EECS 129B Winter 2008 Final Project Report

Team Members

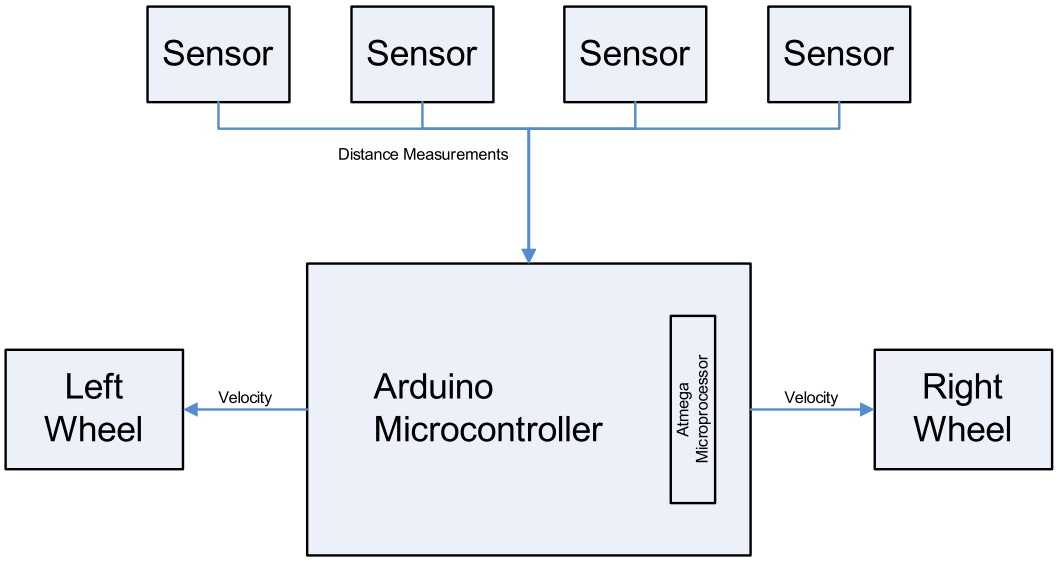
Last Name	First Name
Nguyen	Alan
Muntsinger	Tristan
Dinh	Tuan

Maze-Solving Car

1. Abstract

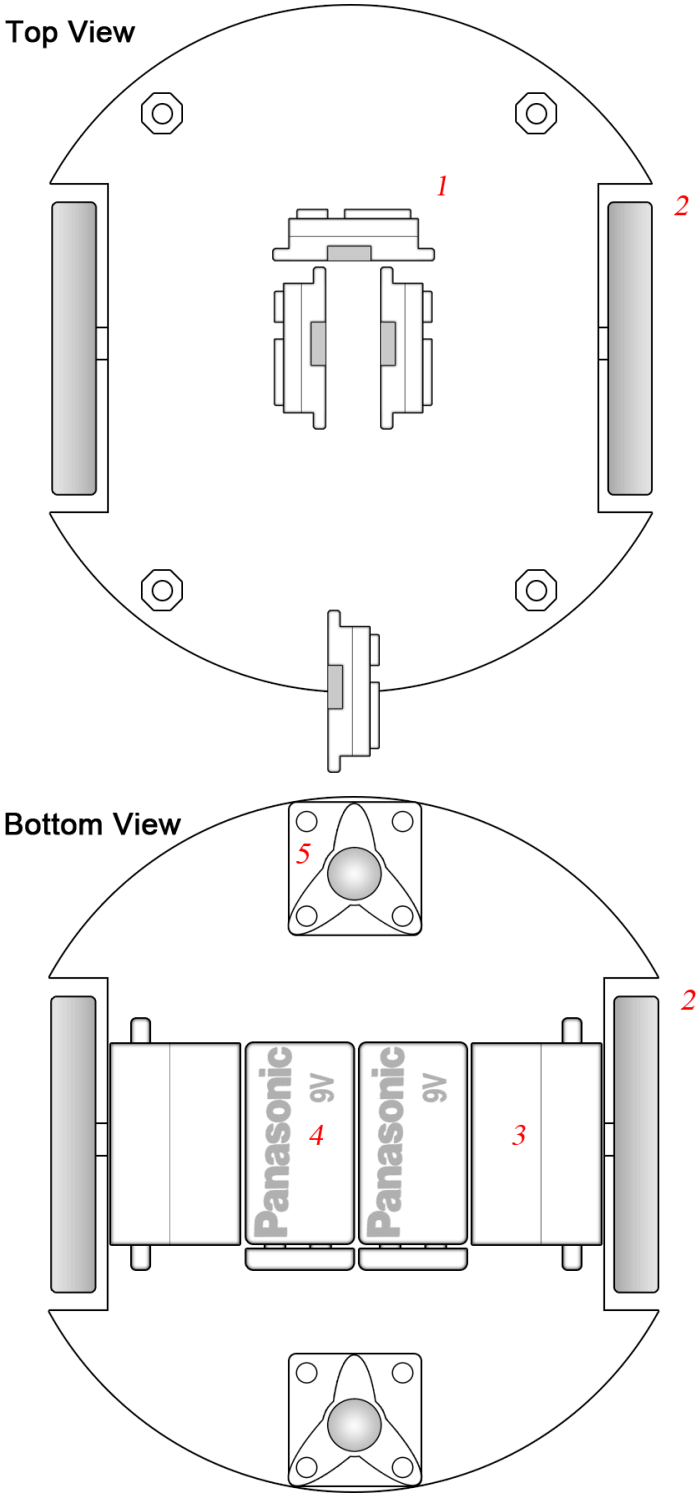
The project involves designing and constructing a car that can navigate itself through an arbitrary maze to find the exit, if it exists. To do so, the car will receive data from the outside world using sensors placed in key positions on the car. The sensors will determine distances of objects (walls) relative to the car and provide data for the car to navigate itself. Data received by the sensors will go to a microcontroller that performs calculations to determine the next course of action. Once the correct action is determined, it will be executed by sending the appropriate signals to the wheels. Because the maze will consist of right angle turns, the simplest approach is to use two bidirectional wheels to ensure that the car remains relatively stationary while performing a 90 degree turn. 2 ball castors are also used to keep the car balanced. The car will perform movements while actively taking sensor data, so that it does not collide with any walls.

