ME 210 Project- 2nd Review

Team 29 - && 0x01:
Benjamin Galligan (bgalliga@stanford.edu)
Ravi Haksar (rhaksar@stanford.edu)
Lars Roemheld (roemheld@stanford.edu)
Dongsuk Shin (ddshin@stanford.edu)

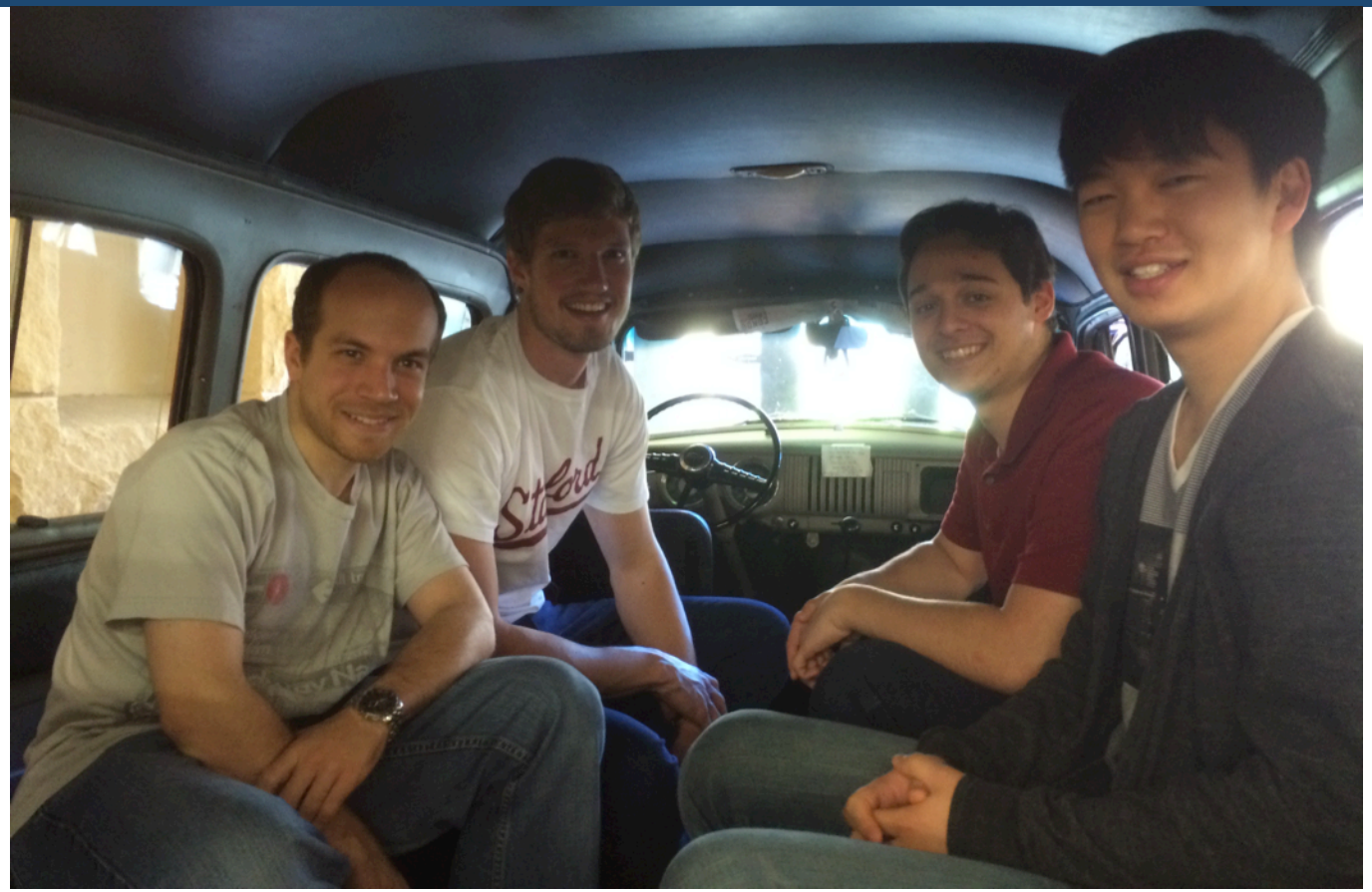
February 19, 2015

Content

- Design overview
- CADs for laser cuts - Driving Platform
- Electrical circuit designs - Driving Platform
- Partslist - Driving Platform
- Motor analysis
- State Machine
- Code Structure

Team 29 - && 0x01

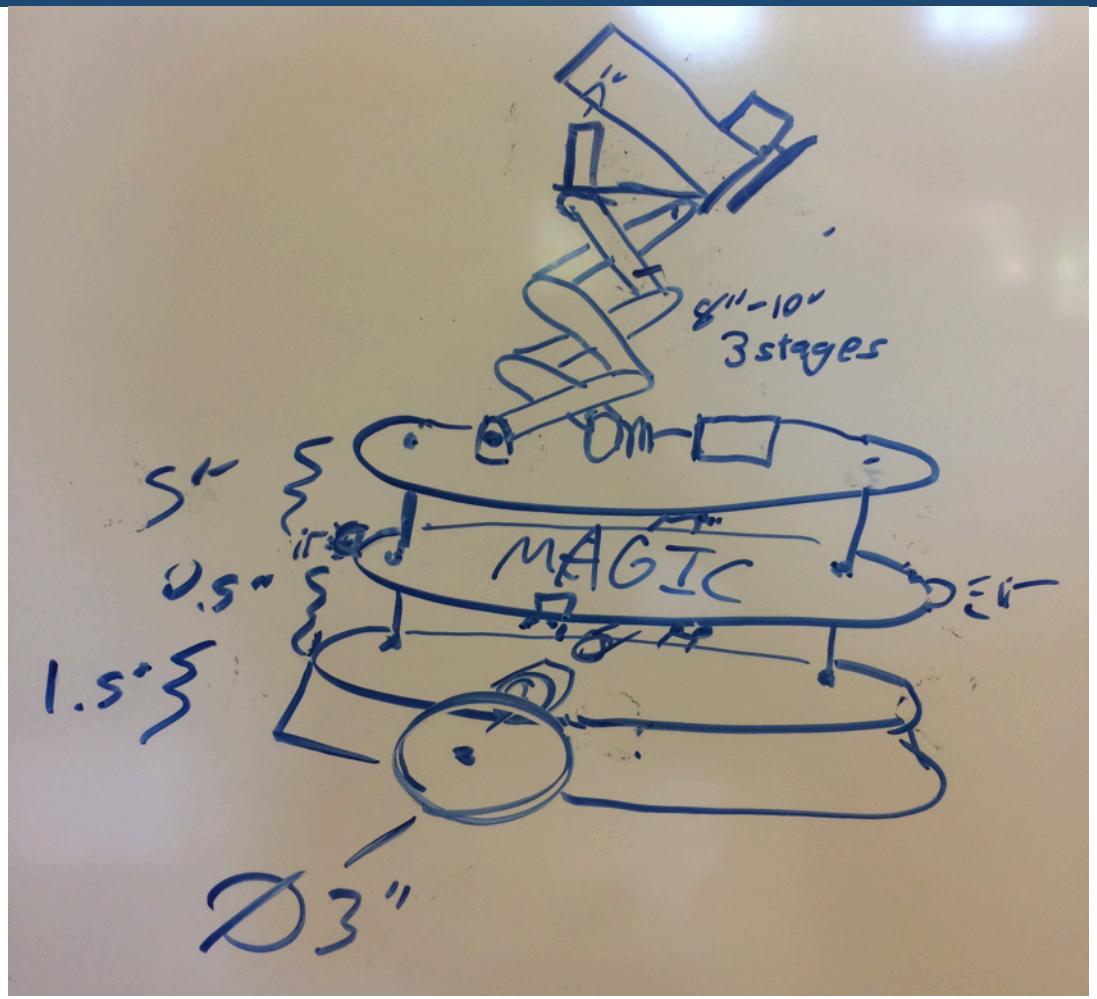
Skip to my loop.



