THOMPSON RIVERS UNIVERSITY

VAULT TEC.

EPHYS 1990

Due date: April 9, 2016

Instructor: Mark Paetkau

UNMANNED WAREHOUSE VEHICLE

Department of Engineering

THOMPSON RIVERS UNIVERSITY

Kamloops, BC, Canada

Winter 2016

DECLARATION

We confirm that this project dissertation we are submitting has been solely done by us. Any materials used from other sources have been clearly identified and properly cited.

Name: Adebayo Ogunmuyiwa

Position: Team Manager

Signature:

Name: Peter Allen Clarke

Position: Software Engineer

Signature:

Name: Matthew Behan-Fossey

Position: Mechanical Engineer

Signature:

Name: Johnathan Schwarz

Position: Electrical Engineer

Signature:

Abstract

The Arduino Line Following robot project was a group project in teams of four. The robot has been designed to navigate a warehouse using a line following prototype at Thompson Rivers University. Each member of the team had specific sub-systems to concentrate on. Consequently, this report details the development of the Unmanned Warehouse Vehicle decision system. The decision system and the software programming was the top-aspect of the robot. The tasks it must accomplish can be roughly broken down into four components;

- Navigating broad 90 degrees curves, as well as 45 degree turns.
- An automated shutdown procedure when it detects the end of the line.
- Navigating broken lines in the warehouse.
- Navigating a line width with 25% tolerance.

Acknowledgements

During this semester long project several people provided many forms of support and help to the team. First and foremost we would like to thank Mr. Mark Paetkau (Project Supervisor) for his support and continuous guidance. Secondly we would like to thank our fellow course colleagues for their support and encouragements throughout this project.

Table of Contents

1.0 OVERVIEW

The Unmanned Warehouse vehicle operates as a line following robot as the name specifies. It uses a photo resistor sensor to follow a line by programming two motors to go straight, left and right along the line. Apart from sensing a line and sticking to it, the robot has to maneuver through set obstacles. Industrially, the line following robot concept can be applied to be used in transporting goods from one point to another.

Although this is a group project, the assessment is based on individual contribution, Hence each member of the team was given specific tasks which counts towards their assessments. These roles includes:

- Adebayo Ogunmuyiwa was responsible for the overall project and report management, organising the weekly meetings and providing a detailed review and summary of the testing stages and challenges.
- Matthew Behan-Fossey: was responsible for the mechanical design and the chassis assembling.
- Peter Clarke was responsible for the software programming and robotics logic
- Johnathan Schwarz was responsible for the electrical components, assembling and wiring of the robot.

1.1 OBJECTIVES OF THE ROBOT

The Robot must follow a line and must have the ability to stay on course in the event of any of the following obstacles:

- Curving path
- 45 and 90 degree turns

- Breaks in line
- T intersections

1.2 LIMITATIONS

- Calibration is difficult and requires the operator to set the robot on the line after calibration
- The steering mechanism requires a complex logic and strict logic controls
- The robot has to reset its position and rotate 360 degrees at a complex curve

1.3 APPLICATIONS

- Automated Cars
- Industrial automated equipment carriers

1.4 ROBOT CONCEPT

The robot requires three main processes for the microprocessor, microcontroller and the program to run. They are:

- Input: Read and calibrate the white/ black surface and condition the input signals.
- Process: based on the input values received, the microcontroller decides what change needs to be made to the robot's speed and direction
- Output: the newly adjusted control signal is sent to the motors

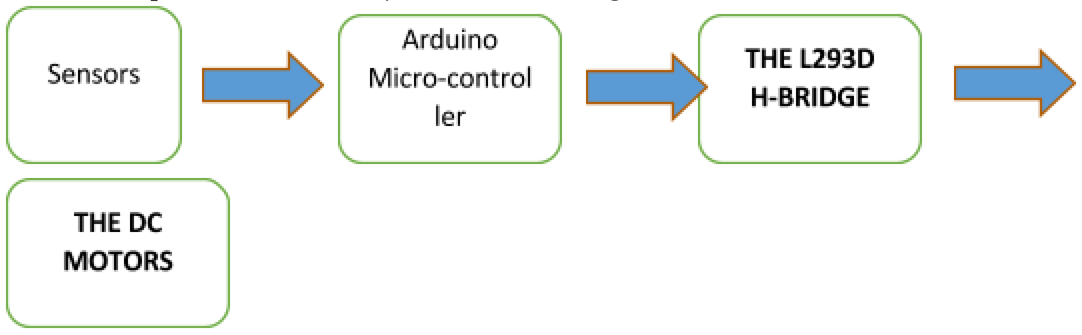


Figure 1: Basic Operation Diagram

As seen from the diagram above, the Pololu IR sensor detects the line as an input and sends the input value to the Arduino microcontroller .The microcontroller then sends the output to the H-bridge which relates the single to the motors. It is important to note that the Pololu sensor gives an analog signal which is converted by the Arduino to a digital logic.

2.0 COMPONENT LISTING

A variety of components can be used to build and control a line following system but in this project, the following components were used:

2.1 Arduino Uno board

The Arduino Uno is a microcontroller board with 14 digital input/ output pins, 6 analog inputs, a 16MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. The Arduino IDE is a programming language based on elements common to C/C++ and java.