2. Project overview (High level)


Sensor
Sensor
Sensor
Sensor
Distance Measurements
Left Wheel
Velocity
Arduino Microcontroller
Atmega Microprocessor
Velocity
Right Wheel

3. Project components

Mechanical engineering



- Car Chassis:

Construction of chassis made from 1/4"-thick High Density Polyethelyne (HDPE) and mounting of various parts:

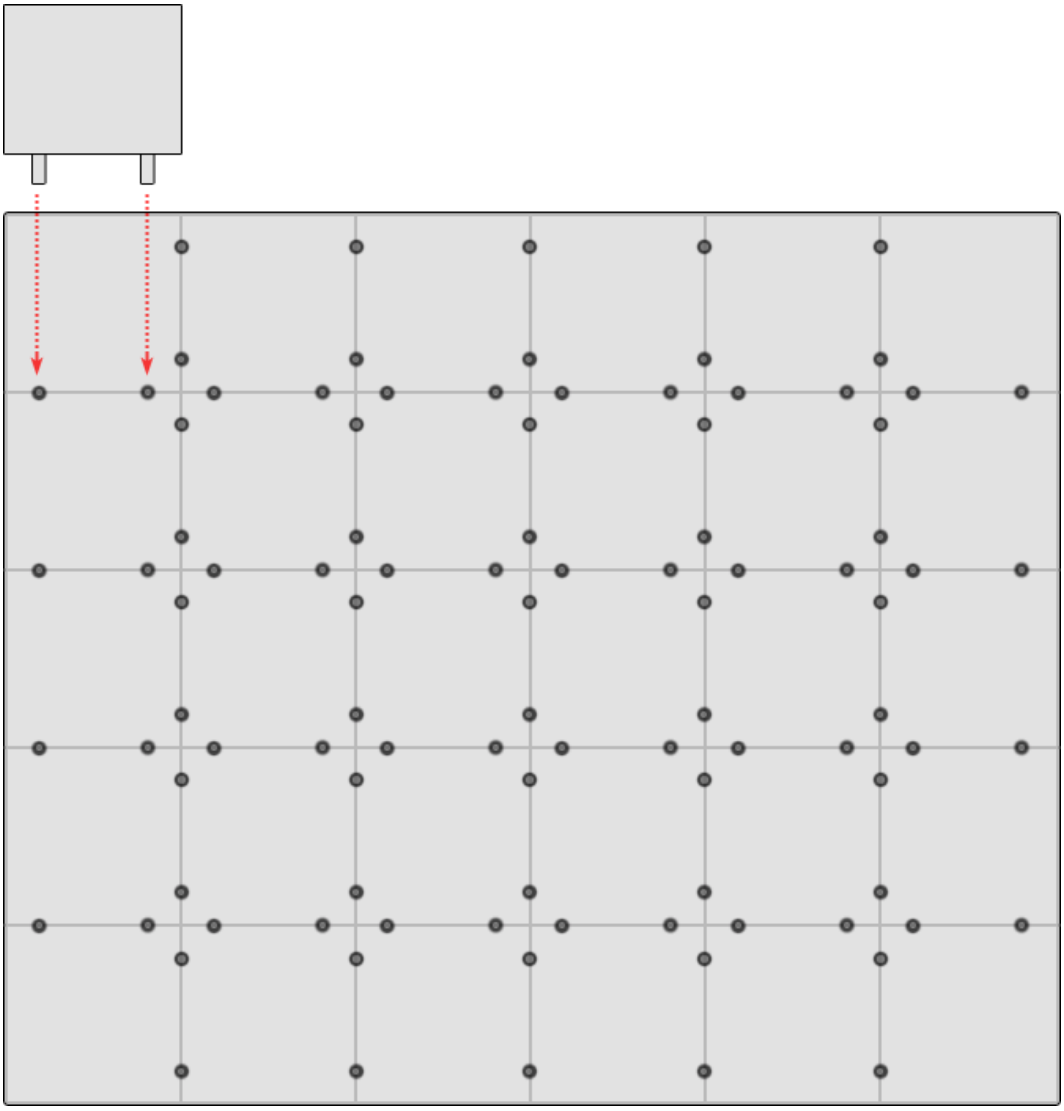
1) Sharp IR sensor
2) Servo-mounted wheel
3) Hitec HS-322HD Servo*
4) 9V battery
5) Tamiya ball castor

*Servos required modification, which involved opening the servos, adjusting the gears, and removing a mechanical stop

Parts were placed onto two sheets of HDPE in appropriate locations and mounted using glue or velcro.

