Project Park Finder

The Parallel Parker

Written By: Obinna Gabriel Edeh

obiedeh@yahoo.com

University of Georgia Athens, USA
Spring 2004

For Directed Studies in Robotics: CSCI 4950

Instructor: Walter D. Potter

Professor of Computer Science

Table Of Contents

Acknowledgments.....

Abstract.....

Executive Summary.....

Introduction.....

Tables & Charts.....

Specifications…….....…

Adding More Power to the System…

The Algorithm.....

Source Code……...

Equipment.....…

Summary……

References.....….

ABSTARCT

The idea behind the Parallel Parker Project is to design and build an autonomous miniature automobile robot that can detect a parking space and then park into the space effectively. The robot is controlled with a Handy Board micro controller, and is equipped with seven infrared sensors, used to measure distances and to detect obstacles along its path, and two Lego motors. A lot of effort was made to ensure that accuracy and consistency in this project, enough test and analysis was carried out to ensure that this is possible, however this project is still subject to further tests, analysis and revision.

Executive Summary

Introduction

Our main goal was to design an autonomous miniature automobile robot that can parallel park. To achieve this we had to design a mini car model, equip it with the necessary sensors and apply an algorithm to the micro controller to output the desired result.

Tables & Charts

The section was created to give detailed characteristics on the SHARP GP2D12 sensors used for the project.

Using the GP2D12 with the Handy board

This section deals with interfacing the GP2D12 sensors with the handy board.

Specifications

Detailed specifications for the project, these specifications must be adhered to get desired results and ensure normalcy.

Figures

Figures have been added to better illustrate the basic processes involved in this project; diagrams referenced from other resources are also available, for better understanding of the entire project.

The Algorithm

A pseudo code describing the algorithm behind the parallel parker

Source Code

The Source Code for the parallel parker

Adding more Power to the System

A detailed description of how to go about adding extra power to the Handy board

Equipment

Listings of Equipment used for the project

Introduction

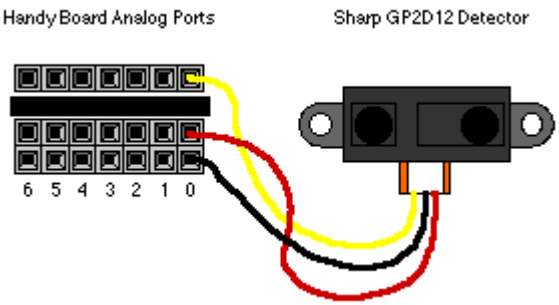
One of the goals of Artificial Intelligence is to help machines find solutions to complex problems by applying characteristics from human intelligence, and applying them as algorithms in a computer friendly way. These complex tasks can then be broken down recursively into simpler tasks, which can be thought of as building blocks.
The parallel parker project is aimed at creating an autonomous automobile robot that can parallel park. Some of the reasons we aspire to achieve this goal is that it can be used as a tool to assist people with disability, beginner drivers and even for the fun of it. For this project we use a mini car prototype equipped with seven sensors and apply an algorithm to the micro controller to output the desired result.

Interfacing the GP2D12 with the Handy board

The major part of the work lies in setting up the hardware so that the individual components can effectively communicate with each other. To interface the GP2D12 range sensor with the handy board, the pull-up resistors will have to be disabled. The analog inputs on the handy board use pull-up resistors, it is necessary to modify the board to disable the pull-ups on any ports being interfaced to the Sharp GP2D12 detector. These pull-ups are provided by a resistor pack that sits in the middle of the 3-port connector for each analog port. There are two ways to modify the board to disable the pull-up(s) and both can be gotten from the sites below respectively http://www.acroname.com, http://www.informatik.uni-oldenburg.de/~trebla/GP2D12/, but the second technique is recommend if you intend to use the expansion board.

