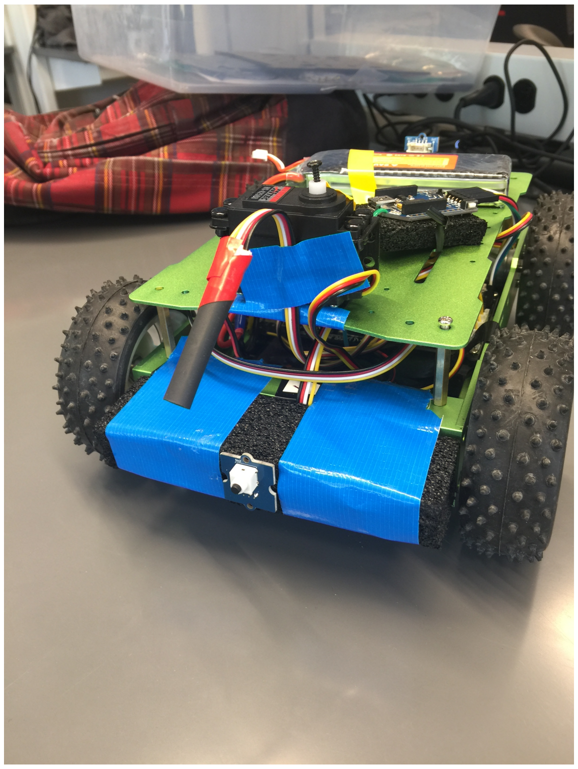
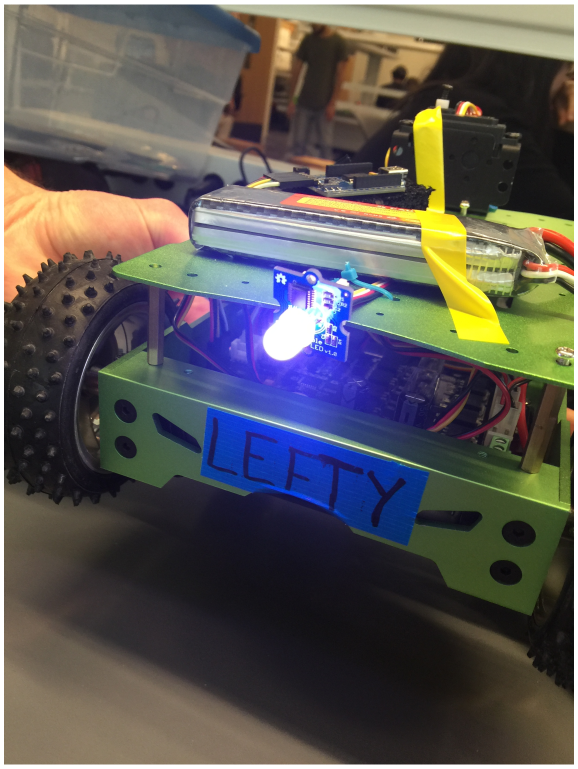
Robert Florence
Stormon Force
5-8-17

Lefty

Intro

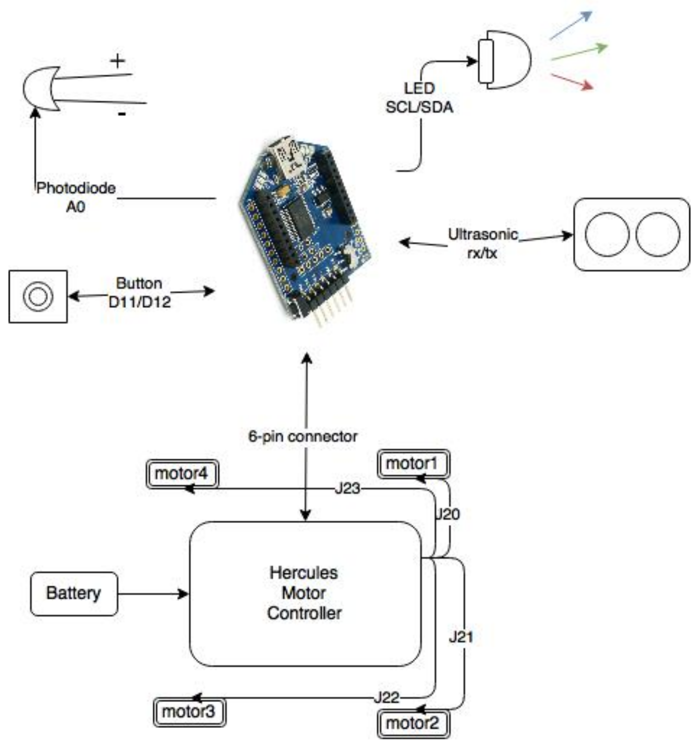
The basis of our semi-autonomous mobile robot was a Seeed Studio Hercules 4WD mobile robotic platform. This ten-week-long project was an incredible opportunity to learn about semi-autonomous robotics. This robot was modeled in a manner of a deterministic finite state machine (DFSM or FSM for short) of 'states' to determine its current behavior throughout a maze, avoiding obstacles and using phototaxis up the 'California incline'. The robot also used a combination of proprioceptive sensors and external environment sensors such as; motor encoders, physical push-button, a photodiode, and an ultrasonic sensor. For testing and style purposes, we also used an RGB Led to signify which state the robot was currently in.