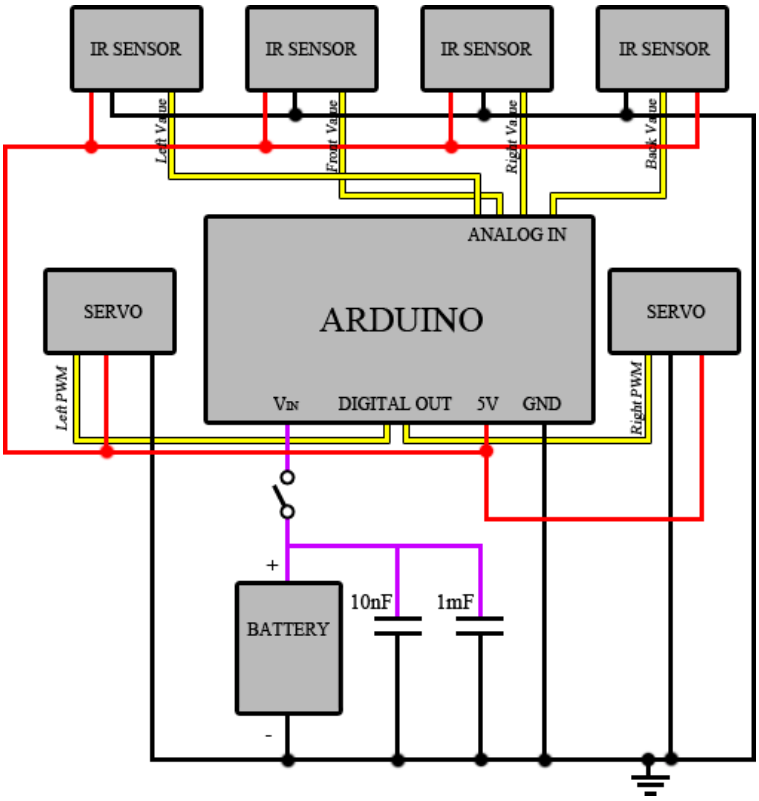
The two sheets of HDPE were then held together using screws and nuts, leaving a space in between the layers for the microcontroller, switch, capacitors, and wires.

- Maze

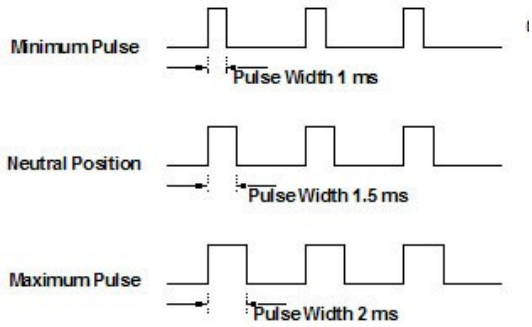


Maze constructed out of three 1/2" thick blocks of wood that, when assembled, became a 5' by 5' piece. Holes were drilled into the wood such that a 6x6 grid was created (each grid block was 10"x10"). Nails were then inserted into smaller blocks of wood, which were made so that the nails could be inserted into the drilled holes in the maze platform, as shown above.