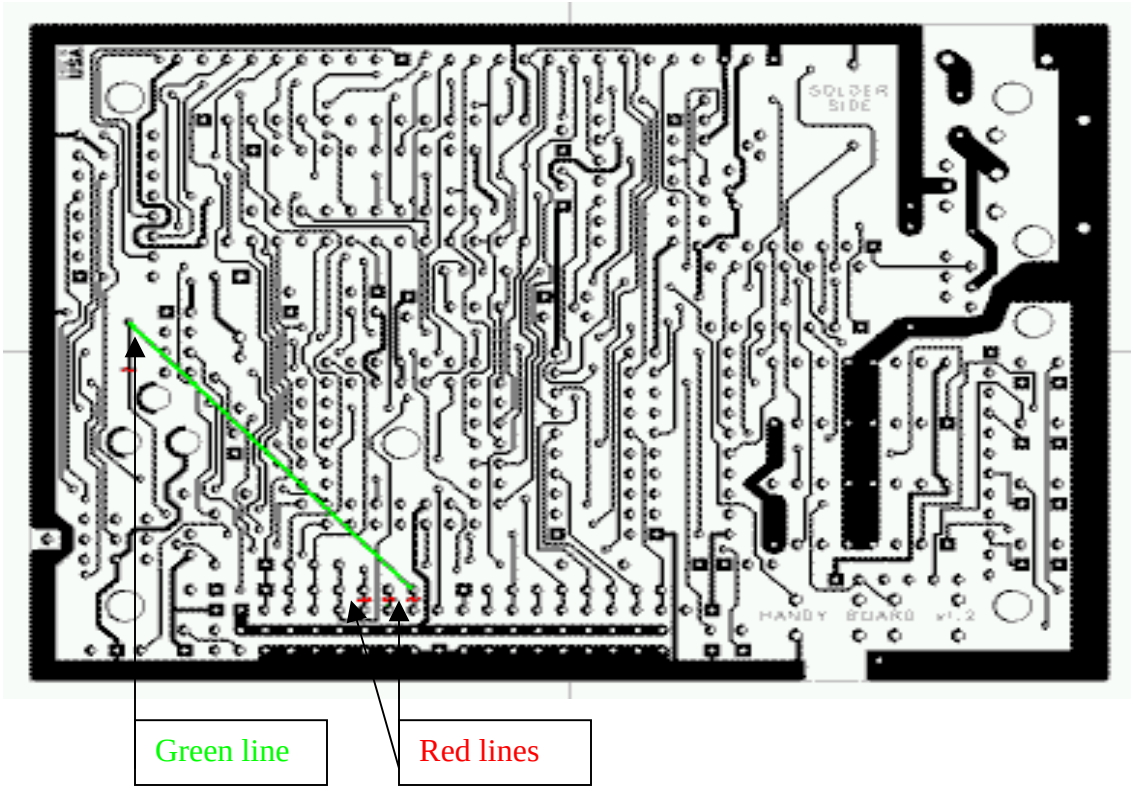
The individual legs of the resistor pack that is directly connected the analog ports that you intend to use, must be cut off. The trace is covered with solder mask so you will need to first cut through the green film and the copper trace as well. Check the discontinuity of the trace with a continuity tester or Ohmmeter to verify that the connection has been cut. Note that configuring port 0 to work with a GP2D12 also enables the analog ports on the expansion board to work with a GP2D12 as well. In the handy board v.1.21, nothing needs to be done on analog port 0.

Other non-GP2D12 sensors can still work without a pull-up resistor, so the ports can still be used for other devices. The major difference is a default reading of 255 with the pull–up resistors and a default reading of 42 without the pull-up resistors when no device is connected to an analog port.



This circuit is very simple and involves just direct wiring of the Sharp GP2D12 to the Handy Board's analog port.