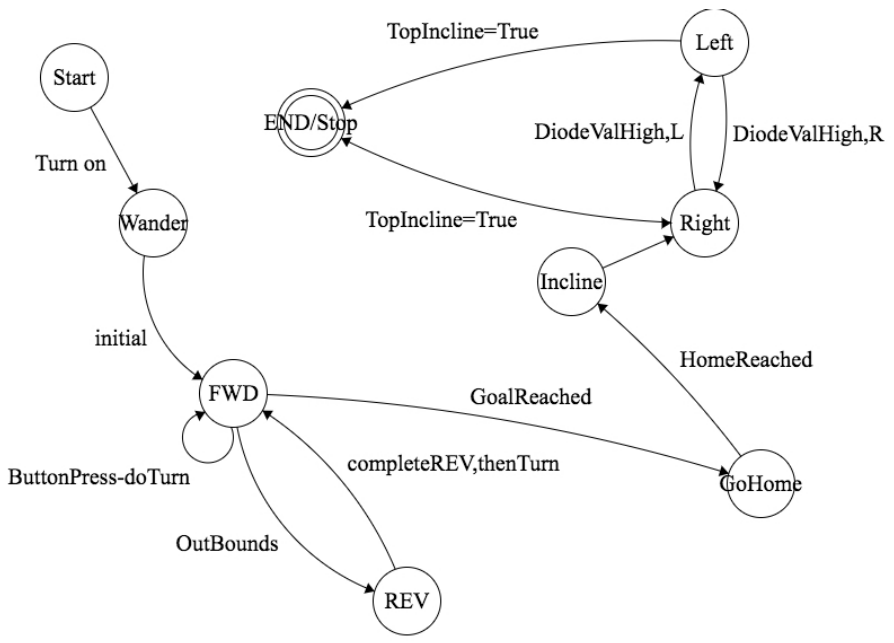
Robot

The most basic components of the robot was the Hercules motor controller that inter-connected all four electric motors and all the electrical components to an Arduino controller. The robot used a combination of proprioceptive sensors and external sensors such as; motor encoders, physical button, a photodiode, and an ultrasonic sensor. The main goal of the proprioceptive sensors (the motor encoders) on the robot was to perform 'dead reckoning', which is the process of calculating one's current position by using a previously determined position, and advancing that position based upon known or estimated speeds over elapsed time and direction. Using that idea, our goal was to build a virtual two-dimensional space, that represented a real 1.5m X 1.5m grid for the robot to traverse though. Ideally, give it a goal square in the grid, have it wander to the goal, then knowing a clear path back home, speed back. The push button was encased between two foam pads and mounted on the front on the robot sticking out farther than the wheels. When the button was pressed it would 'detect' obstacles in front of the robot, mark it within our grid, turn away from the obstacle, and avoid that grid square when it came back through on the return trip. After navigating through the maze, and avoiding obstacles on its return trip home, the next task was to use phototaxis up a ramp with a bright LED strip running up the center. Our robot did this by using a photodiode, which we had mounted/ hanging over the front of the body, like an angler fish uses to lure its prey in. Originally we had the photodiode mounted to the body of the ultrasonic sensor on the front of the robot, but when another group desperately needed a working ultrasonic, we unmounted and hung the photodiode from the front instead. The photodiode works in such a way that, the more light it receives it returns a smaller positive value, and conversely, the less light it receives it returns a value much higher. The values received are the amount of voltage allowed through the sensor depending upon external input.