Electrical

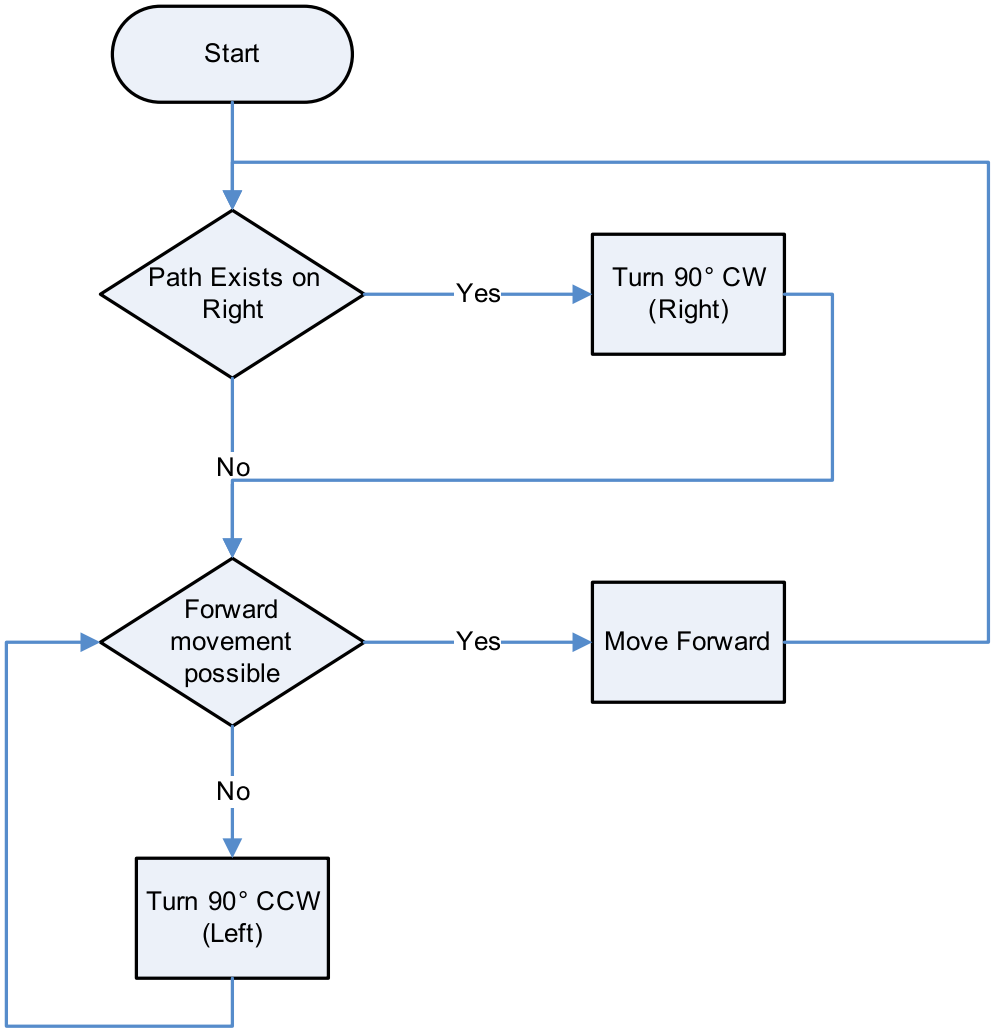


The circuit above shows how the different components are assembled together. The servos and IR sensors require 5V to operate and are, thus, connected to the 5V output of the microcontroller. Signal wires from the sensors go into the analog inputs of the microcontroller, while the servos are connected to the digital output. Once the servers are modified, their rotation speeds can be controlled via Pulse Width Modulation (PWM), in which the speed of the servo is dependent on the width of the pulse. To power the circuit, two 9V batteries are placed in series along with a switch used to turn the circuit on/off. Also, two capacitors are placed in parallel with the batteries: a large one to store charge and maintain a constant flow of current to the microcontroller, and a small one to filter out high-frequency disturbances.