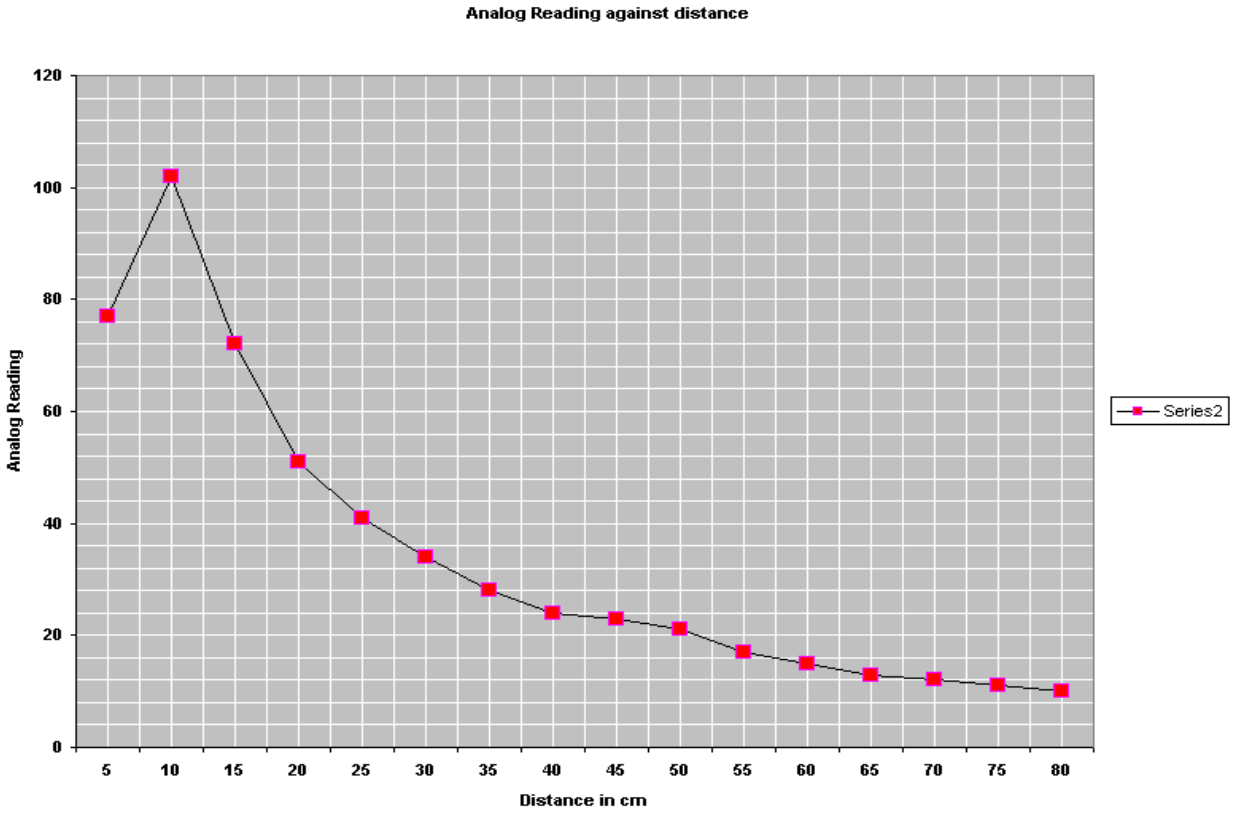
A full schematic diagram of the handy board circuit showing where changes should be made



You need to cut the conductor path at the red marked positions, the red marked positions at the base of the diagram should correspond to the analog inputs you desire to use, then connect a new wire, which is marked green in the picture above. Care should be taken while doing this to avoid damaging the handy board.

You can check the modification by just reading the result of analog(2), analog(4), analog(5), analog(6), and can also check the analog ports on the expansion board, if you have the expansion board connected. Analog ports 0 and 1 will not work while the expansion board is connected because they are multiplexed with the expansion board. The readings should range from 0 to the 100's, when a GP2D12 device is connected to it. To get a better interpretation of these figures you may have to make a chart you can use to map these reading to actual measurements.

Chart and Table showing a typical GP2D12 characteristics



Distance	Analog
5	77
10	102
15	72
20	51
25	41
30	34
35	28
40	24
45	23
50	21
55	17
60	15
65	13
70	12
75	11
80	10

……..

```
// Code Fragment to average Analog Readings from a Port
/* @ port handy board analog port input */
int aveanalog(int port){
    int i= 0;
    int avg= 0;
    /* 5 here is some arbitrary number */
    for(i=1;i<5;i++){
        /* analog(int port) see handy board manual */
        avg+=analog(port);
    }
    avg/=i;
    return avg;
}
```