Behaviors

As previously stated the robot modeled a finite state machine in order to control its behaviors. The goal for this project was to program a robot to Dead Reckon itself onto a physical grid in the real world and keep track of where it was virtually. Using that idea, we wanted to navigate a maze with obstacles, get to the goal square of (4,4), which was the top-most, right-most square in the grid, return home (0,0), and ascend an incline using photo taxis. The machine had three super states to represent its three behaviors we implemented; Wander, GoHome, Incline. Wanders job was to roam through the grid/maze, detect obstacles when it encounters them, turn away from them, and constantly maneuver itself to go toward the goal square. For example, if it hit the top boundary or the left boundary, it would turn right, because it was always our goal to go up and right. Wander also had two sub-states; FWD and REV. The robot would travel forward in FWD until it hit a boundary(then go to REV, reverse and turn), hit an obstacle or reached its goal. Using our heading angle, we would also determine which was the best way to turn. For example, if the robot was going right, turn left, and if we were going left, turn right. Every turn and the locations of said turns are pushed onto a stack and popped off the stack on the return trip home. Once we successfully reached the goal square, we would transition to the GoHome state and using the stack, trace our steps back. Because we already had an unbroken path from home to the goal, and we knew where the obstacles were, we just travel the same distance and turn in the opposite direction to go back home. Once we reached the home square, the robot oriented itself toward the incline and transition to the Incline state. In the incline state, there are two sub-states; Right and Left. It would automatically default to the Right state first, but then when it veers to far away from the light source it would switch over to the Left state and continue doing the same, oscillating up the ramp. The Left and Right state simply drove straight but, depending on the state(L or R) it would have one side drive faster than the other, this would turn the robot in one direction or the other. When the values get beyond a certain hard-coded threshold, it meant that it passed the end of light strip and it was at the top of the incline. This is the final, accepting state of the FSM, in which we stop.



Results/Conclusion

Lefty performed admirably in our two demo runs. He traversed to the goal, and returned home very near the original starting spot on both competition attempts. With slight directional adjustments, Lefty was able to ascend the incline and complete the task on both runs. I am pleased with the way that the final demo runs turned out as we scored within the top 4 teams in the class. Overall, this was an incredibly valuable learning experience for me as I had never had experience working with robotic components, sensors, or Arduino/motor controllers before this class.