Technical Specification

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by

	bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

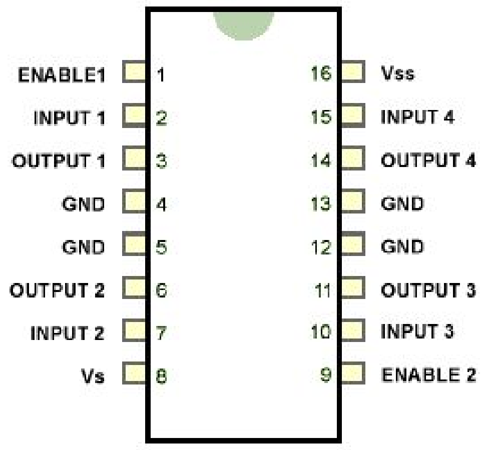
2.2 THE L293D H-BRIDGE

THE L293D Is a high voltage and high current four channel driver designed to accept standard DTL or TTL logic levels. Each pair of channels contains an enable input and a separate supply input is provided for the logic, allowing for operation at a lower voltage.

Technical Specifications

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Values	Units
V_s	Supply Voltage	36	V
V_{ss}	Logic supply Voltage	36	V
V_I	Input Voltage	7	V
V_{out}	Enable Voltage	7	V
I_o	Peak Output Current	1.2	A
P_{tot}	Total Power Dissipation	4	W
T_{stg}	Storage and Junction Temperature	-40 to 150	C



PIN CONNECTIONS (Top view)

2.3 THE QTR-8A REFLECTANCE SENSOR ARRAY

This sensor module is equipped with 8 IR LED/ phototransistor pairs spaced evenly at 0.375” intervals. Each phototransistor is connected to a pull-up resistor to form a voltage divider which produces an analog voltage output between 0V and the input voltage (5V from the Arduino in this case) as a function of the reflected IR.

Technical Specification

- Dimensions: 2.95" x 0.5"
- Operating voltage: 3.3-5.0 V
- Supply current: 100 mA
- Output format for the QTR-8A: 8 analog voltages ranging from 0 V to supplied voltage
- Output format for the QTR-8RC: 8 digital I/O-compatible signals that can be read as a timed high pulse

- Optimal sensing distance: 0.125" (3 mm)
- Maximum recommended sensing distance for the QTR-8A: 0.25" (6 mm)
- Maximum recommended sensing distance for the QTR-8RC: 0.375" (9.5 mm)
- Weight without header pins: 0.11 oz (3.1 g)