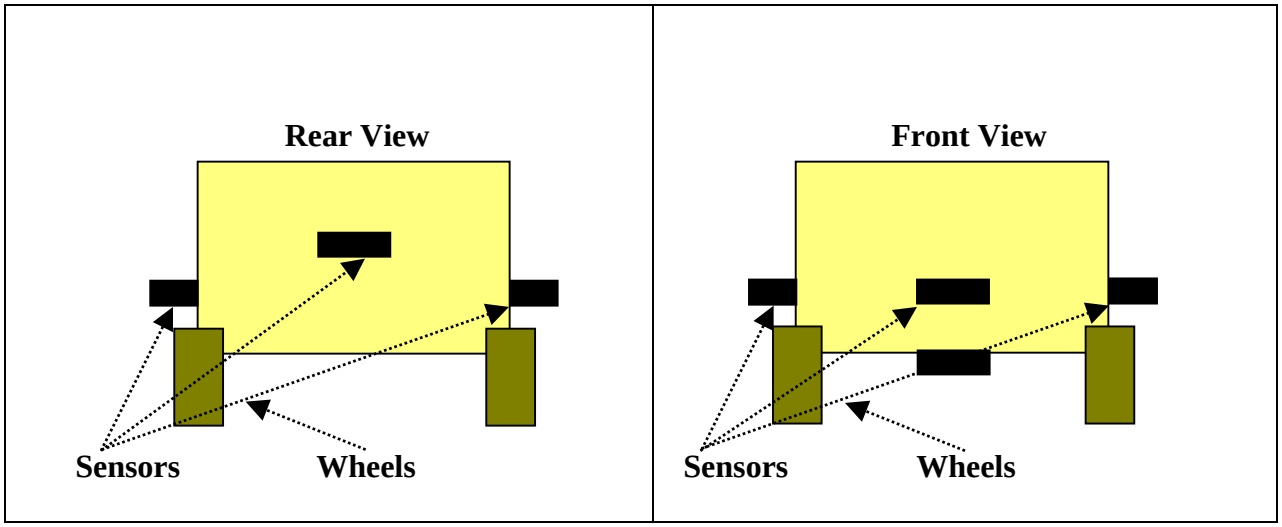
This chart is very much similar to the chart you find on the GP2D12 manual except that the analog values differ, but they typically exhibit the same characteristics under normal circumstances. The normal working range of the GP2D12 is between 10cm to 80cm, the sensors still work beyond this range but will exhibit unpredictable characteristics which conflict with the normal reading when mapped to a chart. So that is why it is better used within the acceptable working range, so there is a distinct value for a particular distance.