Benjamin Galligan, Lars Roemheld, Ravi Haksar, Dongsuk Shin

Overall sketch

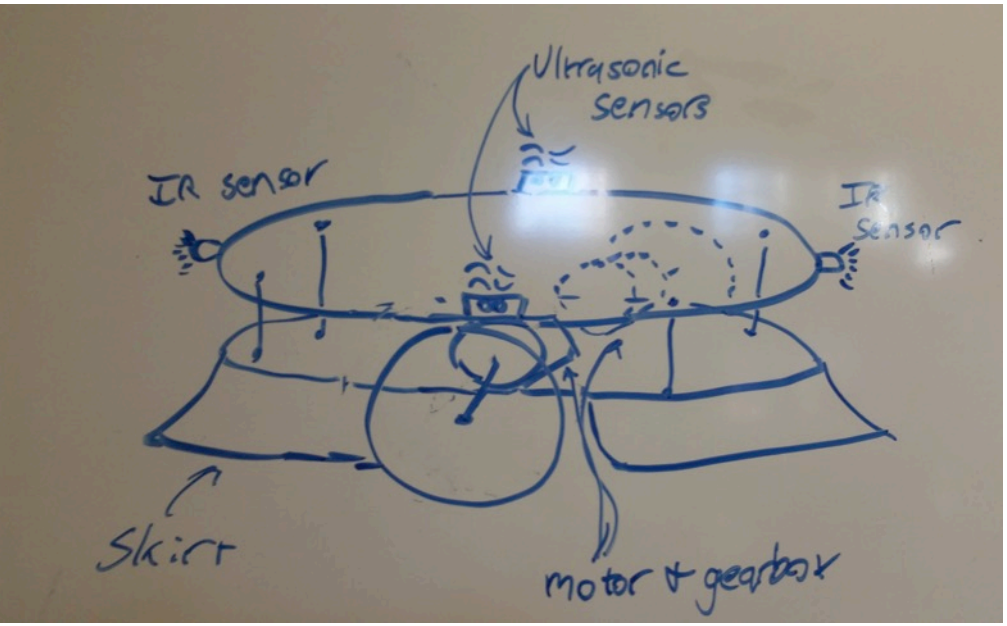
Fast-moving slam dunker.

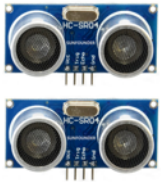


- Driving Platform: similar to cockroach, more sensors
- Skirt: fend off stray balls
- Magic zone
- Scissor-Lift for slamdunk slope

Phase 1: Drive+Navigate

First result is platform that can find bumper & 1pt hoop



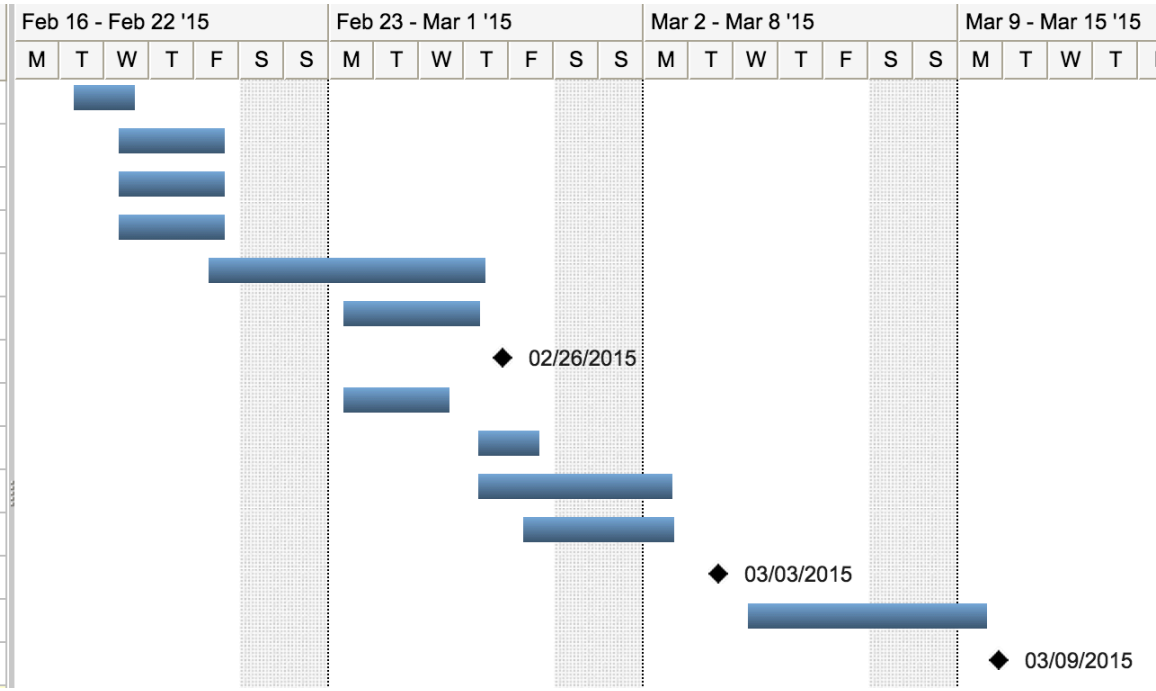


- Sensors:
  - IR proximity front & back
  - Ultrasonic sides (long range)
- Drive chain:
  - Two motors with gearbox for 2twowheels
- Arduino Magic
  - Compute shortest way based on distance to sides?
  - 90deg turns

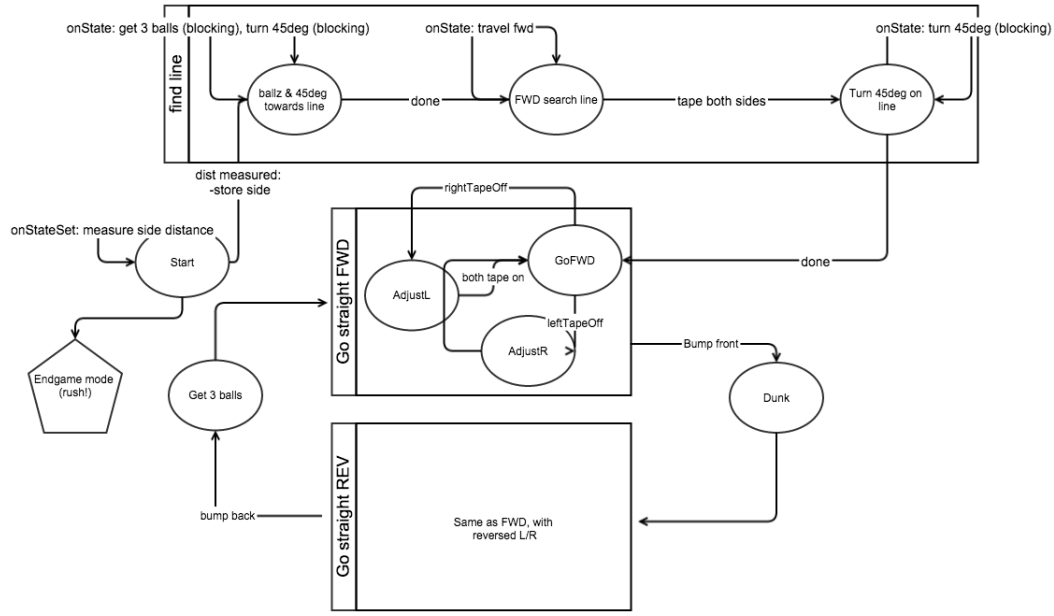
Project timeline

Look, we have a GANTT chart!