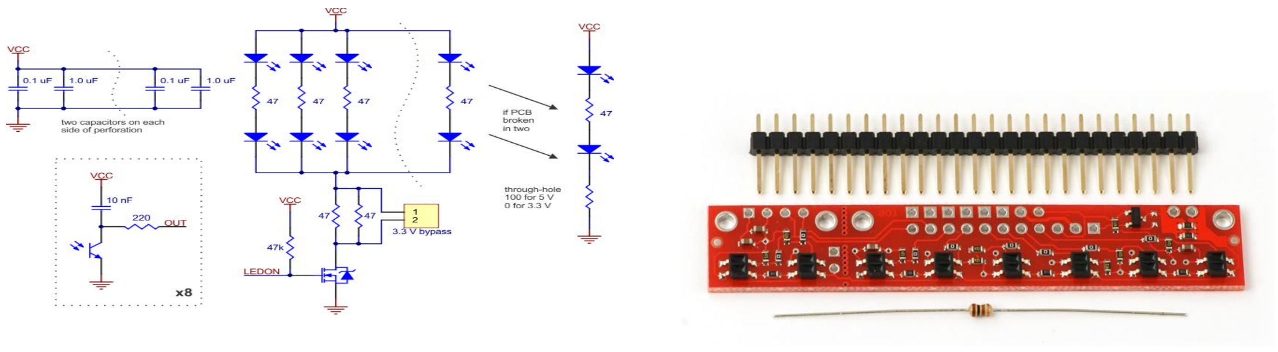


Figure 2: Schematic diagram and Image of the QTR-8A reflectance Sensor Array

2.4 THE GM2 224:1 GEAR MOTOR

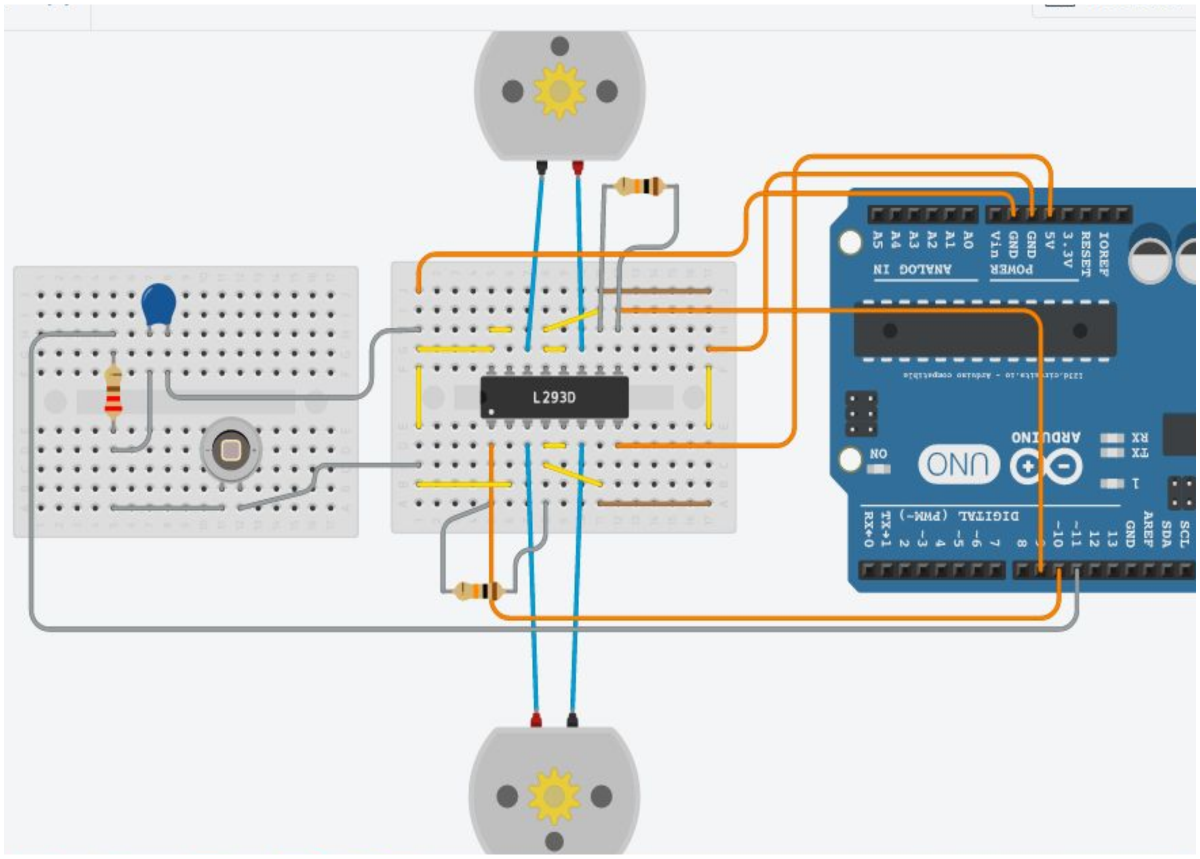
This is a low-current motor with a brushed DC motor within a gearbox. Each motor contains 2 leads and a 7mm in diameter output shaft, designed for the wheels used.

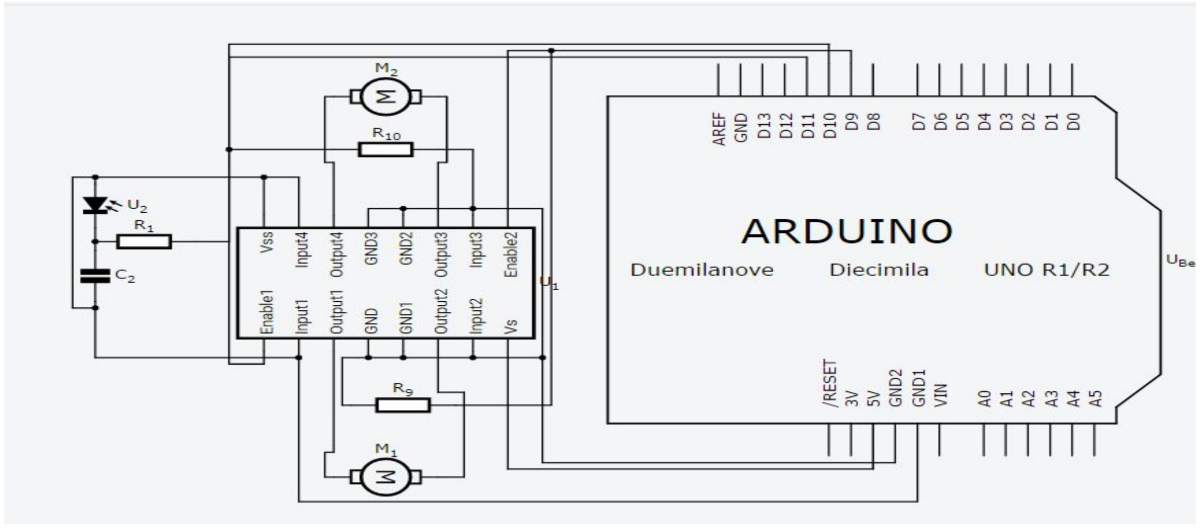
Technical Specifications

● Gear ratio:	224:1
● Free-run speed @ 6V:	46 rpm
● Free-run current @ 6V:	50mA
● Stall current @ 6V:	710mA
● Stall torque @ 6V:	57 oz*in

3.0 PROJECT DESIGN

3.1 CIRCUIT DIAGRAMS





Shown above is the electrical schematic of the robot. Only one sensor is being shown however 8 are connected, with seven more connecting to digital I/O pins 2-8 on the Arduino. The resistors connected to the ground of the H-Bridge serve the purpose of creating a delay to the robot's motors when plugged in, giving it time to be adjusted before moving. A 7.5V supply is required for the circuit as well, which is applied to the Arduino's input (5 AA's).