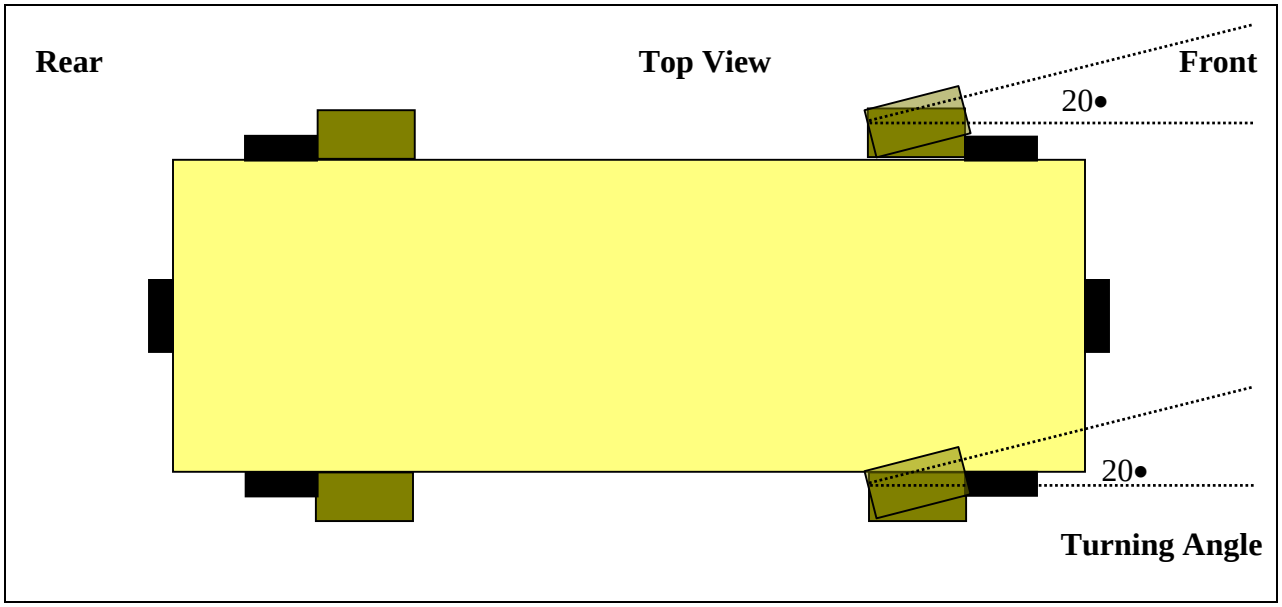
The Algorithm

```
/* Method drive_car used to coordinate all the car events */
drive_car(){
while(true){
        while(there is a clear path in front){
        move forward;
         if(I see a parking space with my right front sensor){
                while(!(I see a parking space with my right rear sensor){}
                        check if there's a clear path in front;
                        stop if there's no clear path;
                        move forward if there is;
                        if(I see a parking space with my right front sensor){
                                while(I see a parking space with my right rear sensor){}
                                        do park parallel parking to the right;
                                        return;
                                }
                        }
        }
        if(I see a parking space with my left front sensor){
                while(!(I see a parking space with my left rear sensor){}
                        check if there's a clear path in front;
                        stop if there's no clear path;
                        move forward if there is;
                        if(I see a parking space with my left front sensor){
                                while(I see a parking space with my left rear sensor){}
                                        do park parallel parking to the left;
                                        return;
                                }
                        }
                }
        }
```

```
        if I am on a table check is I am about to fall;
        stop if true, else continue;
        }
        check is there is a clear path in front;
        stop if false, else continue;

        }
}
```