	Name
1	Validate sensing & turning options
2	Manufacture platform board
3	Buy 2 motors, 2 gearboxes, 2 wheels
4	Buy sensors
5	Assemble drive train
6	Set up state machine, make Arduino drive
7	Test navigation & finish drive train
8	Buy scissor lift parts
9	Assemble scissor lift, lifting motor
10	Assemble dunking slope with IT sensors, servo latch
11	Make Arduino dunk
12	Teaching team checkoff
13	Bugfix, optimize
14	Final presentation



State Machine



Code Structure

```
/*************************************************************
  File:        me210-project.ino
  Contents:    ME210 Project -- Team && 0x01
  Authors:     Lars Roemheld, Benjamin Galligan, Ravi Haksar, Dongsuk Shin
  Notes:       Powers a basketball playing Skip-to-my-Loop robot

               Target: Arduino Uno R1 & R2
               Arduino IDE version: 1.0.6 & 1.5.8 BETA

*************************************************************/

/*------------------ Includes ------------------------------------*/
// ME210 Timers included for timing
#include "Timers.h"
// TODO including headers produced strange errors. . . Fixed for now by (nastily
// Our motor module
#include "Motors.c"
// Our bumber module
#include "Bumpers.c"
// Our tape sensor module
```

```c
#include "TapeSensors.c"


/*------------------ Module Defines ------------------------------*/

/* State Machine. Prefix S_, middle abbreviation for functional component */
#define S_START                 0
// Find line
#define S_FL_GETFIRSTBALLS      1
#define S_FL_FWDSEARCH          2
#define S_FL_TURNONLINE         3
// Go Forward
#define S_GF_FWD                4
#define S_GF_TOLEFT             5
#define S_GF_TORIGHT            6
// Go Reverse
#define S_GR_REV                7
#define S_GR_TOLEFT             8
#define S_GR_TORIGHT            9
#define S_GR_RELOAD            10
#define S_DUNK                 11

// NEVER WRITE TO THIS VARIABLE DIRECTLY, ALWAYS USE setState()!
unsigned char state = S_START; // Global;


/* Timers */
#define T_DEBUG                 0
#define T_DEBUG_INTERVAL     1000


/* Which side of the center line are we on? */
#define SIDE_UNKNOWN        0
#define SIDE_LEFT           1
#define SIDE_RIGHT          2
unsigned char startArenaSide = SIDE_UNKNOWN;


/* Sensor parameters and thresholds */
#define HALF_FIELD_WIDTH 24 // Half the field width, used as threshold for ultra



/*------------------ Module Function Prototypes ------------------*/
```

```
/*———————————— Arduino Main Functions ————————————————————*/
void setup() {  // setup() function required for Arduino
  Serial.begin(9600);
  Serial.println("Skip to my loop!");

  TMRArd_InitTimer(T_DEBUG, T_DEBUG_INTERVAL);
  setState(S_START);
}

void loop() {  // loop() function required for Arduino
  if (TMRArd_IsTimerExpired(T_DEBUG)) timedDebug();

  switch (state) {
    case S_START:
      int startDistanceToLeft;
      // Send 5 pings, average distance
      // SendPing(ultrasonic)
      // WaitForResponse(ultrasonic)
      startDistanceToLeft = 0;

      // TODO: test what if we happen to start right on the line
      startArenaSide = (startDistanceToLeft < HALF_FIELD_WIDTH) ? SIDE_LEFT : SID
      setState(S_FL_GETFIRSTBALLS);
      break;
    case S_FL_GETFIRSTBALLS:
      Serial.println("WARNING: State machine was in S_FL_GETFIRSTBALLS, but this
      break;
    case S_FL_FWDSEARCH:
      if (areBothSensorsOnTape()) setState(S_FL_TURNONLINE);
      break;
    case S_FL_TURNONLINE:
      Serial.println("WARNING: State machine was in S_FL_TURNONLINE, but this sh
      break;
    case S_GF_FWD:
      if (isAnyFrontBumperPressed()) setState(S_DUNK);
      if (!isLeftSensorOnTape()) setState(S_GF_TORIGHT);
      if (!isRightSensorOnTape()) setState(S_GF_TOLEFT);
      break;
    case S_GF_TOLEFT:
      if (isAnyFrontBumperPressed()) setState(S_DUNK);
      if (areBothSensorsOnTape()) setState(S_GF_FWD);
      break;
    case S_GF_TORIGHT:
      if (isAnyFrontBumperPressed()) setState(S_DUNK);
      if (areBothSensorsOnTape()) setState(S_GF_FWD);
      break;
```

```
    case S_GR_REV:
      break;
    case S_GR_TOLEFT:
      break;
    case S_GR_TORIGHT:
      break;
    case S_GR_RELOAD:
      break;
    case S_DUNK:
      break;
  }
}


/*------------------- State Machine Functions -------------*/
void setState (unsigned int newState) {
  signed int arenaTurnSign;

  switch (newState) {
    case S_START:
      break;
    case S_FL_GETFIRSTBALLS:
      requestBalls(3);

      arenaTurnSign = (startArenaSide == SIDE_LEFT) ? 1 : -1;

      turnRightInPlace(200, arenaTurnSign * MAX_MOTOR_SPEED / 2);

      setState(S_FL_FWDSEARCH);
      break;
    case S_FL_FWDSEARCH:
      setMotorSpeed(MAX_MOTOR_SPEED);
      break;
    case S_FL_TURNONLINE:
      arenaTurnSign = (startArenaSide == SIDE_LEFT) ? 1 : -1;

      turnRightInPlace(200, - arenaTurnSign * MAX_MOTOR_SPEED / 2);

      setState(S_GF_FWD);
      break;
    case S_GF_FWD:
      setMotorSpeed(MAX_MOTOR_SPEED);
      break;
    case S_GF_TOLEFT:
      setLeftMotorSpeed(MAX_MOTOR_SPEED * 4 / 5);
      setRightMotorSpeed(MAX_MOTOR_SPEED);
```

```
      break;
    case S_GF_TORIGHT:
      setLeftMotorSpeed(MAX_MOTOR_SPEED);
      setRightMotorSpeed(MAX_MOTOR_SPEED * 4 / 5);
    case S_GR_REV:
      break;
    case S_GR_TOLEFT:
      break;
    case S_GR_TORIGHT:
      break;
    case S_GR_RELOAD:
      break;
    case S_DUNK:
      Serial.println("DUNK!");
      break;
    default:
      Serial.println("Invalid state requested: ");
      Serial.println(newState);
  }

  state = newState;
}




/* Blocking transition functions */

// Will request the specified number of balls by hitting the bumper and waiting
void requestBalls(char numBalls) {
  // assert numBalls correct
  if (numBalls < 1 or numBalls > 3) return;

  for (int iBall = 1; iBall <= numBalls; iBall++) {
    // Make sure we hit the bumper
    while (!isAnyBackBumperPressed()) {
      setMotorSpeed(-MAX_MOTOR_SPEED);
      delay(20);
    }

    // Push a little further, then Wait for a moment to show off pressed state
    delay(100);
    setMotorSpeed(0);
    delay(500);

    // Release the bumper
```

```
    while (isAnyBackBumperPressed()) {
      setMotorSpeed(MAX_MOTOR_SPEED);

      delay(20);
    }

    // Wait for ball
    setMotorSpeed(0);
    delay(2000);
  }
}

// Will turn on the spot. Will turn right for positive turnSpeed, left for negat
void turnRightInPlace(unsigned int milliSecs, signed int turnSpeed) {
  signed int oldLeftSpeed, oldRightSpeed;

  oldLeftSpeed = setLeftMotorSpeed(   -turnSpeed);
  oldRightSpeed = setRightMotorSpeed(  turnSpeed);

  delay(milliSecs);

  setLeftMotorSpeed(oldLeftSpeed);
  setRightMotorSpeed(oldRightSpeed);