Code

```
#include <motordriver_4wd.h>
#include <seeed_pwm.h>
#include <ChainableLED.h>

void markVisited(double x, double y);
void printGrid();
void markBlock(double x, double y);

//-------- PIE constants
const double PIE    = 3.14159265;
const double PIE_O2 = PIE/2.0;
const double PIE2 = PIE*2.0;

//------- Ultrasonic Sensor
const int pingPin = A0;
int Ping(int pingPin);
int dist_cm;

//-------- LED
#define NUM_LEDS 1
ChainableLED leds(SCL, SDA, NUM_LEDS);

//-------- motor control
void TurnLeft90();
void TurnRight90();
void Straight( int speed, int dirn );
void Wander();

//-------- bumper
const int buttonPin = 11;
int button_state = 0;

//-------- dead reckoning
// ticks per rotation
#define TPR 71

// robot measurements (mm)
#define RW    42.5  // radius wheel
#define D     158.0

// robot config variables
double x = 100.0, y = 100.0, dx = 0.0, dy = 0.0;
double theta =  PI/2.0;

// encoder variables
volatile long left_encoder_count = 0, right_encoder_count = 0;
int left_dirn = 1, right_dirn = 1;
```

```
//-------- robot states
enum {WANDER, GOHOME, INCLINE} state;
enum {FWD, REV} wander_state;

//-------- model of environment
double LEFT = 0.0;
double RIGHT = 1500.0;
double BOTTOM = 0.0;
double TOP = 1500.0;

//-------- Mapping Structure
const int granularity = 5;
const int rows = granularity;
const int cols = granularity;
int grid[rows][cols];

//-------- Stack Go home
char Sdir[rows * cols];
int Sxy[rows * cols * 2];
int Sindex = 0 ;

//-------- Light reading
enum {LEFT_incline, RIGHT_incline} stateIncline;
int lightPin = A0;
boolean topOfIncline = false;

//=====Setup=====
void setup(){
     for(int x = 0 ; x < granularity ; x++)
      for(int y = 0 ; y < granularity ; y++)
        grid[x][y] = 0;

    leds.init();
    MOTOR.init();
    attachInterrupt(0, LeftEncoder, CHANGE);
    attachInterrupt(1, RightEncoder, CHANGE);

   Straight( 10, 1 );
    leds.setColorRGB(0, 0, 100, 0);  // green
    wander_state = FWD;

    digitalWrite(lightPin, INPUT_PULLUP);
    Serial.begin(9600);
}

//==============LOOP==============
void loop() {
  int dist_cm = 51;
  //int dist_cm = Ping(pingPin);  Serial.println(dist_cm);
  if (dist_cm < 50){
    x = x + sin(theta)*dist_cm;
    y = y - cos(theta)*dist_cm;
    row = int(x/30);
    col = int(y/30);
    if (row >= 0 && row <= 5 && col >= 0 && col <= 5) {
      if (ogrid[row][col] <= 0.0) {ogrid[row][col] = 0.5;
}
      ogrid[row][col] += (1.0 - ogrid[row][col])/2.0;
    }
  }
  if( digitalRead(buttonPin) == 1)  {
    button_state = 1;
    markBlock(x,y);
  };
```

```
 //---- update robot config (x,y,theta)
 dx = PIE * RW * cos(theta) * ((double)(left_encoder_count + right_encoder_count) / TPR);
 x = x + dx;
 dy = PIE * RW * sin(theta) * ((double)(left_encoder_count + right_encoder_count) / TPR);
 y = y + dy;

 right_encoder_count = left_encoder_count = 0;
 markVisited(x,y);

if (state == WANDER){
   Wander();
   //---- check if we've arrived at goal cell
   if (x > 1400 && y > 1400){
    Straight( 0, 0 );
     delay(100);
     for (int i = 0; i < 4; i++){
     leds.setColorRGB(0, 100, 0, 0);  // red
     delay(100);
     leds.setColorRGB(0, 0, 100, 0);  // green
     delay(100);
     leds.setColorRGB(0, 0, 0, 100);  // blue
     delay(100);
     leds.setColorRGB(0, 100, 100, 0);  // yellow
     delay(100);
     leds.setColorRGB(0, 100, 0, 100);  // purple
     delay(100);
   }
    state = GOHOME;
     leds.setColorRGB(0, 100, 0, 100);    //purple
     TurnLeft90();
     theta  = fmod(theta + PIE_O2, PIE2);
     delay(500);
     TurnLeft90();
     theta  = fmod(theta + PIE_O2, PIE2);
     Straight(20,1);
     Sindex--;
   }
 }else
 if (state == GOHOME){
   goHome();
  //Straight( 0, 0 );
   //delay(100);

   if(x < 300 && y < 200){
       state = INCLINE;
       Straight(0,0);
       delay(200);
       if(theta == (3*PIE)/2)
         TurnRight90();

       leds.setColorRGB(0, 100, 100, 0);  // yellow
       delay(2000);
      }
 }else  if (state == INCLINE){
   if(!topOfIncline)
     inclineLight();
 }
}
void PushToStack(char direction){
     int col = (int)(x / 300);
     int row = (int)(y / 300);
     Sdir[Sindex]  = direction ;
     Sxy[2 * Sindex]   = col;
     Sxy[( 2 * Sindex) + 1] = row;
     Sindex++;
}

//==============Wander()==============
void Wander(){
 if (wander_state == FWD) {
   if (button_state == 1){
     Straight( 0, 0 );
     delay(100);
    leds.setColorRGB(0, 100, 0, 0);  // red
     Straight( 10, -1 );
     delay(500);
     button_state = 0;
    wander_state = REV;
   } else
   if ((wander_state == FWD) && (x >= RIGHT || x <= LEFT || y >= TOP || y <= BOTTOM)) {

    Straight( 0, 0 );
    delay(100);