Software

Flow Chart



Source Code

```
/*

 129 COMPUTER ENGINEERING SENIOR DESIGN PROJECT

 GROUP MEMBERS:
   - Tristan Muntsinger
   - Alan Nguyen
   - Tuan Dinh

*/

// Used mostly to debug the sensor data.
boolean debug = false;

// The following numbers can be anything, it doesn't matter. (Arduino can't do
enumeration)
int AHEAD = 1;
int FRONT = AHEAD;
int RIGHT = 2;
int LEFT = 3;
int BEHIND = 4;
int BACK = BEHIND;
```

```
int CW = 5;
int CCW = 6;
int SLOW = 7;
int MED = 8;
int FAST = 9;
int STOP = 10;

// Pins
int sensorPinRight = 0; // Right Sensor Pin
int sensorPinFront = 1; // Front Sensor Pin
int sensorPinLeft = 2;  // Left Sensor Pin
int sensorPinBack = 3;  // Back Sensor Pin
int servoPinLeft = 2;   // Left Servo Pin
int servoPinRight = 3;  // Right Servo Pin
int ledPinLeft = 4;     // Left Tail Light Pin
int ledPinRight = 5;    // Right Tail Light Pin

// Input (Sensor) Data
int valFront = 0;       // Front Sensor Data
int valRight = 0;       // Right Sensor Data
int valLeft = 0;        // Left Sensor Data
int valBack = 0;        // Back Sensor Data

// Output (Servo) Data
int pulseL = 0;         // Amount to pulse the left servo
int pulseR = 0;         // Amount to pulse the right servo
long lastPulse = 0;     // the time in milliseconds of the last pulse we send
int refreshTime = 20;   // the time needed in between pulses
int CCWPulse[3] = {    // CCW Speeds
       1350            //      Slow
       1100            //      Med
       1100};          //      Fast
int CCWPulse[3] = {    // CW Speeds
       1585            //      Slow
       1625            //      Med
       1653};          //      Fast

// Debug variables
float avg1 = 0;
float avg2 = 0;
float avg3 = 0;
float avg4 = 0;
int count = 0;

// Other Data
int i = 0;             // Used for various loops

// Reads the sensors multiple times, takes the average and stores the data.
void readSensors() {
  valFront = analogRead(sensorPinFront);
  valRight = analogRead(sensorPinRight);
  valLeft = analogRead(sensorPinLeft);
  valBack = analogRead(sensorPinBack);

  valFront += analogRead(sensorPinFront);
  valRight += analogRead(sensorPinRight);
  valLeft += analogRead(sensorPinLeft);
  valBack += analogRead(sensorPinBack);

  valFront += analogRead(sensorPinFront);
  valRight += analogRead(sensorPinRight);
  valLeft += analogRead(sensorPinLeft);
  valBack += analogRead(sensorPinBack);

  valFront += analogRead(sensorPinFront);
  valRight += analogRead(sensorPinRight);
  valLeft += analogRead(sensorPinLeft);
  valBack += analogRead(sensorPinBack);

  valFront /= 4;
  valRight /= 4;
  valLeft /= 4;
```

```
  valBack /= 4;

  if (debug) {
    count++;

    avg1 = (valLeft + (avg1 * (count - 1))) / count;
    avg2 = (valRight + (avg2 * (count - 1))) / count;
    avg3 = (valFront + (avg3 * (count - 1))) / count;
    avg4 = (valBack + (avg4 * (count - 1))) / count;

      Serial.print("Left = ");
      Serial.println((int)avg1);
      Serial.print("Right = ");
      Serial.println((int)avg2);
      Serial.print("Front = ");
      Serial.println((int)avg3);
      Serial.print("Back = ");
      Serial.println((int)avg4);
      Serial.println("");
  }
}


// Finds which path to take based on the current position of the car
// Calls takePath() to take the path it found.
void findPath() {
  // We want to make sure it's a path we see and not a hole in the wall
  if (path(RIGHT) && path(BEHIND)) {
    // When we see a path, we need to move a little to have enough clearing for it
    while (i < 7) {
      i++;
      move();
      takePath();
    }
    i = 0;
    // Then we need to rotate
    rotate(RIGHT);
    // Then we need to move forward, so we don't detect the path we came from
    while (i < 85) {
      i++;
      move();
      if (path(FRONT)) {
        takePath();
      }
    }
    i = 0;
  } else if (path(AHEAD)) {   // if there's no right path, we move forward
    move();
  } else {                    // if there's no right or forward path, we turn left
    rotate(LEFT);
  }
}

// A simple function to check if there's a path based on the direction we're looking
boolean path(int dir) {
  readSensors();
  if (dir == FRONT)
    return (valFront < 240);
  if (dir == BEHIND)
    return (valBack < 155);
  if (dir == RIGHT)
    return (valRight < 175);
  if (dir == LEFT)
    return (valLeft < 175);
}

// Rotate the car in the given direction
void rotate(int dir) {
  // Turn the tail lights off
  digitalWrite(ledPinRight, LOW);
  digitalWrite(ledPinLeft, LOW);
```

```
  // Turning "ahead" is the same as not turning
  if (dir == AHEAD)
    return;
  // Turning "behind" is the same as a uturn or two left turns
  if (dir == BEHIND) {
    rotate(LEFT);
    rotate(LEFT);
  }
  // Turning "right"
  if (dir == RIGHT) {
    digitalWrite(ledPinRight, HIGH);  // turn on the blinker
    pulseR = pulse(CCW,SLOW);
    pulseL = pulse(CCW,SLOW);
    while (i < 26) {                    // Turn right for this many cycles
      i++;
      takePath();
    }
  }
  // Turning "left"
  if (dir == LEFT) {
    digitalWrite(ledPinLeft, HIGH);   // turn on the blinker
    pulseR = pulse(CW,SLOW);
    pulseL = pulse(CW,SLOW);
    while (i < 34) {                    // Turn left for this many cycles
      i++;
      takePath();
    }
  }
  delay(250);                          // A simple delay to see if it's turning 90 degrees
}

// A simple function to call the other pulse() function if we don't declare a speed
int pulse(int dir) {
    return pulse(dir,MED);
}

// Will return the correct pulsewidth based on direction and velocity
int pulse(int dir, int velocity) {
  if (dir == STOP || velocity == STOP)
      return 1500;
  if (dir == CW) {
    if (velocity == SLOW)
      return CWPulse[0];
    if (velocity == MED)
      return CWPulse[1];
    if (velocity == FAST)
      return CWPulse[2];
  }
  if (dir == CCW) {
    if (velocity == SLOW)
      return CCWPulse[0];
    if (velocity == MED)
      return CCWPulse[1];
    if (velocity == FAST)
      return CCWPulse[2];
  }
}

// Calls the other move() if we don't declare velocity
void move() {
  move(MED);
}

// Similar to rotate(); will determine a pulse to travel forward based on sensor data
void move(int velocity) {
  // We only want to move forward in this manner,
  // if we can rely on the right/back sensors for angle positioning
  if (path(FRONT) && !path(BEHIND) && !path(RIGHT)) {
    if (valBack + 40 < valRight) {          // If we're drifting to the right
      pulseR = pulse(CW,FAST);
      pulseL = pulse(CCW,SLOW);
    } else if (valBack + 40 > valRight) {   // If we're drifting to the left
```