}


/*---------------- Module Functions ----------------------------*/


void timedDebug(void) {
  static int Time = 0;

  TMRArd_InitTimer(T_DEBUG, T_DEBUG_INTERVAL);

  Serial.print(" time:");
  Serial.print(++Time);
  Serial.print(" state:");
  Serial.println(state,DEC);
}




/*************************************************************
  File:       Motors.h
```

```
  Contents:  ME210 Project -- Team && 0x01
  Authors:   Lars Roemheld, Benjamin Galligan, Ravi Haksar, Dongsuk Shin
  Notes:     Power our motors

             Target: Arduino Uno R1 & R2
             Arduino IDE version: 1.0.6 & 1.5.8 BETA

************************************************************/

#ifndef H_MOTORS
#define H_MOTORS

#define NULL_VALUE -10000 // used as threshold for non-setting function calls
#define MAX_MOTOR_SPEED 254


#define PIN_LEFT_MOTOR 0
#define PIN_RIGHT_MOTOR 1


void initMotors();

// Set speed of left motor, if newSpeed > (NULL_VALUE). Return old speed.
signed int setLeftMotorSpeed(signed int newSpeed);
// Set speed of right motor, if newSpeed > (NULL_VALUE). Return old speed.
signed int setRightMotorSpeed(signed int newSpeed);
// Set both motor speeds, return nothing.
void setMotorSpeed(signed int newSpeed);

#endif

/************************************************************
  File:      Motors.c
  Contents:  ME210 Project -- Team && 0x01
  Authors:   Lars Roemheld, Benjamin Galligan, Ravi Haksar, Dongsuk Shin
  Notes:     Power our motors

             Target: Arduino Uno R1 & R2
             Arduino IDE version: 1.0.6 & 1.5.8 BETA

************************************************************/

#include "Motors.h"


void initMotors() {
```

```
        // "You do not need to call "pinMode(PIN_LEFT_MOTOR, OUTPUT);" before us

}

// Set speed of left motor, if newSpeed > (NULL_VALUE). Return old speed.
// TODO: how do we handle 2 directions? DIR pin or DC polarity?
signed int setLeftMotorSpeed(signed int newSpeed) {
        static signed int leftSpeed = 0;

        int oldSpeed;
        oldSpeed = leftSpeed;
        if (newSpeed > NULL_VALUE) {
                leftSpeed = newSpeed;

                // Set arduino motor outputs to value
                analogWrite(PIN_LEFT_MOTOR, newSpeed);
        }
        return oldSpeed;
}

// Set speed of right motor, if newSpeed > (NULL_VALUE). Return old speed.
// TODO: how do we handle 2 directions? DIR pin or DC polarity?
signed int setRightMotorSpeed(signed int newSpeed) {
        static signed int rightSpeed = 0;

        int oldSpeed;
        oldSpeed = rightSpeed;
        if (newSpeed > NULL_VALUE) {
                rightSpeed = newSpeed;

                // Set arduino motor outputs to value
                analogWrite(PIN_RIGHT_MOTOR, newSpeed);
        }
        return oldSpeed;
}

void setMotorSpeed(signed int newSpeed) {
  setLeftMotorSpeed(newSpeed);
  setRightMotorSpeed(newSpeed);
}


/*************************************************************
  File:       Bumpers.h
  Contents:   ME210 Project -- Team && 0x01
  Authors:    Lars Roemheld, Benjamin Galligan, Ravi Haksar, Dongsuk Shin
```

```
  Notes:      Read our bumpers

              Target: Arduino Uno R1 & R2
              Arduino IDE version: 1.0.6 & 1.5.8 BETA

*************************************************************/

#ifndef H_BUMPERS
#define H_BUMPERS

#include "Arduino.h"


#define PIN_LEFT_BACKBUMPER    0
#define PIN_RIGHT_BACKBUMPER   1
#define PIN_LEFT_FRONTBUMPER   2
#define PIN_RIGHT_FRONTBUMPER  3


void initBumpers();

unsigned char isLeftBackBumperPressed();
unsigned char isRightBackBumperPressed();
unsigned char isAnyBackBumperPressed();

unsigned char isLeftFrontBumperPressed();
unsigned char isRightFrontBumperPressed();
unsigned char isAnyFrontBumperPressed();

#endif/*************************************************************
  File:       Bumpers.c
  Contents:   ME210 Project -- Team && 0x01
  Authors:    Lars Roemheld, Benjamin Galligan, Ravi Haksar, Dongsuk Shin
  Notes:      Read our bumpers

              Target: Arduino Uno R1 & R2
              Arduino IDE version: 1.0.6 & 1.5.8 BETA

*************************************************************/

#include "Bumpers.h"

void initBumpers() {
        pinMode(PIN_LEFT_BACKBUMPER, INPUT);
        pinMode(PIN_RIGHT_BACKBUMPER, INPUT);
        pinMode(PIN_LEFT_FRONTBUMPER, INPUT);
```

```
        pinMode(PIN_RIGHT_FRONTBUMPER, INPUT);

}

unsigned char isLeftBackBumperPressed() {
        return (digitalRead(PIN_LEFT_BACKBUMPER) == HIGH) ? true : false;
}
unsigned char isRightBackBumperPressed() {
        return (digitalRead(PIN_RIGHT_BACKBUMPER) == HIGH) ? true : false;
}
unsigned char isAnyBackBumperPressed() {
        return (isLeftBackBumperPressed() || isRightBackBumperPressed());
}

unsigned char isLeftFrontBumperPressed() {
        return (digitalRead(PIN_LEFT_FRONTBUMPER) == HIGH) ? true : false;
}
unsigned char isRightFrontBumperPressed() {
        return (digitalRead(PIN_RIGHT_FRONTBUMPER) == HIGH) ? true : false;
}
unsigned char isAnyFrontBumperPressed() {
        return (isLeftFrontBumperPressed() || isRightFrontBumperPressed());
}

/*************************************************************
  File:       TapeSensors.h
  Contents:   ME210 Project -- Team && 0x01
  Authors:    Lars Roemheld, Benjamin Galligan, Ravi Haksar, Dongsuk Shin
  Notes:      Read our tape sensors (IR Sender + Emitter pairs)

              Target: Arduino Uno R1 & R2
              Arduino IDE version: 1.0.6 & 1.5.8 BETA

*************************************************************/

#ifndef H_TAPESENSORS
#define H_TAPSENSORS

#include "Arduino.h"


#define PIN_LEFT_TAPESENSOR    5
#define PIN_RIGHT_TAPESENSOR   6

void initTapeSensors();
```

```
unsigned char isLeftSensorOnTape();
unsigned char isRightSensorOnTape();
unsigned char areBothSensorsOnTape();


#endif


/*************************************************************
  File:        TapeSensors.c
  Contents:    ME210 Project -- Team && 0x01
  Authors:     Lars Roemheld, Benjamin Galligan, Ravi Haksar, Dongsuk Shin
  Notes:      Read our tape sensors (IR Sender + Emitter pairs)

              Target: Arduino Uno R1 & R2
              Arduino IDE version: 1.0.6 & 1.5.8 BETA

*************************************************************/

#include "TapeSensors.h"


void initTapeSensors() {
        pinMode(PIN_RIGHT_TAPESENSOR, INPUT);
        pinMode(PIN_LEFT_TAPESENSOR, INPUT);
}

// TODO: Threshold values, hysteresis (testing)
unsigned char isLeftSensorOnTape() {

}
unsigned char isRightSensorOnTape() {

}
unsigned char areBothSensorsOnTape() {
        return isLeftSensorOnTape() && isRightSensorOnTape();
}
```