Parallel Parker Source Code

```
/* @ author Obinna Edeh
// Computer Science Department, University Of Georgia
// For the Park Finder Project a Directed Studies Course
// Directed by Prof Don Potter, Artificial Intelligence Dept UGA
*/

/* Method used to average sesnor readings */
/* @ int port specifies input port selected */
int aveanalog(int port){
        int i= 0;
        int avg= 0;
        for(i=1;i<5;i++){
                avg+=analog(port);
        }
        avg/=i;
        return avg;
}

/* Method attempts to balance the front steering */
/* @ long sec specifies time - Turning left and right */
void center(long sec){
        if(sec<=0L){ off(2); return;}
        bk(2); msleep(sec);
        fd(2); msleep(sec);
        center(sec-20L);
}

/* Method moves car backward steering right */
/* and then left to park car in a right space */
void back_right_park(){
        move_back();bk(2);
        sleep(3.0);
        fd(2); sleep(3.0);
        ao();
}

/* Method moves car backward steering right */
/* and then left to park car in a right space */
void back_left_park(){
        move_back();fd(2);
        sleep(3.0);
        bk(2); sleep(3.0);
        ao();
}

/* Method moves steering right */
void turn_right(){
        bk(2);
}
```

```
/* Method moves steering right */
void turn_left(){
	fd(2);
}

/* Moves car front */
void move_front(){
	fd(1);
}

/* Checks for obstacles in front*/
void check_front(){
	while(aveanalog(2)>70) stop();
}

/* Checks if car can move forward by checking ground sensor */
int check_ground(){
	if(aveanalog(27)<20){ stop(); return 0;}
	else return 1;
}

/* Moves car back at given power level */
void move_back(){
	motor(1,-80);
}

/* Checks for obstacles behind car */
void check_back(){
	while(aveanalog(25)>70) stop();
}

/* stops all movements */
void stop(){
	ao();
}

/* drives car out right then stops */
void drive_out_right(){
	turn_right();
	move_front();
	sleep(0.8);
	ao();
}

/* drives car out left then stops */
void drive_out_left(){
	turn_left();
	move_front();
	sleep(0.8);
	ao();
}

/* Returns true or false evaluating analog reading from the front right */
int seek_front_right(){
```

```
	if(aveanalog(24)>=20 && aveanalog(24)<=45) return 1;
	else return 0;
}

/* Returns true or false evaluating analog reading from the rear right */
int seek_back_right(){
	if(aveanalog(4)>=20 && aveanalog(4)<=45) return 1;
	else return 0;
}

/* Returns true or false evaluating analog reading from the front left */
int seek_front_left(){
	if(aveanalog(5)>=20 && aveanalog(5)<=45) return 1;
	else return 0;
}

/* Returns true or false evaluating analog reading from the rear left */
int seek_back_left(){
	if(aveanalog(6)>=20 && aveanalog(6)<=45) return 1;
	else return 0;
}

/* Checks if detected parking spot on the right side will fit car */
int check_right_size(){
	if(seek_front_right() && seek_back_right()) return 1;
	else return 0;
}

/* Checks if detected parking spot on the left side will fit car */
int check_left_size(){
	if(seek_front_left() && seek_back_left()) return 1;
	else return 0;
}

/* Used to move car an inch up steering to the right */
void inch_up_right(){
	move_front();
	turn_right();
	sleep(0.5);
	stop();
}

/* Used to move car an inch up steering to the right */
void inch_up_left(){
	move_front();
	turn_left();
	sleep(0.5);
	stop();
}

/* Method drive_car used to cordinate all the car events */
int drive_car(){
	while(1){
		printf("I can move\n");
```

```
                while(aveanalog(2)<70){
                        check_front();
                        move_front();
                        if(seek_front_right()){
                                while(!seek_back_right()){}
                                stop(); sleep(2.0);
                                check_front();
                                move_front();
                                if(seek_front_right()){
                                        while(seek_back_right()){}
                                        sleep(0.5);stop();
                                        back_right_park();
                                        stop();
                                        beep();beep();beep();beep();beep();
                                        return 1;
                                }
                        }
                        if(seek_front_left()){
                                while(!seek_back_left()){}
                                stop(); sleep(2.0);
                                check_front();
                                move_front();
                                if(seek_front_left()){
                                        while(seek_back_left()){}
                                        sleep(0.5);stop();
                                        back_left_park();
                                        stop();
                                        beep();beep();beep();beep();beep();
                                        return 2;
                                }
                        }
                        check_ground();
                }
                check_front();
        }
}

/* Method main starts up the program */
void main(){
        int d= 0;
        printf("Press START\n");
        beep();
        while(!start_button()) {}
        printf("WARMING UP ..... IN 3 SEC\n");
        sleep(3.0);
        d =drive_car();
        if(d==1)inch_up_right();
        if(d==2)inch_up_left();
        if(d==0)drive_car();
        printf("I am done\n");
}
```

Adding More Power to the Handy Board

To ensure consistency while operating the robot and to prevent sudden power drop, an additional power unit should be added to the handy board. Note that this is totally different from increasing or decreasing the servo or Lego motor voltage outputs. To use a separate power source for your motors, please refer to the FAQ section on the handy board website. Here we are simply adding more power to the system. The Handy Board's internal battery is rated for 9.6 volts; this is generally adequate for running devices rated between 6 and 12 volts.

Multiple 9v battery units can be added to the handy board but they must be connected in **parallel** to the handy board's power circuit. WARNING! Connecting these batteries in series will damage the handy board.