3.2.2 POWER CONSUMPTION

The measured current of the robot was 250mA, so then using the equation below we get 1.9W

$P = IV$

$P = (0.250A)(7.5V)$

$P = 1.9W$

Using this measured current rating and the life of AA batteries being 2000mA*H, we get the total robot battery life being approximately 8 hours ((2000mA*H)/ (250mA)). Though the actual run time is likely shorter, we found that the robot could operate during trials between 6-8 hours. The power consumption of each component can be calculated as well. Using figure 2, the current is calculated with the sensor is:

$I = V/R$

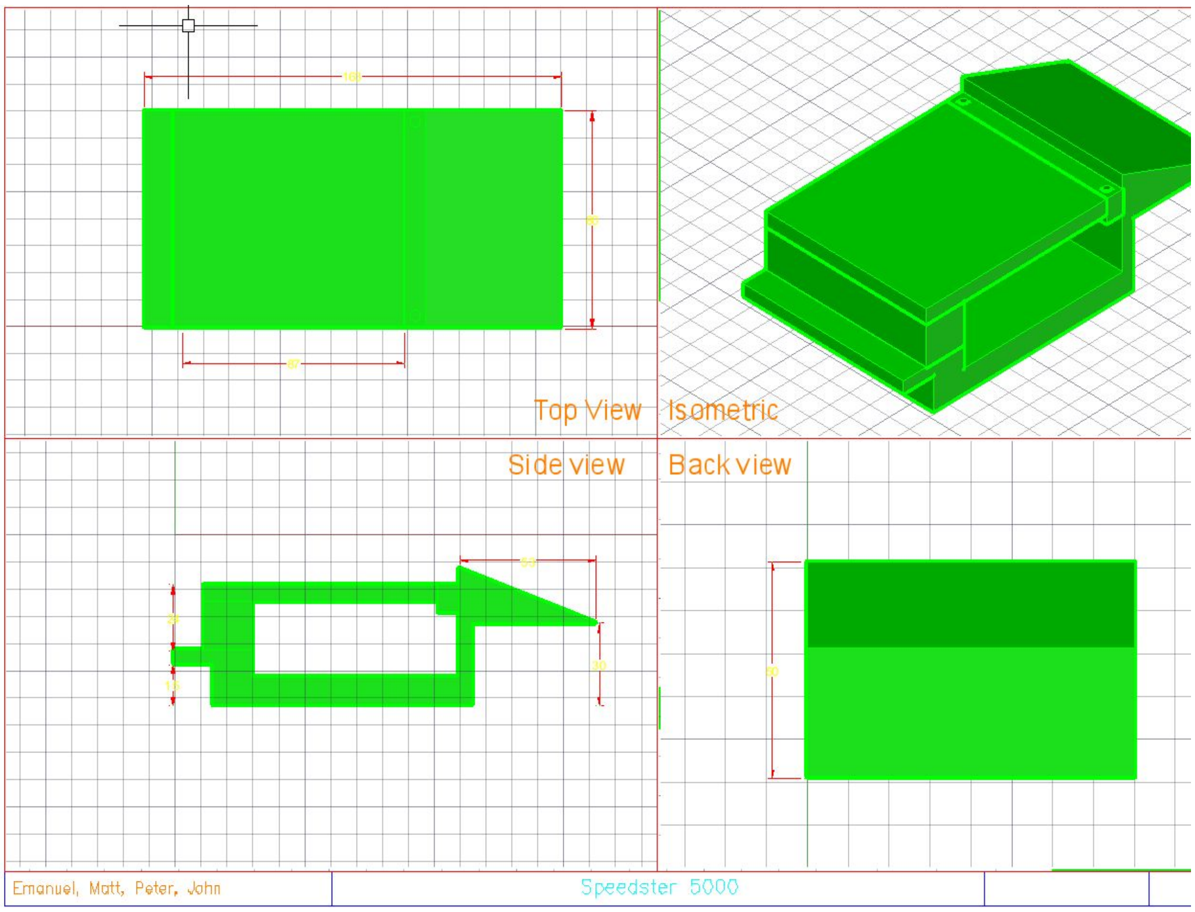
$I = (7.5V/35.25\Omega)$

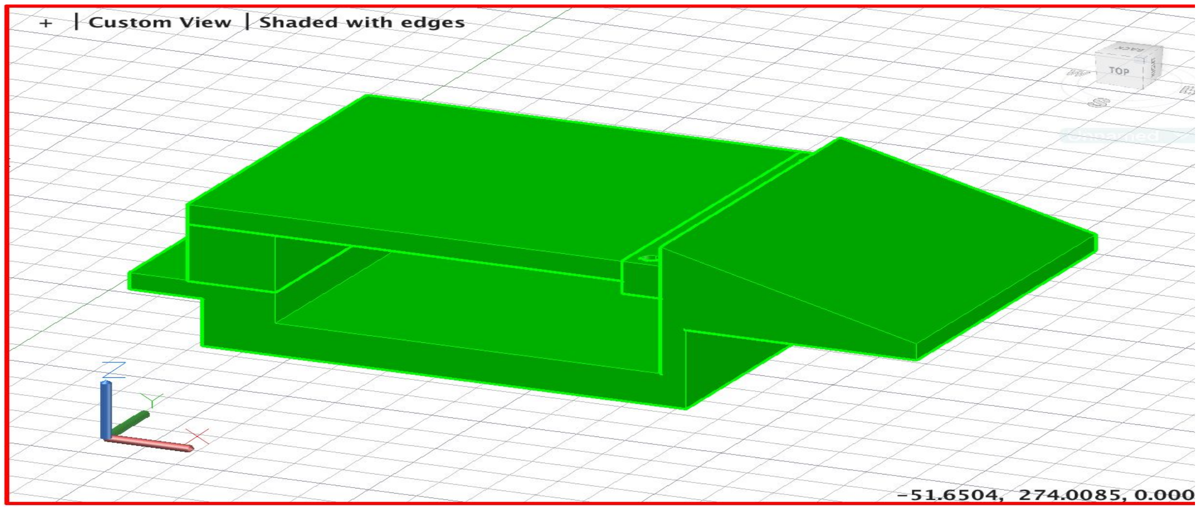
$I = 215mA$

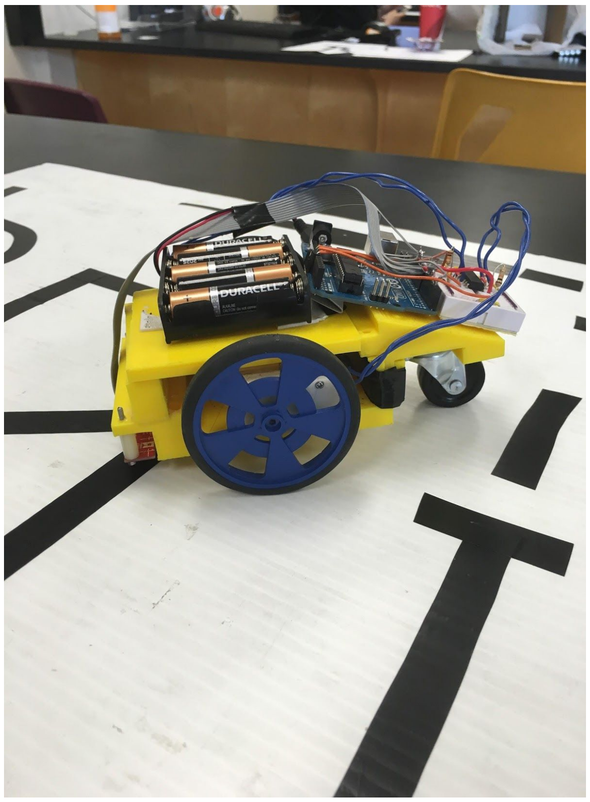
Along with this, the Arduino consumed approximately 80 mA and the motors each used about 65 mA in our case. Using the power equation yields the Arduino to be using 0.01W, the motors to be using 0.500W and the sensor to be using 1.6W, with a total power rating of 2.1W (0.2W away from the measured voltage).