Drive train analysis

Requirements and Considerations

In order to properly size the motors, gearboxes (if necessary), and wheels, expected operating conditions must first be established. Table 1 lists these parameters and their estimated values.

Table 1: Estimated desired operating conditions

Parameter	**Value**
Max supply voltage	7.2 V
Number of driven wheels	2
Max Robot mass	5.6 kg (12.24 lbs)
Min linear speed	0.15 m/s (0.5 ft/s)

In addition, the driver circuit will also effect the feasibility of chosen motors. For simplicity, the L293 driver board was chosen, as it is already available, and requires very little additional components to use. The important characteristics of the driver circuit are shown in Table 2.

Table 2: L293 driver circuit characteristics

Parameter	**Value**
Motor supply voltage	5 to 36 V
Max output drop	3.6 V
Max output current	1 A

Dynamic motor performance

To determine how the motor would perform under the desired parameters, it is necessary to take a force balance and torque balance to determine the linear and angular velocities and accelerations.

$$\sum F_x = ma_x$$
$$\sum F_y = ma_y$$
$$\sum T = I\alpha$$

From the force balance in the y-direction, it is possible to determine the normal force supporting the wheel. Note that since two wheels are driven and two wheels are passive (casters), the normal force that the motor needs to handle is half of the total robot mass.

$$N = \frac{1}{2}mg\cos(\theta)$$

The force balance in the x-direction, it is possible to determine the linear acceleration, which will be in terms of the torques exerted about the driven wheels. This can be rewritten in terms of the angular acceleration of the wheels, from the torque balance.

$$a_x = \frac{\sum T}{mr} = \frac{I\alpha}{mr}$$

Finally, the torque-balance yields the angular acceleration of the wheels. The torque that resists the motor consists of a rolling resistance $C_r N$ and the effects of gravity, if the robot is on an incline.

$$\begin{aligned}
\alpha &= \frac{1}{I}\left(T_m - F_t r\right) \\
&= \frac{1}{I}\left[T_m - \left(C_r N + g\sin\left(\theta\right)\right) r\right] \\
&= \frac{1}{I}\left[2\eta T_m - \left(\frac{1}{4} C_r m g \cos\left(\theta\right) + g\sin\left(\theta\right)\right) r\right]
\end{aligned}$$

In addition to the previous expressions, the relationship between motor torque and angular speed is also required to determine how the motor performs after the initial acceleration from stall. The wheel moment of inertia is also shown below.

$$\begin{aligned}
T_m\left(t\right) &= \frac{\omega_{NL} - \eta\omega\left(t\right)}{R_m} \\
I &= \frac{1}{2} m_{wheel} r^2
\end{aligned}$$

Taking the integral of the expressions for angular and linear accelerations will yield the angular and linear speeds, respectively. These expressions will be used to determine how fast the robot will reach its steady-state speed, and the top speed of the robot.

$$\begin{aligned}
\omega &= \int_0^t \alpha\left(\tau\right) d\tau \\
v_x &= \int_0^x a_x\left(\xi\right) d\xi
\end{aligned}$$

Motor comparison and validation

Using the previous analysis and MATLAB, two motor/gearbox/wheel systems were analyzed for feasibility. Note that the worst case parameters were used; specifically, the motor voltage was reduced by the voltage drop specified in Table 2. Table 3 shows the parameters for the first system, which consists of a 6 V motor connected through a gearbox to 65 mm wheels. The system is from Sparkfun, product number 12866. Some values were estimated, such as the wheel mass and the expected efficiency.

Figure 1 shows that the Sparkfun motors will have a top speed of approximately 0.12 m/s (0.39 ft/s). Using these motors, it will take the robot approximately 20 seconds to traverse 8 ft. In addition, these simulation results are somewhat ideal; it would be wise to estimate the time to be closer to 25 or 30 seconds. Nevertheless, these motors will still work.

$$\boxed{\text{Motor system 1 will work}}$$

Table 3: Motor system 1 parameters

Parameter	**Value**
Motor voltage	6 V
Stall current	1 A
Stall torque	800 gf-cm
No-load speed	80 rpm
Wheel radius	32.5 mm (1.27 in)
Wheel mass	50 g
Efficiency	75%

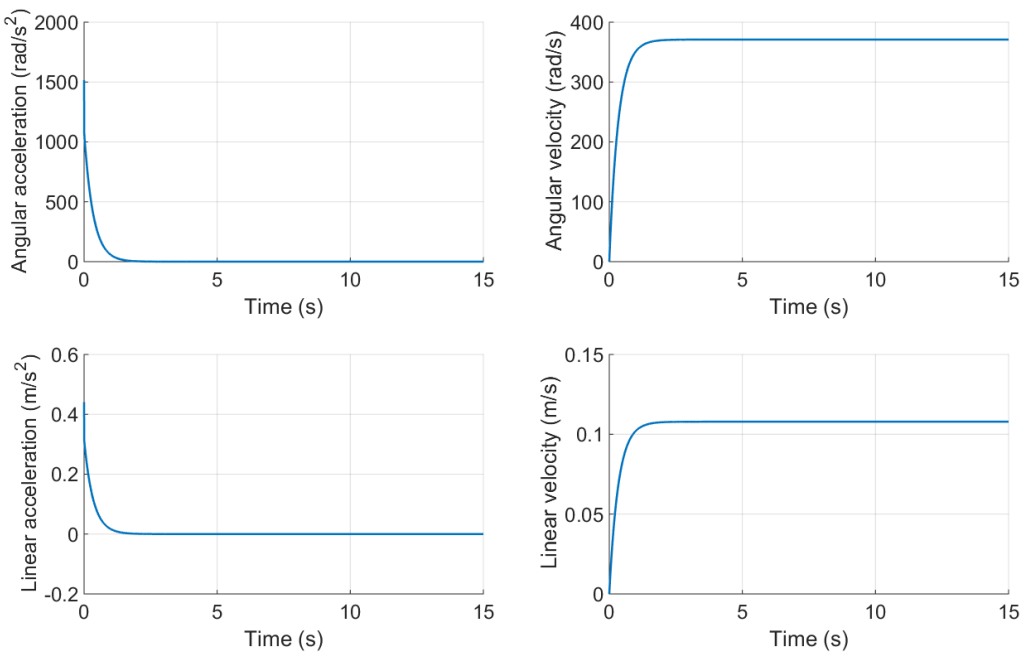


Figure 1: Simulation results for the Sparkfun motors

Table 4: Motor system 2 parameters

Parameter	**Value**
Motor voltage	6 V
Stall current	0.350 A
Stall torque	8800 gf-cm
No-load speed	133 rpm
Wheel radius	50.8 mm (2 in)
Wheel mass	200 g
Efficiency	75%

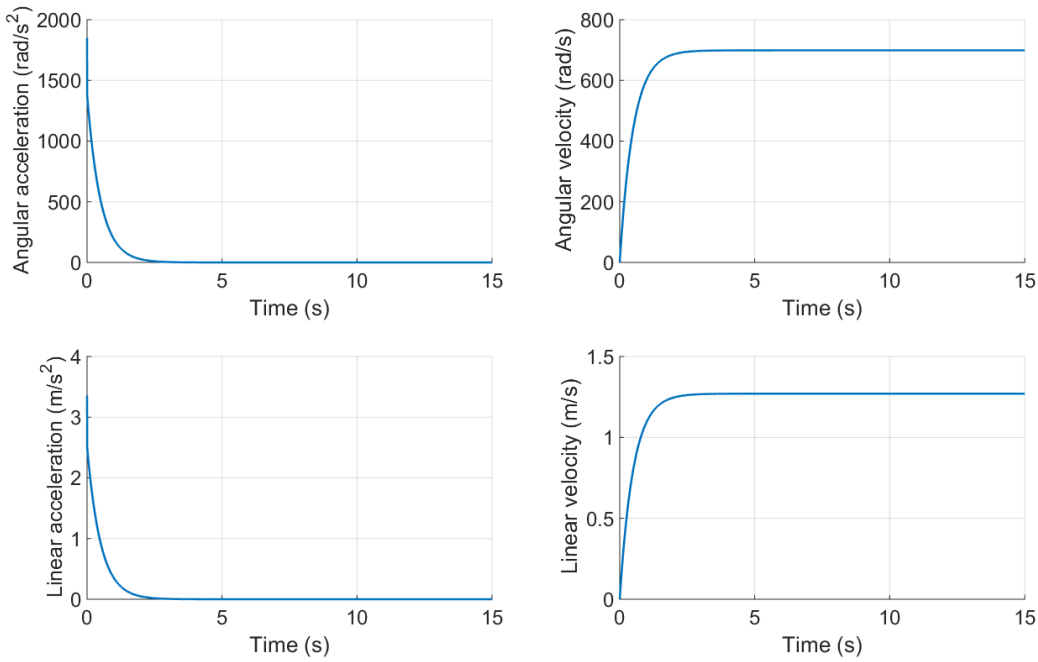


Figure 2: Simulation results for the Jameco motors

Table 4 provides the parameters for the second motor system, which uses two motors from Jameco, and 4 in wheels from the lab supply closet. The product number for the Jameco motors is 2150491.

