFinishTime = millis() + 1500;
            updateMotorStates();
            delay(turnFinishTime - millis());
            //Reset system mode
            turnFinishTime = -1;
            systemMode = SEARCH;
      }
      //MODE: FACING (TRACK)
      //In TinkerCAD, if object is not in range, returned distance is 335, not 0.
Normally, timeout on pulseIn would give 0.
      //Parallax Ping))) (#28015) data sheet:
https://www.digikey.com/htmldatasheets/production/1253021/0/0/1/28015.html
      //Min accurate range is 2 cm. By setting UDS back by at least 2cm from edge of
car, opposing bot cannot get under sensor range
      //Max accurate range is 3 m (300 cm). All distances beyond will be
disregarded.
      else if (udsLeft_cm < 300 || udsCenter_cm < 300 || udsRight_cm < 300){
            systemMode = TRACK;
            defaultSpeed = 255;
            //Opponent sumo bot detected in(F) or (L, F, R) sensors: full forwards
            if (udsCenter_cm < 300 && ((udsLeft_cm < 300 && udsRight_cm < 300) ||
(udsLeft_cm >= 300 && udsRight_cm >= 300)) ){
```

```
                motorLeftSpeed = defaultSpeed;
                motorRightSpeed = defaultSpeed;
        }
    //Detected in (L, F): turn slight left
        else if (udsLeft_cm < 300 && udsCenter_cm < 300){
                motorLeftSpeed = defaultSpeed - 70;
                motorRightSpeed = defaultSpeed;
        }
    //Detected in (R, F): turn slight right
        else if (udsRight_cm < 300 && udsCenter_cm < 300){
                motorLeftSpeed = defaultSpeed;
                motorRightSpeed = defaultSpeed - 70;
        }
    //Detected in (L): turn hard left
        else if (udsLeft_cm < 300){
                motorLeftSpeed = defaultSpeed - 150;
                motorRightSpeed = defaultSpeed;
        }
    //Detected in (R): turn hard right
        else if (udsRight_cm < 300){
                motorLeftSpeed = defaultSpeed;
                motorRightSpeed = defaultSpeed - 150;
        }
    }
    //MODE: SEARCH
    //Continuously rotate CW at slower speed to scan for opponent robot
    else{
        systemMode = SEARCH;
        defaultSpeed = 0;
        motorLeftSpeed = 80;
        motorRightSpeed = -80;
    }

    //Update speeds and direction of each motor
    updateMotorStates();
    previousSystemMode = systemMode;
}

void updateLCD(){
    //Erase display
    lcd.setCursor(0,0);
    lcd.print("                ");
    lcd.setCursor(0,1);
    lcd.print("             ");
    //Print LDR analog readings in middle
    lcd.setCursor(8, 0);
    lcd.print(ldrLeft_state);
    lcd.setCursor(8, 1);
```

```
      lcd.print(ldrRight_state);
      //Print distance sensor distance readings in left
      lcd.setCursor(0,1);
      lcd.print(udsLeft_cm);
      lcd.setCursor(4,1);
      lcd.print(udsRight_cm);
      lcd.setCursor(2,0);
      lcd.print(udsCenter_cm);
      lcd.setCursor(13, 0);
      //Print system mode in top-right
    if (systemMode == OFF){
      lcd.print("OFF");
    }
    else if (systemMode == INITIAL){
      lcd.print("INI");
    }
    else if (systemMode == SEARCH){
      lcd.print("SCH");
    }
    else if (systemMode == TRACK){
      lcd.print("TRK");
    }
    else if (systemMode == NEAR_EDGE){
      lcd.print("EDG");
    }
}

//Pulse the ultrasonic distance sensor to get a return time reading to convert to
distance
long updateUDS(int pin){
      pinMode(pin, OUTPUT);
      digitalWrite(pin, LOW);
      delayMicroseconds(2);
      digitalWrite(pin, HIGH);
      delayMicroseconds(10);
      digitalWrite(pin, LOW);
      pinMode(pin, INPUT);
      return pulseIn(pin, HIGH, 30000);
}

//Motor methods.