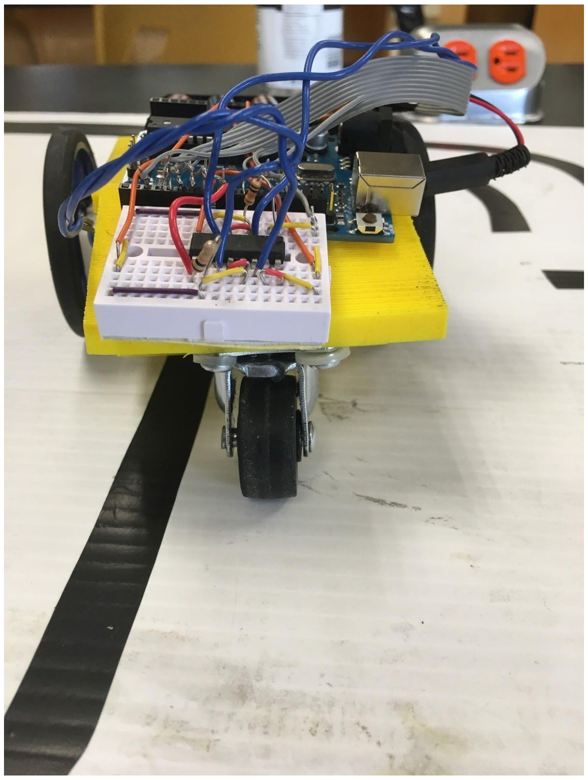
3.3 MECHANICAL DESIGN

The design of the robot went through three versions/ designs, each an improvement on the previous. It started off with a rough design (V1) that was presented to the group and revisions were suggested. V2 was made to better incorporate attachment of the motors and a miniature rotating wheel for the front of the bot. V3 included an enlargement of the whole robot, holes for convenient wiring, and overall reinforcement of weak points. Unfortunately, during finalization V1 was printed accidentally. The chassis was then modified by hand to most closely resemble V3 and improvisations were made, like putting the swivel wheel on the back of the robot, to ensure a functional robot. Finally, spacers were added to bring the sensor closer to the ground and springs were used to attach the swivel wheel.