Figure 2 shows that the robot will have a top speed of approximately 1.25 m/s (4.1 ft/s). As a result, the expected time to traverse 8 ft is about 2 seconds. Considering that the motor specifications were heavily estimated for this motor (as Jameco did not provide a manufacturer datasheet), these results should be taken with a grain of salt. A conservative estimate puts the time to travel the length of the field at around 10 seconds.

$$\boxed{\text{Motor system 2 will work}}$$

Electrical Design Overview

The final robot will use an external power source, i.e., battery, to drive the motors and to deal with sensor signals. Values of resistors required in the circuit will be determined empirically as the team considers the effects of environmental variances. The motors will be controlled using the L293 quadruple half-H driver. The driving platform circuit is shown below.

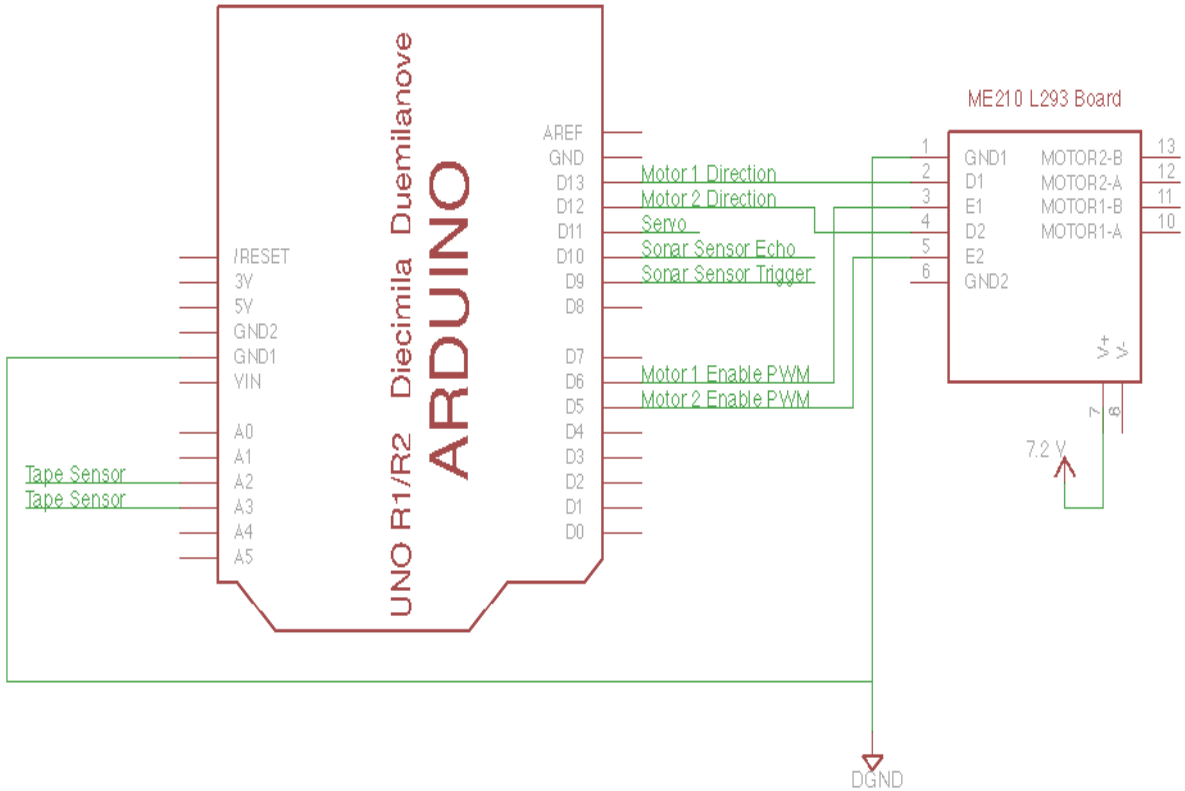


Figure 1. Arduino Pin Configuration with Motor Driver

Figure 1 also shows the Arduino UNO microcontroller pin connection diagram with all the inputs and outputs for the robot:

- Analog Inputs:
  - 2, 3: Tape sensor inputs
- Digital Inputs:
  - 16, 17: Contact switch inputs
  - 11, 12: Limit switch inputs
- Digital Outputs:
  - 5, 6: Enable PWM signals for the 2 motors
  - 12, 13: Directions for the 2 motors
  - 9, 10: Echo and Trigger signals for the sonar sensor
  - 11: Servo PWM signals

Figure 2 shows the circuit configuration for a tape sensor. Each circuit will have an LED part and a photoeresistor part. The photoresistor is in a sourcing configuration and its output is connected to a comparator in an inverting configuration with hysteresis.

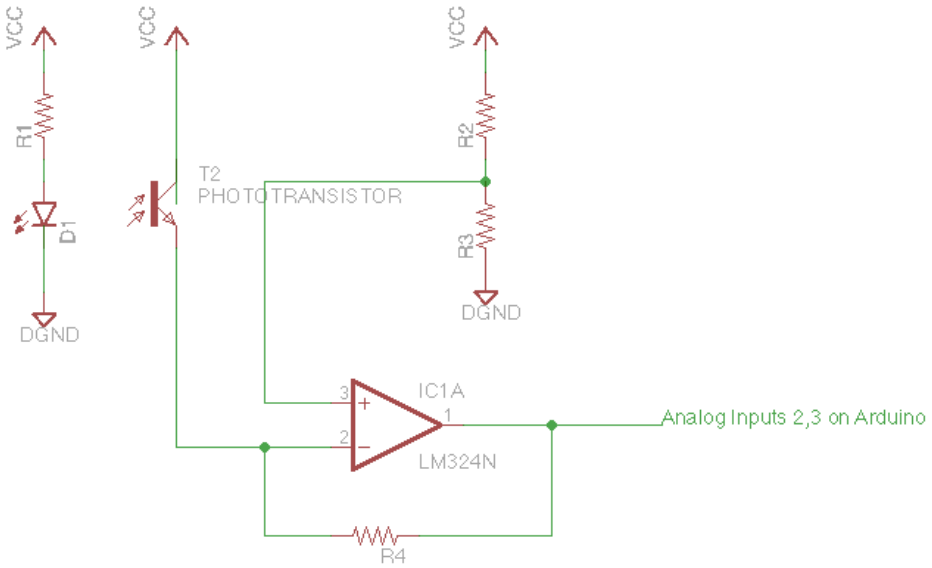


Figure 2. Tape Sensor Circuit

Figure 3 shows the circuit diagram for one contact switch.

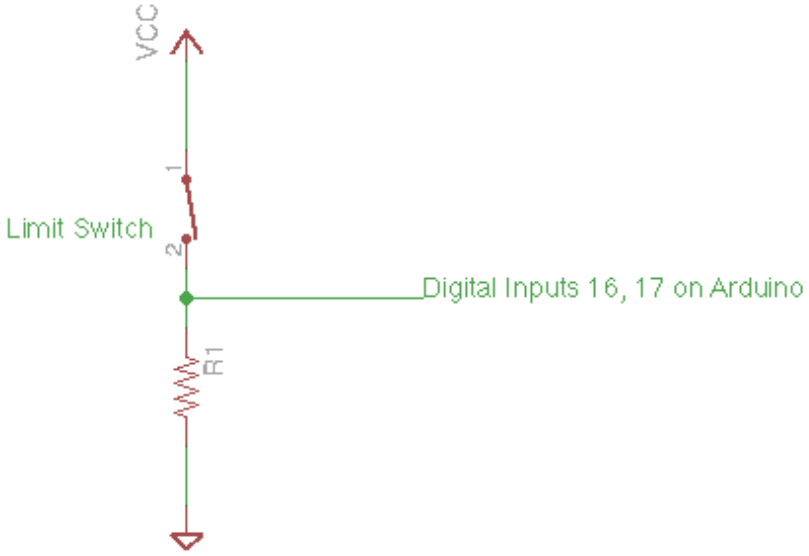


Figure 3. Limit Switch Circuit

Mechanical Design

The current design focuses on the motor housing, wheel placement and supports, and the tape-sensing subsystem of our robot. The figure below shows the exploded view of our design. Note that the upper plate contains slots that fit onto the upper parts of the connecting plates.