    leds.setColorRGB(0, 100, 0, 0);  // red
     Straight( 10, -1 );
     delay(500);
    wander_state = REV;
   }
 }else if ((wander_state == REV) && (x < RIGHT && x > LEFT && y < TOP && y > BOTTOM)) {
     Straight( 0, 0 );
     if (theta <= 0.0 && theta >= -PIE_O2) {
     PushToStack('r');                          //opposite because we'll be facing other direction
     leds.setColorRGB(0, 0, 0, 100);
     TurnLeft90();
     theta  = fmod(theta + PIE_O2, PIE2);
     delay(100);
   }
   else{
     PushToStack('l');
     leds.setColorRGB(0, 0, 0, 100);
     TurnRight90();
     theta  = fmod(theta - PIE_O2, PIE2);
     delay(100);
   }
   leds.setColorRGB(0, 0, 100, 0);  // green
   Straight( 10, 1 );
   wander_state = FWD;
 }
}
 //==============Ultrasonic Sensor ==============
int Ping(int pingPin){
 long duration, cm;
 pinMode(pingPin, OUTPUT);
 digitalWrite(pingPin, LOW);
 delayMicroseconds(2);
 digitalWrite(pingPin, HIGH);
 delayMicroseconds(5);
 digitalWrite(pingPin, LOW);

 pinMode(pingPin, INPUT);
 duration= pulseIn(pingPin, HIGH);
 cm = ((duration/29)/2);
 //delay(100);
 return cm;
 }

//==============Incline Light()==============
void inclineLight(){
 int val = analogRead(lightPin);
 Serial.println(val);

 if(val>950){
   delay(1200);
   Straight(0,0);
   topOfIncline = true;
   return;
 }

 if (val > 100){
   if (stateIncline == LEFT_incline){
     MOTOR.setSpeedDir1(22, DIRF);
     MOTOR.setSpeedDir2(14, DIRR);
     stateIncline = RIGHT_incline;
     leds.setColorRGB(0, 100, 0, 0);                    // red
     delay(1000);
   }else if (stateIncline == RIGHT_incline){
     MOTOR.setSpeedDir1(14, DIRF);
     MOTOR.setSpeedDir2(22, DIRR);
     stateIncline = LEFT_incline;
     leds.setColorRGB(0, 100, 100, 100);                // white
     delay(1000);
   }
 }
}
//==============GoHome ==============
void goHome(){
 double midX = (int)x % 300;
 double midY = (int)y % 300;

 int col = (int)(x / 300);
 int row = (int)(y / 300);
 int midLimit1 = 120;
 int midLimit2 = 180;

 if(col == Sxy[2 * Sindex] && row == Sxy[(2 * Sindex) + 1])
 if( (midX > midLimit1 && midX < midLimit2) || (midY > midLimit1 && midY < midLimit2) ){
   Straight(0,0);
   delay(100);
   if(Sdir[Sindex] == 'r'){
       TurnRight90();
       theta  = fmod(theta - PIE_O2, PIE2);
     } else {
       TurnLeft90();
       theta  = fmod(theta + PIE_O2, PIE2);
     }

     if(Sindex >= 0)
       Sindex--;
     Straight(20,1);
 }
}

void markBlock(double x, double y){
 int col = (int)(x / 300);
 int row = (int)(y / 300);

 double thetaConvert = 0;
 if(theta<0){
   thetaConvert = theta + PIE2;
 } else {
   thetaConvert = theta;
   }
 if(theta == 0.00){
    col++;
 }
 if(thetaConvert <= 1.7 && thetaConvert >= 1.4){
   row++;
 }
 if(thetaConvert <= 3.3 && thetaConvert >= 3.05){
   col--;
 }
 if(thetaConvert <= 4.85 && thetaConvert >= 4.6){
   row--;
 }
 if(row >= 0 && col >= 0)
   grid[row][col] = 2;
}

void markVisited(double x, double y){
 if(grid[(int)(y / 300)][(int)(x / 300)] == 0){
 grid[(int)(y / 300)][(int)(x / 300)] = 1;
 }
}

//==============Print Grid==============
void printGrid(){
   for(int z = granularity - 1 ; z >= 0 ; z--){
       for(int w = 0  ; w < granularity ; w++)
       {
         Serial.print(grid[z][w]);
         Serial.print(" ");
       }
     Serial.println();
     }
}
//==============TurnLeft90==============
void TurnLeft90(){
   right_encoder_count = left_encoder_count = 0;
   left_dirn = -1; right_dirn = 1;
   MOTOR.setSpeedDir1(32, DIRR); MOTOR.setSpeedDir2(32, DIRR);
   while (right_encoder_count < 64){
     delayMicroseconds(1);
   }
   MOTOR.setSpeedDir1(0, DIRF); MOTOR.setSpeedDir2(0, DIRR);
}

//==============TurnRight90==============
void TurnRight90(){
   right_encoder_count = left_encoder_count = 0;
   left_dirn = 1; right_dirn = -1;
   MOTOR.setSpeedDir1(32, DIRF); MOTOR.setSpeedDir2(31, DIRF);
   while (left_encoder_count < 64){
     delayMicroseconds(1);
   }
   MOTOR.setSpeedDir1(0, DIRF); MOTOR.setSpeedDir2(0, DIRR);
}

//==============Straight()==============
void Straight( int speed, int dirn )
{
   left_dirn = dirn; right_dirn = dirn;
   if (speed == 0) {
     MOTOR.setSpeedDir1(0, DIRF);
     MOTOR.setSpeedDir2(0, DIRR);
     return;
   } else if (dirn == 1) {
     MOTOR.setSpeedDir1(speed, DIRF);
     MOTOR.setSpeedDir2(speed, DIRR);
   } else  {
     MOTOR.setSpeedDir1(speed, DIRR);
     MOTOR.setSpeedDir2(speed, DIRF);
   }
}
//==============Interrupt Service Routines for encoders==============
void LeftEncoder()
{
 left_encoder_count = left_encoder_count + left_dirn;
}
void RightEncoder()
{
 right_encoder_count = right_encoder_count + right_dirn;
}
```