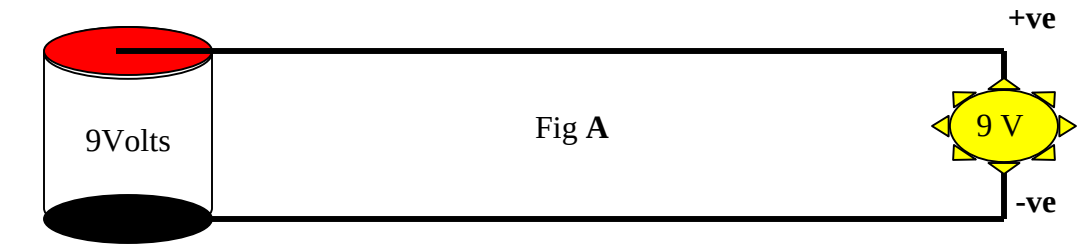


Fig **A**

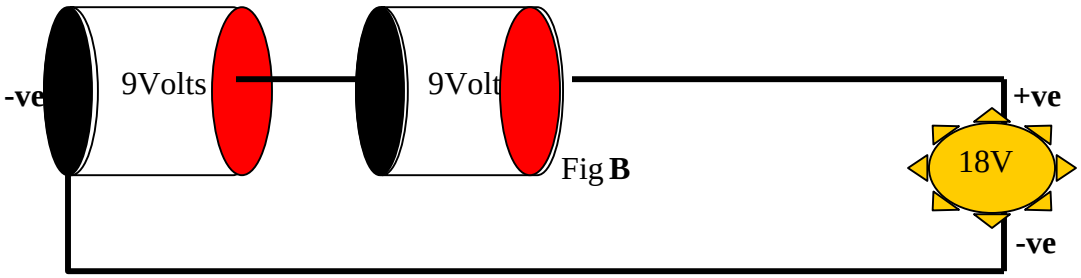


Fig **B**

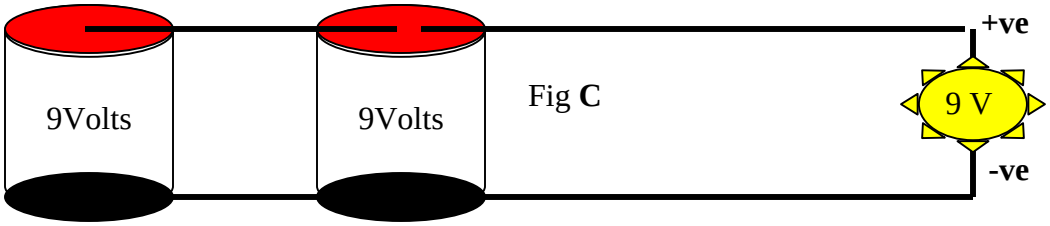


Fig **C**

Fig A. The lamp is connected in parallel across a battery and will yield the normal brightness

Fig B. The batteries are in series and will give the sum voltage of the two batteries. The lamp will be very bright, but may get damaged easily.

Fig C. The batteries are in parallel and will give the same voltage across the circuit. The lamp will glow at normal brightness but will last twice as long as A.

When batteries are connected in parallel the output voltage is the same across the circuit if the all the batteries are the same. However, if the batteries connected in parallel across the circuit have different voltages, the highest voltage battery in the circuit then regulates the effective voltage across the entire circuit until its voltage drops to the same level with the other batteries.

The effective voltage across the circuit V (Total) = V(i) = V(ii) = V(iii) = V(iv) …….

The effective resistance across the circuit I (Total) = I (i) = I(ii) = I(iii) = V(iv) ……..

Recall from Ohms Law, that Voltage (V) = Current (I) x Resistance (R)

Power (P) = V x I

Energy = P x T or V x I x T where T is represents time.

For example, the effective voltage across the circuit in Fig A and Fig C is 9 volts, while effective voltage for Fig B is 18 volts. Clearly we can see that connecting batteries in parallel adds more energy to the circuit because we increase the increase the runtime. The positive and negative terminals of extra battery pack should be soldered to the handy board's +ve and –ve terminal on the solder side. **Red** wires are normally +ve and **Black** wires are normally –ve.

Equipment Used

i Lego Robotics Invention Unit

ii A Handy Board Unit

iii Sharp GP2D12 infrared sensor Unit

iv Interactive Software

v Rechargeable 9.v Batteries

Summary

Multiple tests carried out showed that the robot would mostly attempt to park in an available space if the right conditions were in place. The surrounding environment should be clear of objects that may confuse the robot. It makes about two attempts on doing both the right and to the left, but can be programmed to do this multiple times.

When aligned on a straight line in a space void of surrounding objects the robot would move till it finds a parking space and then park within the space.

Future work on this project may include adding more intelligence to the robot, for example, like returning to the point of origin, and so on.

References http://www.acroname.com

References http://www.informatik.uni-oldenburg.de/~trebla/GP2D12/

References http://www.handyboard.com/