Top View
Isometric
Side view
Back view
Emanuel, Matt, Peter, John
Speedster 5000


+ | Custom View | Shaded with edges
TOP
-51.6504, 274.0085, 0.000





3.4 SOFTWARE PROGRAMING AND ROBOT LOGIC

PSEUDOCODE for Arduino:

```
Include Pololu QTR Library
Define function constants
Declare pins that are connected to sensors
Declare Control Loop Variables/Constants
Setup Loop:
        Initialize Serial Monitor
        Set Control Pins to Outputs
        Begin Calibration
        Sensor Readings Loop:
                Robot Turns in a circle
Reads the sensors for calibration. The sensor values are not returned; instead, the
maximum and minimum values found over time are stored internally and used for
the “readCalibrated()” method
        End of Sensor Readings Loop
        Call on “readLine” function to return an estimated position of the line
        While middle sensor <200 read line value
        While line position is greater that 4350 continue loop until line position near
centre (read line values)
        End Calibration.
End Setup Loop

Begin Main Loop:
        Read Calibrated sensor values and obtain a measure of the line position from
0-7000
        Initialize Variables
        Calibrate Maximum and minimum sensor values (Loop):
                Return Maximum and minimum sensor values after eight readings
        End Loop
        Call on function “read_sensor” and initialize sensor variables respectively
If T junction read sensors 2 through 7 will read black. Sensor readings 1 and 8 are not
used due to light level discrepancies. If sensors 2 through 8 read black the robot will
stop, move forward and turn 360 degrees.
If there is a 45 degree turn the robot will read the outside sensor to be black, 2nd
sensor to the end as white, and the middle sensors as black turn towards outer sensor
that reads black. Once the middle two sensors are black again; continue turn for a
further 200ms to avoid losing line.
If there is a 90 degree turn, the middle sensors will both read black as well as an
outside sensor and the sensor inside of it will both read black. A turn will be initiated
```

```
towards the outer sensor that read black for 1s.
        Else default to regular line following mode
                Error = difference in line position from centre (3500)
                Algorithm attempts to correct the robot to the line over time by
calculating and
                avoiding future error
                Algorithm: [present values of error(PV)=kp*present error]
                           +[Possible future values of error =(kd*change in error)]
                If Present Value of error is larger than allowed (55) limit it to 55
                If Present Value of error is smaller than allowed (-55) limit it to -55
                Adjusted Motor speed = Current speed ±55
End of Main Loop

Function "read_sensor":
This function reads the sensor attached to the appropriate sensor channel (x)
Requires a long value which is the time taken to reach LOW
The less light provided, the more time it takes to reach LOW
The function works by first setting the DIO as an output at +5V for 5ms, discharging
the capacitor
DIO set as input to measure time it takes for the capacitor to charge and input line to
reach LOW
End of "read_sensor" function
```