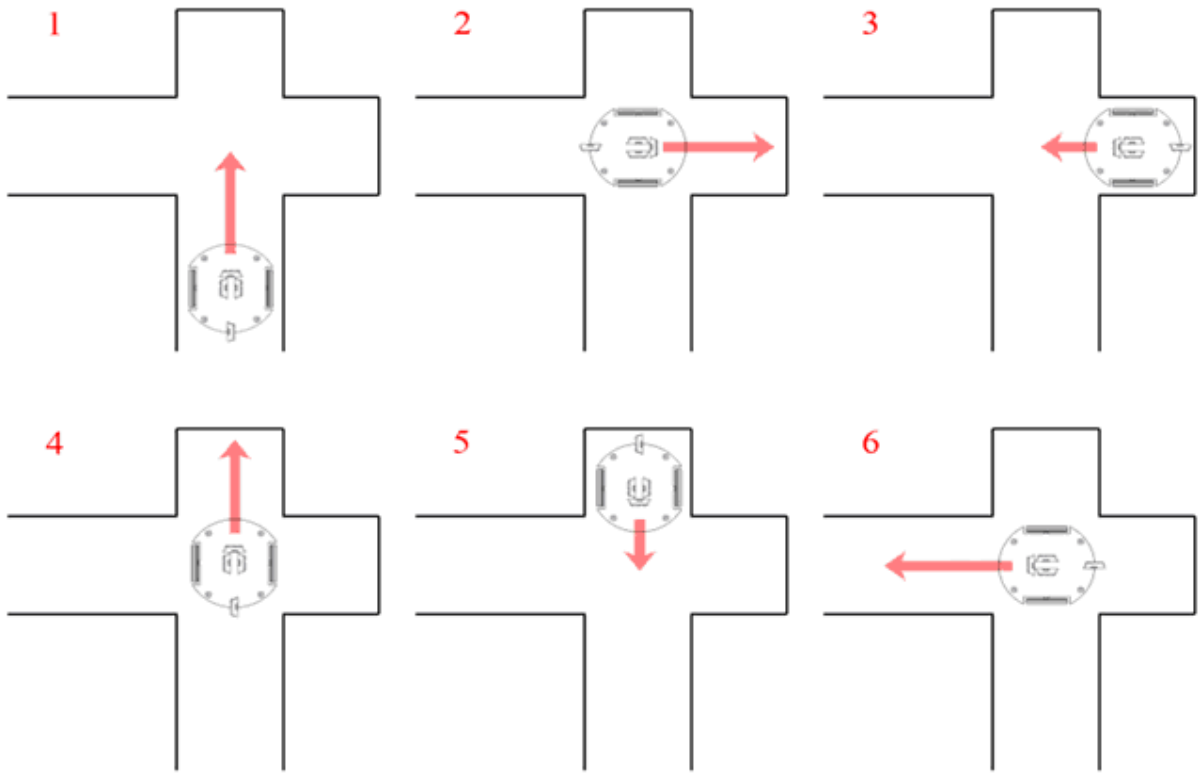
```
      pulseR = pulse(CW,SLOW);
      pulseL = pulse(CCW,FAST);
    } else {                              // If we're not drifting (shouldn't happen)
      pulseR = pulse(CW,velocity);
      pulseL = pulse(CCW,velocity);
    }
  } else if (path(FRONT)) {               // If we can't rely on walls to position us
properly
    pulseR = pulse(CW,velocity);
    pulseL = pulse(CCW,velocity);
  }
}

// This function will take the pulses and send them to the servos
void takePath() {
  // We want to delay sending the pulse until the right time
  while (millis() - lastPulse < refreshTime) {
    delay(1);
  }
  // The right time is when the pulse will actually do something to the servos
  if (millis() - lastPulse >= refreshTime) {
    digitalWrite(servoPinLeft, HIGH);    // Left Servo On
    delayMicroseconds(pulseL);           // Rotate Length
    digitalWrite(servoPinLeft, LOW);     // Left Servo Off
    digitalWrite(servoPinRight, HIGH);   // Right Servo On
    delayMicroseconds(pulseR);           // Rotate Length
    digitalWrite(servoPinRight, LOW);    // Right Servo Off
    lastPulse = millis();                // save the time of the last pulse
  }
}

void setup() {
  Serial.begin(9600);                    // open the serial port to send
  pinMode(servoPinLeft, OUTPUT);         // declare the left servo pin
  pinMode(servoPinRight, OUTPUT);        // declare the right servo pin
  pulseL = pulse(CW);                    // initialization
  pulseR = pulse(CW);                    // initialization
  delay(1000);                           // one second delay after we turn the car on
}

void loop() {
  i = 0;                                // initialization
  readSensors();                        // reads the sensor data
  if (!debug) {
    findPath();                            // finds the correct path
    takePath();                            // takes that path
  }
}
```

4. Design description

The microcontroller receives data from the four IR sensors, and determine whether the car is able to continue forward or make a turn. If at any time, there exists a path to the right of the car, it will rotate 90 degrees to the right and proceed down that path. It will continue down that path until it finds another right turn or until it cannot proceed forward any further. If the latter occurs, it will turn left and check again. Using this right-turn first priority scheme, the car should explore every possible path that will lead to the exit.

In order to verify that the car is moving parallel to the walls, the right and back sensor values are compared for equality. If the right sensor value is equal to the rear sensor value + some offset (since it is closer to the center than the right sensor), the car is parallel with the wall. Otherwise, it will adjust its direction of motion accordingly. Also, to ensure that the car is centered within the walls, the right and left sensor values are compared for equality. If one is larger than the other, the car will move in the direction of greater distance.

Once the correct course of action is determined, the appropriate signal is sent to each of the two servos, which will rotate at a speed and direction based on the pulse width of the sent signal. Sensor readings and car movements occur simultaneously.



5. System test plan

Test Number	Description of Set-up	Input or Stimulus	Expected Behavior
1	Connect microcontroller to computer via USB cable	Send commands to make car turn right, turn left, and move straight	Car will perform the appropriate task.
2	Using 9V batteries. Single hallway w/ width slightly larger than the car	Place car at one end of the hallway	Car will move down the hallway without hitting the walls
3	Using 9V batteries. Single-pathed (does not split) maze with right/left turns	Place car at one end	Car will reach the other end by avoiding walls and making turns
4	Using 9V batteries. Complex maze w/ intersections and dead-ends	Place car at maze entrance	Car will go out the exit or return to the entrance if there is no exit
5	Using 9V batteries. Random Complex maze created using removable walls	Place car in maze at random locations and facing random directions.	Car will adjust itself by assessing the environment and proceed to find the exit

6. Project timeline

Fall Quarter:

Chose project to work on. Created the initial design and schematic. Planned out what we were going to do and when.