/* NOT IMPLEMENTED
//Sets the required motor speeds and turn time required to complete a fixed turn
void initiateTurn(int turnDegrees, int turnSpeed){
  //states: turn time has passed, turn time is ongoing, turn time is not in use
  //Add method to quickly flash maximum power: analogWrite(255) to both motors to
overcome static friction
```

```
}
*/

//Update inputs to the L293D motor control chip to set motor speeds and directions
//Using analogWrite to control the duty cycle of the signal to the Enable pins on the
L293D controls effective power to motors
//by quickly enabling/disabling the chip for a certain fraction of each cycle
void updateMotorStates(){
	//Right motor direction and speed
	if (motorRightSpeed < 0){
		digitalWrite(input1_pin, LOW);
		digitalWrite(input2_pin, HIGH);
	}
	else{
		digitalWrite(input1_pin, HIGH);
		digitalWrite(input2_pin, LOW);
	}
	analogWrite(enable12_pin, abs(motorRightSpeed));
	//Left motor direction and speed
	if (motorLeftSpeed < 0){
		digitalWrite(input3_pin, LOW);
		digitalWrite(input4_pin, HIGH);
	}
	else{
		digitalWrite(input3_pin, HIGH);
		digitalWrite(input4_pin, LOW);
	}
	analogWrite(enable34_pin, abs(motorLeftSpeed));
	//Update the LCD, no more frequently than once every 1/10th of a second to
prevent excessie flashing
	if (millis() >= updateLCDTime){
	updateLCDTime = millis() + 100;
		updateLCD();
	}
}

//Would need mass of car and power curve of motors to make theoretical prediction, or
testing
//Note: there is no use in trying to predict required turn times without physical
testing
//since I don't know the specs of the wheels or motor, or the weight of the robot
void initialTurning(){
	defaultSpeed = 0;   //If you don't want to turn on the spot for whatever
reason, change this
	int dip1_state = analogRead(dip1_pin);
	int dip2_state = analogRead(dip2_pin);
	lcd.setCursor(13, 1);
```

```
	//Conditional statement to specify initial turning angle based on DIP switches 1 and 2
	//(00, 0), (01, 90), (10, 180), (11, 270). Angles in CW direction.
	if (dip1_state < 500 && dip2_state < 500){
	lcd.print("000");
		return;	//No turning
	}
	else if (dip1_state < 500 && dip2_state > 500){
		//send commands to motor to turn 90 degrees CW (right)
		motorLeftSpeed = defaultSpeed + 100;
		motorRightSpeed = defaultSpeed - 100;
		turnFinishTime = millis() + 1000;
	lcd.print("090");
	}
	else if (dip1_state > 500 && dip2_state < 500){
		//send commands to motor to turn 180 degrees CW (right)
		motorLeftSpeed = defaultSpeed + 100;
		motorRightSpeed = defaultSpeed - 100;
		turnFinishTime = millis() + 2000;
	lcd.print("180");
	}
	else if (dip1_state > 500 && dip2_state > 500){
		//send commands to motor to turn 90 degrees CCW (left)
		motorLeftSpeed = defaultSpeed - 100;
		motorRightSpeed = defaultSpeed + 100;
		turnFinishTime = millis() + 1000;
	lcd.print("270");
	}
	updateLCD(); //If this isn't done, LCD won't show correct system mode
	updateMotorStates();
	delay(turnFinishTime-millis());  //Maintain current motor speeds for duration of turn
	turnFinishTime = -1;
}
```

What I would do next time (too late to risk an update)

- Add a method to turn a specified angle in a certain amount of time/certain speed. This would require real-life testing, and may not be a linear function.
- Restructure the motor control logic so that it only updates if a change in motor speed is required. Do this by keeping track of the previous state of motorLeftSpeed and motorRightSpeed. This should prevent the motor speed from fluctuating below 100% speed, or analogWrite(255).
- Add functionality so that if the motor speed increases from 0 to some low speed (below 50% for example), the motor will first be flashed with a 100% duty cycle signal, or analogWrite(255), to overcome static friction and accelerate the motor.