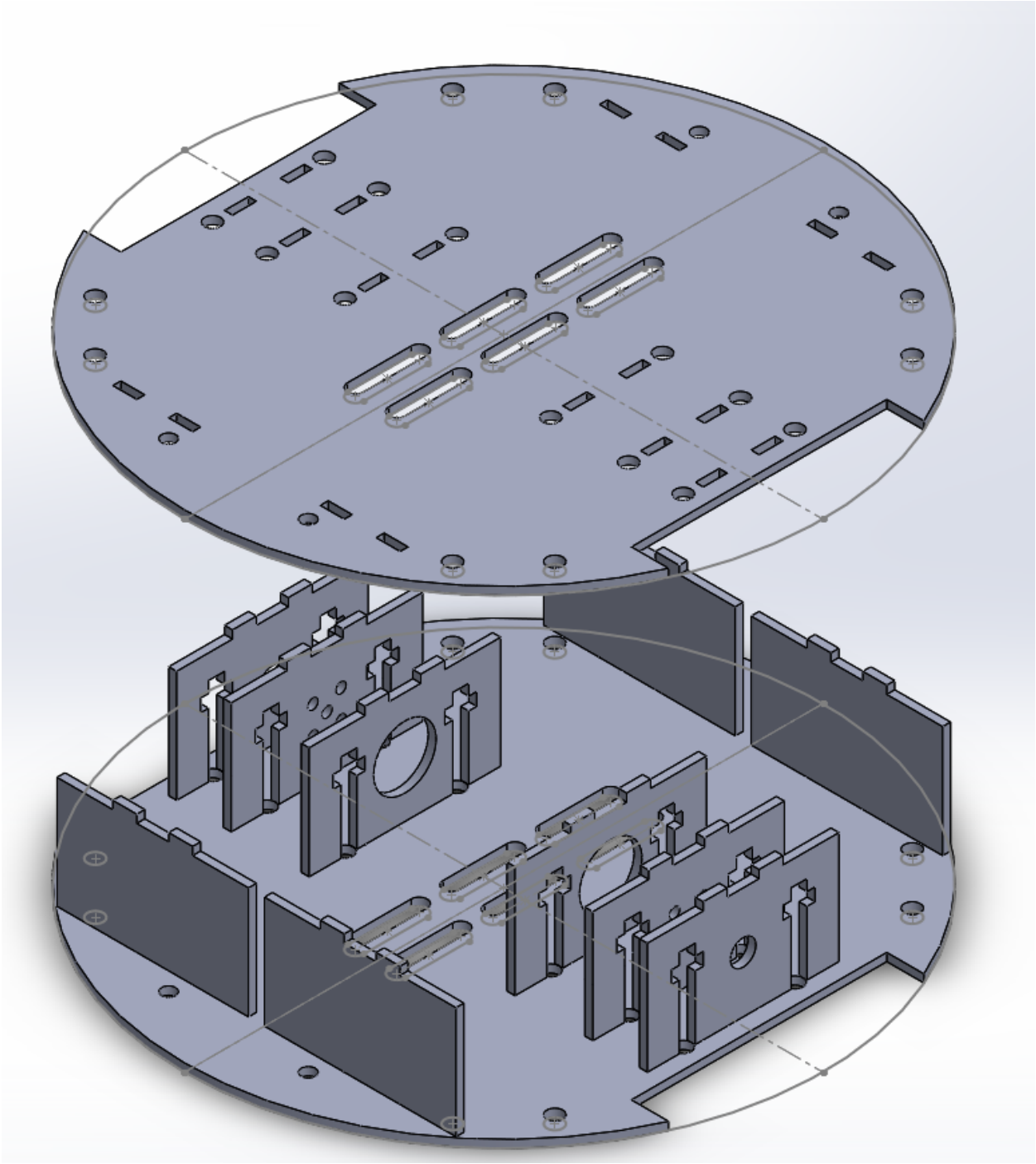


Figure 1: Exploded Assembly of Robot Base

The base and upper plate of the motor housing use the same design, simplifying the manufacturing requirements. Cut edges on the right and left of the robot accommodate up to a 4 inch diameter wheel. Eight holes on the periphery of the plate are designed for ¼ inch bolts. Four additional holes at the front and the back of the robot are designed to work with the ME 210 stockroom ball casters. Six slots ½ inch in width and 1 inch in length are located at the center of the robot. These are intended to allow tape sensors to be attached under the robot, as well as to allow for cabling connections for the motor and tape sensors to upper levels of the robot.

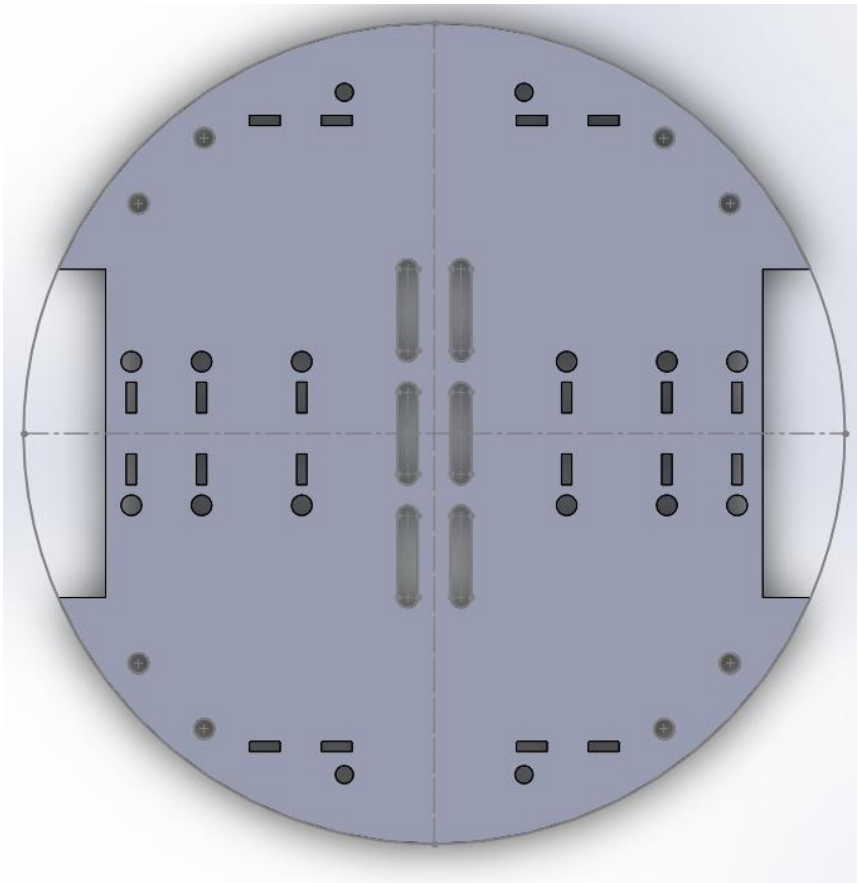


Figure 2: Top View of Base Plate

Ten vertical plates interconnect the base plate and upper plate. These are accommodated by sets of two slots per plate. These slots are 1/8” by 3/8” and are separated by 1/2” to their associated slot. Four plates at the front and back are intended to help provide support to the robot. The remaining six are designed to help support the motor and bearings. Three types of supporting plates are used; one to support the back of the motor, one to screw into the face of the motor, and one to support the bearings. From the interior of the bot to the outside, these are arranged as follows: motor back plate, motor face plate, motor bearing plate. These plates can be seen in Figures 3 to 6 below.