Winter Quarter:

Week 1: Decided on what parts to buy. Bought the parts.

Week 2: Parts arrived, we started playing around with them.

Week 3: Work on chassis started.

Week 4: Car chassis completed. Started to put the car together.

Week 5: Servos, sensors, and tires all mounted onto car. Car completed.

Week 6: Testing various scenarios with the car. Achieved object avoidance and rotating.

Week 7: Maze finished. Maze testing started. Debugging started.

Week 8: More debugging. Achieved driving straight.

Week 9: More debugging. Achieved rotating properly.

Week 10: Made some final minor adjustments to the project. Demonstrated the project.

7. Division of work

Tristan Muntsinger

- System Planning
- Testing the Servos and Sensors
- Putting the Car Together
- Mounting the Servos, Sensors, and Wheels
- More Testing with the Complete Car
- Programming the Code
- Debugging & Analysis

Alan Nguyen

- System Planning
- Ordered Parts
- Testing the Servos and Sensors
- Putting the Car Together
- Mounting the Servos, Sensors, and Wheels
- More Testing with the Complete Car
- Programming the Code
- Debugging & Analysis

Tuan Dinh

- System Planning
- In charge of mechanical aspects of project
- Construction of the Car Chassis
- Purchased the Parts for the Maze
- Construction of the Maze

8. Cost

Part	Unit Cost	Number of Units	Total (after tax+s/h)
1/4" High Density Polyethylene Sheet	$8.33	2	$22.70
Tamiya ball caster kit	$6.00	2	$16.95
HS-322HD Servo	$9.99	2	$25.93
Sharp GP2D120 IR Sensor	$12.50	4	$54.95
9V Battery	$6.50	2	$13.00
Soldering Kit + Capacitors + Switch	$24.92	1	$24.92
Servo-mountable Wheels	$7.68	2	$15.36
Plastic Glue	$4.84	1	$4.84
Velcro	$3.87	1	$3.87
1/2" Thick sheets of wood (various sizes)	$40.00	1	$40.00
Arduino Microcontroller	$34.95	1	$39.95

Total cost of project: **$262.47**

9. Problem encountered and comments

Problem: Our initial idea for the sensors was to have a sensor mounted on a servo, where it would swing around to detect walls and paths. However, this seemed like a terribly complicated solution that would cause us a lot of problems.
Solution: We simply decided to use multiple sensors that were stationary. This way we could easily differentiate between car movement and sensor movement (obviously).

Problem: The servos we got were meant to rotate to a given angle, not rotate continuously as we wanted. A servo controlled tire would have to rotate continuously, so this obviously wouldn't do.
Solution: We modified the servos to rotate continuously. This was much easier than we thought it would be.

Problem: We ordered sensors that detected objects at farther distances, when we wanted them to detect objects closer to the car.
Solution: We couldn't figure out how to use the sensors we ordered any differently, so we simply had to exchange them for the proper sensors.

Problem: The servos could rotate counter-clockwise (CCW) and clockwise (CW), however the maximum speed for rotating CW and CCW were different speeds. This made it difficult to ensure that the car drives straight (because that's the only time the servos would be rotating in different directions).
Solution: We decided to use a feedback loop. Basically, if the car was drifting to the left, we would force the left tire to move fast and the right tire to move slow. Vice versa, if

the car was drifting to the right. This ended up working much better than we thought it would.

Problem: The servos/tires would sometimes speed up and slow down. We noticed this only when the batteries were connected, not with the USB power.
Solution: We figured that the batteries delivered current with minor fluctuations. We decided to put a small capacitor in parallel with the batteries to fix any current fluctuations. This worked quite well.

Problem: Throughout the life of the batteries, we noticed a big power problem. At the beginning, the batteries would drive the car really fast. However, as the battery life dwindled, so did the speed of the servos. We foresaw this to be a big problem later on, if we didn't fix it.
Solution: We decided to use a really large capacitor in parallel for purposes of power storage. If the batteries started dying, we'd still have the large capacitor to deliver the current we needed. Therefore, we'd have a steady current, no matter how dead the batteries were (unless they were completely dead, and the capacitor was drained, at which point we'd just replace the batteries).

Problem: The wheels we chose for the car were terrible. They had very little friction, they were small, and we had a lot of difficulty trying to mount them onto the servos.
Solution: We ordered special “servo-mountable” tires. They were much thinner, much bigger, and had much more friction. Since they were bigger, we had to cut out space on the chassis to accommodate for them. Fortunately, the amount of torque delivered by our servos was enough for the new wheels (since they are about twice the diameter of the old ones).

Problem: We weren't sure how to mount the arduino and various electronics on the car.
Solution: We were planning on using a perfboard, but we realized that wouldn't be much better than just running the wires straight into the arduino. We also decided to “mount” the arduino, by placing it in a cardboard “case” which we constructed out of an empty hot cocoa box and then mounted on the car with velcro.

Problem: The values we read from the sensors weren't always accurate. Not only did the sensors vary within about 10% of the numbers it was sending back to the arduino, but it also had “spikes” where the values would be wildly off. Although this rarely happened, it was causing us problems.
Solution: What we ended up doing in the end was polling the sensors multiple times (very quickly) and taking the average value. If there was a “spike” it would probably throw off the average quite a bit, but it would quickly do another “average polling” and should correct itself (which it did).