ARDUINO CODE:

```
#include <QTRSensors.h>  // Pololu QTR Library
//line sensor defines
#define NUM_SENSORS 8     // number of sensors used
#define TIMEOUT 2500  // waits for 2500 microseconds for sensor outputs to go low
// line sensor declarations
// sensors 0 through 7 are connected to digital pins 2 through 8, respectively (sensor 1
is set to pin 11)
QTRSensorsRC qtrrc((unsigned char[]) {
  11, 2, 3, 4, 5, 6, 7, 8
}, NUM_SENSORS, TIMEOUT);
unsigned int sensorValues[NUM_SENSORS]; // array with individual sensor reading
values
unsigned int line_position = 3500; // value from 0-7000 to indicate position of line
between sensor 0 - 7
int dir_a = 9;  //direction control for Ardumoto outputs A1 and A2 is on digital pin 9
(Left motor)
int dir_b = 10;  //direction control for Ardumoto outputs B3 and B4 is on digital pin 10
(Right motor)
```

```
// motor tuning vars
int calSpeed = 165;   // tune value motors will run while auto calibration sweeping turn
over line (0-255)

// Proportional Control loop vars
float error = 0;
float PV = 0 ; // Process Variable value calculated to adjust speeds and keep on line
float kp = 0.15;  // // kp is the a floating-point proportional constant
int m1Speed = 0; // (Left motor)
int m2Speed = 0; // (Right motor)
float kd = 0.3;//kd is the floating-point derivative constant
int lasterror = 0;
int DEL = 65;//Speed-DEL (Turning)
int SPEED = 65;//Speed

//Define pins
#define S1 11
#define S2 2
#define S3 3
#define S4 4
#define S5 5
#define S6 6
#define S7 7
#define S8 8

//Setup Loop
void setup()
{
  Serial.begin(9600);//Initialize Serial Monitor

  //Set control pins to be outputs
  pinMode(dir_a, OUTPUT);
  pinMode(dir_b, OUTPUT);

  //set both motors to stop
  analogWrite(dir_a, 0);
  analogWrite(dir_b, 0);
  // delay to set the robot on the line
  delay(2000);
  // calibrate line sensor; determines min/max range of sensed values for the current
course
  // auto calibration sweeping left/right, tune 'calSpeed' motor speed at declaration
  // just high enough all sensors are passed over the line
  for (int i = 0; i <= 100; i++)
  {
    analogWrite(dir_a, 0);
```

```
   analogWrite(dir_b, 190);
 qtrrc.calibrate();// Reads the sensors for calibration. The sensor values are not
returned; instead, the maximum and minimum values found over //time are stored
internally and used for the readCalibrated() method
   delay(30);
  }
  // end calibration cycle
  // read calibrated sensor values and obtain a measure of the line position from 0 to
7000
  //this function returns an estimated position of the line.
 //The estimate is made using a weighted average of the sensors multiplied by 1000
  line_position = qtrrc.readLine(sensorValues);


  // read the value of only a single sensor to see the line.
  // when the value is greater than 200 the sensor sees the line.
  while (sensorValues[4] < 200)  // wait for line position to near center
  {
   line_position = qtrrc.readLine(sensorValues);
  }

  // find near center
  while (line_position > 4350) // continue loop until line position is near center
  {
   line_position = qtrrc.readLine(sensorValues);
  }

  // stop both motors
  analogWrite(dir_b, 0); // stop right motor first which kinda helps avoid over run
  analogWrite(dir_a, 0);

  // delay as indicator setup and calibration is complete
  delay(1000);

} // end setup



void loop() // main loop
{

  // read calibrated sensor values and obtain a measure of the line position from 0 to
7000
  //initialize Variables
 unsigned int line_position = qtrrc.readLine(sensorValues);
  unsigned int bmax = 1;
  unsigned int bigmin = 1100;
  float sum;
  float avg;
  int diff;
  int sensor1;
  int sensor2;
  int sensor3;
  int sensor4;
  int sensor5;
  int sensor6;
  int sensor7;
   int sensor8;
  //Calibrate Maximum and minimum sensor values
  for (int i = 0; i < 8; i++)
  {
   if(sensorValues[i] > bmax) {
     bmax=sensorValues[i];}
   if(sensorValues[i] < bigmin){
     bigmin = sensorValues[i];}
   sum = sum + sensorValues[i];
  }//end of for loop

  //Call on function "read_sensor"
   sensor1 = read_sensor(S1);
   sensor2 = read_sensor(S2);
   sensor3 = read_sensor(S3);
   sensor4 = read_sensor(S4);
   sensor5 = read_sensor(S5);
   sensor6 = read_sensor(S6);
   sensor7 = read_sensor(S7);
   sensor8 = read_sensor(S8);
   //If T junction read sensors 2 through 7 will read black. Sensor readings 1 and 8 are
not used due to
   // light level discrepancies. If sensors 2 through 8 read black the robot will stop,
move forward and turn 360 degrees.
   //In this case we are calling on our function "read_sensor"
    if ((sensor5> 900) && (sensor4 > 900) &&
(sensor3>900)&&(sensor6>900)&&(sensor7>900)&&(sensor2>900) ){
   analogWrite(dir_a, 0);
   analogWrite(dir_b, 0);
   delay(1000);
   analogWrite(dir_a, 100);
   analogWrite(dir_b, 100);
   delay(500);
   analogWrite(dir_a, 130);
   analogWrite(dir_b, 0);
```

```
    delay(4800);}

   /*For a 45 degree turn the robot will read the outside sensor to be black, 2nd sensor
to the end as white, and the middle sensors as black
  turn towards outer sensor that reads black. Once the middle two sensors are black
again;     continue turn for a further 200ms to avoid
   losing line*/
   if ((sensorValues[7] > 500 && sensorValues[6] < 50 && sensorValues[0] < 50 ) &&
(sensorValues[5] < 50 || sensorValues[4] < 50)) {
    delay(50);
     do{
    analogWrite(dir_a, 200);
    analogWrite(dir_b, 0);
    delay(1500);
    qtrrc.readCalibrated(sensorValues); //Re-read sensor values
    }while(sensorValues[4] < 700 && sensorValues[5] < 700);//greater that 700 works
awesomely
     analogWrite(dir_a, 200);
    analogWrite(dir_b, 0);
    delay(200);
   }
   if ((sensorValues[0] > 500 && sensorValues[1] < 50 && sensorValues[7] < 50) &&
(sensorValues[2] < 50 || sensorValues[3] < 50)) {
    delay(50);
     do{
    analogWrite(dir_a, 0);
    analogWrite(dir_b, 200);
    delay(1500);
    qtrrc.readCalibrated(sensorValues); //Re-read sensor values
   }while(sensorValues[4] <700 && sensorValues[5] < 700);// greater than 700 works
pretty nice
   analogWrite(dir_a, 0);
    analogWrite(dir_b, 200);
    delay(200);
   }

   //For 90 degree turn the middle sensors will both read black and once an outside
sensor and the sensor inside of
   //it both read black a turn will be initiated towards the outer sensor that read black
for 1s
   if ((sensorValues[0] > 900) && (sensorValues[3] > 900) && (sensorValues[7]>500 &&
sensorValues[6]>500)) {
    analogWrite(dir_a, 0);
    analogWrite(dir_b, 130);
    delay(1000);
   }
   if ((sensorValues[7] > 900) && (sensorValues[4] > 900) && (sensorValues[0]>500 &&
sensorValues[1]>500)) {
    analogWrite(dir_a, 130);
    analogWrite(dir_b, 0);
    delay(1000);
   }
  //If no special situations apply the robot will default to its regular line following mode
  else  {
    /*Regular follow mode  */

   error = (float)line_position - 3500; // 3500 is center measure of 7000 far left and 0 on
far right
    //alter the speeds
    //Calculation of PV: (present values of error=kp*present error)
    //         +[Possible future values of error =(kd*change in error)]
    //This essentially attempts to correct the robot to the line over time as the algorithm
    //attempts to avoid future error
    PV = kp * error + kd * (error - lasterror);
    lasterror = error;
    // this section limits the PV (motor speed pwm value)
    if (PV > DEL)
      PV = DEL;
    else if (PV < -DEL)
      PV = -DEL;
    // adjust motor speeds to correct the path
    // if PV > 0 the robot needs to turn left
    m1Speed = SPEED + PV;
    m2Speed = SPEED - PV;
    analogWrite(dir_a, m1Speed);
    analogWrite(dir_b, m2Speed);
  }

}//loop

//This function reads the sensor attached
//to the appropriate sensor channel (x)
//Requires a long value which is the time
//taken to reach LOW
//The less light provided, the more time it takes
//to reach LOW
//The function works by first setting the DIO as an
//output at +5V for 5ms, discharging the capacitor
//DIO set as input to measure time it takes
//for the capacitor to charge and input line to reach LOW
long read_sensor(int x){
  long To;
  //DIGITAL CHANNEL SET TO HIGH
  pinMode(x,OUTPUT);
  digitalWrite(x,HIGH);
  //Delay 1ms
  delay(1);
  //Dig Channel set to input
  pinMode(x,INPUT);
  //Time to go LOW measured
  To= micros();
  while (digitalRead(x)) {
    if ((micros()-To) > TIMEOUT)
      break;
  }
  //Return
  return (micros()-To);

}//END OF FUNCTION //
```