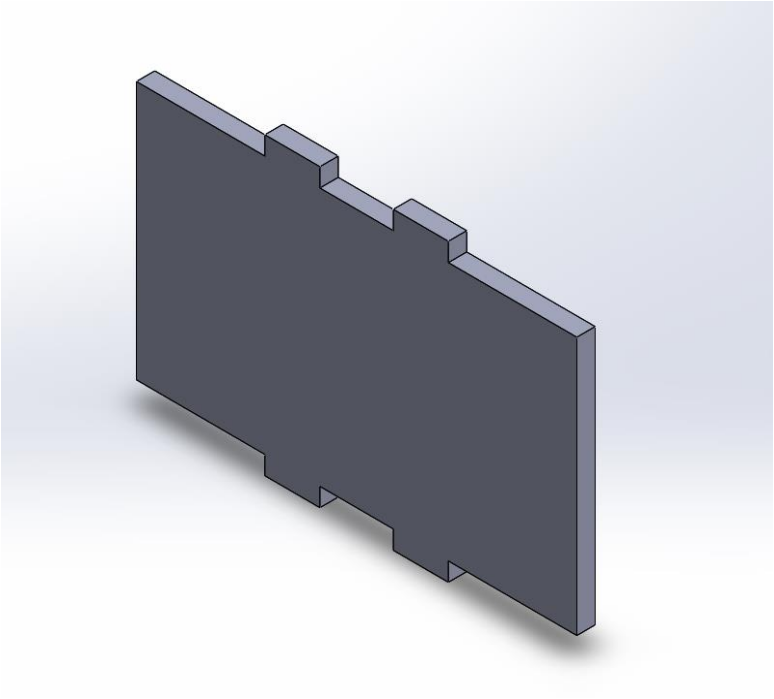


Figure 3: Support Plate

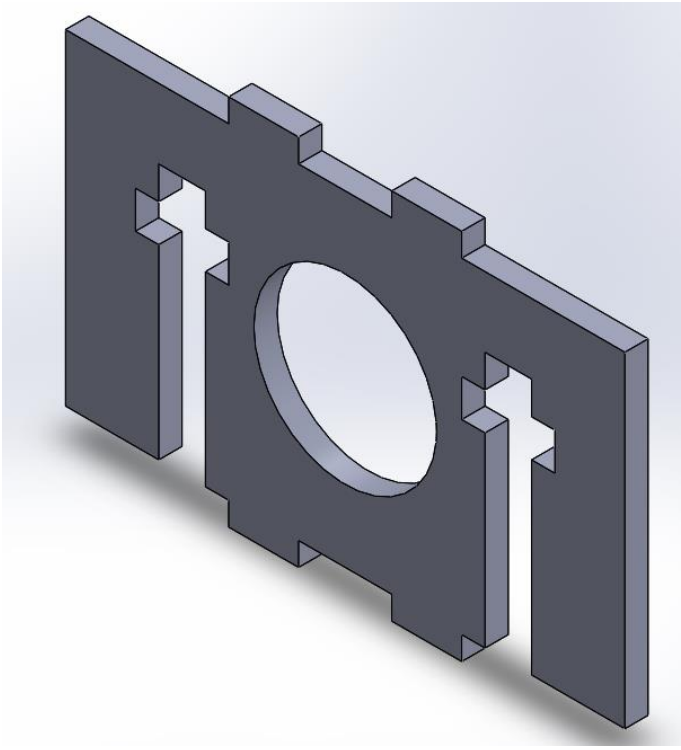


Figure 4: Motor Back Support Plate

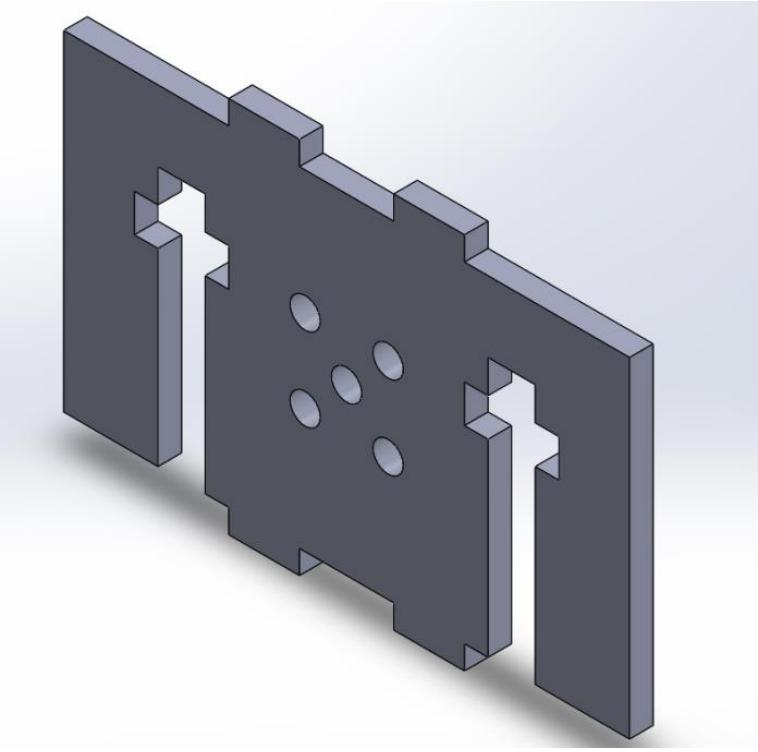


Figure 5: Motor Face Support Plate

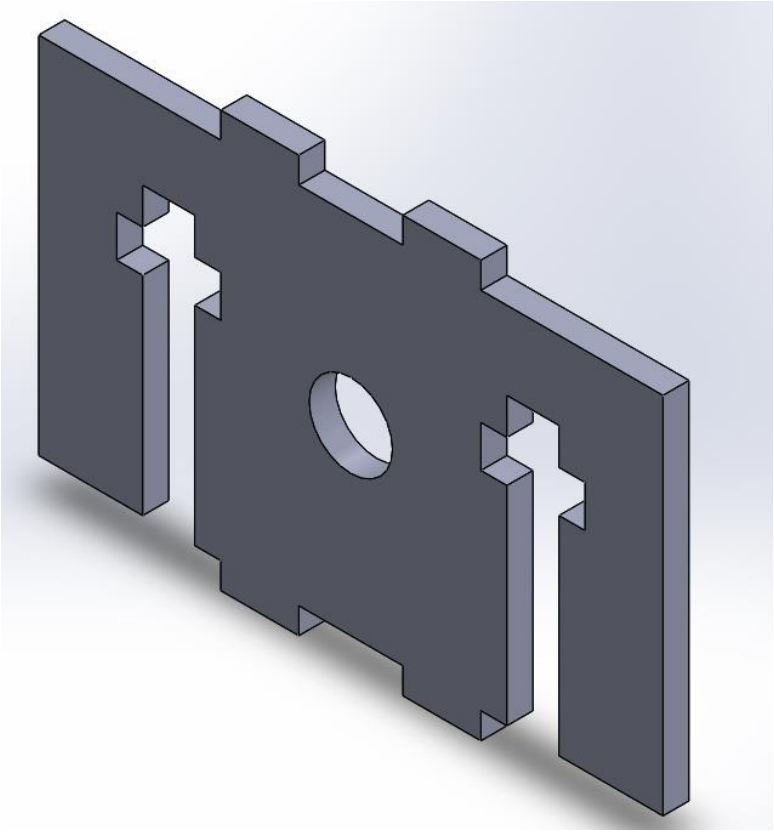


Figure 6: Bearing Support Plate

In addition to the slot connection to the base plate, these plates contain additional slots intended for a 1/4 inch by 1-1/2 inch hex bolt and associated nut. The purpose of this design is to ensure that these critical plates are secure and do not loosen from the robot colliding with obstacles. They also help to stiffen the connecting boards, which reduces mechanical vibrations due to the motor.