Problem: At some point, the car wasn't driving straight.

Solution: The only reason why it wouldn't be driving straight is if the mappings we did from the CCW motion to the CW motion were incorrect. If that was the case, then our system would vary, which is really bad for us. We tried different mappings, and it worked. However, we didn't want our system to change, so we kept this as an ongoing issue for a while. We later figured out that this was due to a power issue with the batteries vis-a-vis the USB power.

Problem: The car would have trouble going over cracks in the maze.
Solution: We weren't sure what to do about this beside sanding down the maze. We tried tape, but that didn't seem to work. So we held off for later. Later on, we still used the tape, and after we fixed a traction issue we had, it also solved this crack issue.

Problem: The car wouldn't turn the right amount.
Solution: We had two options. We could make sure it was "straight" based on the right and back sensors, but if there wasn't a wall to reference, that wouldn't work. The worst situation would be a 4-way intersection, where we'd have no reference except the corner walls. We figured the corner walls would be much too complicated to read sensor values off of, so we decided to do a certain number of cycles until it looked like the right amount to make it turn exactly 90 degrees. We figured out the number of cycles for each turn, used that in our program and hoped it would be consistent throughout. It turned out to work just fine, so we kept it.

Problem: When the car turned right it would detect a path on the right again (the path it came from) and try to take that route.
Solution: This was a bit tough, mainly because it was polling the sensors constantly. So we basically recoded the "right turn" as a "right turn, then move forward for a certain number of cycles". We had to determine the cycles the same way we determined how much to turn right and left for, and it worked, so we kept it.

Problem: The car would sometimes detect a wall as a path and a path as a wall.
Solution: This was one of the most annoying issues, but it turned out to be an easy fix in the end. Basically, what we had before was a threshold, where anything above that value would be a wall, and anything below was a path. We tested this by placing the car about where we wanted it to detect a wall. However, we weren't accounting for error. The thing about walls and paths that we didn't think about before was that detecting a path when there was actually a wall is really bad, because we will definetely crash. If we detect a wall, when there's actually a path, we're okay, because a few microseconds later we'll poll the sensors again, and if there's a path this time, we can still take it. And since it's only a few microseconds later, we don't miss a path we should have taken.

Problem: The 9V alkaline batteries we were using were draining very quickly.
Solution: We were going to solve this issue with NiMH batteries. So we got a 4-pack of AA NiMH batteries. However, for whatever reason, AA NiMH batteries are only 1.2V

each (whereas AA alkaline batteries are 1.5V each). This issue with this is that we needed 5V for our system. Four AA alkaline would provide 6V, where four AA NiMH would provide 4.8V. We tried out the 4.8V anyway, just to see if the devices were sensitive enough for it to matter, and it turns out they were. Since this was a minor issue (as we could just replace the 9V batteries every time they died), we only decided to fix it if we had the time (which we didn't).

Problem: The tires didn't have enough traction, so they were slipping when we were driving straight and more importantly when we were making turns. Usually this wouldn't be an issue, but since we decided on a certain number of cycles to make a turn, this was a big problem. If the tire slipped at all on a turn (which is where it usually slipped), it wouldn't turn enough and crash into a wall.
Solution: We decided that there were two solutions. The first being increasing the traction on the tires themselves and the second being increasing the friction on the wooden maze. We couldn't think about how to do the former, but we realized that for the latter we could use sandpaper. However, we couldn't find that much sandpaper at the time, and it would take a while to glue it all down to our maze, as well as redo all the holes. We didn't have that much time, especially for a solution that may not even work. We decided to go to the local park, steal sand, and throw it onto our maze. This was a ridiculous solution, but we were tired. Obviously, this didn't work. We then realized we could just put rubber bands onto the tires to increase friction. That actually ended up working quite well, and would have saved us a half hour trip to the kiddie park if we realized it sooner.

Problem: The tail lights that we had on the car initially were mostly used for debugging purposes, but we decided to keep them on as they were kinda cool. Unfortunately, nearing the end of the project, they stopped working.
Solution: We tested them in different ways, but we couldn't figure out why they weren't working. We realized they were just aesthitically pleasing anyway and not very practical, so we simply decided to remove them.

Course Comments:

If this is called a “Computer Engineering Senior Project course”, that is what it should be focused on—nothing more. The first quarter of the course turned out to be “Lectures on Contemporary Engineering Issues and How to Design/Implement Engineering Projects”. The lectures should be another class entirely (if included in the curriculum at all). How to take a design and turn it into a physical prototype should also be another class (although very valuable). We also feel the whole course should be spread out over the whole senior year. This way, we have a whole year in a class dedicated to nothing more than working on your project. Instead we were restricted to a quarter to finish a project that isn't very interesting simply due to time constraints. We came up with many ideas for the senior project, but almost all of them were “too grandious” in design (but we’re confident we could have done most of them if given the

entire academic year). Furthermore, the course was only worth 2 units, when much more work was required than is expected out of a 2 unit course. We also feel the course could have been organized better. Instead of spending class lecture times in a huge lecture hall learning about contemporary issues, we could have spent that time meeting with the professor about our project and our progress. While we did learn a lot taking this class, most (if not all) the engineering-related learning that took place was learning we did by ourselves. The contemporary issues were not very informative and we did not learn anything we didn't already know about.