4.0 Conclusions and Review

During the course of this project much progress and update where made to the robotic logic system. The logic system was successfully implemented using proportional controls as explained above. This has helped in making the robot line route safe as the proportional controls keeps the robot centered to the line hence reducing risk of collision.
The proportional controls and the robotic logic system was successfully integrated using C++ programming language on the Arduino. This has not only allowed various integrated logics but it has also reduced the testing time by roughly 30%.

High hopes were placed on the robotic logic system but several changes had to be made to the proportional controls. The main issues that arose during the development of the robotic logic system had mainly to do with the fact that we choose to a forward movement based system, i.e. the motors move in the forward direction only. Ordinarily, in 45 degree turns the motors would need a reverse function in other to avoid major variation from the centre of the line. However, we overcame this challenge by sensing the curves before the robot arrives at the turning point and allowing the robot to begin turning calculations and speed deviation.
During the testing phase, we set the proportional controls values KP and KD to 3 and 5 and we had the motors moving at a speed of 130. This affected the line width tolerance and caused problems with the robotic logic. Several values were tested and it was noted that the most suitable values for the KP and KD where 0.15 and 0.3. However for different track widths, the KD value can be adjusted to half of the set KP value.
Sensor calibration was required to ensure that all the values retrieved were useful. The pololu infra-red sensor was analogue in nature, so interpreting the voltage level correctly for any given distance was important. The unmanned robot has to be calibrated before it begins any routine.

Future Improvements and Developments

The line following robot currently moves on its navigating path perfectly. However, there are some parameters that would have to be fined tuned if the path was changed. The sensor would need a longer calibration period to determine the difference between a “T” intersection and a stop command.
Other possible improvements include the refinement of the robotic logic using the sensor reading to accurately navigate a 45 degrees turns without deviation from the centre path. This would be achieved by rewiring the entire system to include a reverse function. Also, the sensor readings can be further refined to enable a universal coordinate system. This would be achieved by using a probability range for each pixel instead of black or white. As the robot travels it would have an idea of its location at any given time and will use the IR sensor reading to update its location.
Another additional feature that should be added to the robot is a pedometer function.

This would be give a better idea of how far the robot moved, rather than relying on sensor position. This is because the sensor position depends upon the incline and the state of the battery. A pedometer would vastly improve the accuracy of the robot's estimated location.

This project has been successfully developed and satisfies the specifications given to us by Warehouse Technology Inc. The Unmanned Warehouse vehicle can be easily reprogrammed to suit other unforeseen circumstances and upgrades in hardware based system.

REFERENCES

"Pololu - 3.k. Pololu QTR Sensor Functions." *Pololu - 3.k. Pololu QTR Sensor Functions*. Pololu Corporation. Web. 11 Apr. 2016. <https://www.pololu.com/docs/0J20/3.k>.

Pololu - Solarbotics GM2 224:1 Gear Motor Offset Output. (n.d.). Retrieved April 11, 2016, from https://www.pololu.com/product/180

"Arduino - ArduinoBoardUno." *Arduino - ArduinoBoardUno*. Web. 11 Apr. 2016. <https://www.arduino.cc/en/Main/ArduinoBoardUno>.